Fault-Diagnosis Using Neural Networks with Ellipsoidal Basis Functions

S. Jakubek, T. Strasser
Institute for Machine and Process Automation, Vienna University of Technology
Gußhausstr. 27-29, A-1040 Vienna ; email: {jakubek},{strasser}@impa.tuwien.ac.at

Abstract

In this paper a fault detection scheme for applications in the automotive industry is presented. The detection scheme has to process up to several hundreds of different measurements at a time and check them for consistency. Our fault detection scheme works in three steps: First, principal component analysis of training data is used to determine nonsparse areas of the measurement space. Fault detection is accomplished by checking whether a new data record lies in a cluster of training data or not. Therefore, in a second step the distribution function of the available data is estimated using kernel regression techniques.

In order to reduce the degrees of freedom and to determine clusters of data efficiently in a third step the distribution function is approximated by a neural network. In order to use as few basis functions as possible a new training algorithm for ellipsoidal basis function networks is presented: New neurons are placed such that they approximate data points in the vicinity of their centers up to the second order. This is accomplished by adapting the spread parameters using Taylor's theorem.

Application to measured data from a real automotive process show that the proposed algorithm yields good results.

1 Introduction

In order to achieve maximum productivity, modern automotive measuring systems are now highly sophisticated, characterized by complex interrelationships between sub-systems. The developers' growing need for reliable assessments gives rise to an online monitoring system that automatically checks measured data for consistency.

The task is to analyse training data and to build a probability model based on these data using neural networks. However, in a measuring environment in the automotive field there are often several hundreds of measurements to be processed simultaneously which entails the problem that any model has to be built in a high dimensional measurement space.

Therefore, in a first step principal component analysis [1] is carried out on the data in order to find nonsparse regions of the measurement space. This is a common way to deal with a large number of measurements, especially in the field of fault detection in large industrial plants. Recent developments can be found in [2],[3],[4].

In the present application one has to deal with an unknown number of channels whereby physical interrelationships between two or more channels are not known to the system. Rather, it is required that the fault detection system adapts itself to all kinds of test bed configurations automatically. Therefore, we decided to use statistical pattern recognition methods for fault detection.

Before using the training data a transformation into the principal component coordinate system is carried out. This has the advantage that the new "channels" are sorted with respect to their significance. Using the first few principal channels distribution functions of different dimensions involving one or more channels are modeled. Since the training data are not labeled normal or abnormal it was assumed that all available data are normal. We decided to use kernel regression methods for this purpose since they represent a well established means for continuos distribution function estimation [5],[6],[2].

The distribution function resulting from kernel regression has two drawbacks: First, it needs a huge number of parameters which makes it difficult to determine structures like clusters of data points. This is crucial to the present application since training data can originate from merging data of different test cycles resulting in more or less isolated clusters. Second, the distribution function in it's given form is time consuming to evaluate and thus inappropriate for online application.

To overcome this problem in a second step the given distribution function is approximated by a neural network. In order to reduce the number of basis functions as much as possible we decided to use an ellipsoidal basis function network (EBFN). As opposed to RBF-networks, in an EBFN each basis function has ellipses as contour lines allowing closer adaptation to the data. Recent results can be found in [7],[8]. A closely related topic are ellipsoidal basis functions, used as fuzzy member functions in cluster analysis, see [9],[10]. In our approach we use the

neuron structure of a EBFN to fit a neuron to the target function as closely as possible using Taylor's theorem. It turned out that this requires a special training procedure for the EBFN, however, a target function can be approximated with significantly less neurons compared to a RBFN.

2 Kernel regression

After transforming the measurement channels using principal component analysis one has to find a smooth approximation for the probability density function $f(\boldsymbol{x})$ where $\boldsymbol{x}$ resembles an arbitrary point in the measurement space. One way to accomplish this is by placing a *kernel function* $K(\boldsymbol{x})$ of unit probability about each data point $\boldsymbol{x}_i$ ([5]). In the present application Gaussian kernels were used:

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \tag{1}$$

Assuming that $\dim(\boldsymbol{x}) = d$ (i.e. there are d measurements in each record) one gets for the approximation $\hat{f}(\boldsymbol{x})$ of the probability density function

$$\hat{f}(\boldsymbol{x}) = \frac{1}{n h_1 \dots h_d} \sum_{i=1}^{n} \left[\prod_{j=1}^{d} K\left(\frac{x_j - t_{ij}}{h_j} \right) \right] \tag{2}$$

Here n denotes the number of available records for training, x_j is the j-th component of $\boldsymbol{x}$, t_{ij} is the j-th component of the i-th data point and $h_1 \dots h_d$ are so-called *bandwidth* parameters. Given σ_j as the standard deviation of the j-measurement an optimal choice of h_j yields (see [5])

$$h_j = \left(\frac{4}{d+2} \right)^{1/(d+4)} \sigma_j n^{-1/(d+4)} \tag{3}$$

Taking into consideration the large number of data both the sum and the product in (2) make this formula time-consuming to evaluate. Moreover, it is difficult or even impossible to discover structures, like isolated clusters in the data. In our application isolated clusters of data result from merging data from different test-cycles which brings about the difficulty to distinguish between sets of outliers and weakly populated clusters.

Both problems give rise to develop a powerful method to reduce the number of parameters in $\hat{f}(\boldsymbol{x})$ whilst keeping the degree of information almost constant.

3 Network training

We decided to use a neural network with ellipsoidal basis functions to approximate the map $\boldsymbol{x} \to \hat{f}(\boldsymbol{x})$.

The advantage of ellipsoidal basis functions is that they can be fitted to the data with much more accuracy than ordinary radial basis functions. In our design a neuron function $n(\boldsymbol{x})$ even approximates $\hat{f}(\boldsymbol{x})$ up to the second order in the vicinity of it's center. This leads to excellent results with a relatively small number of neurons. The drawback of this method is that all parameters of the neuron, including it's weight are involved in a nonlinear fashion. Therefore, a special training algorithm had to be devised which will be presented in the sequel.

3.1 Neuron structure

In our design $n(\boldsymbol{x})$ is an ellipsiodal Gaussian basis function:

$$n(\boldsymbol{x}) = \gamma \exp\left(-\frac{1}{2} \tilde{\boldsymbol{x}}^T \boldsymbol{A} \tilde{\boldsymbol{x}} \right) \tag{4}$$

The weight is denoted as γ, the expression $\tilde{\boldsymbol{x}}$ denotes the vector from the neuron's center $\boldsymbol{\mu}$ to $\boldsymbol{x}$: $\tilde{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{\mu}$. The matrix $\boldsymbol{A}$ is symmetric and contains "spread parameters" which will be discussed below.

3.2 Neuron design

After the center vector $\boldsymbol{\mu}$ has been chosen, the weight γ is initially set to

$$\gamma = \hat{f}(\boldsymbol{\mu}). \tag{5}$$

Next, the derivatives of $n(\boldsymbol{x})$ with respect to the $x_j's$ are determined:

$$\frac{\partial n}{\partial \tilde{\boldsymbol{x}}}(\tilde{\boldsymbol{x}}) = \gamma \exp\left(-\frac{1}{2} \tilde{\boldsymbol{x}}^T \boldsymbol{A} \tilde{\boldsymbol{x}} \right) \left[-\tilde{\boldsymbol{x}}^T \boldsymbol{A} \right] \tag{6}$$

Since $\boldsymbol{A}$ is symmetric, the following relation also holds:

$$\frac{\partial n}{\partial x_i}(\tilde{\boldsymbol{x}}) = \gamma \exp\left(-\frac{1}{2} \tilde{\boldsymbol{x}}^T \boldsymbol{A} \tilde{\boldsymbol{x}} \right) \left[-\boldsymbol{A}(i,:) \tilde{\boldsymbol{x}} \right] \tag{7}$$

Differentiating (7) with respect to x_j yields

$$\frac{\partial^2 n}{\partial x_i \partial x_j}(\tilde{\boldsymbol{x}}) =$$

$$\gamma \exp\left(-\frac{1}{2} \tilde{\boldsymbol{x}}^T \boldsymbol{A} \tilde{\boldsymbol{x}} \right) \left\{ \left[-\boldsymbol{A}(j,:) \tilde{\boldsymbol{x}} \right] \left[-\boldsymbol{A}(i,:) \tilde{\boldsymbol{x}} \right] - \boldsymbol{A}(i,j) \right\} \tag{8}$$

Evaluating at the center ($\tilde{\boldsymbol{x}} = \boldsymbol{0}$) we get

$$\frac{\partial^2 n}{\partial x_i \partial x_j}(\tilde{\boldsymbol{x}} = \boldsymbol{0}) = -\gamma \boldsymbol{A}(i,j) \tag{9}$$

Hence, if we knew all second derivatives $\hat{f}_{x_i x_j}(\tilde{\boldsymbol{x}} = \boldsymbol{0})$ a good choice for the matrix $\boldsymbol{A}$ would be

$$\boldsymbol{A} = -\frac{1}{\gamma}\begin{pmatrix} \hat{f}_{x_1x_1} & \hat{f}_{x_1x_2} & \cdots & \hat{f}_{x_1x_n} \\ \hat{f}_{x_2x_1} & \hat{f}_{x_2x_2} & \cdots & \hat{f}_{x_2x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{f}_{x_dx_1} & \hat{f}_{x_dx_2} & \cdots & \hat{f}_{x_dx_d} \end{pmatrix}. \qquad (10)$$

If (10) is fullfilled the neuron is fit to it's target function $\hat{f}(\boldsymbol{x})$ up to the second order at it's center. However, there are two reasons why (10) is not a feasible choice: First, it is very time-consuming to compute all derivatives from (2). Moreover, the second derivatives $\hat{f}_{x_ix_j}$ can change rapidly if we move away from the center, and therefore choosing $\boldsymbol{A}$ according to (10) might not even be the optimal choice for the neuron design. Rather, it is desirable to find some "global derivatives" in the vicinity of the neuron's center:

Starting from the center of the neuron, we can expand $\hat{f}(\boldsymbol{x})$ (which is assumed to be analytic at $\boldsymbol{x} = \boldsymbol{\mu}$) into it's Taylor series:

$$\begin{aligned} &\hat{f}(\boldsymbol{x}) - \hat{f}(\boldsymbol{\mu}) = \\ &= [\,\hat{f}_{x_1}(\boldsymbol{\mu}) \quad \hat{f}_{x_2}(\boldsymbol{\mu}) \quad \ldots \quad \hat{f}_{x_d}(\boldsymbol{\mu})\,] \begin{bmatrix} \tilde{x}_1 \\ \vdots \\ \tilde{x}_d \end{bmatrix} + \\ &+\tfrac{1}{2}\left[\hat{f}_{x_1x_1}(\boldsymbol{\mu})\tilde{x}_1^2 + 2\hat{f}_{x_1x_2}(\boldsymbol{\mu})\tilde{x}_1\tilde{x}_2 + \ldots\right] + \ldots \end{aligned} \qquad (11)$$

Switching to matrix notation we get

$$\hat{f}(\boldsymbol{x}) - \hat{f}(\boldsymbol{\mu}) = (\nabla\hat{f})^T\tilde{\boldsymbol{x}} + \frac{1}{2}\tilde{\boldsymbol{x}}^T\boldsymbol{B}\tilde{\boldsymbol{x}} + \ldots \qquad (12)$$

with $\boldsymbol{B}(i,j) = \hat{f}_{x_ix_j}$.

Matching coefficients one gets

$$\boldsymbol{A} = -\frac{1}{\gamma}\boldsymbol{B}. \qquad (13)$$

The parameters $\nabla\hat{f}$ and $\boldsymbol{B}$ in (12) can be approximated as follows: After truncating the Taylor series to second order terms we take all data points $\boldsymbol{x}_v$ in the vicinity of the center $\boldsymbol{\mu}$ and use their associated function values $\hat{f}_v$ to determine $\nabla\hat{f}$ and $\boldsymbol{B}$. This is done by solving (12), truncated to the second order for all $\boldsymbol{x}_v$ in a least-squares sense. In order to guarantee the output of the network is bounded the eigenvalues of $\boldsymbol{A}$ must be all positive. This can be readily seen from (4). One way to accomplish this is by placing neurons only into extrema of the target function (i.e. $\hat{f}(\boldsymbol{x})$) so that the curvature is negative in all directions. Also, at an extremum the gradient $\nabla\hat{f}$ can be set to zero which reduces the computational effort.

For the sake of clarity this procedure is outlined for the case $d = 2$ where $\boldsymbol{x} = [\,x \quad y\,]^T$:

$$\hat{f}(\boldsymbol{x}) - \hat{f}(\boldsymbol{\mu}) = \frac{1}{2}\left(\hat{f}_{xx}\tilde{x}^2 + \hat{f}_{yy}\tilde{y}^2 + 2\hat{f}_{xy}\tilde{x}\tilde{y}\right) \qquad (14)$$

Applying (14) to all r data points $\boldsymbol{x}_{v,1}\ \ldots\ \boldsymbol{x}_{v,r}$ in the vicinity of the center yields

$$\frac{1}{2}\begin{bmatrix} \tilde{x}_1^2 & \tilde{y}_1^2 & 2\tilde{x}_1\tilde{y}_1 \\ \tilde{x}_2^2 & \tilde{y}_2^2 & 2\tilde{x}_2\tilde{y}_2 \\ \vdots & & \\ \tilde{x}_1^r & \tilde{y}_1^r & 2\tilde{x}_r\tilde{y}_r \end{bmatrix}\begin{bmatrix} \hat{f}_{xx} \\ \hat{f}_{yy} \\ \hat{f}_{xy} \end{bmatrix} = \begin{bmatrix} \hat{f}(x_{v,1}, y_{v,1}) - \hat{f}(\boldsymbol{\mu}) \\ \hat{f}(x_{v,2}, y_{v,2}) - \hat{f}(\boldsymbol{\mu}) \\ \vdots \\ \hat{f}(x_{v,r}, y_{v,r}) - \hat{f}(\boldsymbol{\mu}) \end{bmatrix} \qquad (15)$$

Of course, this procedure does not produce accurate results for the actual derivatives. However, the resulting $\boldsymbol{A}$-matrix fits the neuron closely to the given data in the vicinity of it's center. From (13) it becomes obvious that the weight γ also influences the spread parameters. As a consequence standard training algorithms cannot be applied to the given network and a special training procedure had to be devised:

3.3 Iterative Network training

As already mentioned earlier, the proposed neuron structure (4) needs a special training procedure. It is required that this procedure has a high degree of reliability since it is supposed to operate in an automated fault detection system that runs without human interaction. The training algorithm we propose works iteratively: In the first iteration loop neurons are placed until a certain degree of accuracy is reached. In the following loops each of these neurons is re-designed successively to improve the overall net performance.

First loop:

1. Place the center of the first neuron $n_1(\boldsymbol{x})$ at the data point $\boldsymbol{x}_{max,1}$ where $\hat{f}$ has it's maximum:

   $$\boldsymbol{\mu}_1 = \arg\max(\hat{f}_i(\boldsymbol{x}_i)) \quad i = 1 \ldots n. \qquad (16)$$

   Compute the remaining parameters of the neuron as described above.

2. Place the center of the second neuron $n_2(\boldsymbol{x})$ at the data point $\boldsymbol{x}_{max,2}$ where $f_2 = \hat{f} - n_1$ has it's maximum:

   $$\boldsymbol{\mu}_2 = \arg\max(f_2(\boldsymbol{x}_i)) \quad i = 1 \ldots n. \qquad (17)$$

Compute the remaining parameters of the neuron as described above.

3. Continue to place neurons as described until the maximum of f_p lies below a certain threshold ϵ:

$$\max(f_p(\boldsymbol{x}_i)) \leq \epsilon \quad i = 1 \ldots n. \tag{18}$$

Now the number of neurons is fixed to p. (The target function f_k for the k-th neuron is defined recursively: $f_k = f_{k-1} - n_{k-1}$, $f_1 = \hat{f}$).

General loop:

1. Re-position the center of the first neuron $n_1(\boldsymbol{x})$ at the data point $\boldsymbol{x}_{max,1}$ where $f_1 = f_p + n_1$ has it's maximum:

$$\boldsymbol{\mu}_{1,new} = \arg\max(f_1(\boldsymbol{x}_i)) \quad i = 1 \ldots n. \tag{19}$$

(The function f_1 describes the remaining error of the network if only the first neuron is removed). Compute the remaining parameters of the neuron $n_{1,new}$ as described above.

2. Re-position the center of the the second neuron $n_2(\boldsymbol{x})$ at the data point $\boldsymbol{x}_{max,2}$ where $f_2 = f_1 + n_{1,new} - n_2$ has it's maximum:

$$\boldsymbol{\mu}_2 = \arg\max(f_2(\boldsymbol{x}_i)) \quad i = 1 \ldots n. \tag{20}$$

(Again, f_2 describes the remaining error of the network if the second neuron is removed). Compute the remaining parameters of the neuron $n_{2,new}$ as described above.

3. Continue to re-position and re-design all p nerons.

The iteration is repeated until a truncation condition is satisfied (in our case if the accuracy of the network ceases to improve). To conclude the training all weights γ_k $(k = 1 \ldots p)$ are re-computed using standard least-squares. Changes of the γ_k are not taken into account in (13). This step is used to finally rule out overlapping effects between the single neurons in the above design process. It turned out, however, that the concluding least-squares optimization does not change the γ_k significantly any more. The training procedure described above proved to work reliably on all the available training data and under all desired model dimensions $d = 1$ up to $d = 5$.

4 Residual determination

The probability of a data point $\boldsymbol{x}$ lying in a domain $D(\boldsymbol{x}_0)$ about an arbitrary $\boldsymbol{x}_0$ is given by

$$P(D, \boldsymbol{x}_0) = \int_{D(x_0)} f(\boldsymbol{x}) d\boldsymbol{x} \tag{21}$$

For a fixed D, $P(D, \boldsymbol{x}_0)$ can be used to check different regions of the measurement space for their probability of containing a data point. The maximum probability is given about $\boldsymbol{x}_{max} = \arg\max f(\boldsymbol{x})$:

$$P_{max}(D) = \int_{D(x_{max})} f(\boldsymbol{x}) d\boldsymbol{x} \tag{22}$$

Now a residual $r(\boldsymbol{x})$ can be generated by comparing any $P(\boldsymbol{x})$ to P_{max}:

$$r(\boldsymbol{x}) = \frac{P(D, \boldsymbol{x})}{P_{max}(D)} \tag{23}$$

If we let $D \to 0$ we obtain

$$r(\boldsymbol{x}) = \frac{f(\boldsymbol{x})}{f_{max}} \tag{24}$$

The residual definition given by (24) compares the "local" probabilities of presence of a data points. Values close to one can be found in areas of high data density whereas values close to zero indicate a possible fault.

Fig. 1 shows a sample FDI system where four one-dimensional models based on PCA-channels #1 through #4 and two two-dimensional models based on channels #1,#2 and #2,#3 respectively are used to generate six residuals. In the actual application various models of different dimensions are built from the PCA-data. Two dimensional models have as inputs PCA-channels (#1,#2), (#2,#3), (#3,#4), three dimensional models are built using combinations (#1,#2,#3), (#2,#3,#4) and so on. In general, residuals from all models are used to decide wether a given data record is faulty. If more then 50% of the residuals fall below a certain threshold the system issues an alarm.

5 Results

Fig. 2 illustrates a simulated example: An analytic function $x \to y(x)$ (solid black curve) is corrupted by white noise, "measured" data are represented by black dots. The resulting network that was built from the data consists of 12 ellipsoidal basis functions, illustrated by contour-lines. Despite the strong noise the ridge-line (not plotted here) fits the actual function $y(x)$ accurately. The power of the proposed method when applied to function approximation lies in the fact that virtually all relationships – even non-unique maps – can be approximated. For the given application this is extremly important since the direction of a mutual dependancy between two or even more channels is not known a priori.

Figures 3 and 4 illustrate a two-dimensional model using real data. It can be seen that $\hat{f}(x_1, x_2)$ consists of iso-

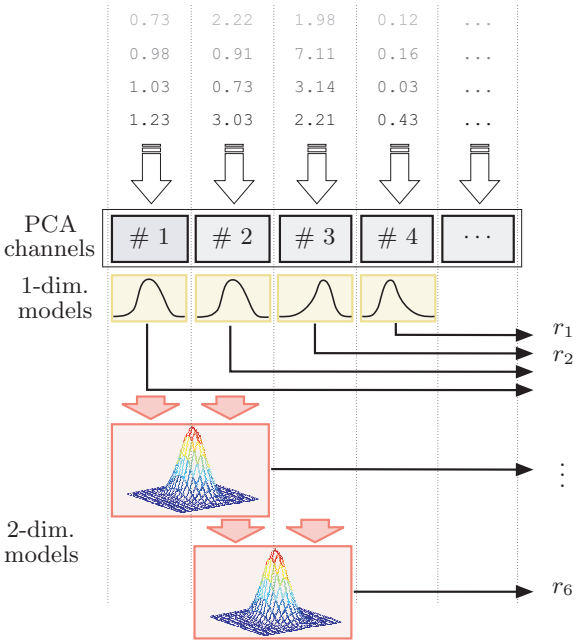

Figure 1: Residual generation scheme

lated clusters with varying densities. The network consists of 13 ellipsiodal neurons. The root mean square error over all data points was 1%. In order to illustrate the power of the proposed training method an ordinary radial basis function network was trained for comparison. Even after optimization of the spread parameter it took 22 Neurons to match the accuracy of the EBF-network. Our general experience was that the EBF-network with the proposed training algorithm needs about half the number of neurons compared to a RBF-network of the same accuracy.

6 Conclusion

In this paper an approach for fault detection in automotive measuring systems was presented. The main problem that had to be solved was that a large number of measurements has to be monitored with as few parameters as possible and with reasonable computational effort. The proposed method transforms the data using principal component analysis and then models relationships of differnet dimensions between the data channels using kernel regression techniques. In order to reduce the amount of necessary model parameters a new training algorithm for ellipsoidal basis function networks is presented. The resulting network uses significantly less basis functions than a radial basis function network of the same accuracy. Application examples to a simulated example and to real data show that the proposed method yields excellent results.

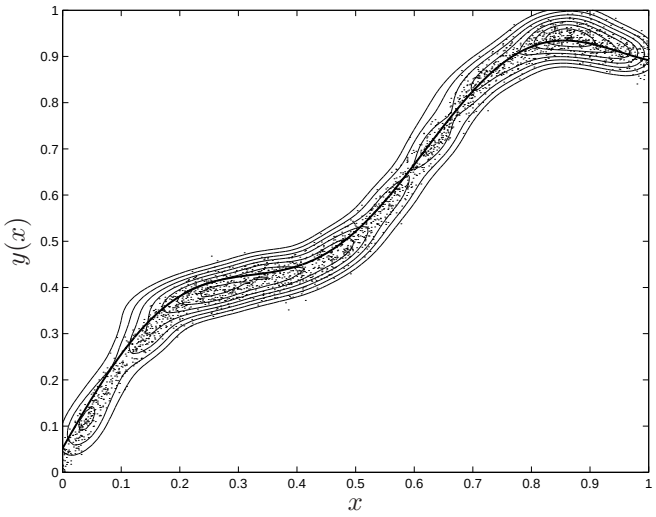

Figure 2: Sample function approximation example with very noisy data

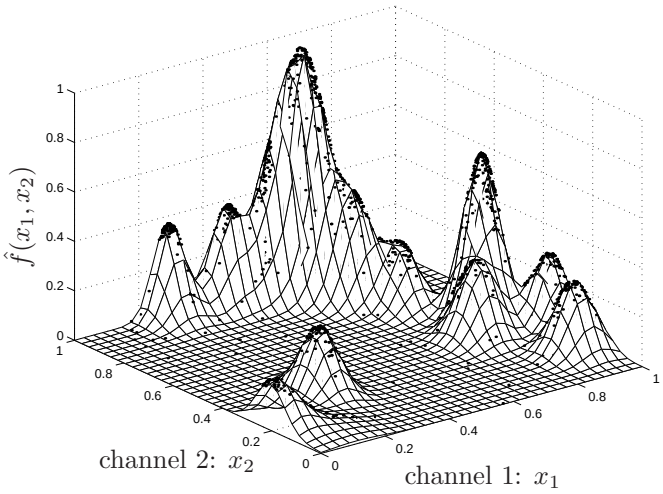

Figure 3: pdf-approximation example with isolated clusters

References

[1] J.E. Jackson. *A User's Guide to Principal Components*. John Wiley and Sons, New York, 1991.

[2] A. Ruiz. Nonlinear kernel-based statistical pattern analysis. *IEEE Transactions on Neural Networks*, 12:16–32, January 2001.

[3] J. Gertler. Isolation enhanced principal component analysis. *AIChE Journal*, 45:323– 334, 1999.

[4] N. Kaistha and B.R. Upadhyaya. Incipient fault detectionin a pwr plant using principal component analysis. In *Proceedings of the American Control Conference, Alrington, VA, June 25-27,2001*, pages 2119–2120, 2001.

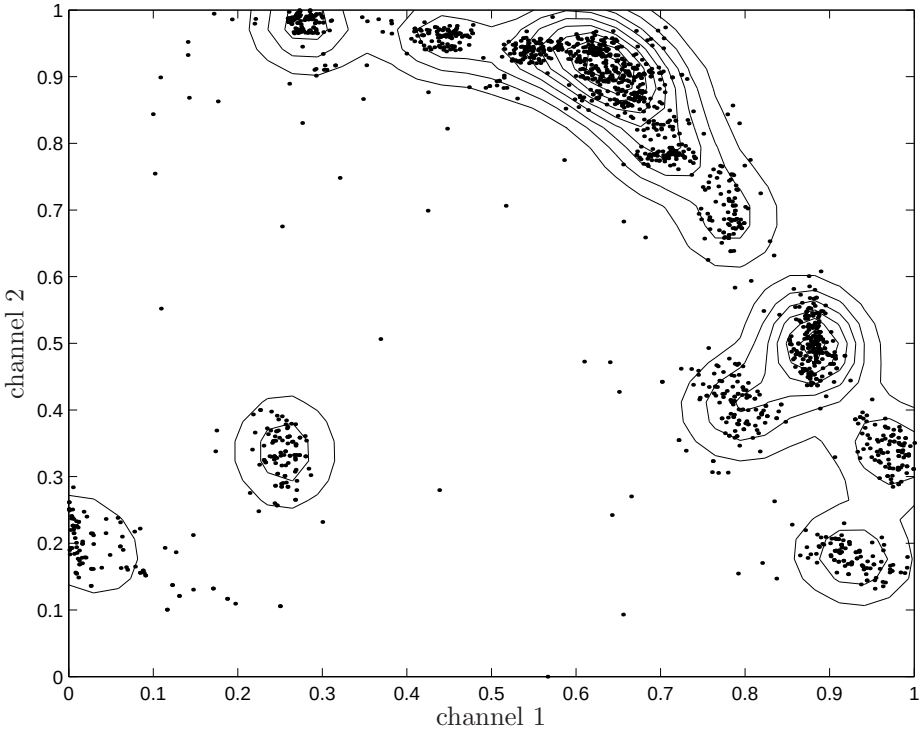

Figure 4: pdf-approximation example with isolated clusters

[5] Kyle S. Cranmer. Kernel estimation for parametrization of discriminant variable distributions. *http://www-wisconsin.cern.ch/∼cranmer/keys.html*, 1999.

[6] A. Zaknich. Efficient kernel functions for the general regression and modified probabilistic neural networks. In *International Joint Conference on Neural Networks, IJCNN '99*, volume 2, pages 1446 –1449, 1999.

[7] Min Woong Hwang, Mun Hyuk Kim, and Jin Young Choi. Second order multilayer perceptrons and its optimization with genetic algorithms. In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 1, pages 652–658, 2000.

[8] Hiroyasu Kubota, Hisashi Tamaki, and Shigeo Abe. Robust function approximation using fuzzy rules with ellipsoidal regions. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, volume 6, pages 529–534, 2000.

[9] Shigeo Abe and Ruck Thawonmas. Fast training of a fuzzy classifier with ellipsoidal regions. In *Proc. the Fifth IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'96), New Orleans, LA*, volume 3, pages 1875–1880, Sept. 1996.

[10] A. Shigeo. A fuzzy classifier with ellipsoidal regions for diagniosis problems. *IEEE Transactions Systems, Man and Cybernetics*, 29(1):140–149, February 1999.