Monograph:
Frieb, Thilo-Thomas and Harrison, R.F. (1998) Perceptrons in Kernel Feature Spaces. Research Report. ACSE Research Report 720 . Department of Automatic Control and Systems Engineering

Perceptrons in Kernel Feature Spaces

T-T Frieß and R.F. Harrison

Date: 16. July 1998

Research Report 720

The University of Sheffield,
Dept. of Automatic Control and Systems Engineering
Mappin Street, S1 3JD, England

Abstract: The weight vector of a perceptron can be represented in two ways, either in an explicit form where the vector is directly available, or in a data dependent form where the weight is represented by a weighted sum of some training patterns.
Kernel functions allow the creation of nonlinear versions of data dependent perceptrons if scalar products are replaced by kernel functions. For Muroga's and Minnick's linear programming perceptron, a data dependent version with kernels and regularisation is presented; the linear programming machine which performs about as well as support vector machines do by only solving *linear* programs (support vector learning is based on solving *quadratic* programs). In the decision function of a kernel-based perceptron nonlinear dependencies between the expansion vectors can exist. These dependencies in kernel feature space can be eliminated in order to compress the decision function without loss by removing redundant expansion vectors and updating multipliers. The compression ratio obtained can be considered as a complexity measure similar to, but tighter than, Vapnik's leave-one-out bound.

Key Words: Kernel Perceptron Machines, Linear Programming Machines, Minimal Expansion Set Reduction, Kernel Adatron, Support Vector Machine

1 Introduction

The weight vector of a perceptron can be represented in two ways, either in an explicit form where the vector is directly available, or in a data dependent form where the weight is represented by a weighted sum of some training patterns. Kernel functions allow the creation of nonlinear versions of data dependent perceptrons if scalar products are replaced by kernel functions.
The linear programming machine (LPM) is a novel algorithm. It operates in batch mode to learn a perceptron with kernels using weight decay regularisation. It is simple to implement, and its performance is comparable to that of support vector machines (SVM). In LPMs, learning is based on optimising *linear* cost functions while in SVMs a *quadratic* cost functions must be optimised. Quadratic programming is nontrivial to implement, and may be subject to some stability problems.
After learning a data dependent perceptron (e.g. a SVM) the decision function may contain nonlinear dependencies between the expansion vectors (as pointed out in [5], [29]). It is shown how these dependencies in kernel feature space can be identified in order to compress the decision function *without loss.* This allows the determination of the *true* compression ratio which is upper bounded by Vapnik's leave-one-bound and leads to a speedup of the learning machine in prediction mode.

This document is strucured as follows. In the following section perceptrons will be rewritten in the data dependent representation. It is shown how kernels can be "plugged into" a Rosenblatt-perceptron in order to create nonlinear algorithms. This result has been known for decades [1]; parts are presented again to underline the crucial link between this work and the linear programming machine, as well as to Vapnik's support vector machine. In section 3 it is shown how data dependent decision functions can be compressed without loss by removing redundant patterns. Finally, section 4 presents the results of computer experiments which allow the comparison of the performance of the LPM to that of KNN classifiers, kernel perceptrons, SVMs, and kernel Adatrons.

2 Kernel Perceptrons

2.1 Rosenblatt's Perceptrons

Rosenblatt's learning algorithm for perceptrons [25] can find a linear discriminant function, $f(x)$, for a given set of labelled training patterns. Each training pattern, x_i, is a vector in R^d and has a label $y_i \in \{+1, -1\}$ $\forall i \in \{1..l\}$.
The vector w of the linear decision function

$$f(x) = \langle w, x \rangle \tag{1}$$

is found by the following algorithm:

Rosenblatt-Perceptron

1. Choose a starting point (e.g. $w_d = 0 \; \forall d$)
2. WHILE not all patterns are correctly classified
3. choose pattern $x_i (i \in 1..l)$
4. If NOT $sign(f(x_i)) = y_i$ update w by: $w \leftarrow w + \eta x_i y_i$ END IF
5. END WHILE

For the learning rate, η, usually a small positive value is chosen (or alternatively η is set to 1 and training patterns are re-scaled). Similar algorithms (e.g. neural networks) sometimes use an adaptive learning rate which can vary during the learning process.
If it is possible to classify all training patterns correctly, the algorithm terminates in a finite number of iterations. Then a weight vector, w, has been found such that $y_i = sign(f(x_i)) \; \forall i$. Clearly then the weight vector, w, may be expressed as a weighted sum of patterns: $w = \sum_{i=1}^{l} \alpha_i x_i$ where the α_i are multipliers for individual patterns, x_i. Another way to expand the vector, w, is given by: $w = \sum_{i=1}^{l} y_i \alpha_i^* x_i$, in this case the α_i^* are non-negative integers.

Similar to perceptrons, the expansion of w onto a weighted sum of some training patterns also holds for Vapnik's support vector (SV) machine - therefore the SV machine can be regarded as a species of the Rosenblatt-perceptron, or alternatively Rosenblatt's-perceptron as a special form of a "support pattern" machine.

2.2 Perceptrons in the Data Dependent Representation

Decision function (1) may be rewritten in expanded form:

$$f(x) = \langle w, x \rangle = \langle (\sum_{i=1}^{l} \alpha_i x_i), x \rangle = \sum_{i=1}^{l} \alpha_i \langle x_i, x \rangle \tag{2}$$

and Rosenblatt's update rule (as described above) can be rewritten as:

$$\alpha_i \leftarrow \alpha_i + \eta y_i \tag{3}$$

Note that in (3) the elements of the vector of the multipliers, α, are updated instead of updating the weight vector, w, directly.
To avoid target function ((1),(2)) passing through the origin it is common practice to use augmented training patterns, x^a. Then an affine function: $f(x) = \langle w, x \rangle + b = \langle w^a, x^a \rangle$ will be obtained which has one more degree of freedom.
A pattern can be augmented by simply increasing its dimensionality by one and setting the $d+1$'th component of the augmented pattern vector to 1. Obviously, augmenting patterns is similar to introducing a bias unit which always has an activation equal to one. It is also possible to introduce a bias unit in the data dependent representation, this requires a change in the update rule (3) to:

$$\alpha_i \leftarrow \alpha_i + \eta y_i \quad ; \quad b \leftarrow b + \eta y_i \tag{4}$$

In data dependent perceptrons it is not necessary to use augmented patterns for learning because parameter b can be updated directly.

2.3 Kernel Functions and Nonlinear Perceptrons

The perceptron with kernels, known as the method of potential functions [1], has the ability to learn nonlinear decision functions. The algorithm is based on the idea of nonlinear kernel functions which represent dot products in Hilbert spaces. A kernel function allows the mapping of two patterns (x_u, x_v) at first into a high dimensional feature space $(\phi(x_u), \phi(x_v))$, and then the calculation of a dot product there. This is expressed by:

$$k(x_u, x_v) = \langle \phi(x_u), \phi(x_v) \rangle = \langle z_u, z_v \rangle \tag{5}$$

where the z are images of patterns x in feature space.
Any function k which satisfies Mercer's conditions may be used as a dot product in kernel-feature space [4].
Initially the kernel idea was applied in the potential function algorithm; an algorithm which uses kernel dot products to run a perceptron in a linearization space - the kernel feature space.
There has been an increased interest in using kernel dot products to create nonlinear algorithms. Examples are support vector machines [5], kernel principal component analysis [29], kernel Adatrons [13], and kernel clustering [16]. It is also possible to shine some new light onto gaussian-based algorithms by considering Gaussians as dot products in some feature space [9].
Two examples for kernel functions are the radial basis function (RBF) kernel and the polynomial kernel:

$$k_{RBF}(x_u, x_v) = exp(-||x_u - x_v||^2/\sigma^2) \tag{6}$$

$$k_{pol}(x_u, x_v) = (\langle x_u, x_v \rangle + 1)^d, \; d = 1, 2, .. \tag{7}$$

Kernel-based algorithms are elegant in the sense that the "kernel trick" allows algorithms to operate implicitly in very high (sometimes infinite-) dimensional feature spaces without explicitly expanding patterns into their feature space representation. Undesired side effects have been observed using support vector machines with polynomial kernels which are not scale or shift invariant. In these learning machines the optimal margin decision function will not lie in the middle of the two classes ([3], [12]). Further information about kernel functions can be found in [31].

To plug kernels into Rosenblatt's algorithm the dot products from the data-dependent decision function (2) are simply replaced by kernel functions:

$$f(x) = \langle w, \phi(x) \rangle = \langle \sum_{i=1}^{l} \alpha_i \phi(x_i), \phi(x) \rangle = \sum_{i=1}^{l} \alpha_i k(x_i, x) \tag{8}$$

On the right hand side of (8) the weight vector, w, resides now in the feature space defined by the kernels, it cannot be accessed any more for updates. Recall that the dot product in feature space is available, but not the transform $\phi(x) = z_x$ for a pattern x.
However in the data dependent representation of the kernel perceptron, multipliers are still accessible therefore the learning scheme is now given by:

Kernel-Perceptron Learning (Method of Potential Functions)

1. Choose a starting point (e.g. $\alpha_i = 0 \ \forall i \in 1..l$)
2. WHILE not all patterns are correctly classified (using decision function (8))
3. choose pattern $x_i (i \in 1..l)$
4. If NOT $sign(f(x_i)) = y_i$ update w by $\alpha_i \leftarrow \alpha_i + \eta y_i$ END IF
5. END WHILE

The b-parameter can be introduced in this perceptron by using update rule (4) instead of the rule suggested in step 4.
In step 2 of the algorithm, $f(x)$, is evaluated many times for training patterns. To speed up the algorithm the kernel correlation matrix $M_{i,j} = k(x_i, x_j)$ should be calculated once and stored in the computer's memory. During learning an evaluation of $f(x)$ can then be realised efficiently by a simple l dimensional dot product between the i'th row-vector of M and the vector of multipliers, α. In the kernel perceptron and kernel Adatron used in the following experiments, and also in [13], this coding scheme was used.

2.4 The Cost Function

Learning a set of training patterns can be implemented by minimising a cost function. Minimising only the error rate on the training patterns is not always sufficient and leads often to the well known overfitting phenomenon. Complexity regularisation allows the addition of an inductive bias into the process of choosing a model (a decision function) by minimising a cost function which minimises the empirical error (empirical risk) and an additional penalty term to punish hypotheses (decision functions) of a high complexity. Occam's Razor, a well known principle in machine learning, states basically that simple explanations which work well should be preferred over more complex ones. Popular approaches are the minimal description length (MDL) principle [7] and structural risk minimisation [34].

Often it is not possible or useful to classify all training patterns correctly. Outliers, wrongly labelled patterns, and noise in the training patterns may be tolerated to avoid overfitting. In cases where a misclassification of some training patterns is more expensive (implies a higher risk) than the misclassification of some other training patterns the loss (e.g. L_1 or L_2 loss) of each pattern, $e(x_i)$, can be weighted by a cost factor, c_i.
For this purpose an error function of the following form can be used [8]:

$$E(x_i) = \sum_{i=1}^{l} c_i e(x_i) \tag{9}$$

In batch learning this approach is useful, while in on-line (neural network-) algorithms it is common practice to use multiples of some training patterns to increase their cost.

Support vector machines are perceptrons using batch learning in the data dependent representation; there a good set of multipliers, α, is sought by minimising a *quadratic* function:

$$g(w) = C\ E(s_i) + \langle w, w \rangle \tag{10}$$

while insisting on constraint $y_i\langle w, \phi(x_i)\rangle \geq 1 - s_i \quad \forall i \in \{1..l\}$ (s_i is a non-negative slack variable required for cases where pattern i cannot be classified correctly).
This cost function consists of two parts, one which minimises the L_2 norm of w, and one which minimises the empirical error (denoted by error function E).
For capacity control, a constant C is used which allows a balance of the proportion of error minimisation against maximising the margin by minimising $||w||_2$. Minimising the L_2 norm of the weight vector can be considered as weight decay regularisation in feature space, that is to find the large margin perceptron (recall that in perceptrons the margin is inversely related to the norm of weight vector, w). The large margin perceptron is the one which separates two classes such that the distance from the *closest* pattern of each class to the decision boundary (that is the margin) is maximised. By definition the expected generalisation ability of a perceptron is high if there is a large margin between the separating perceptron-plane and the closest training patterns [35].

Perceptrons of optimal stability have already been studied in the framework of statistical mechanics. It has turned out that the optimal perceptron has a large margin and is nearly identical to the support vector machine [18]. On-line and batch learning algorithms for perceptrons with optimal-margin have been developed. The Adatron and kernel Adatron, respectively, are perceptrons of optimal stability; their learning algorithm converges fast to the optimal solution [23].

Adatrons and support vector machines use weight decay regularisation based on minimising the L_2 norm of w. Other ideas to implement regularisation are to minimise the L_1 or L_2 norm of the vector of multipliers, α. This leads to cost functions of the form:

$$g(w) = E(x) + C||\alpha||_1 \tag{11}$$

$$g(w) = E(x) + C||\alpha||_2 \tag{12}$$

Algorithms which implement function (11) or (12) do no longer find a large-margin perceptron in feature space, but they aim to maximise the margin of the perceptron in "hidden unit" space; in other words they minimise some function of the overall embedding strength. The hidden unit space is the vector space spanned by the row vectors of the kernel correlation matrix M (as defined above). The vector which is obtained by propagating a training pattern, x_i, through the first layer of a two-layered (data dependent) perceptron is equivalent to row i in M.

2.5 Linear Programming Machines

In 1961 Minnick and Muroga proposed a batch algorithm for learning perceptrons using linear programming. Since then some different and interesting approaches have been developed, see e.g. [2], [27] for further details.

Minnick's and Muroga's algorithm works as follows; use linear programming to solve:

$$min\ g(w) = \sum_{i=1}^{l} s_i \tag{13}$$
$$subject\ to: s_i \geq 0;\ y_i(\langle w, x_i\rangle + b) \geq \lambda - s_i$$

The constraints ensure that during the optimisation process a weight vector, w, is determined such that training patterns x are correctly classified. In some cases this may not be possible, therefore non-negative slack variables s_i were introduced. The overall slack (L_1 error) is minimised in cost function (13). Constant, λ, was introduced for the purpose of numerical stability. Any non-negative choice for λ is possible; in the following $\lambda = 1$ is assumed.

Similar to the Rosenblatt perceptron, it is possible to rewrite the decision function and learning procedure in the data dependent representation using kernel functions. In this type of kernel perceptron the decision function is given by:

$$f(x) = \sum_{i=1}^{l} \alpha_i y_i k(x_i, x) \tag{14}$$

and batch learning is performed by minimising a linear program:

$$min\ g(\alpha) = \sum_{i=1}^{l} s_i \tag{15}$$
$$subject\ to: s_i \geq 0;\ \alpha_i \geq 0;\ y_i f(x) \geq \lambda - s_i$$

In this notation weight vector $w = \sum_{i=1}^{l} \alpha_i y_i x_i$, has been substituted by a weighted sum of patterns and labels, therefore multipliers, α, will always be non-negative.
Owing the circumstance that this kernel perceptron operates in high dimensional spaces implied by the kernels, it may tend to learn overly complex decision boundaries. Therefore capacity control by the rightmost term in (11) is introduced. As pointed out above, this type of regularizer maximises the margin of the multiplier-vector in the "hidden unit" space.
The final algorithm is called the linear programming machine LPM. It uses decision function (14) for predictions after optimising cost function:

$$\min\ g(\alpha) = \sum_{i=1}^{l} s_i c_i + C \sum_{i=1}^{l} \alpha_i \tag{16}$$
$$subject\ to: s_i \geq 0;\ \alpha_i \geq 0;\ y_i f(x) \geq \lambda - s_i$$

(the c_i are the costs of making errors on individual patterns).
So far it has been assumed that the function to be learned can be completely characterised by the weight vector, w, or its data-dependent representation. In many cases it may be desired to learn a regularized function with a bias unit (a non-regularized parameter, b). This can be achieved by simply changing expression (14) to: $f(x) = \langle w, x\rangle + b = \sum_{i=1}^{l} \alpha_i y_i k(x, x_i) + b$ and using (16) to optimise the values for all α and b.

3 Minimal Expansion of the Decision Function

Data dependent kernel perceptrons, like support vector machines and kernel Adatrons, use a decision function of the form:

$$f(x) = \langle w, \phi(x) \rangle + b = \sum_{i=1}^{l^{sv}} \beta_i k(x_i^{sv}, x) + b \tag{17}$$

In this expansion all patterns with a non-zero multiplier, β, are denoted as "sv" patterns, there are l^{sv} such patterns. As stated above, in data dependent perceptrons the weight vector, w, exists only in feature space; it lies in the subspace spanned by the vectors $\phi(x^{sv})$. In ([5], Endnote 6) it is explained that in support vector machines sometimes more patterns, x^{sv}, are found than necessary to expand w in feature space (*"the decision function is unique but not its expansion on support vectors"*).
It is a nontrivial task to eliminate the linear relationships in feature space because transform $\phi(x)$ is not available. Even if this transform were available it would be nearly impossible to calculate linear dependencies between the vectors, $\phi(x^{sv})$, because these vectors can have a very high number (possibly infinite) of dimensions.
If linear dependencies in feature space could be found, their images in input space could be removed from the decision function in a straightforward manner. Assuming that multipliers from the decision function are updated correctly, patterns from expansion (17) can be removed *without* loss.
In the following, a way to find linear dependencies between patterns in feature space is given, then the expansion is rewritten such that the corresponing images in input space can be removed. If, in feature space, all linear dependencies are removed, an optimal decision function can be calculated (optimal means here the use of a minimal number of expansion vectors leading to a speed up in decision making proportional to the compression ratio obtained).
In [29] the reduced set algorithm is proposed, which is conceptually different to the one presented here because there the "compression" of the decision function is not loss-less (compression *with* loss), and the expansion vectors obtained by the reduced set algorithm are no longer elements from the set of training patterns.

Matrix P will be defined such that $P_{g,h}$ is the h'th component of vector $\phi(x_g)$; $g \in \{1..l^{sv}\}$. Note that matrix, P, is an l^{sv} by d^{fs} matrix where fs denotes the dimensionality of the feature space.
It is useful to define an l^{sv} by l^{sv} kernel correlation matrix $M = PP'; M_{i,j} = k(x_i, x_j)$ which has the following property:

$$rank(M) = rank(P) \tag{18}$$

This relationship is a property of a Gram matrix [17].
Except for cases where $rank(M) = l^{sv}$ there will be patterns $\phi(x_i)$ which can be expressed as a linear combination of l^{*sv} other patterns in feature space ($l^{*sv} \leq l^{sv}$).

$$\phi(x_i) = \sum_{j^*=1}^{l^{*sv}} \gamma_{j^*}\, \phi(x_{j^*}) \quad (i \neq j^* \;\forall j^*) \tag{19}$$

Linear dependencies between row-vectors of M are defined by:

$$\forall h \in \{1..l^{*sv}\} \;\; M_{i,h} = \sum_{j^*=1}^{l^{*sv}} \delta_{j^*} M_{j^*,h} \tag{20}$$

It can be shown that the linear dependencies between the row vectors in M are exactly the same as the linear dependencies between the row vectors in P.

Theorem 1: *For each linear dependency of the form (19) there exists exactly one linear dependency in M where the linear factors in (19) and (20) are equivalent:*

$$\forall i \;\; \gamma_i = \delta_i \tag{21}$$

Proof:

$$\begin{aligned}\forall_h : M_{i,h} = k(x_i, x_h) = \langle \phi(x_i), \phi(x_h) \rangle = \langle \sum_{j^*=1}^{l^{*sv}} \gamma_{j^*} \phi(x_{j^*}), \phi(x_h) \rangle = \\ \sum_{j^*=1}^{l^{*sv}} \gamma_{j^*} k(x_{j^*}, x_h) = \sum_{j^*=1}^{l^{*sv}} \gamma_{j^*} M_{j^*,h}\end{aligned} \tag{22}$$

Therefore it follows that $\gamma_i = \delta_i \forall i$. This relationship allows the computation of the γ from matrix M, this leads to the δ which can be used to update the multipliers, β, if a pattern, x_i, is to be removed from the decision function.

All that's required is to find the γ is to solve linear equation systems where one row vector of M is expressed as a linear combination of other row vectors of M. This can be done using standard techniques for solving linear equation systems (e.g. Gauss elimination or Householder's method [19]).
Once linear dependencies, δ, in feature space have been found the decision function can be rewritten in a way such that one redundant (linearly dependent) pattern will vanish. Function (17) can be expanded:

$$\begin{aligned}f(x) = b + \sum_{i=1}^{l^{sv}} \beta_i k(x_i, \phi(x)) = \\ b + \beta_1 \langle z_1, \phi(x) \rangle + \beta_2 \langle z_2, \phi(x) \rangle + ... \beta_i \langle z_i, \phi(x) \rangle + ... \beta_{l^{sv}} \langle z_{l^{sv}}, \phi(x) \rangle\end{aligned} \tag{23}$$

By using (19) and multiplying out

$$\begin{aligned}f(x) = b + \beta_1 \langle z_1, \phi(x) \rangle + \beta_2 \langle z_2, \phi(x) \rangle + ... \\ \beta_i \langle (\sum_{j^*=1}^{l^{sv}} \gamma_{j^*} z_{j^*}), \phi(x) \rangle + ... \beta_{l^{sv}} \langle z_{l^{sv}}, \phi(x) \rangle\end{aligned}$$

$$\begin{aligned}= b + \beta_1 \langle z_1, \phi(x) \rangle + \beta_2 \langle z_2, \phi(x) \rangle + ... (\beta_j + \beta_i \gamma_j) \langle z_j, \phi(x) \rangle + ... \\ (\beta_k + \beta_i \gamma_k) \langle z_k, \phi(x) \rangle + ... \beta_{l^{sv}} \langle z_{(l^{sv}-1)}, \phi(x) \rangle\end{aligned} \tag{24}$$

a function is obtained where one unit has been removed from the expansion.

So the update rule for multipliers β from (17) is given by:

$$if\ k \neq i \quad u(\beta_k) = \beta_k + \beta_i \gamma_{k^*} \tag{25}$$

To obtain the minimal expansion the process of removing redundant units and updating multipliers is repeated until no further updates are possible.

An algorithm for minimal expansion set reduction can now be summarised in pseudocode:

minimal expansion set reduction (MESR) algorithm

1. calculate correlation matrix M using all l^{sv} patterns
2. REPEAT the following loop exactly $(l^{sv} - rank(M))$ times:
3. choose index i of a row-vector from M which can be expressed as a linear combination of other row-vectors of M
4. find the multipliers, δ, which allow one to express row-vector i of M as a linear combination of some other row-vectors of M
5. recalculate M using all patterns used in the last iteration except pattern i chosen in step 3
6. update all multipliers by $\beta^t \leftarrow u(\beta^{t-1})$
7. END REPEAT
8. use all patterns whose corresponding rows and columns in M were not deleted during the update process; use also the latest set of updated multipliers, β^t, for the new expansion of the decision function

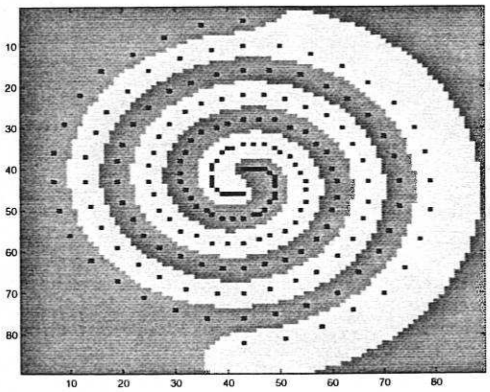
In each iteration of the algorithm a linear dependency of form (19) is identified, one pattern is removed from the decision function (point 3), and then multipliers β are updated.

At step 3 of the algorithm it is expected to find a pattern which is a linear combination of other patterns. If a number, n, of patterns are linearly dependent, then obviously each of the n patterns could be chosen and expressed by $n-1$ or fewer patterns.
If the algorithm is applied to support vector machines it is suggested to pick the support pattern with the largest slack variable. These patterns are often the misclassified patterns or patterns which lie close to the margin, for those support patterns "classification constraint" $y_i\langle w, z_i\rangle \geq 1$ is violated. As pointed out in [24] Vapnik's leave-one-out bound works only in cases where all patterns are correctly classified by a support vector machine. In those cases the "compression ration" (expected generalisation ability) given by the leave-one-out bound [35] is an upper bound on compression function $T_{MESR} = rank(P)/l$.
Therefore it is suggest to estimate the model complexity and expected generalisation performance by using function T_{MESR}.

4 Computer Experiments

Four experiments with linear programming machines are presented below. In the first experiment the two-spiral benchmark has been chosen to compare the decision funcions found by the LPM and SVM. In the three following experiments the performance obtained by the LPM is compared to that obtained by k-nearest neighbourhood (KNN) classifiers, kernel perceptrons with bias unit, kernel Adatrons, and SVMs. In these experiments the following validation strategy has been applied; parameters were chosen such that the performance on a validation set (a subset of the training set) was optimal.
The experiments merely aim to allow a first comparison between LPM and other classification techniques. Except for the KNN classifier the RBF kernel (6) has been used in all experiments. All algorithms have been implemented in MATLAB, for linear and quadratic programming LOQO [33] was used.

4.1 Two Spiral Classification



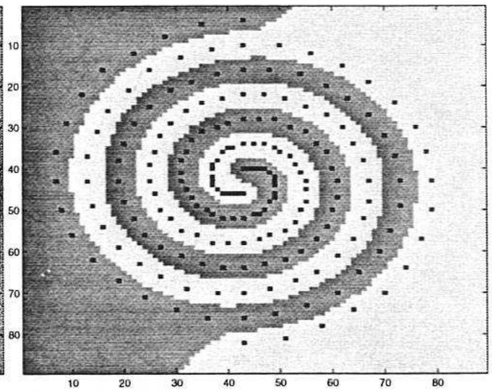


Figure 1: Two Spiral classification for linear programming machine (left) and support vector machine (right); ($\sigma_{LPM} = \sigma_{SVM} = 0.8$; $C_{LPM} = 0$; $C_{SVM} = 1000$).

In the two spiral benchmark the patterns which must be classified lie on two spirals which are coiled three times around each other. The benchmark [11] is widely known and has been used with a variety of classification techniques. The two dimensional nature of the benchmark allows the visualisation of the decision function learned by the two algorithms. Both algorithms have determined all 194 training patterns to be support patterns (hidden units). The diagrams show that in both cases (LPM, SVM) a classification was learned such that a large margin lies between the decision function and most patterns from the two classes. The two results are slightly different, note that in the centre of the two spirals the LPM's decision boundary is *smoother* than that of the support vector machine where the decision boundary always has the *largest* possible margin. Note also that the linear programming machine chose smaller values for multipliers, α, than the support vector machine (this can be seen by comparing the size of the grey "band" around the outmost patterns).
Patterns which have a high distance to the decision plane in feature space (that is, roughly, a large distance to the training patterns in input space) are always assigned to one class

in RBF kernel machines (both in SVM and LPM). This can be explained by considering decision function (14) which always assigns patterns which lie exactly on the decision plane in feature space to one class. Owing to assumptions which are always made when algorithms are implemented on computer systems (with static number representation) all numbers which lie in a small interval around zero are considered to be zero (rounding). All patterns which have a small distance to the decision plane in feature space will lie in such a zero-interval and therefore be assigned to one class. This computational bias (how to assign labels to the two classes) represents a *prior* which one should be aware of when using radial basis function learning machines.

4.2 Sonar Classification

The task here is to classify the sonar set of 208 patterns representing metal cylinders and rocks, where each pattern has 60 dimensions. As suggested in [15] the angle dependent data has been split into a training set and test set each of 104 patterns.

SONAR	Cycles	$\sigma(RBF)$	# Trn.Err	# Tst.Err
KNN (k=1)	-	-	0	10
KNN (k=3)	-	-	0	19
k-Perc.	-	0.8	0	6
k-Perc.	-	1.0	0	8
k-Perc.	-	1.2	0	13
SVM	-	0.8	0	8
SVM	-	1.0	0	7
SVM	-	1.2	0	7
LPM	-	0.8	1	11
LPM	-	1.0	1	9
LPM	-	1.2	0	8
k-Adatron	10	0.8	0	6
k-Adatron	10	1.0	0	9
k-Adatron	10	1.2	0	8
k-Adatron	250	0.8	0	6
k-Adatron	250	1.0	0	6
k-Adatron	250	1.2	0	7

Table 1: Comparision of classifiers using the sonar patterns.

Table 1 compares the performance of the linear programming machine with other algorithms. In this classification task no significant improvement can be observed when comparing the kernel perceptron's (small margin) performance with the one of the SVM (large margin). Interestingly the LPM is the only algorithm which tends to allow misclassified patterns in the training set (due to its regularizer); however the performance on the test set is comparable to that of the SVM.

4.3 Mirror Symmetry Classification

All patterns from the mirror symmetry benchmark are synthetic patterns (200 patterns for the training set, 100 patterns in the test set). The aim of the benchmark is to distinguish patterns which are symmetric about their centre from patterns which are not symmetric.

Mirr.Sym	Cycles	$\sigma(RBF)$	# Trn.Err	# Tst.Err
KNN (k=1)	-	-	0	25
KNN (k=7)	-	-	0	22
k-Perc.	-	5	0	25
k-Perc.	-	6	0	26
k-Perc.	-	7	0	27
SVM	-	5	0	3
SVM	-	6	0	2
SVM	-	7	0	5
LPM	-	5	0	6
LPM	-	6	0	3
LPM	-	7	0	2
k-Adatron	10	5	0	4
k-Adatron	10	6	0	5
k-Adatron	10	7	0	6
k-Adatron	250	5	0	3
k-Adatron	250	6	0	4
k-Adatron	250	7	0	5

Table 2: Mirror symmetry; comparison of classifiers.

In this benchmark both the KNN classifier and the kernel perceptron have shown a poor performance compared to the LPM, SVM, and kernel Adatron. In all cases, where regularisation has been used a substantially better performance was obtained.

4.4 Wisconsin Breast Cancer Classification

The original Wisconsin breast cancer database consists of 699 instances. For this experiment a subset of 200 patterns was randomly chosen for the training set and 200 patterns were chosen for the test set.

Can.Diag	Cycles	$\sigma(RBF)$	# Trn.Err	# Tst.Err
KNN (k=1)	-	-	0	13
KNN (k=3)	-	-	0	9
k-Perc.	-	0.4	0	13
k-Perc.	-	0.5	0	10
k-Perc.	-	0.6	0	9
SVM	-	0.4	0	11
SVM	-	0.5	0	10
SVM	-	0.6	0	11
LPM	-	0.4	9	16
LPM	-	0.5	0	10
LPM	-	0.6	0	9
k-Adatron	10	0.4	0	8
k-Adatron	10	0.5	0	9
k-Adatron	10	0.6	0	11
k-Adatron	250	0.4	0	10
k-Adatron	250	0.5	0	9
k-Adatron	250	0.6	0	10

Table 3: Classification performance on the Wisconsin database.

Similar to the sonar experiment the simple kernel perceptron seems to perform as well as the LPM, kernel Adatron, and SVM. One possible explanation is that the kernel perceptron stopped in a lucky position where the plane has, by accident, a large margin. Another explanation is that the "large" margin between the closest patterns from two classes and the plane in feature space is only very small (for the σ given in the table), and that, therefore, the kernel perceptron always stops at a good position. It could also be possible that there is some "noise" in the patterns such that the large margin between the patterns *closest* to the plane is not similar to a large margin for *most* patterns from the two classes.
The experiment was repeated five more times with the kernel perceptron using different starting points, in all cases a similar performance to the one indicated in the table was obtained.

5 Conclusion

Perceptrons, both on-line and batch, have been rewritten in the data dependent representation. This allowes to plug kernels into the algorithms and to develop non-linear models; one of them is the linear programming machine which implements weight decay regularisation (large margin) in *hidden unit space*, similar to neural networks and Ada-Boost [26] where the weight of the final "neurone" is regularized by weight decay. In contrast, support vector machines use weight decay regularisation in *feature space*.
Computer experiments compare the performance of the LPM, SVM, and kernel perceptron. In these experiments the LPM achieved a performance which is comparable to the one obtained by support vector machines. This is interesting because linear programming is computationally much simpler than solving quadratic optimisation problems as required by support vector machines.
The minimum expansion set reduction algorithm allows a speed up of kernel perceptrons in prediction mode by analysing linear dependencies in kernel feature space. The algorithm can be considered from two perspectives; on the one hand it allows the compression and speed up of support vector machines and thus a decrease in the description length of the hypothesis, on the other hand it can determine a compression ratio of a learning machine which can be considered as a complexity measure. The compression ratio is a useful complexity measure for model selection compared to the method of estimating the generalisation ability using the leave-one-out bound (as suggested in [35]) because it allows the removal of redundant instances from the decision function and does not fail in cases where a high overlap between the two classes occurs.

References

[1] Aizerman, M., Braverman, E., and Rozonoer, L. (1964). Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning, Automations and Remote Control, 25:821-837.

[2] Bennett K., Mangasarian O.L., (1998), Multicategory Discrimination via Linear Programming, Optimization Methods and Software 3, pp. 27-39

[3] Boser, B., Guyon, I., Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In Proceedings of the Fifth Annual Workshop of Computational Learning Theory, 5, 144-152, Pittsburgh, ACM

[4] Courant A., Hilbert D. (1951). Methods of Mathematical Physics, Wiley Interscience

[5] Cortes, C., and Vapnik, V. (1995). Support Vector Metworks, Machine Learning 20:273-297.

[6] Cortes, C. (1995). Prediction of Generalization Ability in Learning Machines. PhD Thesis, Department of Computer Science, University of Rochester.

[7] Cover, Thomas. (1985). Elements of Information Theory, Wiley

[8] Duda R. O., Hart P.E. (1973), Pattern Classification and Scene Analysis, Wiley Interscience, New York

[9] Cristianini, N., Shawe-Taylor, J., Sykacek, P., (1998). Bayesian Classifiers are Large Margin Hyperplanes in a Hilbert Space, in Shavlik, J., ed., Machine Learning: Proceedings of the Fifteenth International Conference, Morgan Kaufmann Publishers, San Francisco, CA.

[10] Fukunaga K., (1990), Introduction to Statistical Pattern Recognition, Academic Press, San Diego

[11] Frieß T-T., Harrision R.F., (27.02.1998), Linear Programming Support Vector Machines and the Set Reduction Algorithms, Research Report 706, Dept. of Automatic Control & Systems Engineering, University of Sheffield

[12] Frieß T-T., Harrison R.F., (1998), Pattern Classification using Support Vector Machines, Eng. and Int. Sys. (EIS98 Conf. Proc.)

[13] Frieß T-T., Cristianini N., & Campbell C. (1998). The Kernel-Adatron Algorithm: a Fast and Simple Learning Procedure for Support Vector Machines, in Shavlik, J., ed., Machine Learning: Proceedings of the Fifteenth International Conference, Morgan Kaufmann Publishers, San Francisco, CA.

[14] Freund Y., Schapire R. E., (1998) Large Margin Classification Using the Perceptron Algorithm, Proceedings of the Eleventh Annual Conference on Computational Learning Theory (COLT 98)

[15] Gorman R. P. & Sejnowski, T. J. (1988) Neural Networks 1:75-89.

[16] Grappel T., Obermayer K. (1988). Fuzzy Topographic Kernel Clustering, in Brauer W., ed,. Proceedings of the 5th GI Workshop Fuzzy Neuro Systems '98

[17] Horn A. R., Johnson H. R. (1985). Matrix Analysis, Cambridge University Press, New York

[18] Kinzel, W.,(1990) Statistical Mechanics of the Perceptron with Maximal Stability. Lecture Notes in Physics (Springer-Verlag) 368:175-188.

[19] Kreyszig E., (1993) Advanced Engineering Mathematics, Wiley, New York

[20] Minnick, (1961). Linear Input Logic, IRE Transactions on Electronic Computers, 10, 6-16

[21] Minsky M.L. & Papert, S.A. (1969) *Perceptrons*, MIT Press: Cambridge.

[22] Muroga et al., (1961). Theory of Majority Decision Elements, Journal of Franklin Institute, 271, 376-419

[23] Opper M., Learning Times of Neural Networks: Exact Solution for a Perceptron Algorithm. Physical Review A38:3824

[24] Ripley B.D., (1996). Pattern Recognition and Neural Networks, Cambridge University Press, Cambridge

[25] Rosenblatt F., (1961). Principles of Neurodynamics, Spartan Press, New York

[26] Schapire. R., Freund, Y., Bartlett, P., & Sun Lee, W. (1997). Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods, Proceedings of International Conference on Machine Learning.

[27] Smith, (1968). Pattern Classifier Design by Linear Programming, IEEE Transactions on Computers, 17, 367-372

[28] Smola A.J., Schölkopf B., Müller K-R., (1998). The connection between Regularzation Operators and Support Vector Kernels, to Appear in Neural Networks

[29] Schölkopf, B., (1997). Support Vector Learning. PhD Thesis. R. Oldenbourg Verlag, Munich.

[30] Shawe-Taylor J., Bartlett, P., Williamson, R. & Anthony, M. (1996). Structural Risk Minimization over Data-Dependent Hierarchies NeuroCOLT Technical Report NC-TR-96-053 (ftp://ftp.dcs.rhbnc.ac.uk /pub/neurocolt/tech_reports).

[31] Smola A., Schölkopf B., Müller K-R., (1998). The Connection between Regularization Operators and Support Vector Kernels, to appear in Neural Networks

[32] (Sonar dataset) http://www.boltz.cs.cmu.edu/benchmarks/sonar.html

[33] Vanderbei R. J., (1994) LOQO: An interior point code for quadratic programming, TR SOR-94-15, Statistics and Operations Research, Priceton University, NJ

[34] Vapnik, V., Chervonenkis A. (1979) Theory of Pattern Recognition, German Translation, Akademie Verlag, Berlin

[35] Vapnik, V. (1995). The Nature of Statistical Learning Theory, Springer Verlag.

[36] Weston J. et. al (02.02.1998). Density Estimation Using SV Machines, Technical Report CSD-TR-97-23, Dept. of Computer Science, Royal Holloway University of London http://www.cs.rhbnc.ac.uk/research/compint/areas/comp_learn/sv/pubs.html

[37] Zell A. (1993), Neuronale Netze, Springer Verlag,