A Kernel Method for Classification

Donald MacDonald[1], Jos Koetsier[1], Emilio Corchado[1],

Colin Fyfe[1] and Juan Corchado[2]

[1]School of Information and Communication Technologies
The University of Paisley, High Street, Paisley, PA1-2BE, Scotland.
corc-ci0@paisley.ac.uk
[2]Departamento de Informática y Automática, Universidad de Salamanca, Spain.

Abstract. Kernel Maximum Likelihood Hebbian Learning Scale Invariant Maps is a novel technique developed to facilitate the clustering of complex data effectively and efficiently and that is characterised for converging remarkably quickly. Kernels were originally derived in the context of Support Vector Machines which identify the smallest number of data points necessary to solve a particular problem, such as regression and classification. The combination of Maximum Likelihood Hebbian Learning Scale Invariant Map and the Kernel Space provides a very smooth scale invariant quantisation which can be used as a clustering technique. The efficiency of this method have been used to analyse an oceanographic problem.

1 Introduction

This paper presents an efficient technique for data clustering. Kernel Maximum Likelihood Hebbian Learning Scale Invariant Map (K-MLSIM) is based on a modification of a new type of topology preserving map that can be used for scale invariant classification [6]. Kernel models were first developed within the context of Support Vector Machines [16]. Support Vector Machines attempt to identify a small number of data points (the support vectors) which are necessary to solve a particular problem to the required accuracy. Kernels have been successfully used in the unsupervised investigation of structure in data sets[15] [11] [9]. Kernel methods map a data set into a Feature space using a nonlinear mapping. Then typically a linear operation is performed in the feature space; this is equivalent to performing a nonlinear operation on the original data set. The Scale Invariant Map is an implementation of the negative feedback network to form a topology preserving mapping. A kernel method is applied in this paper to an extension of the Scale Invariant Map (SIM) which is based on the application of the Maximum Likelihood Hebbian Learning (MLHL) method [4]and its possibilities are explored. The proposed methodology groups cases with similar structure, identifying clusters automatically in a data set in an unsupervised mode. The method has been successfully used to improve the reasoning process of a distributed oceanographic system developed for monitoring and predicting toxic episodes in coastal waters.

2. Kernel Scale Invariant Map

This section reviews the techniques used to construct the K-MLSIM method.

2.1 Scale Invariant Map

Consider a network with N dimensional input data and having M output neurons. Then the activation of the i^{th} output neuron is given by

$$act_i = \sum_{j=1}^{N} w_{ij} x_j \tag{1}$$

Now if we invoke a competition between the output neurons, it is possible to have a number of different competitions between output neurons, two obvious examples are:
Type A: The neuron with greatest activation wins.
Type B: The neuron closest to the input vector wins.
The Kohonen network typically uses the second since the first requires specific re-normalisation to ensure that all neurons have a chance of winning a competition. We have shown that the scale invariant mapping produced by the new network does not require any additional competition limiting procedure when using the first criterion. In both cases, the winning neuron, the p^{th}, is deemed to be maximally firing (=1) and all other output neurons are suppressed(=0). Its firing is then fed back through the same weights to the input neurons as inhibition.

$$e_j \leftarrow x_j - w_{pj}.1\_for\_all\_j \tag{2}$$

where p is the winning neuron. Now the winning neuron excites those neurons close to it i.e. we have a neighbourhood function $\Lambda(p,j)$ which $\Lambda(p,j) \le \Lambda(p,k)$ for all $j,k : \|p-j\| \ge \|p-k\|$ where $\|.\|$ is the Euclidean norm. In the simulations described in this paper, we use a Gaussian whose radius is decreased during the course of the simulation. Then simple Hebbian learning gives

$$\Delta w_{ij} = \eta_t \Lambda(p,i).e_j = \eta_t \Lambda(p,i).(x_j - w_{pj}) \tag{3}$$

where we have used x_j as the activation of the j^{th} input neuron and w_{ij} is the weight between this and the i^{th} output neuron. For the ^{pth} winning neuron, the network is performing simple competitive learning but note the direct effect the p^{th} output neuron's weight has on the learning of other neurons. This algorithm introduces competition into the same network used in [5] to perform a Principal Component Analysis and in [7] to perform an Exploratory Projection Pursuit.

2.2 Kernel K-means Clustering

We will follow the derivation of [14] who has shown that the k means algorithm can be performed in Kernel space. The basic idea of the set of methods known as kernel methods is that the data set is transformed into a nonlinear feature space

$(\phi : x \to \phi(x))$. Any linear operation now performed in this feature space is equivalent to a nonlinear operation in the original space.

The aim is to find k means, $\mathbf{m}_\mu$, so that each point is close to one of the means. First we note that each mean may be described as lying in the manifold spanned by the observations, $\phi(\mathbf{x}_i)$ i.e. $m_\mu = \sum_i w_{\mu i}\phi(x_i)$. Now the k means algorithm chooses the means, $\mathbf{m}_\mu$, to minimise the Euclidean distance between the points and the closest mean

$$\left\|\phi(\mathbf{x}) - m_\mu\right\|^2 = \left\|\phi(\mathbf{x}) - \sum_i w_{\mu i}\phi(\mathbf{x}_i)\right\|^2 = k(\mathbf{x},\mathbf{x}) - 2\sum_i w_{\mu i}k(\mathbf{x},\mathbf{x}_i) + \sum_{i,j} w_{\mu i}w_{\mu j}k(\mathbf{x}_i,\mathbf{x}_j) \qquad (4)$$

i.e. the distance calculation can be accomplished in Kernel space by means of the K matrix alone. Let $M_{i\mu}$ be the cluster assignment variable. i.e. $M_{i\mu}$=1 if $\phi(\mathbf{x}_i)$ is in the μ^{th} cluster and is 0 otherwise. [14] initialises the means to the first training patterns and then each new training point, $\phi(\mathbf{x}_{t+1}), t+1 > k$, is assigned to the closest mean and its cluster assignment variable calculated using

$$M_{t+1,\alpha} = \begin{cases} 1\_if\_\left\|\phi(x_{t+1}) - \mathrm{m}_\alpha\right\| < \left\|\phi(x_{t+1}) - \mathrm{m}_\mu\right\|, \forall \mu \neq \alpha \\ 0\_otherwise \end{cases} \qquad (5)$$

In terms of the kernel function (noting that k(**x**,**x**) is common to all calculations) we have

$$M_{t+1,\alpha} = \begin{cases} 1\_if\_\sum_{i,j} w_{\alpha i}w_{\alpha j}k(\mathbf{x}_i,\mathbf{x}_j) - 2\sum_i w_{\alpha i}k(\mathbf{x},\mathbf{x}_i) \\ < \sum_{i,j} w_{\mu i}w_{\mu j}k(x_i,x_j) - 2\sum_i w_{\mu i}k(\mathbf{x},\mathbf{x}_i), \forall \mu \neq \alpha \\ 0\_otherwise \end{cases} \qquad (6)$$

We must then update the mean, $\mathbf{m}_\alpha$ to take account of the $(t+1)^{th}$ data point

$$\mathbf{m}_\alpha^{t+1} = \mathbf{m}_\alpha^t + \zeta\left(\phi(x_{t+1}) - \mathbf{m}_\alpha^t\right) \qquad (7)$$

where we have used the term $\mathbf{m}_\alpha^{t+1}$ to designate the updated mean which takes into account the new data point and

$$\zeta = \frac{M_{t+1,\alpha}}{\sum_{i=1}^{t+1} M_{i,\alpha}} \qquad (8)$$

Now (7) may be written as

$$\sum_i w_{\alpha i}^{t+1}\phi(\mathbf{x}_i) = \sum_i w_{\alpha i}^t\phi(\mathbf{x}_i) + \varsigma\left(\phi(\mathbf{x}_{t+1}) - \sum_i w_{\alpha i}\phi(\mathbf{x}_i)\right) \qquad (9)$$

which leads to an update equation of

$$w_{\alpha i}^{t+1} = \begin{cases} w_{\alpha i}^t(1-\zeta)\_for\_i \neq t+1 \\ \zeta\_for\_i = t+1 \end{cases} \qquad (10)$$

2.2.1 Kernel Self Organising Map

We have previously used the above analysis to derive a Self Organising Map [10] in Kernel space. The SOM algorithm is a k means algorithm with an attempt to distribute the means in an organised manner and so the first change to the above algorithm is to update the closest neuron's weights and those of its neighbours. Thus we find the winning neuron (the closest in feature space) as above but now instead of (6), we use

$$M_{t+1,\mu} = \Lambda(\alpha, \mu), \forall \mu \tag{11}$$

where α is the identifier of the closest neuron and $\Lambda(\alpha, \mu)$ is a neighbourhood function which in the experiments reported herein was a gaussian. Thus the winning neuron has a value of M=1 while the value of M for other neurons decreases monotonically with distance (in neuron space) away from the winning neuron. For the experiments reported in this paper, we used a one dimensional vector of output neurons numbered 1 to 20 or 30. The remainder of the algorithm is exactly as reported in the previous section.

The Kernel SOM was reported in [10][2] to have very fast convergence and results in that paper were based on Gaussian kernels. In [2], we showed that the reason for this very fast convergence was the interaction between the learning method (a combination of one-shot learning and incremental decay) and the fact that we are working (for linear kernels) in data space but using the data points as basis vectors for that space. We now investigate the application of similar methods to the Scale Invariant Map.

2.2.2 The Kernel Scale Invariant Map

We will, in this Section, consider only linear kernels though the extension to other kernels is straightforward. Since every point in data space can be represented by a linear combination of the training data set $\{\mathbf{x}_i, i = 1,..., N\}$, we have $\exists v_i : x = \sum_{i=1}^{N} v_i \mathbf{x}_i$ for all x in the data space. Similarly the weight vectors can be represented in the same way. Thus we can represent

$$y_\mu = \sum_{j=1}^{N} w_{\mu j} \mathbf{x}_j \tag{12}$$

as

$$y_\mu = \sum_j \sum_i v_{\mu i} x_i x_j = \sum_j \sum_i v_{\mu i} k(\mathbf{x}_i \mathbf{x}_j) \tag{13}$$

2.2.2.1 The Competition

Now we wish to have a competition to find out which output will win for a particular input, $\mathbf{x}_l$, and so the first question to be addressed is the nature of the competition. If we are working in data space with the above overcomplete basis, then every member of the training set is representable as a vector which is all zeros except for the l^{th} position which is set to 1. Therefore we identify the l^{th} column of the kernel matrix, and determine the output which wins the competition to represent $\boldsymbol{x}_l$ as

$$\alpha = \arg\max_\mu y_\mu = \arg\max_\mu \sum_i v_{\mu i} x_i x_l = \arg\max_\mu \mathbf{k}_l \mathbf{v}_\mu \tag{14}$$

where we have used $\mathbf{k}_l$ as the vector from the l^{th} column of the kernel matrix.

2.2.2.2 The Weight Update

We have experimented with two methods:
1. The Kernel method of the previous section. With the notation above, we have

$$v_{\alpha i} = \begin{cases} v_{\alpha i}(1-\zeta)\_ for\_ i \neq l \\ \zeta\_ for\_ i = l \end{cases} \tag{15}$$

with ζ defined as for the Kernel SOM. This continues to be one-shot learning with a subsequent gradual decay.

The neural method used with the standard SIM. Note that since we are working in the space spanned by the data as basis vectors, the input vector has a zero everywhere but the l^{th} position and we are subtracting $\mathbf{v}_\alpha$ (α being the winning neuron). Thus

$$\mathbf{e} = \mathbf{x} - \mathbf{v}_\alpha \tag{16}$$

in this basis. We then apply the standard learning rule so that

$$\Delta v_{\mu i} = \eta \mathbf{\Lambda}(\alpha, \mu) e_i \tag{17}$$

Even though this method is an iterative method as is usual in neural methods, we find that only a few (less than 10, often just 2 or 3) iterations through a data set are enough to form the mapping.

2.3 Maximum Likelihood Hebbian Learning

We have previously [2][10] considered a general cost function associated with a negative feedback PCA network.

$$J = 1^T E\left\{(\mathbf{x} - Wy)^2\right\} \tag{18}$$

If the residual after the feedback has probability density function

$$p(\mathbf{e}) = \frac{1}{Z}\exp(-|\mathbf{e}|^p) \tag{19}$$

Then we can denote a general cost function associated with this network as

$$J = -\log p(\mathbf{e}) = |\mathbf{e}|^p + K \tag{20}$$

where K is a constant. Finding the minimum of J corresponds to finding the maximum of $p(\mathbf{e})$ i.e. we are maximising the probability that the residual comes from a particular distribution. We do this by adjusting the weights. Therefore performing gradient descent on J we have

$$\Delta W \propto -\frac{\partial J}{\partial W} = -\frac{\partial J}{\partial \mathbf{e}}\frac{\partial \mathbf{e}}{\partial W} \approx y(p\,|\mathbf{e}|^{p-1}\, sign(\mathbf{e}))^T \tag{21}$$

We would expect that for leptokurtotic residuals (more kurtotic than a Gaussian distribution), values of p < 2 would be appropriate, while platykurtotic residuals (less kurtotic than a Gaussian), values of p > 2 would be appropriate. We have previously [8] [3] [13]shown that this network can perform Exploratory Projection Pursuit.

2.3.1 Application to Kernel Scale Invariant Map

Now the SIM was originally derived by introducing competition to the negative feedback PCA network. Therefore we introduce the Maximum Likelihood Hebbian learning concept of the last section to the Scale Invariant Map (SIM). Consider the feedback in 16. Let us present a particular input, $\mathbf{x}_l$= (0,..,0,1,0,..0) which has 1 in the l^{th} position, to the network. Then if neuron α wins the competition, it is because the weights $\mathbf{v}_\alpha$ have a high dot product with the elements of *l*'s column of the k matrix; either $\mathbf{v}_{\alpha l}$ is large or $\mathbf{v}_{\alpha k}$ (for input k which will be grouped with *l*) is large. Thus the residuals after feedback will have a bimodal distribution - either the residuals will tend towards 0 or the residuals will be large.
This suggests maximising the likelihood that the residuals come from a sub-gaussian distribution; therefore, we proposed the learning rules

$$\Delta v_{\mu j} = \eta \mathbf{\Lambda}(\alpha, \mu) sign(\mathbf{e}_j) |\mathbf{e}_j|^2 \tag{22}$$

This has an interesting effect on the learning rules in that a pie slice of the data is learned but the actual positions of the neuron centres themselves (when transformed back into data space) lie outside the data set. This enables a very smooth scale invariant quantisation as shown in Figure 1. The **v** weight vectors are shown in Figure 2. The data set, drawn uniformly from [-1,1]*[-1,1], is shown by the crosses. The weights of the converged Kernel Scale Invariant Map have been joined to form almost a circle. The corresponding vector in data space based on the data as basis vectors is shown in Figure 2 each line of the diagram represents the weights of one output neuron in terms of the data points (the **v** weights in fact).

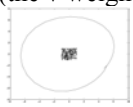

Fig. 1. The data set is shown by the red crosses. It was drawn uniformly from[-1,1]*[-1,1]. The weights of the converged KSIM have been joined to form almost a circle.

Fig. 2. The **v** weights as represented in the data basis. Each line is the weight vector into an output neuron and is shown.

2.3.2 The *p* parameter

We have found that even a very small departure from the standard K-SIM parameter (with p=2) gives a visible change to the representation of the data set. In the top left of Figure 3, we show the converged weights after 7 iterations of the K-SIM algorithm when p=1.8; the weights are well enclosed in the data. In the top right of that figure, we show the weights when p=2.1 was used; the weights are beginning to move outside the data set. In the bottom figure, we show what happens when p=2.5. Now the weights are well outside the data.

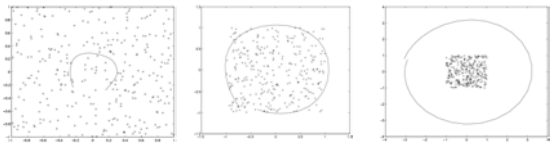

Fig.3: The left figure shows the weights after 7 iterations of the K-SIM algorithm with p=1.8 on the standard artificial data set. The one in the middle shows the weights with p=2.1. The right figure shows the weights when p=2.5.

Consider the situation in which there are *n* points in the pie slice won by y_α. Without loss of generality let us write $w_\alpha=(a_1\},a_2,...,a_n,0,0,...,0)$ i.e. the vector w_α has non-zero components corresponding to the points (in the training set), $\mathbf{x}_1,\ldots,\mathbf{x}_n$ while its components corresponding to $\mathbf{x}_{n+1}\ldots\mathbf{x}_N$ are all zero. Then

$$\Delta w_\alpha = \eta\Lambda(\alpha,\mu)sign(\mathbf{e})|\mathbf{e}|^{p-1} \tag{23}$$

Let us consider only the situation in which the points $\mathbf{x}_1,\ldots,\mathbf{x}_n$ are presented to the network and so the output y_α is the winner; thus $\Lambda(\alpha,\mu)$=1. Let point $\mathbf{x}_1$ be presented and so **x**=(1,0,0,...,0). Then

e = $(1-a_1,-a2,...,-a_n, 0,...,0)$ (24)

When point $\mathbf{x}_2$ is presented, **x**=(0,1,0,...,0) and

e = $(-a_1,1-a2,...,-a_n, 0,...,0)$ (25)

We will consider the effect of the update rules on this weight vector for different values of p.

p=1. Focus now on the first element of $\mathbf{w}_\alpha$, the element of the weight vector linking input $\mathbf{x}_1$ to output y_α, then at convergence $E(\Delta w_{\alpha 1})=E(sign(e_1))=0$. Clearly if $a_1<0$, sign(e_1) is always positive. Therefore $a_1>0$ .Thus there can only be two non-zero elements in this vector. Impossible.

p=2. This is the standard K-SIM. Then at convergence $E(\Delta w_{\alpha 1})=E(sign(e_1)|e_1|^1)=0$ and so

$1-a_1 + (n-1)*(-a_1)=0$ (26)

Thus $a_1=1/n$. This argument applies equally to all non-zero elements of $\mathbf{w}_\alpha$ and so $\mathbf{w}_\alpha=\left(\frac{1}{n},\frac{1}{n},...,\frac{1}{n},0,...,0\right)$ which when we translate back to the original basis means that the centre of the KSIM is given by

$$c_\alpha=\frac{1}{n}\mathbf{x}_1+\frac{1}{n}\mathbf{x}_2+...+\frac{1}{n}\mathbf{x}_n \tag{27}$$

the mean of the data points for which neuron α is responsible for representing.

p=3. At convergence $E(\Delta w_{\alpha 1})=E(sign(e_1)|e_1|^2)=0$ and so, for $0<a_1<1$,

$(1-a_1)^2 - (n-1)*(a_1)^2=0$ (28)

Solving this, we find that $a_1=\frac{-1\pm\sqrt{(n-1)}}{n-2}$. Thus for n=10, $a_1=\frac{1}{4}$ or $-\frac{1}{2}$. In practise, we have never seen the latter result but it seems, in principle, possible. Note that the solution is an equally weighted sum of the data points where the weights are

greater than $\frac{1}{n}$. Thus the centre, $c_\alpha^* = C * c_\alpha$ where C is a constant and c_α is the mean of the data points defined in 26.
This gives the broad picture. Intermediate values of *p* will give solutions somewhere between the results given above. In general, at convergence $E(\Delta w_{\alpha 1}) = E\left(sign(e_1)|e_1|^{p-1}\right) = 0$ and so

$(1-a_1)^{p-1} - (n-1)*(a_1)^{p-1}=0$ (29)
if $a_1>0$. This implies $a_1 = \frac{1}{1+(n-1)^{\frac{1}{p-1}}}$
The effect of varying p with n=10 is shown in Figure 4. We see that for values close to 1, a_1 remains close to 0, while for values approaching 2, we reach the $\frac{1}{10}$ value. Subsequently, a_1 continues to rise but shows signs of levelling off (right figure).

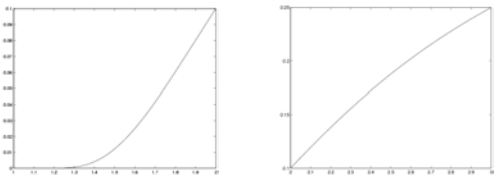

Fig. 4: As we vary *p* from 1 to 2, a_1 climbs slowly to 0.1 (left figure) and grows more slowly subsequently (right figure).

3. Clustering Real Data using the K-MLSIM

In the following section we detail the results of the K-MLSIM on the red tides data. The K-MLSIM is specially suited to be used on the red tides data because of the flexibility of the p-parameter. Changing this parameter affects the update of the centres and it determines how far each centre will move. This property allows the selection of different clustering combinations, penalising, or accentuating the representation of outliers in the clustering. In the case of the red tides data, as each instance of a red tide would be considered an outlier, it is necessary for us to use an algorithm to deal with these important data points in an appropriate manner. In this section we will show that the K-MLSIM is a powerful method that can extract the relevant clustering information.
We have shown previously that the ε-insensitive SIM [12] in data space, which is equivalent to using a *p* value of 0, will place each centre in the median of the cluster. This property was extended by [1] to use different values of p to achieve different clustering combinations. The larger the value of p the greater the contribution outliers will have in the clustering.

The effect that this parameter has on the clustering is important as it allows us to change the clustering to be more representative of the properties of the red tides data as it contains small numbers of outliers which are very important for the classification. If we were to cluster with a contemporary algorithm, we would likely be left with fewer centres that identify instances of red tides than we would like. In contrast the K-SIM can be tailored to give us more centres identifying red tides, resulting in a more expressive clustering.

Figure 6 shows the clusters produced by the K-MLSIM with different values of the *p* parameter. This parameter will penalize the effect of outliers in the data. This results in the centres being placed in the median of the data cloud.

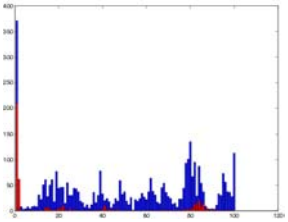

Fig. 6.a

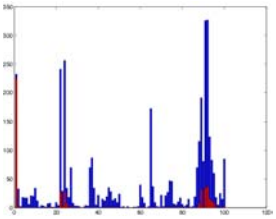

Fig. 6.b

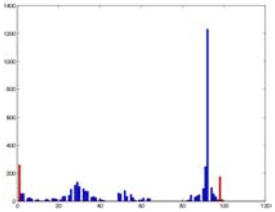

Fig. 6.c

Fig. 6: clustering of K-SIM on red tides data using 100 centres and a value of $p = 0$ (Fig.6a), a value of $p = 0.5$ (Fig 6b) and a value of $p = 2$ (Fig 6c).

As we can see from Figure 6a, the clustering is a compact coding with at most 800 data samples being assigned to one centre. This coding is less interesting for the red tides data as the abnormal points are those which are important, and so we wish to find a clustering which promotes rather than penalises them. In the figure 6b we use a value of $p = 0.5$ which places more emphasis on the larger changes in the weights, which in turn means that outliers, or abnormal data, will have greater effect on the learning. Thus we can ensure that the data points representing red tides will be strongly represented in the clustering, which is ideal for this problem. In Figure 6b we can see that there is a more sparse representation as there is a greater emphasis placed on large differences between the winner and the input data. In this figure we can see that the dense clusters are assigned fewer centres and that less dense clusters that contain outliers are more favoured. Assigning more centres to the less dense clusters also allows us to get a better representation within clusters. Commonly in contemporary clustering, one centre in this sparse region of the data would capture a mixture of red tides and spurious outliers. As we are now placing more centres in these regions, we have a better chance to separate the true red tides from other outliers.

Figure 6c. shows the clustering of the K-MLSIM on the red tides data using a p value of 2. The K-MLSIM is more penalizing to small changes in the weights than with $p = 0.5$. As can be seen from figure 6c it has provided an even more sparse representation of the clustering. This is a far more suitable clustering of the red tides data where we used smaller values of p. By changing the value of p in the weight update rule the K-MLSIM can be adapted to penalize or promote outliers in its clustering.

4 Conclusions

We have demonstrated a new technique for clustering. Of interest too is the fact that the method allows investigation of the nonlinear projection matrix K that readily reveals when a new situation behaves similarly, which may be very important in the identification of toxin episodes in coastal water.

References

1. Corchado E. and Fyfe C. The Scale Invariant Map and Maximum Likelihood Hebbian Learning. KES2002. Sixth International Conference on Knowledge-Based Intelligent Information Engineering Systems. Italy. (2002)
2. Corchado E. and Fyfe C. Relevance and kernel self-organising maps. In International Conference on Artificial Neural Networks, ICANN2003. (2003)
3. Corchado E., MacDonald D. and Fyfe C. Optimal projections of high dimensionald data. *In IEEE International Conference on Data Mining, ICDM02.* (2002)
4. Corchado E., MacDonald D. and Fyfe C. Maximum and Minimum Likelihood Hebbian Learning for Exploratory Projection Pursuit. *Data Mining and Knowledge Discovery. In Press*.
5. Fyfe C. PCA properties of interneurons. In From Neurobiology to Real World Computing, ICANN 93, (1993) pp. 183-188.
6. .Fyfe C. A scale-invariant feature map. *Network:Computation in Neural Systems,* 7: 269-275,1996.
7. Fyfe C. and Baddeley R. Non-linear data structure extraction using simple hebbian networks. Biological Cybernetics, (1995) 72(6):533-541.
8. Fyfe C., Baddeley R. and McGregor D.R. Exploratory Projection Pursuit: An Artificial Neural Network Approach, University of Strathclyde Research report/94/160. (1994).
9. Fyfe C. and Corchado J. M. Automating the construction of CBR Systems using Kernel Methods. International Journal of Intelligent Systems. Vol 16, No. 4, April 2001. ISSN 0884-8173.
10. Fyfe C. and MacDonald D. Epsilon-insensitive Hebbian Learning. Neurocomputing, (2002) 47:35-57.
11. Fyfe, C., MacDonald, D., Lai, P. L., Rosipal, R. and Charles, D. Unsupervised Learning with Radial Kernels in Recent Advances in Radial Basis Functions, Editors R. J. Howlett and L. C. Jain, Elsevier. (2000).
12. MacDonald D. Unsupervised Neural Networks for the Visualisation of Data. PhD Thesis, University of Paisley. (2002).
13. MacDonald D., Corchado E., Fyfe C. and Merenyi E. Maximum and Minimum Likelihood Hebbian Learning for Exploratory Projection Pursuit. In International Conference on Artificial Neural Networks, ICANN2002. (2002).
14. Scholkopf B.,The Kernel Trick for Distances. Technical report, Microsoft Research, May 2000.
15. Scholkopf B., Smola A. and Muller K. R. Nonlinear component analysis as a kernel eigenvalue problem. Neural Computation, (1998) 10:1299-1319.
16. Vapnik V. The nature of statistical learning theory, Springer Verlag. (1995)