Institut für Informatik
Fachbereich Informatik und Mathematik

GOETHE
UNIVERSITÄT
FRANKFURT AM MAIN

Real-valued Feature Selection for Process Approximation and Prediction

Frank Heister, Rüdiger Brause

Nr. 1/09

Frankfurter Informatik-Berichte

Institut für Informatik • Robert-Mayerst.11-15 • D-60054 Frankfurt am Main, Germany

ISSN 1868-8330

Real-valued Feature Selection
for process approximation and prediction

Frank Heister, Rüdiger Brause
Department of Computer Science and Mathematics
Johann Wolfgang Goethe-University
60054 Frankfurt, Germany
r.brause (at) informatik.uni-frankfurt.de

Abstract

The selection of features for classification, clustering and approximation is an important task in pattern recognition, data mining and soft computing. For real-valued features, this contribution shows how feature selection for a high number of features can be implemented using mutual information. Especially, the common problem for mutual information computation of computing joint probabilities for many dimensions using only a few samples is treated by using the Rènyi mutual information of order two as computational base. For this, the Grassberger-Takens correlation integral is used which was developed for estimating probability densities in chaos theory. Additionally, an adaptive procedure for computing the hypercube size is introduced and for real world applications, the treatment of missing values is included. The computation procedure is accelerated by exploiting the ranking of the set of real feature values especially for the example of time series.

As example, a small blackbox-glassbox example shows how the relevant features and their time lags are determined in the time series even if the input feature time series determine nonlinearly the output. A more realistic example from chemical industry shows that this enables a better approximation of the input-output mapping than the best neural network approach developed for an international contest.

By the computationally efficient implementation, mutual information becomes an attractive tool for feature selection even for a high number of real-valued features.

Key words: feature selection**,** process approximation, Rènyi mutual information, classification, clustering, Takens-Grassberger correlation integral,

1 Introduction: Feature selection methods

For many applications the selection of proper input features is very important. Good features are essential for good diagnosis, prognosis, classification and approximation used in the medical, financial and industrial area. What are "good" features and how are they obtained? If we have many input features, how do we know which are the most salient ones? This is the classical task for feature selection and depends heavily on the application.

This paper shows how real-valued features can be selected using information as performance measure. We concentrate on features which contribute most of the information to the target of the application, e.g. a diagnosis or a prognosis. As information measure, we start with the traditional Shannon information using mutual information I(**X**;Y) between the input features **X** and a target Y.

For qualitative features, e.g. features exhibiting a final number of states or qualitative labels like "green", "red", "good", "sweet", the computation of mutual information between the features and the target variable is quite common for building decision trees, see [1], and are based on the probability evaluation of the states ("counting the states"). For real valued features, this is not possible any more, because we have an infinity of states. Here, other methods have to be considered.For real-valued input, there are many different ways of extracting relevant features for a target, like non-supervised linear transformations as the Principal Component Analysis (PCA) or the Independent Component Analysis (ICA). Both methods are based on a linear transformation of the basis vectors. For PCA, after transformation the components of maximal variance are selected, all others are dropped ("transform coding”). Unfortunately, the results of PCA depend on the scaling of the input variables. For ICA, those components are manually selected which have a high semantic meaning. Both transformation methods can not be applied to situations where input data measurements are sparse or expensive or just not possible like all clinical tests or examinations; PCA and ICA need always all features for computing the new ones. In many cases, a feature selection process has to be applied before any transformation; measuring all possible features is not possible and acceptable.

Another common approach for real valued feature selection is the sensibility analysis, e.g. [1] which checks the influence of each input for the approximated output when an approximating system is given. For instance, the error of an approximation model for the desired output gives us a performance measure of that model. Other approaches like the correlation coefficient try to predict one variable by another variable [3]. The disadvantage of this method is that it can not be applied to non-linear dependencies.

Before we discuss the main task of this paper how to compute the mutual information, let us consider how to select the most important features, based on mutual information. There are two simple greedy procedures for selecting features based on a performance criterion: the sequential forward selection and the sequential backward selection approach [4].

> The sequential *forward selection* approach starts with one feature and checks the performance. The best one is chosen and kept. Then a second one is chosen which gives the best result under the condition of using the first feature. Then a third feature is selected, using the first and second selected ones. This is iterated and gives us a ranked list of inputs. If we choose a threshold of performance, we might drop all features of that list which have a lower rank and are not necessary for the performance.
>
> The sequential *backward selection* approach uses all features first, and then drops the feature which affects the performance at least. This gives us the first feature to drop. For the second one, the same procedure applies as before: only that feature is dropped which has the weakest influence on the model performance. Continuing the iterated procedure gives us also a ranked list of input features.

In general, both methods are greedy algorithms, and, like most greedy algorithms, not optimal: they tend to get stuck in local optima and are not to guarantee a global optimum, i.e. the optimal subset of features of all possible 2^N subsets. Nevertheless, it is clear that for a high number *N* of possible features (e.g. N= 200) a systematic evaluation of the whole search space of 2^N subsets is out of question. Both methods are simple to implement and fit to most needs.

From the computational point, the forward method is more advantageous if the desired number *k* of features is closer to *1* than to *N*; otherwise, the backward method requests fewer steps. On the other hand, if there is a subset of features where all members are necessary, the backward selection method shows better results: it avoids breaking up the subset. This situation is explained best by an example.

Example 1

Consider for instance the CorrAL dataset of [1] with the binary feature set M_6 = {A0, A1, B0, B1, C, D} of n = 6 features where binary target Y is determined by the subset of four features {A0, A1, B0, B1} by Y = $(A0 \wedge A1) \vee (B0 \wedge B1)$. Feature C is assumed to correlate to Y in 75% of all cases and gives a mutual information of I(Y;C) = 0.31 whereas feature D is independent of Y with I(Y;D) = 0.01. Assuming equal probabilities for features A0, A1, B0, B1, D of P(1) = P(0) = 0.5 the involved conditional probabilities and mutual information can be easily computed, see Appendix A.

The sequential forward selection strategy using mutual information will select feature C with I(Y;C) = 0.3 as most important one, all other features with I(Y;A0) = 0.1 afterwards, and feature D with I(Y;D) = 0.01 as last one. If we look for the feature set of the best k=4 features of n = 6 ones we will get for instance M_4 = {C, A0, A1, B0} which gives a mutual information of I(Y;M_4) = H(Y) – 1/8 = 0.8637.

In contrast to this, the backward selection strategy will first drop feature D and then C, because dropping any other feature will not conserve the initial mutual information of I(M_6) = H(Y) = 0.9887, and with the reduced feature set of M_4 = {A0, A1, B0, B1} the mutual information I(Y;M_4) = H(Y) is still maximal, see Appendix A. Therefore, the backward selection scheme selects a better subset of k = 4 features than the forward selection scheme.

Please note that A0, A1, B0 and B1 are independent features, i.e. the mutual information between them is zero. Nevertheless, the grouping effect takes place. Therefore, independency of features is not sufficient for reaching the global optimum in the forward selection procedure.

In conclusion, closed feature groups are better treated by backward selection. Nevertheless, if there are no closed feature groups (which is indicated by no sudden jumps in mutual information when adding features continuously) the forward selection method is faster for selecting only a few features out of many.

Certainly, there are other methods which are more efficient than greedy algorithms, like the *floating search* method [5] which includes features again already dropped, or the *branch and bound* method [6][7] which depends on the monotony of the performance criterion. Nevertheless, since we have no grouped features in our examples and the methods above are computationally more complex than simple greedy algorithms, they are not used here.

The usual precondition for using the forward and backward procedures is the existence of an already defined model for measuring the performance, e.g. the error of an approximation or classification. In contrast to this, in our case we want first to choose the inputs and then subsequently build up an adaptive model based on that choice of features. Therefore, for selecting the most relevant input features we choose a special performance function: The probability of frequent input-output pairs of sample values which does not need an explicit model. This is done by taking the mutual information between input and output. Let us formalize this concept in detail.

2 Mutual information based feature selection

Let us use the concept of mutual information in selecting the input features. Given all input features, we have to select a proper subset of them. The concept of evaluating all possible subsets is prohibited by the combinatorial explosion of the number of subsets to be tested. Instead, we use the simple forward procedure already introduced using mutual information as performance criterion.

Let us start with the observed *n* input features $X_1 \ldots X_n$ and the output *Y*. With the definition of the mutual information [8]

$$I(X;Y) = H(Y) - H(Y|X) = H(Y) + H(X) - H(Y,X) \tag{1}$$

and the Shannon entropy [9]

$$H(X) = -\sum_{i=1}^{m} P_i \log(P(x_i)) = -\langle \log p(x_i) \rangle_i$$

we know that for random variables X and Y we have $H(Y) \geq H(Y|X)$ or

$$I(X;Y) \geq 0 \tag{2}$$

which becomes zero only for independent variables, see [8]. On the other hand, the more *Y* depends on feature x_i the amount of mutual information increases independently of the kind of input-output function. On this observation we build our selection procedure. The base for the greedy forward and backward selection algorithms is the *chain rule for mutual information*, see [8], Theorem 2.5.2:

$$I(X_1,X_2,\ldots,X_n;Y) = I(X_1;Y) + I(X_2;Y\,|X_1) + I(X_3;Y\,|\,X_1,X_2) + \ldots + \ldots$$

$$= \sum_{i=1}^{n} I(X_i; Y \mid X_{i-1}, X_{i-2},\ldots, X_1)$$

Thus, the mutual information between the *n* random variables and target *Y* can be obtained by adding *n* terms of mutual information between the target and a feature on the condition of a sequentially growing feature set. In every step, the mutual information is conditional upon the joint distribution of the already selected features (random variables). Thus, we only count the additional information a feature gives us about the target in the light of the already included features, not the full information of the feature including redundancy.

If we choose the normalized version of mutual information

$$\tilde{I}(X;Y) = \frac{H(Y) - H(Y \mid X)}{H(Y)} = \frac{H(Y) + H(X) - H(X,Y)}{H(Y)}$$

we have the property

$$0 \le \tilde{I}(X;Y) \le 1 \qquad (3)$$

If not denoted otherwise, we use the normalized version of mutual information in the rest of the paper. For the subsequent examples, we choose the mutual information forward selection algorithm as follows:

```
Forward selection algorithm
  i = 1. As first input feature $X_{k1}$, select the feature with the highest mutual information I(.)
      $X_{k1} = \arg \max_{X_j} I(X_j;Y)$
  FOR i :=1 TO n DO
      i := i+1.  Select the next variable $X_{ki}$ giving the highest mutual information
      $X_{ki} = \arg \max_{X_j} I(X_j; Y \mid X_{k1},\dots,X_{ki-1})$        (4)
  ENDFOR
```

The algorithm gives us a ranked list of variables $\{X_{k1},\dots,X_{kn}\}$. For a set of only relevant features, we might stop the algorithm as soon as possible, e.g.

- we have reached the number of predefined input variables,
- or the amount I(.) gets no significant information increase any more,
- or the amount I(.) surpasses a predefined threshold θ,
- or the computation time surpasses a predefined threshold.

3 Computing mutual information

The main reason why information based real valued feature selection is not commonly used, is an efficient procedure for computing the mutual information. The computation of mutual information I(.) is based on the computation of the entropies H(Y), H(**X**), and H(**X**,Y) using the joint distributions P(**X**) = $P(X_{k1},\dots,X_{kn})$ and P(**X**,Y). The computation of an entropy H(.) is impeded by the fact that in the standard case we do not have the necessary number of input samples for the computation to avoid the curse of dimensionality for a high number *n* of dimensions.

For the Shannon information measure, a correct sample frequency can hardly be measured in a high-dimensional interval. For example, as rule of thumb a histogram of at least 10 intervals is necessary for a rough estimation. If we use 100 samples for these 10 intervals, the resulting histogram gives a fair distribution approximation. Now, using n = 20 variables, the same average density of 10 samples per interval can only be obtained by using $100^{20} = 10^{40}$ samples, which is prohibitively high.

The basic problem for computing the information lies in the difficulty of computing the joint probability density of many random variables based on only a few samples. There are several possibilities for estimating the probability density. One idea is to circumvent the problem by approximating the mutual information between the feature set and the target output by a weighted sum of the pairwise mutual informations of the feature set, see for example Battini [10]. Certainly, for more complicated interactions this does not replace the real conditional mutual information.

Another approach is based on the Parzen window approach [8]: Each sample is taken as the center of a Gaussian distribution of variance 1; the superposition of all Gaussians is taken as the desired density function. For mutual information, this approach was introduced by Principe et al. [12] and used for learning e.g. by Torkkola [13].

Let us follow another approach. We might try to circumvent the problem that we do not have sufficient samples per histogram interval by taking also the samples in the neighboured intervals into account, i.e. by averaging the number of samples. For the Shannon entropy the average is taken after computing the probability and its logarithm, not before. So, averaging the number of samples is not possible. If we could inverse the sequence, i.e. first compute the average of the probability and then take the logarithm, we can profit not only from the neighbours, but also avoid costly computations of the logarithm. This is performed by the following approach.

By Jensen's inequality for the convex function f(Z) (see theorem 2.6.2 in [8]) which is true for random variables Z,

$$\langle f(z_i)\rangle_i \geq f\left(\langle z_i\rangle_i\right)$$

we know that for the negation of the relation

$$H(X) = -\langle \log P_i\rangle_i \leq -\log\langle P_i\rangle_i \equiv H_2(X)$$

holds. Here, the average can be computed very efficiently, see appendix A. The function

$$H_2(X) = -\log\langle P(x_i)\rangle_i = -\log\sum_{i=1}^{m} P_i P(x_i) = -\log\sum_{i=1}^{m} P_i^2 \qquad (5)$$

is called the *Rényi entropy H_α of order $\alpha = 2$* (see [23]) which is a generalization of the Shannon entropy.

Formally, the Rényi entropy is defined as follows. Let ξ be a discrete random variable with the probability distribution

$$P(\xi=x_i) \equiv \{P_i\},\ i = 1..m \quad \text{and} \quad \sum_{i=1}^{m} P_i = 1$$

Then, the Rényi-Entropy of order $\alpha \in \Re$, $\alpha \geq 0$ is defined by

$$H\alpha(X) := H\alpha(\{Pm\}) = \begin{cases} \dfrac{1}{1-\alpha}\log_2\sum_{i=1}^{m} P_i^\alpha & : \alpha \geq 0, \alpha \neq 1 \\ -\sum_{i=1}^{m} P_i \log_2 P_i & : \alpha = 1 \end{cases} \qquad (6)$$

with $0^0 := 0$ and $0\cdot\log_2(0) := 0$. Here, for α = 1 we get the Shannon entropy.

For independent random variables *X*, *Y* we know that the mutual information I(X;Y) in eq.(1) becomes zero, which is also valid for $I_2(X;Y)$. Nevertheless, in our case $I_2(X;Y)$ may also become negative, and from $I_2(X;Y) = 0$ we can not deduce the independence of *X* and *Y* because relation (2) does not hold any more. Therefore, our selection procedure may be no longer accurate. This problem is illustrated in Appendix B by an example.

There is one possibility of ensuring proportion (2): for a uniformly distributed *Y* we prove in Appendix B that

$$\min\{H_2(X), H_2(Y)\} \geq I_2(X;Y) \geq 0 \qquad (7)$$

with I_2 equal to zero only in the case where X and Y are independent distributions. So, our selection procedure still holds for uniform target distributions. This is the reason why we transfer the target time series *Y* to uniform distribution before we use it for computing $I_2(X;Y)$. The resulting time series *Y* is changed, but it still reflects the time series dynamics.

4 Computing mutual information of order 2

Most of the work in using mutual information feature selection is bound to the implementation. There are several problems and their solutions which are described here. Let us start with the basic computation procedure and then consider the acceleration of the computation afterwards.

4.1 The basic computations

Let us assume that our samples $\{\mathbf{x}\}$ are from the *n*-dimensional space $\Re^n$. For time series, the i-th sample $\mathbf{x}(t_i)$ is taken at time point t_i. The number *c* of samples $\mathbf{x}(t_2)$ in its neighbourhood is the number of all samples within a hypercube of length ε for index $t_1 \neq t_2$, i.e. within interval $[x_i(t_1) - \varepsilon/2, x_i(t_1) + \varepsilon/2]$ for all dimensions *i*.
We have

$$c(t_1) \; = \left| \left\{ (t_1,t_2) \,|\, \bigwedge_{i=1}^{n} \left\| x_i(t_1) - x_i(t_2) \right\| < \tfrac{\varepsilon}{2}, t_2 = 1,...,T \right\} \right| \tag{8}$$

$$= \left| \left\{ (t_1,t_2) \,|\, B(t_1,t_2) = \mathrm{TRUE}, t_2 = 1,...,T \right\} \right|$$

$$\text{with } B(t_1, t_2) = \bigwedge_{i=1}^{n} \left\| x_i(t_1) - x_i(t_2) \right\| < \tfrac{\varepsilon}{2} \tag{9}$$

$$= \sum_{t_2} b(t_1,t_2) \quad \text{with } b(t_1,t_2) = \begin{cases} 0 & B(t_1,t_2) = \mathrm{FALSE} \\ 1 & B(t_1,t_2) = \mathrm{TRUE} \end{cases}$$

All decisions $\{b(t_1,t_2)\}$ can be represented by a binary matrix $\mathbf{b} = [b(i,j)]$ containing them. The number $c(t_1)$ is the number of co-occurrences of the sample events within the ε-hypercube.
The average number over all *T* possible values for t_1 is

$$C = \sum_{i=1}^{T} p(c_i)c_i \; = \sum_{i=1}^{T} p(c_i) \sum_{j=1}^{T} b(i,j) \; = \frac{1}{T} \sum_{i=1}^{T} \sum_{j=1}^{T} b(i,j) \tag{10}$$

This is referenced as the "correlation integral" introduced by Takens [19] and Grassberger & Procaccia [19] and used there for probability estimation in chaotic systems.
The relative number of samples per ε-hypercube, an estimation for the average probability within a cube, becomes

$$\langle P \rangle = C/T = \sum_{i=1}^{T} p(c_i) \frac{c_i}{T} \; = \sum_{i=1}^{T} p(c_i) p(c_i) \; = \sum_{i=1}^{T} p(c_i)^2$$

and the negative logarithm of it becomes

$$H\,(\mathbf{x}) = -\log\langle P(\mathbf{x}) \rangle = -\log \left(\sum_{i=1}^{T} p(c_i)^2 \right) = -\log \; \frac{1}{T^2} \sum_{i=1}^{T} \sum_{j=1}^{T} b(i,j) \tag{11}$$

4.2 Accelerating the computations

Thus, by eq. (11) the final procedure for computing the multi-dimensional entropy is reduced to counting samples b(i,j) within ε-hypercubes. Since we have T samples, we have to compute $T \cdot T = T^2$ decisions to cover all possible (t_1,t_2) tuples; the runtime complexity is $O(T^2)$.

4.2.1 Acceleration by symmetry

Since we have symmetric decisions $b(t_1,t_2) = b(t_2,t_1)$ we might facilitate the task by computing only half of the decisions, i.e. the upper triangular part of the matrix . The T^2 decisions are separated into the *T* trivial elements b(i,i) = 1 and the T(T−1) non-trivial elements b(i,j), i ≠ j

$$\langle P(\mathbf{x})\rangle = \frac{1}{T^2}\sum_{i=1}^{T}\sum_{j=1}^{T} b(i,j) = \frac{1}{T^2}\left(\sum_{i=1}^{T} b(i,i) + \sum_{i=1}^{T}\sum_{\substack{j=1\\ j\neq i}}^{T} b(i,j)\right)$$

$$= \frac{1}{T^2}\left(\sum_{i=1}^{T} b(i,i) + 2\sum_{i=1}^{T}\sum_{j=i+1}^{T} b(i,j)\right) = \frac{1}{T\cdot T}(T + 2C_x) = \frac{1}{T}\left(1+\frac{2C_x}{T}\right)$$

$$= \frac{T}{T^2} + \frac{2C_x}{T^2} = \frac{2C_x + T}{T^2} \qquad (12)$$

The number C_x of comparisons in the upper triangular matrix of decisions (b_{ij}) is doubled, since they are symmetric, and complemented by the value of the main diagonal b_{ii}. The relation to the number of all possible decisions is the sample average probability.
Therefore,

$$H(\mathbf{x}) = -\log\frac{1}{T}\left(1+\frac{2C_x}{T}\right) \quad \text{with } C_x = \sum_{i=1}^{T}\sum_{j=i+1}^{T} b(i,j) \qquad (13)$$

4.2.2 Acceleration by ranking

Further acceleration can be found by decreasing the number of decisions. Equation (13) still suggests that we have to compare all *T* samples with the rest of the samples. Since we have to rank at least the target samples before computing the information, we might as well profit by the already existing index arrays of the ranking, see by Pompe and Heilfort [21]. The main idea is illustrated by drawing of the probability density function p(f) of a time series f(t) in Fig. 1.

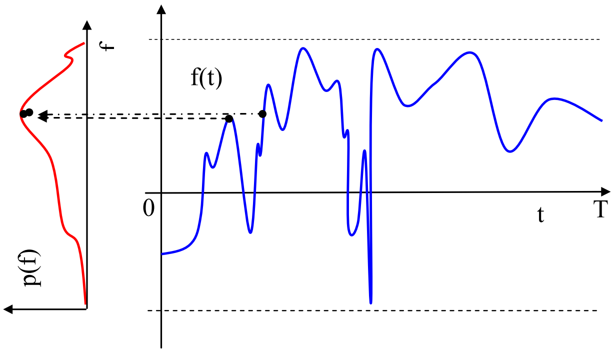

Fig. 1 *The probability density function of a time series*

Each function value f(t) of the time series is represented in the histogram (*pdf*) on the left hand side. Similar values, even if they occur at very different time instances, are neighbours here. This neighbourhood is reflected by the ranking index field: two neighboured samples y_i, y_j have also a small distance in their ranking index

$$| \mathrm{index}(y_i) - \mathrm{index}(y_j) | < \delta \tag{14}$$

For instance, for the uniform distribution of the ranked output *y* we know that the *T* samples are transformed into the ranked time series with indices 0,1,2,…,T–1. Replacing the original samples by their index values and scaling them to 0.0, 1/(T–1), 2/(T–1), …, (T–1)/(T–1) = 1.0 in the interval [0.0,1.0] produces a uniformly distributed time series. Within this time series, the linearly changing index value also limits the maximal value difference of the samples. In our case, only all δ samples from index $t_1 = i+1$ to $t_2 = i+\delta$ within the index array fulfil relation (14), because

$$|y_i - y_j| < \varepsilon/2 = \varepsilon' \Leftrightarrow \left| \frac{i}{T-1} - \frac{j}{T-1} \right| < \varepsilon' \Leftrightarrow | i-j | < \varepsilon'(T-1) \equiv \delta$$

Therefore, it is sufficient to check only for those δ samples of time series *y* if the corresponding compound input samples **x**(k) also fulfil relation (9). If YES, the comparison is counted for C_{xy}. If NO, it is omitted.

By this limitation, we are able to lower the runtime complexity of the computation algorithm from $O(T^2)$ to $O(T\delta)$ which is much lower.

4.3 The missing value problem

There is one problem often found in real world data: the data are not complete. This might due to acquisition errors like broken or stuck sensors, or because the measurements are too expensive or not available, e.g. x-ray data of humans or laboratory data like tissue analysis which has not a high sampling frequency. For all these missing values, the comparison with the other time series samples at the same time point can not be done. This has some implications on the entropy computation:

- The uniform distribution can not be normalized by the number of all samples (length of the ranked array), but only by the number of valid ones.

- The number of possible comparisons which is necessary for computing the entropy in eq.(13) is reduced by the number of missing values. This has to be taken into the computation formula.

Let us investigate this more concretely.

4.3.1 Treating missing values

Let us assume that we have marked *k* missing values in the involved *n* time series, e.g. by assigning them the maximal or minimal possible input value. How should they be treated for the computation of the mutual information? We know that for a given time point *t*, if there is only one missing value in one of the *n* series, all other samples for point *t* can not be used, too. Therefore, the number of possible comparisons b(i,j) is restricted by the *k* missing values, not by dimension *n*.

For the main diagonal of matrix **b** = [b(i,j)] (see Fig. 2) we have only T−k valuable comparisons. Each missing value, denoted by a filled circle in the T×T matrix **b**, causes other comparisons to be invalid. Each missing value is involved in the *T* comparisons on the horizontal line and on the vertical line in Fig. 2, and therefore invalidates them. To be exact, including the missing value on the main diagonal the first missing value invalidates T+(T−1) = 2T−1 comparisons. The next one also invalidates T+(T−1) comparisons, but the two comparison at the crossings of the vertical and horizontal lines with those of the first missing value are counted double. So, we have only T+(T−3) = 2T−3 additional invalidations for the 2nd missing value.

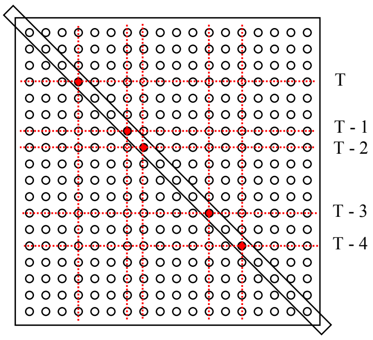

Fig. 2 The invalidations by missing values

This is also valid for the 3rd missing value which invalidates 2T−5 comparisons. In conclusion, we have

$$\begin{aligned} m &= (2T-1) + (2T-3) + \ldots + (2T-2k+1) \\ &= \sum_{i=1}^{k}(2T-2i+1) = k(2T+1) - 2\sum_{i=1}^{k} i \end{aligned} \qquad (15)$$

invalidations. With

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$$

we have

$$m = k(2T+1) - 2\frac{k(k+1)}{2} = 2kT+k-k^2-k = k(2T-k)$$

invalidations. Therefore, in eq. (12)

$$\langle P(\mathbf{x})\rangle = \frac{T+2C_x}{T^2}$$

the number T of main diagonal elements with b(i,i) = 1 becomes (T−k) for *k* missing values. Additionally, the total number of valid comparisons becomes T^2−m. Therefore, we get for the estimated average probability having *k* missing values

$$\langle P(\mathbf{x})\rangle = \frac{2C_x+T-k}{T^2-m} = \frac{2C_x+T-k}{T^2-k(2T-k)} \qquad (16)$$

4.3.2 Treating equal values

Another problem is the appearance of equal values within a time series. Sometimes, they are due to a broken sensor and should therefore be treated as missing values. In other cases, the accuracy of the sensors is limited to a restricted number of discrete values. In the latter case, we have to treat the features differently, especially if this occurs in the target time series. For equal values, the order of the ranked samples will be very arbitrary, depending on the sorting algorithm. For two different sorting algorithms the order in the ranking might be different, resulting in two different ranked and scaled time series and therefore in two different numerical values of the mutual information for the same feature data. There are several ideas how to treat this problem:

1. we might not do anything special, since the differences are small in the example
2. we always use only the same sorting algorithms
3. we change the input data by adding a small increment to all equal valued samples
4. we treat all equal valued samples as discrete states, compute the mutual information by the conventional symbolic approach of counting the states and integrate it into the I_2-estimation

The first two ideas are very arbitrary and not very appealing. The third one is also arbitrary, but provides the means for consistent results in different software environments. The fourth one is the best, but most complicated one: How should we integrate two different kinds of probability estimation into one schema?

In this paper, we use the third approach, leaving the fourth one for further research. Please note that the smallest possible increment of a floating point number x is not a constant, but depends on the value of x. It can be obtained by converting the normalized floating point number, e.g. in the 64 bit IEEE 754 format, into a bit string (long integer), increment it, and then convert it back again into a floating point number.

4.4 The hypercube size

In this paper, for each computation of the mutual information a hypercube size ε for each counting is used. What size should it have to give optimal performance? It should not be too small, giving no result, or too big, giving an imprecise result.

- There are complicated algorithms for determining a cube size depending on a performance criterion, for instance the recursive division of input space by Frazer [14]. However, computational efficient approaches taking the finite Epanechnikov kernel [12] show that the results depend heavily on the initial kernel width and on the binning algorithm. Here, we do not depend on a recursive binning algorithm because we can average the probability estimation before taking the logarithm which is much more stable. Therefore, we can use a fixed hypercube size for the whole feature space.

- With this idea, we can use a much simpler algorithm, an interval nesting, based on the target value of a certain percentage s of samples which should be contained in the hypercube. The interval nesting should choose a cube size ε such that the entropy $H_{xy}(\varepsilon)$ is equal to the entropy $H_s = -\log(s)$. The algorithm is described in appendix C.1.

- Interestingly, the mutual information computation changes also when adding additional features as precondition. How does an increase in the number of features change the resulting information? This is treated in appendix C.2. As result, the mutual information becomes smaller.

- The accuracy of the mutual information computation using hypercubes of size ε depends heavily on the number of examples (samples in a time series). How does the squared error, the variance, in the number of samples per hypercube depend on the size of the hypercube? This is discussed in appendix C.3 giving a formula where the error depends nonlinearly on the cube length ε.

4.5 Number representation

There are different approaches for computing the mutual information. Originally, the mutual information is a real valued number which means it is represented in a floating point format. Nevertheless, we have to rank the target time series according to theorem eq.(24) in order to have the minimum of mutual information at zero. The operation of ranking induces an order, expressed by an index which is an integer value. In general, subtracting and comparing two integers is faster than the same procedures for floating point numbers. Since the accuracy is approximate the same for using integers or floating point for the same numbers, the results should not differ much and are in favour for integer comparison. However, in the counting process for each comparison we have to convert either the integers to floating point representation or to convert the floating point numbers to integer, e.g. by ranking the array. In each case, we have a conversion overhead.

Additionally, in the two different cases we get different results. Why? Although the number values by the different representations are very close, the computation of mutual information by the adaptation procedure used in C.1 depends on the kind of numbers we use because the ε-values are different: the interval nesting stops when a ε is determined which produces an H_{xy} sufficiently close to H_s.

Since ε is determined in the middle of the interval $[\varepsilon_0,\varepsilon_1]$ as $\varepsilon = (\varepsilon_1-\varepsilon_0)/2$, the division by two returns different results depending on the fact using either integer ε or floating point ε. In the former case, rounding will occur and produces a different ε than in the latter case. In both cases, the resulting ε will fulfil the requirements, but the computed approximate H_{xy} will be different for both cases. There is no favour for each of both solutions, they all are correct in the approximate sense.

Now, what should we choose as number representation? Integer or floating point? Although the integer solution performs a bit faster even when the ranking pre-processing is included, we prefer the approach of not ranking too many time series, only the necessary one in order not to change any data unnecessarily. This implies the conversion of the index array of the ranked array *y* into a floating point representation, using only floating point numbers in each comparison and for ε.

5 A simple application example

Now, as introducing example we take a system which we regard in two ways: on one hand as a block box which has inputs and output and a input-output mapping function which we have to approximate, and on the other hand as a glass box system, i.e. a nonlinear system where we know everything about [24] and can compare the real mapping with the approximated one.
Let us assume that we have four inputs and one output. The system is depicted by the following figure.

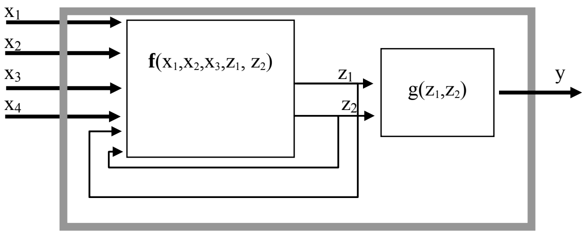

Fig. 3 *An example of a dynamical system*

From outside, we can only observe the time series of x_1, x_2, x_3, x_4 and the output y. The structure within the grey boarder is hidden. This structure might be described by the following equations of a simple, non-linear system

$$z_1(t+1) = z_1(t) - ½\, z_2(t) + 0.8x_2^2(t\text{-}10) + x_4^2(t\text{-}20)$$

$$z_2(t+1) = z_1(t)/(1+ 0.1z_2^2(t)) + 0.8x_2^2(t\text{-}10) + x_4^2(t\text{-}20)$$

and $y(t) = 1.8 \tanh(0.32z_1(t)) - 0.63$

We note that the system states and the output do not use inputs 1 and 3, but inputs 2 and 4 at different time delays 10 and 20.
As input we might take a random input which will cause the observed output in Fig. 4.

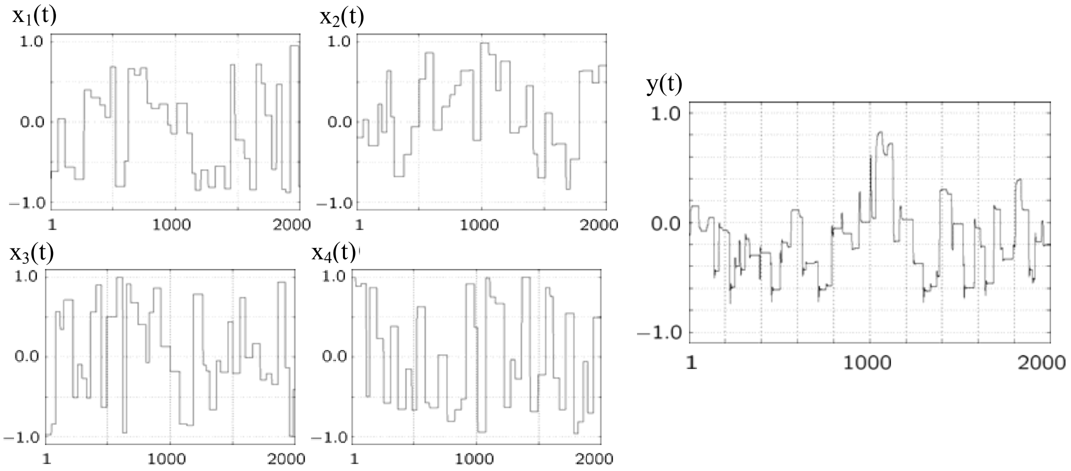

Fig. 4 The random input and resulting output signals

Now, the mutual information I_2 between all delayed input lines x_i(t-r) and the desired output y(t) is shown in Fig. 5

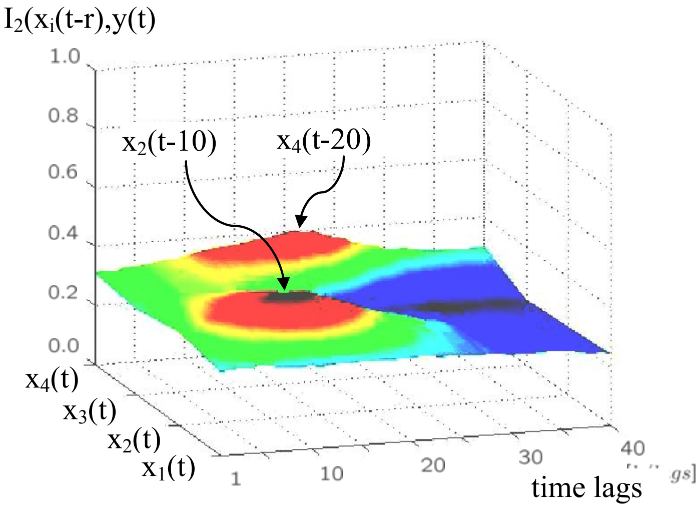

Fig. 5 The mutual information between the delayed input and the output

We see that, although the output is a nonlinear function of the interdependent states which are themselves non-linear functions of the inputs, both facts are reflected by the mutual information [14]: The mutual information between the not used input lines and the output is significant lower, and the mutual information between the involved input and the output has its peaks at the two time lags r = 10 and r = 20. Thus, in our small example the mutual information feature selection works well and is even valid for time delayed features. Let us now regard a more realistic example: the approximation of a chemical process output.

6 Approximating an industrial process output

The feature selection method introduced so far is now used to approximate the time series of a chemical process. The approximation benchmark was introduced as a competition within the European NISIS network at 2006. It consists of a data base of 5867 samples of 14 input lines and one output ("catalyst activity") for training and four periods for prediction and retraining, see Fig. 6.

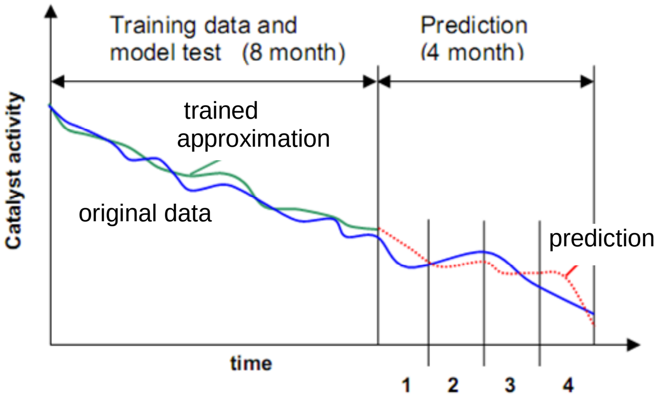

Fig. 6 The training period and the test/retraining intervals

The input/output data of each period was provided only after a prognosis was given for that interval. The prognosis was based on all data available before that period. The resulting error between the prognosis and the real data was accumulated. After the end of the contest, the best solution was marked.
Now let us regard the task a bit closer. The chemical process can be described as follows.

6.1 The process

The chemical process to be modelled consists of a reactor containing some 1000 tubes filled with catalyst, used to oxidize a gaseous feed (ethane is taken as example). It is cooled with a coolant supposed to be at constant temperature. The description of the reaction speed is taken from literature and depends strongly non-linearly from temperature. Its exothermal reaction is counteracted by the cooling and leads to a temperature maximum somewhere along the length of the tube. As the catalyst decays, this becomes less pronounced and moves further downstream. The catalyst activity usually decays within some time to zero, a year is taken as example here. The process to be modelled takes input from other, larger processes, so that the feed will vary over the days. The operating personal reacts to this by choosing appropriate operating conditions. The catalyst decay is however much slower than these effects. The process is equipped with measurements to log all the variations of the feed and the operating conditions. In addition, there are measurements showing some concentrations, flows and a lot of temperatures along the length of a characteristic tube to identify the processes state.

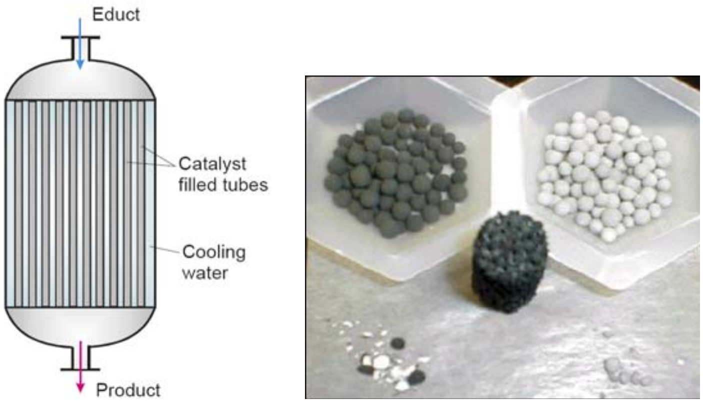

Fig. 7 The reactor with catalyst tubes and its content

All measurable influences are considered as input variables for a mathematical multi-input-single-output-model describing relevant process variables (model outputs) representative for chemical processes:

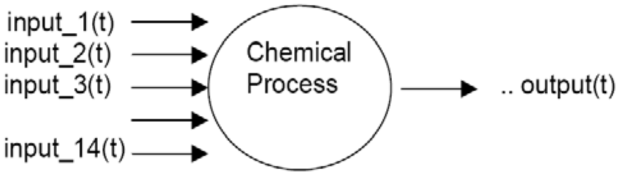

Fig. 8 The input-output modelling situation

16 columns, where the content of the columns is as follows:

column1: time in hours (1/24) since last catalyst change
column2-15: input data at this time
column16: output data at this time.

Input:

- Measured flow of air, kg/hr
- Measured flow of combustible gas, kg/hr
- Measured concentration of combustible component in combustible gas feed in mass fraction
- Total feed temperature, Cooling temperature
- Temperature at length 1/20, 2/20, 4/20, 7/20, 11/20, 16/20, 20/20 of reactor length in Celsius
- Product concentration of oxygen in mass fraction
- Product concentration of combustible component in mass fraction

Output: Catalyst activity. All time series have 5807 samples.

6.2 Selecting relevant inputs

First, the input feature selection process can be applied to the 14 input lines of the process example described above. In a first attempt, we just set up a ranking according to the mutual information I_2 between the last 3800 samples of the output time series (s=2%) and the corresponding input feature time series. This gives us Table 1 .

Table 1 Ranking of inputs on I_2 basis

rank	Name of var	I_2
1	QI X ORG	0,6138
2	QI EX C*	0,4829
3	FI ORG	0,3994
4	TI ROR 7	0,3296
5	TI ROR11	0,2504
6	QI EX O2	0,2250
7	TI ROR16	0,2154
8	TI ROR 4	0,2074
9	TCOOL	0,1801
10	TI ROR20	0,1763
11	FI AIR	0,1674
12	TI FEED	0,1399
13	TI ROR 1	0,1311
14	TI ROR 2	0,1266

The ranking shows a peculiarity: the input “QI X ORG” is nearly constant due to a sensor failure. Since the activity of the catalyst degrades slowly, a constant value seems to have some information in common with it. This is also partially true for other input values which can be labelled as “missing values”. Therefore, we weight the input sources by their amount of missing values and select only the most reliable ones. This eliminates the input lines “QI X ORG” and “QI EX C*”.

Additionally, for each time series the normalized mutual information with the output time series is computed. As we discussed in section 2, this takes not the mutual dependencies into account. Therefore, we revise the list: we apply the forward selection procedure described in section 2 and rank the inputs according to their conditional normalized mutual information. The results are shown in Table 2.

Table 2 Ranking of inputs on conditional I_2 basis

rank	Name of var	I_2 incr	I_2
1	FI ORG	0,3994	0,3994
2	TI ROR 7	0,2118	0,6111
3	TI ROR11	0,0828	0,6939
4	FI AIR	0,0602	0,7541
5	TI ROR16	0,0562	0,8103
6	TI ROR 4	0,0142	0,8245
7	TCOOL	0,0127	0,8373
8	QI EX O2	0,0043	0,8415
9	TI ROR20	0,0044	0,8459
10	TI ROR 2	-0,0179	0,8280
11	TI ROR 1	-0,0297	0,7984
12	TI FEED	-0,0136	0,7848

We also remark that for high dimensional cells (10, 11 and 12 variables) which have to be enlarged in order to hold still 2% of the samples, the I_2 computation precision decreases and gives lower absolute I_2 values leading to negative I_2 differences: Those inputs may also be cancelled.

Now we pose the question: Does the activity depend on input values of the past? Are there time delays in the system which makes the output in a time step depend on input values of prior time steps? Like in our synthetic example of Fig. 5, we test this by introducing as new inputs the old inputs delayed by different time delays, from 0 to 2000 samples in steps of 100 samples. This gives us the ranking in Table 3.

Table 3 Ranking of inputs on conditional I_2 basis including their delayed versions

rank	Name of var	delay	I_2 incr	I_2
1	FI ORG	700	0,5075	0,5075
2	FI ORG	1700	0,2951	0,8026
3	FI ORG	1000	0,0933	0,8959
4	FI ORG	100	0,0622	0,9581
5	FI ORG	800	0,0100	0,9681
6	FI ORG	1300	0,0046	0,9727
7	FI ORG	0	0,0145	0,9871
8	FI ORG	400	0,0051	0,9923
9	FI ORG	1200	0,0003	0,9926
10	FI ORG	0	0,0000	0,9926
11	FI ORG	0	0,0000	0,9926
12	FI ORG	0	0,0000	0,9926

We remark a stupefying result: It suffices to take the delayed versions of just one input for completely determining the output. The last three variables in the list reflect the fact that no other variable gives a contribution which increases the I_2 any more; the best variable is one which contributes zero.

Here, the I_2 depends not only on the variable but also on its delay; for each possible delay we obtain one I_2 value. The results for different input variables may be plotted as a function of the delay. This is shown in Fig. 9.

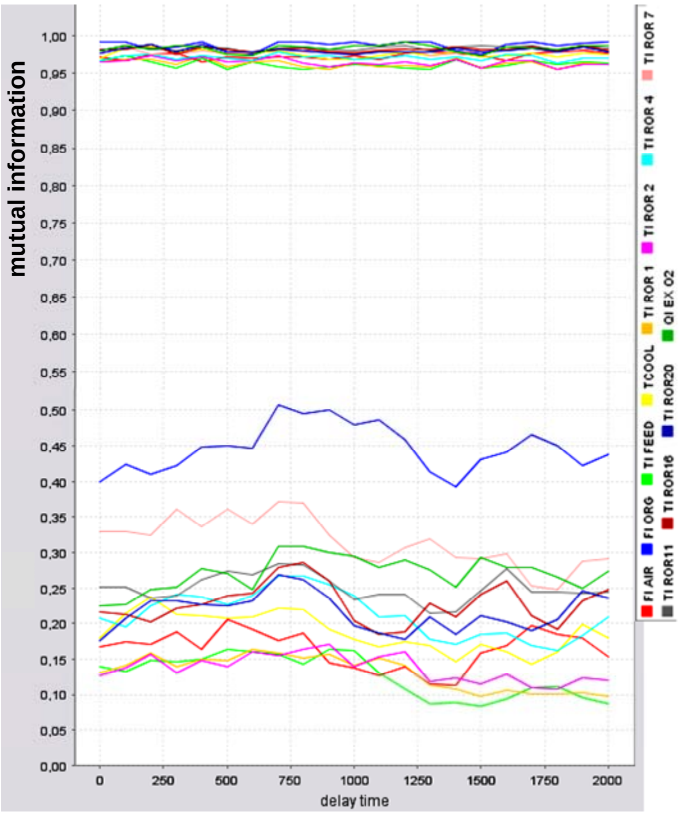

Fig. 9 Different I_2 values for variables as functions of the delays

The I_2 increase by different contributing variables is shown for two states: in the lower part the first approximation is shown, corresponding to Table 1; in the upper part the same variables for the 11th approximation are drawn. It is obvious that the variance is reduced due to the small possible enhancements.

The whole ranking of the forward selection process of section 2 can be visualized in Fig. 10. The contributions are shown as bars while the resulting I_2 is shown as function plot. The contributing variable names are shown under the bars.

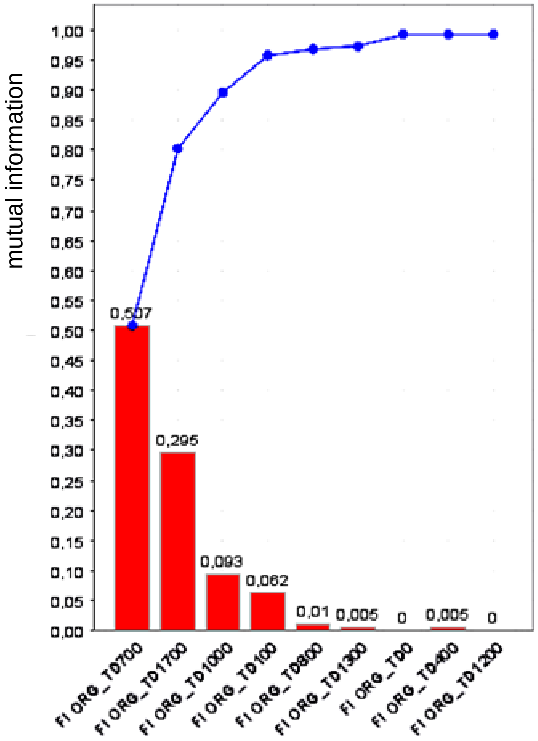

Fig. 10 Approximating maximal I_2 by input feature selection

6.3 Approximation by neural networks

The contribution of the best four variables to a neural network approximation is shown in Fig. 11. Here, different approximation states for the training samples are shown. Starting with a rough approximation by one feature, the approximation becomes smoother and smoother as we add further features.

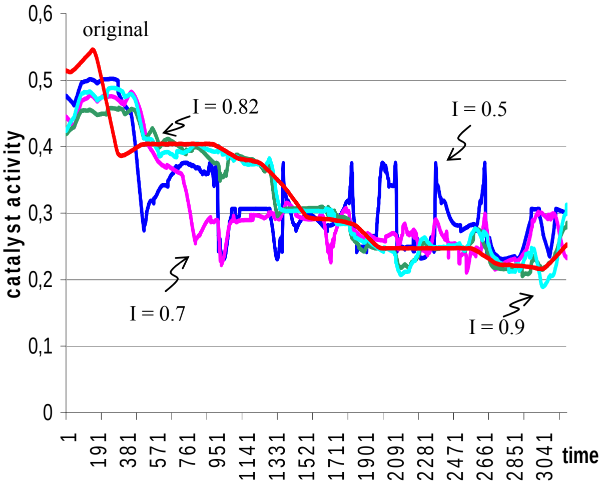

Fig. 11 Output approximation by input selection of maximal I_2

The relative error E_A of the approximation is defined within the contest by

$$E_A = \sum_{j=1}^{4} \frac{100}{N} \sum_{i=1}^{N} \frac{|y_i - L_i|}{|y_i|} \tag{17}$$

as sum of the 4 periods containing N=15 selected samples each.

After selecting the best features we trained two kind of neural networks: A standard two-layer RBF network and a standard two-layer perceptron network with backpropagation learning.

6.3.1 RBF approximation

Adaptive approximations by growing RBF networks are hindered by the RBF proportion of local reference. RBF networks are good in interpolation, whereas our problem has non-stationary input and needs extrapolation. This is true for different architectures as growing cell structures and neural gas algorithms.

For this reason we chose a fixed RBF architecture where the centres of the RBF neurons are given and fixed within a regular grid of the input space and overlap sufficiently. The only layer to adapt is the linear second layer with its weights. In Fig. 11 the coverage of the input space is shown for two inputs by 9 RBF neurons with a Gaussian activation function. The input values are filtered (smoothed) and start on the left lower end in the figure. The training period stops with the values shown on the right upper end.

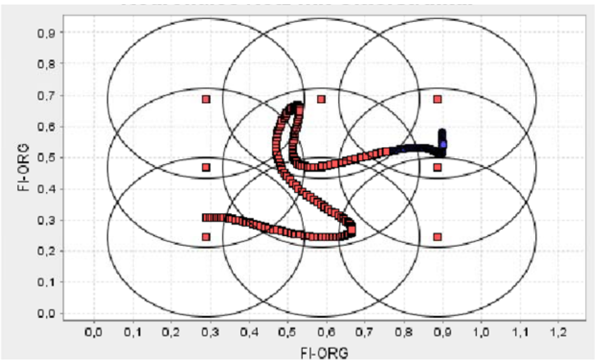

Fig. 12 Input space coverage

Different experiments showed that the number of neurons for the RBF network should not be too high: For too many neurons the influence of neighboured learning for neurons in the prognosis period is too small; they do not profit of the experience of the others. Here, a mix of multiple grid structures with different widths does not help. As optimal number of neurons we got 9 neurons for 2 inputs.

6.3.2 Multilayer Perceptron Backpropagation approximation

The input-output mapping of non-stationary input my also be approximated by the sigmoidal activation functions of a standard multilayer backpropagation network. Here, the training of one neuron affects all the output and we have no local input space sensitivity. The best results were achieved with 50 neurons and four inputs. The number of neurons does not affect much the prediction error.

6.3.3 Approximation results

The original contest had a variety of system architectures. The best architecture was a set of multilayer-backpropagation networks learning by genetic algorithms. Each network was characterized by a parameter set and adapted using genetic operations.

In the following table, the errors of the best three approximations of the competition (cont1, cont2, cont3) are shown. Additionally on the last lines, we show the results of our best RBF and Multilayer Perceptron approximation.

Table 4 Prediction error of the best competition networks

#	Test 1	Test 2	Test 3	Test 4	**Sum**
cont 1	21.01	12.87	19.14	20.13	**73.06**
cont 2	43.41	18.38	52.31	24.14	**138.26**
cont 3	63.15	17.83	33.89	28.91	**143.79**
RBF	1.76	6.10	5.57	15.47	**28.90**
ML	3.12	16.86	17.59	18.54	**56.11**

We see that the two standard approaches using only the selected features of maximal mutual information give much better predictions than all other specialized approaches.

7 Selecting real time features for combustion control in automotive environments

The necessity for feature selection is especially appreciated in real time applications. The following chapter gives an example for real time control of motor combustion.

7.1 Technical background

Today combustion motor management systems rely on numerous sensor signals to accomplish their tasks. From the viewpoint of combustion control, the so called 50%-energy conversion point (50%-ECP) is of particular interest. However, this variable cannot be measured directly. It has to be determined from a measurable surrogate quantity which implicitly characterizes the combustion quality. Since the in-cylinder pressure describes the course of the entire combustion cycle, this measurable quantity is used to determine the desired 50%-ECP.

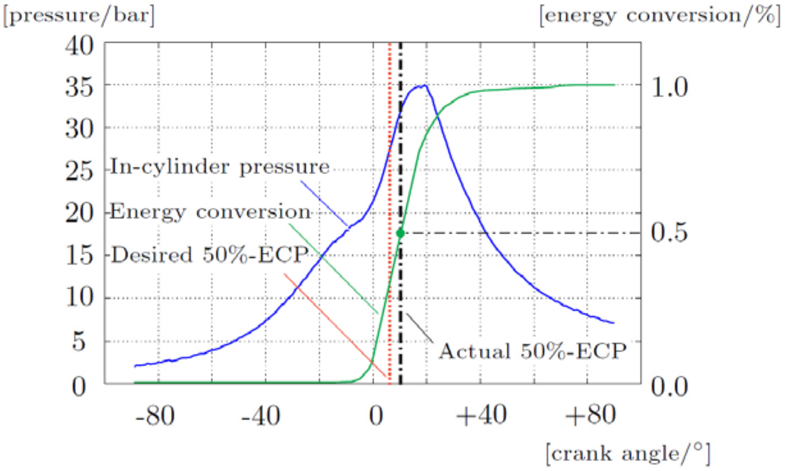

Fig. 13 In-cylinder pressure, the 50%-ECP and the course of the energy conversion during one combustion cycle

Fig. 13 displays the in-cylinder pressure curve and the according energy conversion as a function of the crank angle position during one combustion cycle. The 50%-energy conversion point is defined as the crank angle position at which 50% of the chemically stored energy has been released. In case of an optimal combustion with maximum efficiency, the position of the 50%-ECP has to be at a crank angle position of approximately +8° (±2°) after the top-dead-center position (TDC) of the piston [15]. The desired crank angle is shown by a thin dotted line, while the actual 50%-ECP is shown by a bold dotted line. In order to keep the 50%-ECP at the desired crank angle position, we employ a closed loop PI controller on top of a neural combustion analysis shown in Fig. 14.

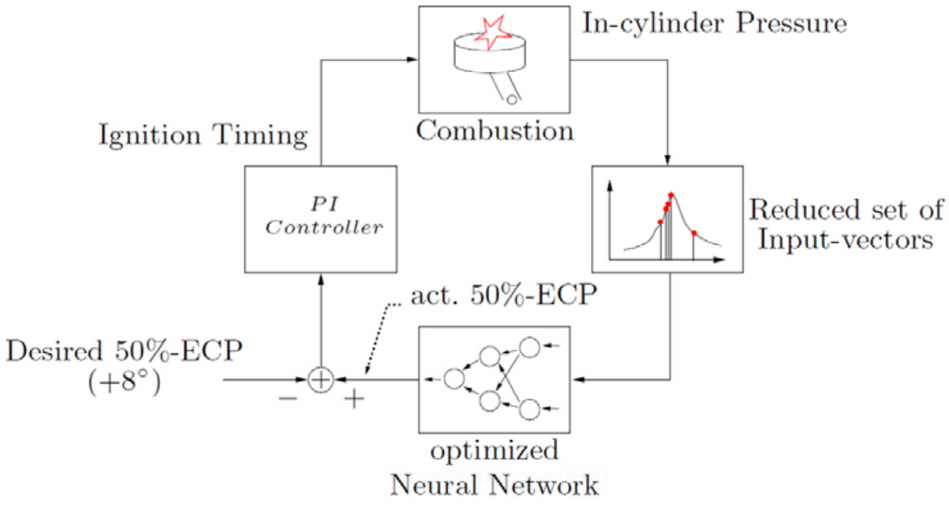

Fig. 14 Basic structure of a closed loop neural combustion control

The figure depicts the basic structure of such a closed loop combustion control. The desired crank angle position depends on the actual operating condition of the engine. The in-cylinder pressure is constantly measured during the course of combustion and the crank angle position of the 50%-ECP is determined from the obtained pressure curve through a neural network. The deviation of the current 50%-ECP from the desired value is fed into the PI controller, which then adjusts the ignition timing for the next combustion cycle. The 50%-ECP might be obtained with a high degree of accuracy from the measured pressure curve through a thermodynamic computation. Since this calculation is far too complex to be accomplished in real time, a neural network has been employed as an alternative. Once the training has been successfully completed, the neural network can be regarded as a so called virtual sensor for the 50%-ECP. Finally, it is integrated into the control structure shown in Fig. 14.

7.2 Selection of the relevant input variables

Since a neural network should be employed to model the nonlinear mapping between the samples of the in-cylinder pressure and the 50%-ECP, the most relevant pressure samples have to be selected, see the stage at the right hand side of Fig. 14. In order to construct an optimized data set for neural network training, the input variables with the highest information content are iteratively selected from a 180-dimensional data set with the forward-selection strategy [16], [17]. How is this obtained? Here again, mutual information can be used.

In Fig. 15, the so called I_2 function for the selection of the first feature can be observed.

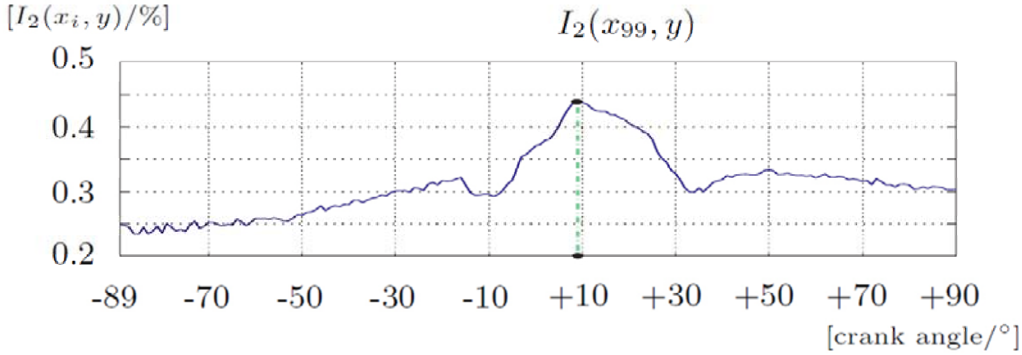

Fig. 15 The I_2 function reaches its maximum at the crank angle position with the highest information content.

This particular I_2 function depicts the values of the mutual information of order two (I_2) against the crank angle position. The position with the highest I_2 value is interpreted as the most relevant sample point of the pressure curve for determining the 50\%-ECP. In this case, the first sample point to be chosen would be at 9° relative to the top dead center position of the piston.

As mentioned earlier in this report, the correlation of coefficients is not capable for the identification of nonlinear dependencies. Since thermodynamic processes are known to be highly nonlinear, the comparison between the mutual information and the coefficient of correlation might be of particular interest for our problem context.

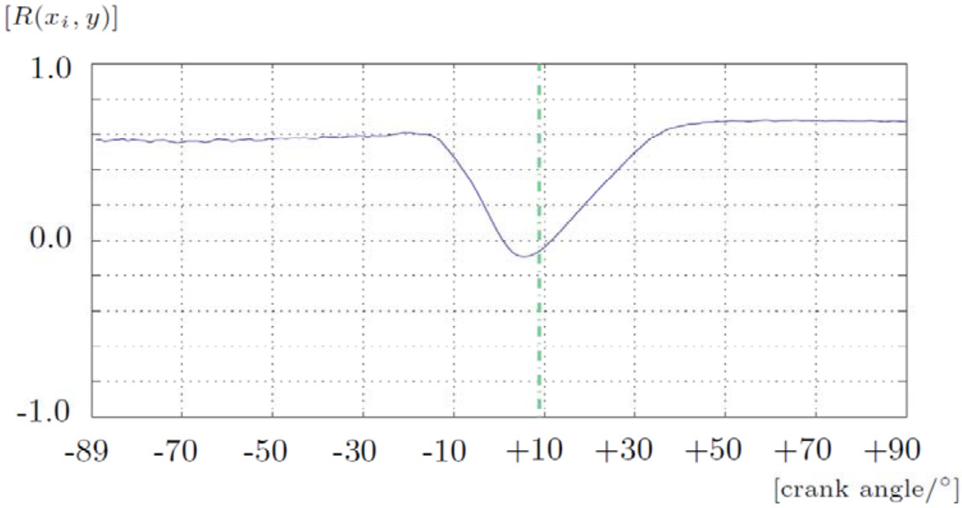

Fig. 16 Spearman's rank correlation coefficients adopt values around zero for crank angle position where the I2 reaches its maximum.

In Fig. 16, the Spearman rank correlation coefficients [18] are depicted. It can be clearly observed that the correlation coefficients adopt values around zero whereas the I_2 function in Fig. 15 reaches its maximum. If we would choose the correlation coefficients to tell us where the interesting sample points are, we could not conclude anything in the range -5° to +15°. Hence, we would have a blind spot in the so called high-pressure phase of the combustion. Since the high-pressure phase is definitely the most interesting part of the combustion, employing the linear methods does clearly not contribute to the solution of our selection problem.
Although the I_2 function in Fig. 15 might us remind of the coefficient of correlation, it differs from the latter in multiple points:

- The I_2 is defined also for multivariate time series, while the coefficient of correlation is not.
- All linear- and nonlinear dependencies between the possible multivariate sequences are captured by the I_2. The conventional coefficient of correlation is only sensitive to linear dependencies.
- If the I_2 value tends to zero, it can be concluded that there are no statistical dependencies between the investigated time series. From a coefficient of correlation of zero no conclusion can be drawn at all for arbitrarily distributed time series.

Continuing with the forward selection strategy delivers an ordered set of crank angle positions, specifying the positions with the highest information content for the determination of the 50%-ECP.

Fig. 17 illustrates the I_2 functions of successive iteration steps which were used to identify the most relevant sample points of the in-cylinder pressure curve. The described procedure is carried out while the information gain is significant. If additional input variables will not yield further information about the output variable, the iteration process is stopped. It can be observed that the selected points are not equally distributed, like one might assume naively. However, from the viewpoint of information theory, the identified set of crank angle positions is indeed optimal.

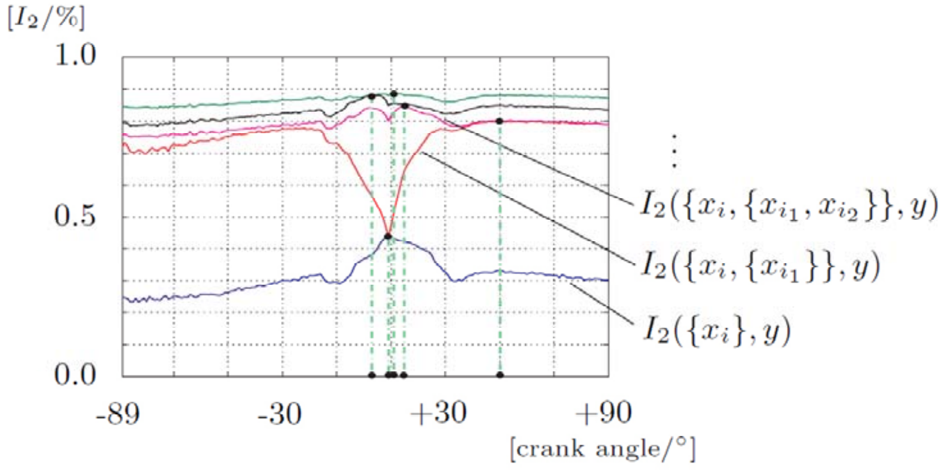

Fig. 17 Sequence of I_2 functions for successive iteration of the forward selection process.

Employing the concept of mutual information resulted in a significant reduction of the input dimension, the dimensionality of the input vector could be decreased from 180 down to only five sample-points. Finally, the identified crank angle positions are further employed for the construction of a neural training set.

7.3 Results

In the previous section, the most relevant input variables of the in-cylinder pressure has been selected. In order to implement an optimized closed loop combustion control, this process has to be identified in advance. For this purpose, we employed a feed forward neural network, which was trained by the Extended Kalman Filter rules.

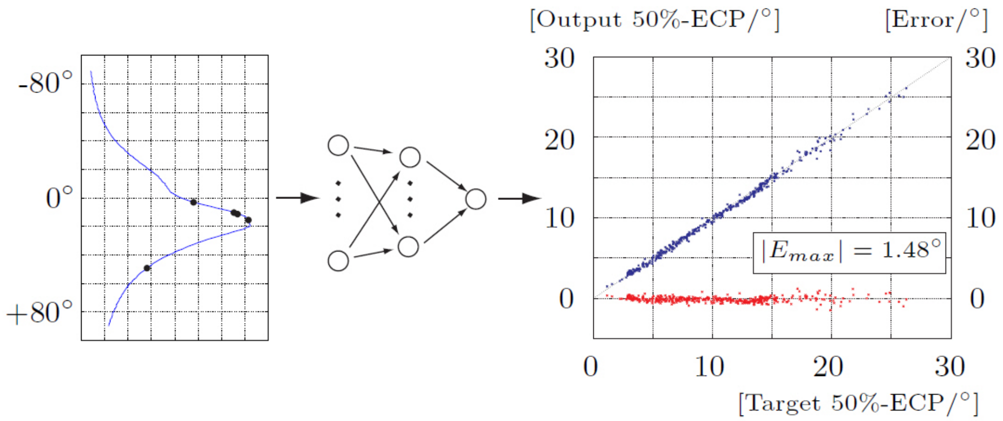

Fig. 18 Determination of the 50%-ECP from the I_2 selected points of the in-cylinder pressure curve with a Kalman Filter trained neural network.

In Fig. 18, the performance of the derived neural network is assessed with a test data set of 400 input patterns. The maximum absolute error turned out to be 1.48° for this test set. This is below the predefined error tolerance of ±2°, which is the requirement for the implementation of a closed loop combustion control. Hence, this neural network can be employed as a so called *virtual sensor* for the determination of the 50%-ECP, as depicted in Fig. 13.

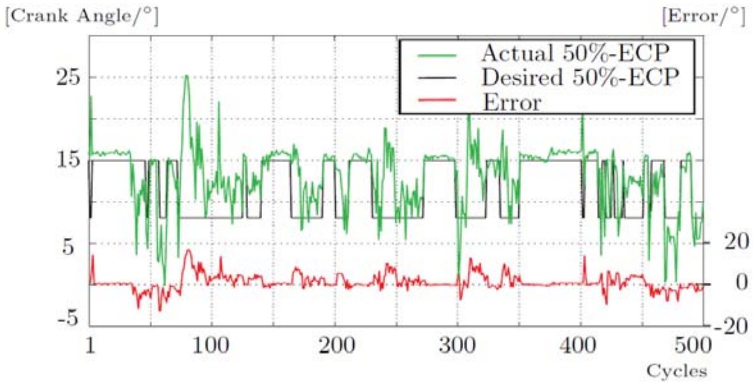

Fig. 19 The 50%-ECPs of cylinder 3 over 500 cycles with conventional open-loop combustion control

In Fig. 19, the actual and the desired 50%-ECPs are plotted over 500 combustion cycles. In this case, the ignition angle is determined from a conventional ignition map inside the engine control unit as a function of engine speed and load. The deviation of the actual and the desired value of the 50%-ECP varies between -11° and +18°.

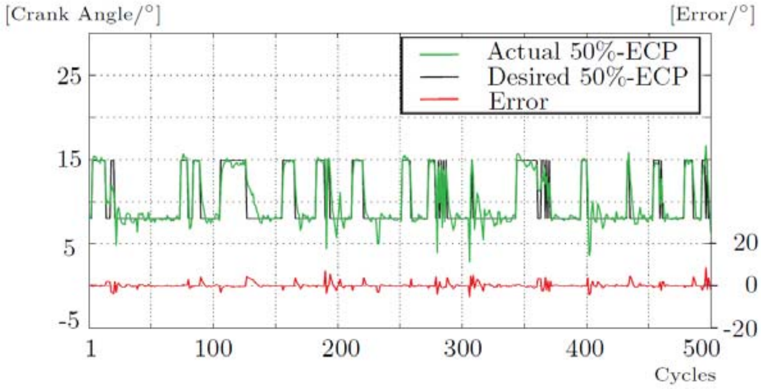

Fig. 20 The 50%-ECPs of cylinder 3 over 500 cycles with optimized neural combustion control

In contrast to this, Fig. 20 depicts the actual- and the desired 50%-ECPs of 500 cycles while the combustion control is active. Compared to Figure 7, the 50%-ECPs follow the desired value quite accurately with a significantly smaller deviation between -3° and +8°.

8 Conclusion

In this contribution we have shown how the selection of real valued features for mutual information can be approximated using only a few samples. This enables us to drop all irritating, irrelevant features in the process of approximating an input-output function, even if we do not know this function explicitly. Thus, the approximation process becomes much more efficient: the convergence is accelerated and the resulting error is decreased. It was shown that even the influence on the input-output mapping of time delays of the time series can be included in the feature selection process.

For the implementation of the density estimation process, the approach using the Rènyi information measure was introduced and its limitation discussed. A theorem is proved which guarantees that the minimal mutual information stays positive as long as we use at least one uniformly distributed random variable. Several algorithmic acceleration procedures were proposed and the influence of “missing values” were discussed and included in the computation scheme.

Three examples show the usefulness of feature selection for real world applications. First, a simple example of a nonlinear input-output mapping using time delays show how the relevant inputs and their time delays are determined. Second, a chemical process was approximated using delayed versions of only one input variable out of 14. The results are better than that of the best solution in an international contest. Third, the use of feature selection in real time process control is demonstrated for the example of automotive combustion control selecting only 5 samples out of 180. The results show the advanced control featuring a better control by fewer demands for hardware speed.

In summary, real valued feature selection shows to be a promising tool for facilitating the building of all approximation and diagnostic tools.

All program code can be obtained at the web site of the working group [25].

Acknowledgements

We thank Sven Förster for performing the simulations on RBF and Multilayer network approximation of section 6.3.

References

[1] Dash M., Liu H.: "*Feature Selection for Classification*". Intelligent Data Analysis - An International Journal, Elsevier, Vol. 1, No. 3, pages 131 - 156, 1997

[2] Saltelli A., Ratto M., Tarantola S. and Campolongo F.: *Sensitivity Analysis for Chemical Models*, Chemical Reviews, 105(7) pp 2811 – 2828. (2005)

[3] Ben-Bassat M.: *Pattern recognition and reduction of dimensionality*. in P.R.Krishnaiah, L.N.Kanal (eds.), Handbook odf Statistics, pp. 773-791. North Holland, 1982

[4] Theodoridis S., Koutroumbas K.: *Pattern Recognition*, 2nd ed., Elsevier Academic Press, London 2003

[5] Pudil P., Novovicova J., Kittler J.: *Floating search methods in feature selection*, Pattern Recognition Letters, Vol.15, pp. 1119-1125, 1994

[6] Yu B.,Yuan B.: *A more efficient branch and bound algorithm for feature selection*, Pattern Recognition Vol 26 (6), pp. 883-889, 1993

[7] P. Somol, P. Pudil, F.J. Ferri and J. Kittler. *Fast branch and bound algorithm in feature selection*. Proc. of SCI/ISAS 2000. Volume VII, (B. Sanchez, J.M. Pineda, J. Wolfmann, Z. Bellahsene, y F.J. Ferri, eds.), 646-651, 2000

[8] Cover T, Thomas J: *Elements of Information Theory*. John Wiley& Sons, New York 1991

[9] Shannon CE, Weaver W: The *mathematical theory of information*. University of Illinois Press, Urbana, 1949

[10] Battiti R.: *Using mutual information for selecting features in supervised neural net learning*, IEEE Transactions on Neural Networks, vol.5, pp.537-550 (1994)

[11] Parzen E.: *On the estimation of probability density function and the mode*. The Annals of Mathematical Statistics, 33, 1065. (1962).

[12] Principe J., Fisher III J., Xu, D.: *Information theoretic learning*. In S. Haykin (Ed.), Unsupervised adaptive filtering. Wiley, New York, NY (2000).

[13] Torkkola K., Campell W.: *Mutual Information in Learning Feature Transformations*, Proc. of the 17th Int.Conf.on Machine Learning, Morgan Kaufmann, pp.1015-1022, 2000

[14] Rényi A.: Probability Theory, Noth-Holland, Amsterdam 1970.

[15] Bargende, M.: *Schwerpunkt-Kriterium und automatische Klingelerkennung*, MTZ Motortechnische Zeitschrift, 56 (10): 623:638, 1995

[16] Heister F., Schock G.: Nonlinear, statistical Data-Analysis for the optimal Construction of Neural-Network Inputs with the Concept of a Mutual Information, In: ESANN'2000 proceedings- European Symposium on Artificial Neural Networks, Bruges(Begium), pages 439–444, 26-28 April 2000. 2000

[17] Heister F., Froehlich M.: *Non-linear time series analysis of combustion pressure data for neural network training with the concept of mutual information*, Journal of automobile engineering, 215(D2):299–304, 2001 2001

[18] Spearman, C., The Proof and Measurement of Association Between Two Things, American Journal of Psychology, 15:72–101 1904

[19] Takens, F.: *Invariants related to dimension and entropy*. In: Atas do 13. Coloquio Brasileiro de Mathematica, Rio de Janeiro 1983

[20] Grassberger P., Procaccia I.: *Estimation of the Kolmogorov entropy from a chaotic signal.* Phys. Rev. A 28, 2591-2593 (1983)

[21] Pompe B, Heilfort M: *On the Concept of the Generalized Mutual Information Function and Efficient Algorithms for Calculating It*, unpublished (1995). Available at http://www.physik.uni-greifswald.de/~pompe

[22] Fraser A. M. and Swinney H. L.: *Independent coordinates for strange attractors from mutual information*, Phys. Rev. A **33**, 1134 (1986)

[23] Bonnlander B., Weigend A.: *Selecting Input Variables Using Mutual Information and Nonparametric Density Estimation*, Proc.Int. Symp. on Artificial Neural Networks ISANN'94, Taiwan, pp.42-50 (1996)

[24] Heister F.: *Nonlinear feature selection using the general mutual information*, Johann Wolfgang Goethe-Universität, Frankfurt am Main, FB Informatik und Mathematik, Institut für Informatik 2008, available at http://publikationen.ub.uni-frankfurt.de/volltexte/2008/5462/

[25] *Mutual Information Software*: http://www.informatik.uni-frankfurt.de/asa/software/MI2/

[26] Pompe B.: *Measuring statistical dependencies in a time series*. J Stat. Phys., 73:587–610, 1993.

Appendix A The example dataset

The example presented here is a probability-corrected version of the simplified CorrAL dataset of [1]. It consists of a binary feature set $M_6 = \{A0, A1, B0, B1, C, D\}$ of $n = 6$ features where binary target Y is determined by the subset of four features {A0,A1,B0,B1} by $Y = (A0 \wedge A1) \vee (B0 \wedge B1)$. Feature C is assumed to correlate to Y in 75% of all cases whereas feature D is independent of Y. Assuming equal probabilities for features A0, A1, B0, B1, D of $P(1) = P(0) = 0.5$ the involved conditional probabilities and mutual information can be easily computed. The following table shows our dataset.

#	A0	A1	B0	B1	D	C	Y	#	A0	A1	B0	B1	D	C	Y
1	0	0	0	0	0	1	0	9	1	0	0	0	1	1	0
2	0	0	0	1	1	1	0	10	1	0	0	1	1	0	0
3	0	0	1	0	0	1	0	11	1	0	1	0	0	1	0
4	0	0	1	1	1	0	1	12	1	0	1	1	1	0	1
5	0	1	0	0	0	1	0	13	1	1	0	0	0	0	1
6	0	1	0	1	1	1	0	14	1	1	0	1	0	1	1
7	0	1	1	0	1	0	0	15	1	1	1	0	1	0	1
8	0	1	1	1	0	1	1	16	1	1	1	1	0	0	1

We have

$H(Y) = P(0)\cdot ld(P(0)^{-1}) + P(1)\cdot ld(P(1)^{-1}) = –9/16\cdot ld(9/16) – 7/16 \cdot ld(7/16) = 0.4669 + 0.52178$
$= \mathbf{0.9887}$

With

$P(Y=0 \wedge D=0) = 4/16 \approx 4.5/16 = 9/16 \cdot 0.5 = P(Y=0)\cdot P(D=0)$
$P(Y=0 \wedge D=1) = 5/16 \approx 4.5/16 = 9/16 \cdot 0.5 = P(Y=0)\cdot P(D=1)$
$\Rightarrow P(Y=0 \wedge D=0)+ P(Y=0 \wedge D=1) = P(Y=0)= 9/16$
$P(Y=1 \wedge D=0) = 4/16 \approx 3.5/16 = 7/16 \cdot 0.5 = P(Y=1)\cdot P(D=0)$
$P(Y=1 \wedge D=1) = 3/16 \approx 3.5/16 = 7/16 \cdot 0.5 = P(Y=1)\cdot P(D=0)$
$\Rightarrow P(Y=1 \wedge D=0)+ P(Y=1 \wedge D=1) = P(Y=1)= 7/16$

we see that the two features Y and D are independent.
For feature **D**, the corresponding mutual information is $I(Y;D) = H(Y) – H(Y|D)$ with
$H(Y|D) = P(D=0)H(Y|D=0) + P(D=1)H(Y|D=1)$ with
$H(Y|D=0) = P(Y=0|D=0)\cdot ld(.) + P(Y=1|D=0)\cdot ld(.) = –4/8 \cdot ld(4/8) –4/8\cdot ld(4/8) = 1.0$
$H(Y|D=1) = P(Y=0|D=1)\cdot ld(.) + P(Y=1|D=1)\cdot ld(.) = –5/8 \cdot ld(5/8) –3/8\cdot ld(3/8) = 0.95443$
$\Rightarrow \mathbf{I(Y;D)} = H(Y) – H(Y|D) = 0.9887 – 0.9772 = \mathbf{0.01148}$

For feature **C**, we see that in 12 of 16 cases (75%) we have $Y = \neg C$, i.e. a highly correlated feature. The mutual information is $I(Y;C) = H(Y) – H(Y|C)$ with
$H(Y|C) = P(C=0)H(Y|C=0) + P(C=1)H(Y|C=1)$ with $P(C=0) = P(C=1) = 0.5$ and
$H(Y|C=0) = P(Y=0|C=0)\cdot ld(.) + P(Y=1|C=0)\cdot ld(.) = –2/8 \cdot ld(2/8) –6/8\cdot ld(6/8) = 0.811278$
$H(Y|C=1) = P(Y=0|C=1)\cdot ld(.) + P(Y=1|C=1)\cdot ld(.) = –7/8 \cdot ld(7/8) –1/8\cdot ld(1/8) = 0.543564$
$\Rightarrow \mathbf{I(Y;C)} = H(Y) – H(Y|C) = 0.9887 – 0.67742 = \mathbf{0.3113}$

For feature **A0** we have $I(Y;A0) = H(Y) – H(Y|A0)$.
$H(A0) = P(A0=0)\cdot ld(P(A0=0)^{-1}) + P(A0=1)\cdot ld(P(A0=1)^{-1}) = 0.5\cdot ld(2/1) + 0.5\cdot ld(2/1) = 1.0$
$= H(A1) = H(B0) = H(B1)$
$H(Y|A0) = P(A0=0)H(Y|A0=0) + P(A0=1)H(Y|A0=1)$ with
$H(Y|A0=0) = P(Y=0|A0=0)\cdot ld(.) + P(Y=1|A0=0)\cdot ld(.) = –6/8 \cdot ld(6/8) –3/8\cdot ld(3/8) = 0.84192$
$H(Y|A0=1) = P(Y=0|A0=1)\cdot ld(.) + P(Y=1|A0=1)\cdot ld(.) = –2/8 \cdot ld(2/8) –5/8\cdot ld(5/8) = 0.9238$
$\Rightarrow H(Y|A0) = 0.88286$

$\Rightarrow$ **I(Y;A0)** = H(Y) – H(Y|A0) = 0.9887 – 0.88286 = **0.10584**

Since the computation Y = (A0$\wedge$A1) $\vee$ (B0$\wedge$B1) is completely insensitive to a relabeling of the features, the mutual information of A1, B0 and B1 with Y are equal to that of A0:
I(Y;A1) = I(Y;B0) = I(Y;B1) = I(Y;A0).

The mutual information of the set M_4 = {A0, A1, B0, B1} is
I(Y;M_4) = H(Y) – H(Y|M_4) with

$$H(Y|M_4) = -\sum_{i=1}^{16} P(S_i) \sum_{j=0}^{1} P(Y_j | S_i) \mathrm{ld}(P(Y_j | S_i)$$

using the 16 states of the joint random variable S = (A0, A1, B0, B1) and $P(S_i)$ = 1/16 of each state. This becomes

$$H(Y|M_4) = -\frac{1}{16} \sum_{i=1}^{16} P(Y=0|S_i) \mathrm{ld}\ P(Y=0|S_i) + P(Y=1|S_i) \mathrm{ld}\ P(Y=1|S_i)$$

All states are different and determine the value of Y. Therefore, each term of the sum

$$P(Y=0|S_i)\cdot \mathrm{ld}\ P(Y=0|S_i) + P(Y=1|S_i)\cdot \mathrm{ld}\ P(Y=1|S_i)$$

becomes zero, either because P(.) = 1 and ld(P(.)) = 0, or because P(.) = 0 and therefore P(.)·ld(P(.)) also becomes zero when ld(.) $\rightarrow -\infty$. In summary, H(Y|M_4) = 0 and therefore
I(Y;M_4) = H(Y) – H(Y|M_4) = **H(Y) = 0.9887.**

In contrast to this, the mutual information of the set M_4 = {C, A0, A1, B0} is
I(Y;M_4) = H(Y) – H(Y|M_4). The computation gives nearly the same results as above; nearly all sums become zero except one. Here, the state S = (A0=0, A1=1, B0=1, C=0) occurs two times (P(S) = 1/8) with two different results Y=0 and Y=1 of equal occurrence and we get

$$H(Y|M_4) = -\ 1/8\ [P(Y=0|S)\cdot \mathrm{ld}\ P(Y=0|S) + P(Y=1|S)\cdot \mathrm{ld}\ P(Y=1|S)]$$

$$= -\ 1/8\ [0.5\cdot \mathrm{ld}\ 1/2 + 0.5\cdot \mathrm{ld}\ 1/2] = 1/8\ [\mathrm{ld}(2)\ -\mathrm{ld}(1)] = 1/8$$

I(Y;M_4) = H(Y) – H(Y|M_4) = 0.9887 – 0.125 = **0.8637**

Appendix B The proportions of mutual information of order 2

One of the severe drawbacks of the quadratic Rényi information measure gives a mutual information, which fulfil the condition

$$I_\alpha(X;Y) \geq 0 \quad \text{only for } \alpha = 0,1 \qquad (18)$$

for arbitrary distributions of *X* and *Y*, see [23]. This means that the mutual information also may become negative for $\alpha > 1$. Especially, if $I_\alpha(X;Y) = 0$ for $\alpha > 1$ we can not deduce that X and Y are independent

Example

Let us assume that we have two binary random variables X and Y which can take the values x_1, x_2 and y_1, y_2. Both should have the same distribution, i. e. $P(x_1) = p$, $P(x_2) = 1–p$ and $P(y_1) = p$, $P(y_2) = 1–p$.

The joint distribution of both should be for $1–1/\sqrt{2} \leq p \leq 1/\sqrt{2}$, see [26]:

$$\begin{aligned} P(x_1,y_1) &= -½ +2p - p^2 \\ P(x_1,y_2) &= ½ - p + p^2 \\ P(x_2,y_1) &= ½ - p + p^2 \\ P(x_2,y_2) &= ½ \qquad - p^2 \end{aligned} \qquad (19)$$

which gives us the correct marginal distributions

$$\begin{aligned} P(x_1) &= P(x_1,y_1) + P(x_1,y_2) = p \\ P(x_2) &= P(x_2,y_1) + P(x_2,y_2) = 1–p \\ P(y_1) &= P(x_1,y_1) + P(x_2,y_1) = p \\ P(y_2) &= P(x_1,y_2) + P(x_2,y_2) = 1–p \end{aligned} \qquad (20)$$

The two random variables are only independent, iff

$$P(x_i,y_j) = P(x_i)P(y_j) \qquad (21)$$

holds. For the first joint probability of eq. (19) this is the case iff

$$\begin{aligned} &P(x_1,y_1) = -½ +2p - p^2 = p^2 = P(x_1)P(y_1) \quad \Leftrightarrow \quad 2p^2 -2p + ½ = 0 \\ &\Leftrightarrow \quad p = ½ \end{aligned} \qquad (22)$$

The same goes for the other three joint probabilities.

Now, let us compute the mutual information $I_2(X;Y)$. By eqs.(1) and (5) we have

$$I_2(X;Y) = H_2(X)+ H_2(Y) - H_2(X,Y) \qquad (23)$$

$$= -\log (P(x_1)^2+P(x_2)^2) \quad -\log (P(y_1)^2+P(y_2)^2) + \log(P(x_1,y_1)^2+P(x_1,y_2)^2+P(x_2,y_1)^2+P(x_2,y_2)^2)$$

$$= -2\log (p^2 + (1–p)^2) +\log((–½ +2p –p^2)^2 + 2(½ –p +p^2)^2 + (½ –p^2)^2)$$

$$= -\log (p^2 + (1–p)^2)^2 +\log (4p^4–8p^3+8p^2–4p +1) = 0$$

Thus, in the whole range of $1–1/\sqrt{2} < p < 1/\sqrt{2}$ the mutual information becomes zero, not only for p =

½ . This means that for α=2 we can not deduce the independence of random variables from their zero mutual information.

How can we overcome this problem? This is obtained by the following theorem:

Theorem: For uniformly distributed random variable *Y* with $P_i = 1/T$ and random variable *X* of unknown distribution we have

$$I_\alpha(X;Y) \geq 0 \quad \text{for } \alpha = 2 \qquad (24)$$

Only iff the random variables *X* and *Y* become independent, this becomes zero.

Proof:

Assuming an uniformly distributed random variable *Y* with $P_i = 1/T$ we have with eq.(23)

$$I_2(X;Y) = H_2(X) + H_2(Y) - H_2(X,Y)$$

$$= -\log\sum_{i=1}^{T} P^2(x_i) - \log\sum_{i=1}^{T} P^2(y_i) + \log\sum_{i,j=1}^{T} P^2(x_i, y_j)$$

$$= -\log\sum_{i=1}^{T} P^2(x_i) - \log\sum_{i=1}^{T}\left(\frac{1}{T}\right)^2 + \log\sum_{i,j=1}^{T} P^2(x_i, y_j)$$

$$= \log\left(\frac{\sum_{i,j=1}^{T} P^2(x_i, y_j)}{\sum_{i=1}^{T} P^2(x_i)\sum_{i=1}^{T}\left(\frac{1}{T}\right)^2}\right) = \log\left(\frac{\sum_{i,j=1}^{T} P^2(x_i, y_j)}{\sum_{i=1}^{T} P^2(x_i)\frac{1}{T^2}\sum_{i=1}^{T} 1}\right) = \log\left(\frac{\sum_{i,j=1}^{T} P^2(x_i, y_j)}{\frac{1}{T}\sum_{i=1}^{T} P^2(x_i)}\right)$$

$$= \log\left(\frac{\frac{1}{T}\sum_{i=1}^{T} P^2(x_i) - \frac{1}{T}\sum_{i=1}^{T} P^2(x_i) + \sum_{i=1}^{T}\sum_{j=1}^{T} P^2(x_i, y_j)}{\frac{1}{T}\sum_{i=1}^{T} P^2(x_i)}\right)$$

$$= \log\left(1 + \frac{-\frac{1}{T}\sum_{i=1}^{T} P^2(x_i) + \sum_{i=1}^{T}\sum_{j=1}^{T} P^2(x_i, y_j)}{\frac{1}{T}\sum_{i=1}^{T} P^2(x_i)}\right)$$

This is always greater or equal zero if we can prove that the numerator of the fraction is always positive. With equations

$$\frac{1}{T}\sum_{i=1}^{T} P^2(x_i) = \sum_{j=1}^{T}\sum_{i=1}^{T}\frac{P^2(x_i)}{T^2}$$

and

$$\frac{1}{T}\sum_{i=1}^{T} P^2(x_i) = \frac{1}{T}\sum_{i=1}^{T} P(x_i)P(x_i) = \frac{1}{T}\sum_{i=1}^{T} P(x_i)\sum_{j=1}^{T} P(x_i, y_j)$$

$$= \frac{1}{T}\sum_{i=1}^{T}\sum_{j=1}^{T} P(x_i, y_j)P(x_i)$$

we get for the numerator of the fraction

$$-\frac{1}{T}\sum_{i=1}^{T}P^2(x_i) + \sum_{i=1}^{T}\sum_{j=1}^{T}P^2(x_i,y_j) = \frac{1}{T}\sum_{i=1}^{T}P^2(x_i) - \frac{2}{T}\sum_{i=1}^{T}P^2(x_i) + \sum_{i=1}^{T}\sum_{j=1}^{T}P^2(x_i,y_j)$$

$$= \sum_{j=1}^{T}\sum_{i=1}^{T}\frac{P^2(x_i)}{T^2} - \frac{2}{T}\sum_{i=1}^{T}\sum_{j=1}^{T}P(x_i,y_j)P(x_i) + \sum_{i=1}^{T}\sum_{j=1}^{T}P^2(x_i,y_j)$$

$$= \sum_{j=1}^{T}\sum_{i=1}^{T}\left(\frac{P^2(x_i)}{T^2} - 2P(x_i,y_j)\frac{P(x_i)}{T} + P^2(x_i,y_j)\right)^2 = \sum_{i=1}^{T}\sum_{j=1}^{T}\left(\frac{P(x_i)}{T} - P(x_i,y_j)\right)^2$$

The expression for the numerator can only become greater than or equal zero. Therefore, the logarithm becomes

$$I_2(X;Y) = \log\left(1 + \frac{\sum_{i=1}^{T}\sum_{j=1}^{T}\left(\frac{P(x_i)}{T} - P(x_i,y_j)\right)^2}{\frac{1}{T}\sum_{i=1}^{T}P^2(x_i)}\right) \qquad (25)$$

and the argument of the logarithmic function is always greater equal one. So, for a uniformly distributed random variable *Y* the desired relation (18) also holds for $\alpha = 2$.

For independent random variables X and Y, we have $P(x_i,y_j) = P(x_i)P(y_j)$. Then, with $P(y_i) = 1/T$ we get

$$\sum_{i=1}^{T}\sum_{j=1}^{T}\left(P(x_i,y_j) - P(x_i)/T\right)^2 = \sum_{i=1}^{T}\sum_{j=1}^{T}P(x_i)^2\left(P(y_j) - 1/T\right)^2 = 0$$

and therefore

$$I_2(X;Y) = \log(1) = 0 \qquad (26)$$

Conversely, eq. (26) holds only iff the quotient in eq. (25) is zero. This is only the case for the positive denominator iff the numerator is zero which in turn is only the case for positive terms

$$\left(P(x_i,y_j) - P(x_i)/T\right)^2 = 0$$

or with $P(y_i) = 1/T$

$$P(x_i,y_j) - P(x_i)P(y_j) = 0 \quad \text{or} \quad P(x_i,y_j) = P(x_i)P(y_j)$$

which is the definition of independency. Thus, for zero mutual information both random variables have to be independent. □

Please note that $P(x_i)$ is the probability of the i-th event of random variable X. This is also true for a compound random variable $\mathbf{x}_i \in \Re^n$. Therefore, the theorem above (eq.(24)) is also true for compound random variables **X**.

Appendix C The size of the Hypercube

C.1 Adapting the hypercube size

The size ε of the hypercube is essential for estimating correctly co occurrences of the events. If it is taken too small, the number of samples within the hypercube changes too much from location to location. Conversely, if it is taken too big, the sample average does not reflect the local distribution well. Therefore, an adaptive algorithm is used for estimating a "good" ε. As goal, we chose a certain percentage s of samples per hypercube of the joint distribution p_{xy}, i.e. the corresponding entropy $H_s = –\log(s)$. For adaptation, let us use interval nesting.

Defining the function $f(\varepsilon) = H_s – H_{xy}(\varepsilon)$ for ε within the interval $[\varepsilon_0=0, \varepsilon_1=1]$ we know by eq. (13) that for zero cube size no samples are captured except the central sample of the cube

$$f(0) = H_s – H_{xy}(0) = –\log(s) + \log(1/T) < 0$$

because $1 > s > 1/T$. Additionally, when we capture all the samples, we have a ratio of one in eq. (12) and we get

$$f(1) = H_s – H_{xy}(1) = –\log(s) + \log 1 > 0$$

In fact, $f(\varepsilon)$ has one zero crossing within the interval. By choosing a new boundary within the old interval and defining whether it is on the right side or in the left side, we maintain the property of the interval that it contains a zero crossing of $f(\varepsilon)$. Therefore, we use the following steps:

```
1. Choose a new value within the interval, e.g. $\varepsilon(n) = (\varepsilon_1-\varepsilon_0)/2$.
2. IF $f(\varepsilon(n)) > 0$ THEN $\varepsilon_1 = \varepsilon(n)$      // use it as new right boundary
                    ELSE $\varepsilon_0 = \varepsilon(n)$.       // use it as new left boundary
3. Increment $n$ and continue at step 1.
```

The whole process should be stopped when $f(\varepsilon)$ is small enough. The obtained ε should be used also for the computation of $H(x)$ and $H(y)$.

C.2 Mutual information, feature dimensionality increase and hypercube size

When we use the forward selection procedure described in section 2, the dimension of the random variable X for computing the mutual information $I_2(X;Y)$ increases after each iteration and changes the mutual information. In order to understand this change, we have to take the following insights into account:

- For a fixed hypercube size ε we see that the mutual information

  $$I_2(X;Y) = H_2(X) + H_2(Y) – H_2(X,Y)$$

  $$= -\log\left(h(C_y)\right) - \log\left(h(C_x)\right) + \log\left(h(C_{xy})\right) \text{ with } h(C) = \frac{1}{T}\left(1+\frac{2C}{T}\right) \qquad (27)$$

  depends on the monotonously increasing function $h(C)$. Since the negative logarithm is decreasing, the whole function $H(h(C))$ is monotonously decreasing. If we add the *n+1*-th variable with its additional time series, we implement it by adding an additional AND condition in eq. (8). Here, the number of terms $B(t_1,t_2)$ with the value *true* will be the same or less, but not more than that without the *n+1*-th variable. This leads to a constant or decreasing C_x and therefore to a constant or increasing $H_2(h(C_x))$.

  For the correlation integral C_{xy} we have to include additional conditions for the time series samples of Y. Therefore, we have $C_x \geq C_{xy}$ which means that $H_2(h(C_x)) \leq H_2(h(C_{xy}))$. If we add a variable, C_x will decrease, $H_2(X)$ will increase, and so will $H_2(X,Y)$ and stay always bigger than $H_2(X)$. With constant $H_2(Y)$, the resulting $I_2(X;Y)$ will stay the same or become smaller.

- When an additional input variable is considered, the high-dimensional event space has less samples. Therefore, the hypercube size has to increase in order to hold the same number of samples as before. For increasing ε, the counters C_y and C_x increase, making $H_2(Y)$ and $H_2(X)$ smaller. Since ε is increased such that H_{xy} takes the same value as before adding the additional feature, the resulting mutual information becomes smaller.

In conclusion, increasing the number of features will decrease the amount of mutual information.

C.3 Variance and hypercube size

The size of the hypercube which we consider here directly depends on the desired number of samples to catch within, e.g. p = 2% of *T* samples. For *n* time series we have a *n*-dimensional space where we place our hypercube. Assuming that we have uniformly distributed samples within the hyperspace, the *pT* samples are located in a hypercube with a volume $V_\varepsilon = \varepsilon^n$ out of the whole volume $V = T^n$, i.e. $V_\varepsilon = pT^n = \varepsilon^n$. Therefore, we have

$$\varepsilon = p^{1/n}T \quad \text{or} \quad p = \left(\frac{\varepsilon}{T}\right)^n \tag{28}$$

Ideally, this is the hypercube length. In reality, we have to include two facts:

- The distributions involved are not uniform
- There is a variance within each hypercube which can not be neglected

Therefore, the actually needed ε will be different. Because we compute the correlation integral of eq. (10) by averaging over all hypercubes, we have to consider the variance within this averaging process in order to predict the variability (the error) in measuring the mutual information.

The number of samples within a hypercube initially does not depend on the other hypercubes. Counting samples within a hypercube is equivalent to counting random events of a random source sorted into a bin. Assuming that we have *k = pT* samples in a bin we have a probability of *p* that random events are put into the bin and *(1–p)* that this is not the case. The probability of a discrete random event to be in the bin is therefore

$$P = p(1-p)$$

After *T* random events we have *k* of those in the bin. There are $\binom{T}{k}$ possibilities of selecting a subset of *k* events of all *T* events. Each of this subsets has the probability of $p^k(1\text{-}p)^{T\text{-}k}$ to occur. Therefore, the probability to count *k* samples within a hypercube is

$$P_B(k|p,T) = \binom{T}{k}\, p^k(1-p)^{T-k} \qquad \textit{Binomial distribution} \tag{29}$$

This probability is well known, the corresponding distribution is called the Binomial distribution and the cumulative distribution function is

$$F_X = \sum_{k=1}^{\lfloor x \rfloor} \binom{T}{k} p^k (1-p)^{T-k}$$

Principally, for T→∞, the Binomial distribution becomes a Gaussian distribution. As a rule of thumb this is already the case for T·p and T(1–p) > 5.

Therefore, the variance within a bin is $\sigma^2 = p(1-p)$ or with eq. (28) to

$$\sigma^2 = p(1-p) = p - p^2 = \left(\frac{\varepsilon}{T}\right)^n - \left(\frac{\varepsilon}{T}\right)^{2n}$$

This means, the variance of the samples within a bin is determined by the hypercube size length ε nonlinearly.

The average for all the bins is the sum of them

$$\sigma_B^2 = \sigma_1^2 + \ldots + \sigma_T^2 \tag{30}$$

For uniformly distributed samples the probabilities of all bins are equal and we have

$$\sigma_B{}^2 = T\, p(1\text{–}p) = T\left[\left(\frac{\varepsilon}{T}\right)^n - \left(\frac{\varepsilon}{T}\right)^{2n}\right]$$

This is true for independent bins. Now, in the averaging process the bin window is shifted only a small length within the input space. Therefore, are certain fraction of the bin under consideration contains samples in common with the neighbouring bins. Therefore, the bins are not independent; the variance of the random variables is not the simple sum of eq.(30) but it is smaller.

Appendix D Computation pseudo code

The main ideas are demonstrated by the following pseudo code which takes all the considerations into account. It uses the symmetry approach acceleration of eq.(13) and the ranking approach of eq.(14) as well as the missing value evaluation of eq.(15).

```
double hx(int[][] X, int[] index, int epsilon){
int end = VIP_List.length;
int Cx = 0;      // init correlation integral values
int M = 0;       // init counter for comparisons;
int t1,t2;       // absolute and relative time index
for (int j = 0; j < T-1; j++){            // check all point indices
   t1 = index[j];                         // and corresponding time points
boolean c = true; int i = 0;
   while (c && (i <= end)) {              // check for missing values in x
        c = c && (X[i][t1] != SPECIAL_VALUE);
        i++;
   }
   if (!c) { M++;                         // if missing value in t1: count it
   } else {                               // look for missing value in t2
        int k = j+1; c1++;
        while ( (k < j + epsilonInt) && (k < T) ){ // and all neighbors within ε
        t2 = index[k];   c2++;
        c = true; i = 0;
        while (c && (i <= end)) {// check co-occurrence for already selected features
              c = c && ( Math.abs(X[i][t1]- X[i][t2]) < epsilon )
                     && ( X[i][t2] != SPECIAL_VALUE );
              i++; } if (c) Cx++;         // All conditions for x met? count Cx on.
        k++;
        }  // end while k
 }  // end if..else
  }  // end for j
  return g(Cx,M);
}  // end hx()
```

The computation compares the time series X[0] with all other time series X[i]. The precondition for this is the uniform distribution of one time series. This is accomplished by the ranking of the *y* = X[0] time series prior to comparison. The ranking algorithm is a variant of the well-known quicksort algorithm, presented e.g. in [21]. In Java this can be done for instance by the following code for the time series `data` of length `T` which returns the ranked array and the index array:

```
public int[][] rank(double[] data, int T){
 this.data = data;                     // copy reference to unranked data
 RankX = new int [T];
 Tindex = new int [T];

 for (int j = 0; j < T; j++) { Tindex[j] = j; }  // init Tindex
 QuicksortIndex(0,T-1);                          // reorder it according to sample rank
 for (int j = 0; j < T; j++){                    // set invers transform
      RankX[Tindex[j]] = j;
 }// end for j

 int [][] result = {RankX, Tindex};
 return result;
}// end ranking

private void QuicksortIndex(int left, int right){
 int t; double middle;
 int j = left;
 int k = right;

 if (right > left){
     middle = data[Tindex[(left + right)/2 ]];
     while (j <= k){
          while (data[Tindex[j]] < middle){ j = j+1;  }
          while (data[Tindex[k]] > middle){ k = k-1;  }
          if (j <= k){
            t = Tindex[j];
            Tindex[j] = Tindex[k];
            Tindex[k] = t;
            j = j+1;
            k = k-1;
          }//end if
     } //end while j
     QuicksortIndex(left,k);
     QuicksortIndex(j,right);
 } //end if
}//end QuicksortIndex
```