Proc. 11th Online World Conference on Soft Computing in Industrial Applications, September-October, 2006, in Industrial Applications Recent and Emerging Methods and Techniques, Series: Advances in Soft Computing , Vol. 39, Saad, A.; Avineri, E.; Dahal, K.; Sarfraz, M.; Roy, R. (Eds.), Springer, Berlin, 2007, pp.144-155

A Computationally Efficient SUPANOVA: Spline Kernel Based Machine Learning Tool

Boleslaw K. Szymanski[1], Lijuan Zhu[1], Long Han[1], Mark Embrechts[1], Alexander Ross[2], and Karsten Sternickel[2]

[1] Rensselaer Polytechnic Institute, Troy, NY, USA
{szymab,zhul4,hanl4,embrem}@rpi.edu
[2] Cardiomag Imaging, Inc., Schenectady, NY, USA
{aross,karsten}@cardiomag.com

Summary. Many machine learning methods just consider the quality of prediction results as their final purpose. To make the prediction process transparent (reversible), spline kernel based methods were proposed by Gunn. However, the original solution method, termed SUpport vector Parsimonious ANOVA (SUPANOVA) was computationally very complex and demanding. In this paper, we propose a new heuristic to compute the optimal sparse vector in SUPANOVA that replaces the original solver for the convex quadratic problem of very high dimensionality. The resulting system is much faster without the loss of precision, as demonstrated in this paper on two benchmarks: the iris data set and the Boston housing market data benchmark.

1 Introduction

Kernel transformations are frequently used in machine learning methods to transform the input domain into a feature domain so that linear methods can be used to find an optimal solution to the learning problem. The most prominent examples of such methods are Support Vector Machines (SVM [1]) and Partial Least Square (PLS) approaches [2]. Despite their predictive power and efficient implementations, they share a fundamental weakness with other machine learning techniques, namely that although they provide answers, they do not give hints on how these answers were produced or on what basis they were reached. To address this weakness, Gunn and Brown [3] and later Gunn and Kandola [4] proposed to use spline kernels and combine them with a full combinatorial decomposition of the feature set. These models explicitly identify feature subsets that are used in producing the answers. Those subsets can then be used to discern the reasons for predictions. The key element of this approach is a sparse solution to the fully decomposed spline kernel prediction function. Therefore, such a sparse solution can be used for hypothesis forming [4]. However, SUPANOVA is not without its challenges, which arise when an efficient and scalable implementation is desired. These challenges were addressed in our initial work on application of SUPANOVA to machine learning for magnetocardiography [5].

The rest of the paper is organized as follows. Section 2 describes the original SUPANOVA approach published in the literature. Its implementation is discussed in Section 3. Section 4 describes two benchmarks that we used to measure the performance of the implementation and Section 5 provides the results of these measurements. Section 6 offers conclusion and outlines future work in this area.

2 SUPANOVA

Before we discuss the models, we start with some basic definitions and notation. In this paper, we assume that there are N training data points in the form of vectors $x_i = [x^i_1, x^i_2, \ldots, x^i_n] \in R^n$ for $i = 1, 2, \ldots, N$. Each vector represents values of n features and has the corresponding output value, $y_i \in R$. We will denote the matrix containing these vectors (or training data points) as x and the vector of the corresponding output values as y. We assume that the data Mahalanobis scaled [6], that is that each feature has its average and standard deviation computed and then each value is replaced by the difference of the original and average values divided by the standard deviation.

We want to find a function f that is represented by these data points and their values, such that for all training points, as well for any new data point $x_0 = [x^0_1, x^0_2, \ldots, x^0_n]$ with the associated value y_0, we have $y_i \approx f(x_i), i = 0, 1, 2, \ldots, N$. To reconstruct the function f from the training data points we use the following basic kernel model:

$$f(x_0) = \sum_{i=1}^{N} a_i K(x_i, x_0) = K(x, x_0) \cdot a \tag{1}$$

where a kernel function, $K(x_i, x_0)$, yields the value that is a measure of similarity between vectors x_i and x_0 and kernel vector:

$$K(x, x_0) = [K(x_1, x_0), \ldots, K(x_N, x_0)].$$

$a \in R^N$ is the usual weight of data point vector and $\cdot$ denotes the dot product.

The basic idea of the SUPANOVA method based on spline kernels is to represent the solution to a machine learning problem as a sum of kernels that decompose functions of the order n into a sum of terms that are 1-ary, 2-ary,. . . ,n-ary order functions of the original arguments. Each function higher than first order uses a product of spline functions to represent its arguments.

This basic model (1) can be extended by replacing kernel function with a sum of kernels $K_j(x_i, x_0)$ with each one measuring similarity of vectors x_i, x_0 on a subset of features, with $M = 2^n - 1$ we get: $f(x_0) = \sum_{j=0}^{M} c_j K_j(x, x_0) \cdot a$, where $c_j \geq 0$. In this representation, a linear sum of kernels weighted by nonnegative coefficients c_j is used in which each kernel K_j yields the value $K_j(x_i, x_0)$ that defines the j^{th} component of ANOVA decomposition [4] in which an order n function $g(u)$ over an n element vector u is represented as: $g_0 + \sum_{i=1}^{n} g_i(u_i) + \sum_{i<j} g_{i,j}(u_i, u_j) + \ldots + g_{1,2,\ldots,n}(u_1, u_2, \ldots, u_n)$. The appropriate multivariate

ANOVA kernel of two vectors u, v is given by a tensor product of a univariate kernel plus a bias term as follows:

$$K_A(u,v) = \prod_{i=1}^{n}[1 + k(u_i, v_i)] = 1 + \sum_{i=1}^{n} k(u_i, v_i) \ldots + \prod_{i=1}^{n} k(u_i, v_i).$$

As an operator, following Kandola [7], we will use a spline kernel in the form of a piece-wise cubic polynomial: $k_{spline}(u_i, v_i) = u_i v_i + \frac{(u_i+v_i)\min(u_i,v_i)}{2} - \frac{\min(u_i,v_i)^3}{6}$. The resulting kernel is used in a SUPANOVA technique first proposed by Gunn and developed by Kandola [7].

Clearly, the number of potential terms in this representation is very large, $M = 2^n - 1$. However, vector c should be very sparse, so only a few terms in the decomposition will be significant.

The loss function for this kind of representation consists of three (and not two as is the case in traditional kernels) terms:

1. The error of modeling that measures the distance of the prediction from the real results (equal to traditional kernels) and for a quadratic loss can be expressed as $\left|y - \sum_{j=0}^{M} c_j K_j \cdot a\right|_2^2$.
2. The smoothness of the representation, defined by vector a as $\lambda_a \sum_{j=0}^{M} c_j a^T \times K_j \cdot a$. is again the same term as used in traditional kernels to maintain generalizibility of the representation.
3. The sparseness of the representation is specific to the spline kernels, and is defined ideally by the number of non-zeros in vector c, that is by zero-norm of the vector c as $\lambda_c \sum_{j=0}^{M} |c_j|_0$. Since the optimization with zero-norm (that is not differentiable) is difficult, Gunn and Kandola [4] approximated it with the first-norm and used the absolute value of the sum of all vector c elements as a measure of sparseness. Even this first-norm metric leads to the optimization problem that is difficult to solve because of very large numbers of feature subsets that arise even for modest numbers of features. Indeed, for a problem with n features, the corresponding optimization problem would be of 2^n dimensionality. Therefore even for relatively modest numbers of features in the range of 30 to 50, the corresponding optimization problem would have 10^9 to 10^{15} dimensions and would be expensive to solve.

Hence, for the quadratic loss for the entire training data set, the error function is just the sum of the three defined above terms: $\Phi(a,c) = \left|y - \sum_{j=0}^{M} c_j K_j \cdot a\right|_2^2 + \lambda_a \sum_{j=0}^{M} c_j a^T \times K_j \cdot a + \lambda_c \sum_{j=0}^{M} |c_j|_0, \quad c_j \geq 0$. The sparseness and smoothness terms are weighted by regularization parameters λ_a and λ_c, that have to be identified heuristically as they strike the balance between the quality of the predictions on the training data (that is the distance between the predicted and real values for each training data point) and the model's ability to generalize to the testing or new data (that is the distance between the predicted and real values for each testing or new data point).

3 SUPANOVA Implementation

λ_a and λ_c are not initially known but need to be found. In an iterative search for those values, both λ_a and λ_c should initially be set large and reduced gradually (by 2-3 percentage points of the initial value per iteration). Based on Kandola's experience that subsequent iteration steps did not change the solution significantly [7], we follow a solution procedure that consists of an initialization step followed by the single iteration.

Initialize: $c' = 1$ $a' = argmin_a(\Phi(a, c'))$. **Single step:** $c^* = argmin_c(\Phi(a', c))$.

The single iteration step used in [4] included also recomputation of the optimal smoothness vector a for the newly found sparseness vector c, but our experience indicates that this additional step does not improve the solution, so we are not executing it in our solution.

In the initialization step, regularization parameter λ_c is not used, so it does not need to be determined in the respective optimization. λ_a is determined by cross-validation (like [4], we use an automatic search procedure to locate a local minimum of the validation error in 8-fold cross-validation runs). The difficult part of the optimization of the loss function is determining the regularization parameters in the single step that computes vector c. Following [4], we use a method based on the best empirical performance. We set $\lambda_a = 0$ and select such λ_c that the loss is equal to the loss of the validation error in the initialization step. As the result, λ_c can be readily computed from the following equation. $\lambda_c = \frac{\lambda_a \sum_{j=0}^{M} c'_j a'^T \times K_j \cdot a'}{\sum_{j=0}^{M} c'_j} = \frac{\lambda_a}{M} \sum_{j=0}^{M} a'^T \times K_j \cdot a'$. This leads to the following revised procedure.

S0: $c' = 1$

S1: $a' = argmin_a(\Phi'(a))$, where $\Phi'(a) = \left| y - \sum_{j=0}^{M} K_j \cdot a \right|_2^2 + \lambda_a a^T \times \sum_{j=0}^{M} K_j \cdot a$, and λ_a is computed by minimizing (possibly locally) validation error in 8-fold cross validation procedure (for problems in which $N < 14$, we use $N/2$-fold cross validation procedure instead).

S2: $c^* = argmin_c(\Phi''(a', c))$, $\Phi''(a, c) = \left| y - \sum_{j=0}^{M} c_j K_j \cdot a \right|_2^2 + \lambda_c \sum_{j=0}^{M} |c_j|$, and λ_c set so the loss is the same as it is in the initialization step.

In step **S1**, by taking the partial derivatives for all elements of vector a, we obtain a system of N linear equations with N unknowns that we represent as $d \times a = f$, where d is a matrix of coefficients for the unknown vector a elements and f is a vector of free terms for those linear equations. Hence, $f_l = 2 \sum_{j=0}^{M} c'_j \sum_{i=1}^{N} y_i K_j(x_i, x_l)$ and each element of matrix d is defined as $d_{l,p} = \sum_{j=0}^{M} c'_j [2 \sum_{i=1}^{N} K_j(x_i, x_l) \sum_{q=0}^{M} c'_q K_q(x_i, x_p) + \lambda_a(K_j(x_l, x_p) + K_j(x_p, x_l))]$.

In step **S2**, Kandola in his Ph.D. thesis [7] replaced the sparseness term with the sum of the elements of vector c (so effectively replaced the zero norm with the first norm measure). This change allowed him to use a convex quadratic problem solver, called BPMPD [8] based on a robust interior point solver technique. However, it is not scalable to a very large number of variables (in our case $M = 2^n - 1$ grows very rapidly with the number of features n). The size of the quadratic problem that we face for realistic data is prohibitively large (when $n > 20$, $M = 2^n - 1 > 10^6$) even for the most modern quadratic problem solvers and computers.

Fortunately, the minimization problem of step **S2** is a very special case of a quadratic problem in which the number of quadratic terms (N) is much smaller than the number of linear terms (M). This fact guided our development of a heuristic that uses a greedy selection of the non-zero entries in vector c, one-by-one, and measures the corresponding error of the solution until this error cannot be further minimized. Indeed, the difficult part of optimization needed in step **S2** is determining which elements of vector c should be non-zero. Once we know that, finding their values is just a matter of solving a set of linear equations of the order equal to the number of non-zero elements chosen. We also know that this number is limited by the number of data points, that is N. To make selection efficient, the heuristic assumes that the subvector of non-zero entries of size $k + 1$ that minimizes the loss function in step **S2** is simply the subvector of non-zero entries of size k that minimizes the loss function plus one more non-zero elements. This approach is often termed a "greedy" selection. The results that we obtained from benchmarks confirmed that a vector selected in such a way is close to the vector that minimizes the loss function in step **S2**.

The additional advantage of the heuristic is that it allows us to use exact value of the sparseness term in the loss function, that is zero norm of the vector c elements. Indeed, as in each step of heuristic the number of non-zero elements is constant, only their values are recomputed, so non-differentiability of the loss function sparseness term does not impact our optimization. Hence, comparing ours with Gunn's approach, we notice that we optimize approximately (by using greedy selection of the vector c non-zero elements) the exact loss function, whereas Gunn optimizes exactly (by using a quadratic problem solver) the approximate expression of the loss function.

The final advantage of the heuristics is its memory efficiency. The size of the sparse kernel array is $O(MN^2)$, however, as shown later, our heuristic requires storing only one element for each column of the spline kernel. Hence, the size of the storage needed for our implementation is just $O(MN)$. This reduction in the memory size by a factor of N enables us to solve much larger problems than solvable on the same machine using the quadratic problem solver. The gain in computational and memory efficiency combined with the quality of the solutions that we obtained justify our approach.

The computation in step **S2** proceeds as follows.

First, we compute λ_c such that the loss is equal to the loss of the validation error in the initialization step, assuming that the maximum number of non-zero elements in vector c is N (as this is the number of independent equations in the corresponding optimization problem which we use to compute the values of non-zero elements of vector c), so $\lambda_c = \frac{\lambda_a}{N} \sum_{j=0}^{M} a'^T \times K_j \cdot a'$.

The loss function contains N approximation error terms of the form

$$\left[y_i - \sum_{j=0}^{M} c_j \sum_{k=1}^{N} K_j(x_i, x_k) a_k \right]^2 = \left[y_i - \sum_{j=0}^{M} c_j P_{i,j} \right]^2 ,$$

where $P_{i,j} = \sum_{k=1}^{N} K_j(x_i, x_k)a_k$. Assuming that only $k \leq N$ positive elements of vector c should be selected and all other elements should be set to 0 (so this is the k-th step of the "greedy" selection in our heuristic), the choice is easy. Taking the derivatives for all non-zero elements selected, each derivative for some non-zero element c_p yields the equation $\sum_{i=1}^{N} \sum_{j=0}^{k} c_j P_{i,j} P_{i,p} = \sum_{i=1}^{N} y_i P_{i,p}$. If solving this set of k equations with k unknowns, we obtain all the non-zero values positive, then we can easily compute the corresponding error as $\sum_{i=1}^{N} \left[y_i - \sum_{j=0}^{M} c_j P_{i,j} \right]^2 + k\lambda_c$. To select the next non-zero value in vector c, we can now substitute the found optimal values of the k elements of vector c so far selected, computing the new vector y^k of approximation error terms in the loss function as $y_i^k = y_i - \sum_{j=1}^{k} c_j \sum_{m=1}^{N} K_j(x_i, x_m)a_m$. Then, we can compute the derivative for each of the $M - k$ unselected elements c_r's of vector c as $\sum_{i=1}^{N} c_r P_{i,r}^2 = \sum_{i=1}^{N} y_i^k P_{i,r}$ to find the optimal value of c_r and the corresponding value of the loss function. Selecting that element c_r that yields the smallest approximation error term of the loss function, we can extend the non-zero value subvector of c with the newly found non-zero element and continue the heuristic.

The more detail description of the implementation of step **S2** follows.

1. **Initialization.** We create initially empty set S of all selected elements of vector c that are positive, and a set E contains all the remaining elements of vector c.
2. **Selection.** We select, one by one, elements e_r in set E and compute, according to Equation 3 the minimum value of the loss function with this selection. Then, we choose that element e_r that yields the smallest minimum among all elements of set E and move this element from set E to set S.
3. **Adjustment.** With the newly created set S, we compute the solution to the set of linear equations obtained by taking derivatives of the error expression for the elements of set S. If all elements of the solution are non-negative, the solution is accepted. Otherwise, the previously found solution is retained and the heuristic stops.
4. **Control loop.** If set E becomes empty, heuristic stops, otherwise step 2 is executed with the extended set S.

In pseudo-code, the heuristic can be written as follows.

```
$S = \emptyset$, $E = \{c\}$, $k = 1$, toterr=$\sum_{i=1}^{N} \left[ y_i - \sum_{j=0}^{M} \sum_{p=1}^{N} K_j(x_i, x_p)a_p \right]^2 + \lambda_c$
do {
    $e_1^{opt} = \sum_{i=1}^{N} y_i^k P_{i,1} / \sum_{i=1}^{N} P_{i,1}^2, kerror = \sum_{i=1}^{N} \left[ y_i^k - \sum_{p=1}^{N} e_1^{opt} P_{p,1} \right]^2$
    for r=2 to $|E|$ do
        $e_r^{opt} = \sum_{i=1}^{N} y_i^k P_{i,r} / \sum_{i=1}^{N} P_{i,r}^2, kerror = \sum_{i=1}^{N} \left[ y_i^k - \sum_{p=1}^{N} e_r^{opt} P_{p,r} \right]^2$
        if error<kerror then kerror=error, ksel=i endif
endfor
$S = S \cup \{e_{ksel}\}$
compute $k$ derivatives and solve a system of k linear equations
if all elements of $S$ are positive then
```

```
    error=$\sum_{i=1}^{N}\left[y_i - \sum_{j=0}^{k} s_j \sum_{m=1}^{N} K_j(x_i, x_m)a_m\right]^2$
    continue=(error<toterr)
endif
if continue then k=k+1, toterr=kerror+$k\lambda_c$ endif
until continue
```

Let $s(M)$, which we will call a sparse factor, denote the number of non-zero entries in the final sparse vector c of size M. Since we have at most N approximation terms in the loss function, then clearly $s(M) \leq N$. The computational complexity of the entire computation is $O(MN^2)$. Indeed, the spline kernel has $O(MN^2)$ elements, which of each has to be computed. In step **S1**, we solve the set of N linear equations repeatedly to find the optimal value of regularization parameter λ_a, operation of complexity $O(N^3) = O(MN^2)$ for $M > N$. Finally, in step **S2**, we compute coefficients $P_{i,j}$, again an operation of complexity $O(MN^2)$, then we solve $s(M)$ sets of linear equations varying in size from 1 to $s(M)$, so operation of complexity $O(s(M)^3)$. Since $s(M) \leq N < M$ then also the last operation is no worse than $O(MN^2)$. The memory complexity is also reasonable. It is easy to notice that in step **S1**, all elements of vector c are equal to 1, so instead of the full kernel, we need to store only the sum of all values along the vector c dimension, in total $O(N^2)$ memory requirement. In step **S2**, the only arrays $P_{i,j}$ and similarly sized arrays storing the sum of squares of elements $P_{i,j}$ and products of elements $P_{i,j}$ by elements y_i^k need to be stored. So in total only $O(MN)$ data items need to be stored. This is a huge improvement over storing entire spline kernel that would require memory by factor N larger.

4 Performance Metrics and Data Benchmarks

We use two metrics evaluate the prediction performance. The first one is the Root Mean Square Error index or RMSE, which is defined as the average value of the squared error: $RMSE = \sqrt{\frac{1}{n}\sum_i(\hat{y}_i - y_i)^2}$, where $\hat{y}_i = f(x_i)$ is the predicted value for the data vector x_i. The second one is r^2, defined as the correlation coefficient squared between target values and predictions for the response, $r^2 = \frac{(\sum_{i=1}^{n_{train}}(\hat{y}_i-\bar{\hat{y}})(y_i-\bar{y}))^2}{\sum_{i=1}^{n_{train}}(\hat{y}_i-\bar{\hat{y}})^2\sum_{i=1}^{n_{train}}(y_i-\bar{y})^2}$. Two related metrics, $q^2 = 1 - r^2$ and $Q^2 = 1 - R^2$, are used to assess the performance of validation or test data. The smaller the values for q^2 the better. Ideally, both values should be similar. Detailed information about these metrics is given in [6].

The model quality was measured on two machine learning benchmarks: the petal iris data and the Boston housing market data. Both data sets are members of the UCI Repository for Machine Learning Data-Bases [9] and are described below.

Petal iris data defines a classification problem with linear relationship. We use this simple data set to verify the implementation. This is dataset is widely used in the pattern recognition literature and was adopted from a classic paper in botany [10] perhaps because of its simplicity for classification. The dataset contains 3 classes of 50 instances each, for a total of 150 data points. Each class refers to a type of iris plant. One class is linearly separable from the other two that are not linearly separable from each other.

The Boston housing dataset has been extracted from the work of Harrison and Rubinfield [11] who were studying the effects of air pollution on housing prices. The data provides the median price for 1970 of owner-occupied houses in 506 census tracts within the Boston metropolitan area. Thirteen features characterizing each census tract are available for use in predicting the median price, including crime rate, mean no. of rooms, distance to job center, etc.

5 Experimental Results

The described above two data sets were subject to many machine learning experiments, including [4, 7, 12]. For this reason, we have chosen them to illustrate the performance of the SUPANOVA method in general and our heuristic in particular.

The results from processing the petal iris dataset using our initial implementation are shown in Figure 1. The available data were split as follows: 50 samples were chosen for training and 30 samples are used for validation. This data set has only four features so only 2^4 elements in vector c were created. The optimal solution uses nine out of these 16 elements. Values of q^2 and Q^2 indicate a high quality of the results, as shown in Figure 1. The positive and negative cases are separated nearly perfectly. Measuring from the ROC curve, the goodness of fit is close to 1.

The results from the Boston housing market data set are shown in Figure 2. They display very similar, high quality results as shown in the previous figures. This data set has a large number of dimensions, 13, and therefore a relatively high number of feature combinations ($2^{13} = 8192$). The non-linearity of the data is limited, so the model that we ran included only empty feature set, single features and pairs of features, yielding 92 feature combinations in total. From this set of 92 potential features, the heuristic selected only 40 features for the solution, as indicted by the number of non-zero elements in vector c. Hence, sparseness of the solution, $40/8192 \approx 0.5\%$, was excellent. Finally, despite of the large size of the model, the results are good, as indicated by the high values of q^2 metrics and Q^2 metrics.

It is interesting to note that in these plots, the largest divergence between the measured and predicted values is at the extremes of the pattern sequence numbers. This is not surprising, because prices of the most expensive houses were censored at the value of $50k$ causing the model to predict wrongly for them.

To simplify the analysis of input variables, we introduce a parameter in the algorithm to control the maximum number of features in feature combinations considered in the solution. In general, it could be expected that the significant elements of vector c may correspond only to single features and their binary and triple combinations. As already mentioned, the results for Boston housing market data set are obtained using at most binary combination of features. To

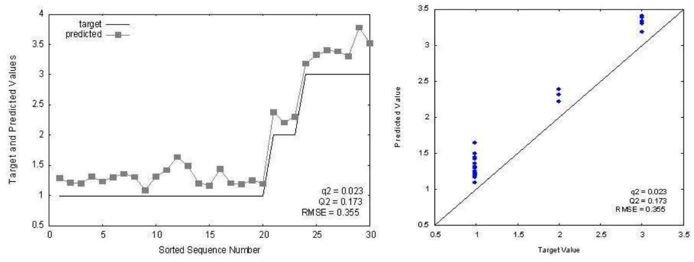

Fig. 1. Predicted vs. target values and scatterplot for the petal iris test data

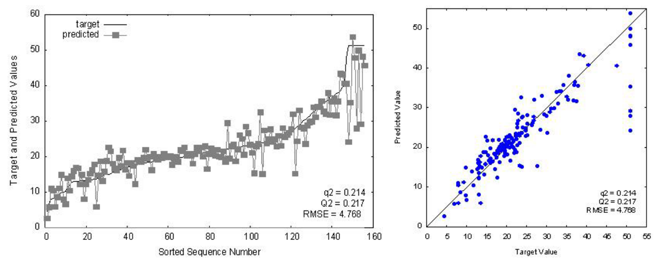

Fig. 2. Processing results for the test Boston housing market data; closeness of the data points to the diagonal in the scatterplot indicates quality of predictions

investigate if adding triple feature combination improves the quality of the prediction, we extend feature space to include 378 feature combinations, including all possible triples, but there are no obvious improvements in the predictions. We may conclude that most of the information for Boston housing market data lies in feature compositions that are generated up to binary combination of features. This result is significant as it yields an extreme reduction in the search space for vector c while retaining good quality of predictions. This is useful for pruning features based on relatively simple binary combination of inputs. In fact, this could be a basis of an efficient and systematic approach to pruning, in which we start with binary feature combinations and then continue with the increasing numbers of features in combinations until no further improvement to the quality of the solution can be obtained. As we expect this iteration to finish after three to five steps, (as it did in the test cases), the size of vectors c is greatly reduced from $M = 2^n - 1$ to M in the range $\frac{n^3}{6}$ to $\frac{n^5}{120}$, making analysis of problems with up to 100 features feasible.

Our results agree with both Husmeier (who used an ensemble of Bayesian neural networks trained using an Expectation-Maximization (EM) algorithm and incorporating automatic relevance determination (ARD) [13]) and Gunn and Kandola (who used SUPANOVA [4]). concluded that Feature 1 in the Boston housing market data, "Crime rate", seems to have a low-impact on house prices, and this conclusion is confirmed in our approach. Likewise, Feature 2, "Percentage of residential land", was also found not to be significant. More interesting is a comparison with the original SUPANOVA analysis provided by Gunn [4]. The paper listed the following single features as significant: mean number of rooms, percentage built before 1940, distance to job center, and percentage of minorities. and binary combination of features with the mean number of rooms and with percentage built pre 1940 as the most important in the 14 non-zero vector c entries selected, but the paper did not provide the prediction quality measurements. On the other hand, the quality of prediction of our implementation of SUPANOVA methods matched other advanced machine learning such as KPLS kernels [6]. Our implementation selected only one single feature: mean number of rooms. It also selected the same binary combinations with mean number of rooms and with percentage built pre 1940 as the Gunn and Kandola's implementation [4] did. In total, both approaches selected 14 non-zero entries in vector c and agreed on 9 of those selections. These results demonstrate that our new heuristic achieves a high accuracy of predictions while preserving the transparency of the solution supported by the original SUPANOVA implementation. We also noticed that the adjustment improves significantly the quality of the results, as measured by the reduction in the total error by over 30% compared with the unadjusted solution. At the same time, the sparseness of the solution can be easily adjusted because of the sequential growth of the size of vector c. For example, the last 20 elements of vector c, that is elements $21 - 40$, improve the final result by less than 3% and therefore can be discarded, improving sparseness of the solution.

Another interesting conclusion is that the values of the c vector element mainly decrease in the order of their selection. For example, the values of the first 20 elements selected by the heuristic vary from 4.86 to 0.36, whereas values of the next 20 elements range from 0.11 to 0.29. This is not surprising, as our heuristic was designed to do exactly that, meaning that it selects the non-zero elements in the order of their importance for the solution. The fact that the order of the selection is very strongly correlated with the magnitude of the impact confirms that the heuristic works correctly, and in fact the difference between the order of selection and the order of magnitude of the value of non-zero elements is a measure of optimality of the heuristics. In fact, at the moment when the heuristic completes, we can compute the largest shift of an element in the vector c in this order to estimate how many additional steps the heuristics should be run to make sure that no better non-zero vector exists. Such an extended vector, if found, can than be truncated using the following observation. Since the magnitude of the entry in the vector c is correlated with its impact on the solution, the corollary is that the elements with small magnitude can be dropped without decreasing the quality of the approximation error but improving the sparseness of the solution. This is important, because the more sparse the solution is the more transparent the model becomes, thereby increasing the value of spline kernel solutions to the users.

Finally, it should be noted that the presented heuristic is capable of eliminating linearly or nearly linearly dependent feature combinations (including single features). Indeed, if two feature sets are linearly dependent, only one of them would be selected as having a non-zero coefficient in vector c, whereas the other (or others if there are more than one) would have their coefficients kept at zero, as the inclusion of any of them would not improve the solution.

6 Conclusion and Future Work

So far we have used the benchmark data to examine a new implementation of the SUPANOVA technique. our current work focuses on applying the developed sparse kernel implementation with the new heuristic presented above to biological data, such as magnetocardiograms. In this special case, the number of data points, N, (the number of patients tested) is now relatively modest (on the order of a few hundred, but it is expected to grow quickly to a few thousand). This range of values of N is not a problem from a computational point of view for the current implementation to handle. The challenge is that the number of features that dictates the number of the feature subsets defining the size of the sparseness vector, c. There is a need to limit the number of features considered to $100 - 200$, as larger numbers of features would result in our implementation exceeding the current limits of memory in modern workstations (for example, with triples of 200 features and $1,000$ data points in the training set, our implementation requires about 8 GB of memory). Our heuristic is instrumental here, as without it, the same 8 GB memory workstation would be able to process at most 23 feature problem with $1,000$ data points in the training set and triples of features using the quadratic problem solver [14].

Acknowledgement

Content of this paper does not necessarily reflect the position or policy of the U.S. Government - no official endorsement should be inferred or implied.

References

1. Vapnik, V.: The Nature of Statistical Learning Theory. Springer, New York, NY (1995)
2. Wold, S., Sjstrm, M., Eriksson, L.: PLS-Regression: a Basic Tool of Chemometrics. Chemometrics and Intelligent Laboratory Systems **58** (2001) 109–130
3. Gunn, S., Brown, M.: Supanova - a sparse, transparent modelling approach. Proc. IEEE Workshop on Neural Networks for Signal Processing, Madison, WI (1999) 21–30
4. Gunn, S., Kandola, J.: Structural Modelling with Sparse Kernels. Machine Learning **48**(1) (2002) 137–163
5. Embrechts, M., Szymanski, B., Sternickel, K., Naenna, T., Bragaspathi, R.: Use of Machine Learning for Classification of Magnetocardiograms. Proc. IEEE Conference on System, Man and Cybernetics Washington DC (2003) 1400–1405

6. Embrechts, M., Szymanski, B., Sternickel, K.: Introduction to Scientific Data Mining. Computationally Intelligent Hybrid Systems: The Fusion of Soft Computing and Hard Computing. Wiley, New York, NY (2005)
7. Kandola, J.: Structural Modelling with Sparse Kernels. Department of Electronics and Computer Science, University of Southampton, U.K (2001)
8. Meszaros, C.: The BPMPD interior point solver for convex quadratic problems. TR WP 98-8, Computer and Automation Institute, Academy of Sciences, Budapest, Hungary (1998)
9. Merz, C.J., Murphy, P.M.: UCI repository for machine learning data-bases. http://www.ics.uci.edu/ mlearn/MLRepository.html (1996)
10. Fisher, A.: The use of multiple measurements in taxonomic problems. Annual Eugenics **7** (1936) 179–188
11. Harrison, D., Rubinfield, D.: Hedonic housing prices and the demand for clean air. Journal of Enviromental Economics and Management **5** (1978) 81–102
12. Neal, R.: Bayesian Learning for Neural Networks. Springer, New York, NY (1995)
13. Husmeier, D.: Neural Networks for Conditional Probability Estimation. Springer, New York, NY (1999)
14. Szymanski, B., Han, L., Embrechts, M., Ross, A., Sternickel, K., Zhu, L.: Using efficient supanova kernel for heart disease diagnosis. Proc. 16th Conf. Artificial Neural Networks Industrial Engineering, ANNIE (2006)