2011 American Control Conference
on O'Farrell Street, San Francisco, CA, USA
June 29 - July 01, 2011

Reliable Nearest Neighbors for Lazy Learning

Tobias Ebert, Geritt Kampmann and Oliver Nelles
Department of Mechanical Engineering
University of Siegen
D-57068 Siegen, Germany
tobias.ebert@uni-siegen.de

Abstract— **A key problem of memory based learning methods is the selection of a good smoothing or bandwidth parameter that defines the region over which generalization is performed. In this article we present a novel algorithm to answer this question by utilizing the information from confidence intervals, to compute a bandwidth. The basic idea is the usage of confidence intervals to get a statistical statement about the quality of fit between estimated model and process. As long as the prediction intervals of a certain model include the neighboring data points of an incremental growing validity region, it is considered to be a good fit.**

I. INTRODUCTION

Lazy learning [1], also known as Just-in-Time modeling [2], [3], Model on Demand [4] or instance-based learning [5], is a powerful approach for data based modeling. Until a query, also called novel instance, needs to be answered, data is stored without further processing. Only then relevant data is extracted from the database to allow the estimation of a model to answer the query. This design allows an inherent ability for adaptation, a desperately needed attribute, if the adaptive qualities of other approaches like neural networks is insufficient. There are several Examples of possible application, where a lazy learning approach is advantageous: Systems with shifting limits [6] or rapidly changing systems in online applications, where frequent re-training, to adapt a model, is impractical.

A recurring question is, which data should be used to fit a model in memory-based learning methods for good results [7]. This article presents a robust algorithm, which is able to estimate a reasonable region size for a valid neighborhood in the data volume. The Goal is, to select those samples from the whole data set, which are required for a good estimation of the model in the current operating point. The paper is organized as follows: Section II delivers some insight into Model on Demand theory. Section III briefly explains confidence and prediction intervals. In Sec. IV a novel algorithm to find a reasonable size for a valid neighborhood is presented. An useful improvement to the algorithm is presented in Sec. V. Last but not least Sec. VI states the conclusion of this paper and sums up the results.

II. MODEL ON DEMAND

For an n-dimensional input space a model output is to be calculated. Model on Demand aims to compute an answer for a query point, where no observation is given. Only a data cloud $\mathbb{M}$ is given, which in most of all cases can not give a direct answer to a query. Model on demand approximates a model for the query, such that some kind of interpolation can be evaluated.

The evaluation of a model always requires four steps. First, a valid neighbor region $\mathbb{N}$ around the query point $\boldsymbol{q}$ must be selected out of the data set $\mathbb{M}$, to approximate a model:

$$\mathbb{N} \subseteq \mathbb{M} \in \mathbb{R}^p \,. \tag{1}$$

This neighborhood $\mathbb{N}$ contains all data points that are necessary, to reliable estimate the parameters of the model. The aim is to find a $\mathbb{N}$ to describe the query $\boldsymbol{q}$ with a chosen model as good as possible. The relevance of each data point is determined by its distance d from the query. Each data point consists of an output value y and an input vector $\boldsymbol{x}_i$ or $\boldsymbol{z}_i$. Similar to premise and consequent variables used in Fuzzy Systems, not all information within the data input vector needs to be used for the distance calculation or approximation of the model parameters. They may, but do not need to, include the same information. Therefore we distinguish between an input vector $\boldsymbol{z}_i$ to calculate the distance and an input vector $\boldsymbol{x}_i$ to estimate the parameters, which may be equal, but that is not a requirement as stated above,

$$\boldsymbol{x}_i,\, \boldsymbol{z}_i \in \mathbb{N} \,. \tag{2}$$

A typical distance function is the Euclidean distance

$$d_{\text{Euclidean}} = \sqrt{\sum_{j=1}^{p} \left(\boldsymbol{z}_j - \boldsymbol{q}_j\right)^2} = \sqrt{(\boldsymbol{z}-\boldsymbol{q})^T\,(\boldsymbol{z}-\boldsymbol{q})}\,, \tag{3}$$

where p is the dimension of $\boldsymbol{z}$. There are several ways [8] to define a reasonable size of the neighborhood. One common and very simple approach is the limitation of $\mathbb{N}$ to the k nearest neighbors (knn) [9]. Then, $\mathbb{N}$ consists only of those k data points with the k smallest distances to the query, see Fig. 1. The most important question is how to choose k, or the size of $\mathbb{N}$. Common approaches for an approximated solution to this problem are methods like leave-one-out cross validation [10], [11], [12]. These are able to find a global optimum for k, i.e. valid for the entire input space. But this is not the desired result, since we are looking for a local model. A good choice of k may depend strongly upon where the query is, for a good trade-off between bias and variance error. Additionally, if there are multiple queries and the underlying data amount changes, as in a time varying process, it must be reapplied to account for the changed data set. Obviously, this is very time consuming, especially, if $\mathbb{M}$ consists of many data points. A new algorithm is presented in this paper, which is an outperforming alternative to existing approaches, see Sec. IV. The size of the neighborhood problem for the knn approach can also be interpreted as a bandwidth problem with a uniform weighting function, see [8].

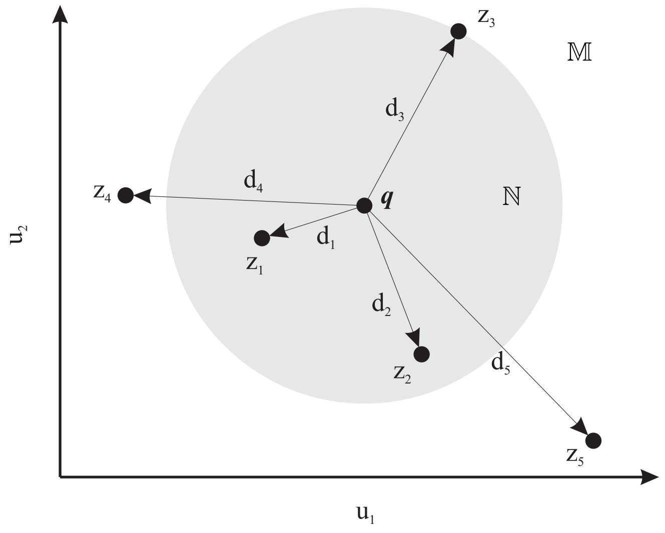

Fig. 1. A two-dimensional example ($p = 2$) of a query and neighborhood including its three nearest neighbors

The second step of modeling is to choose the model structure. Typically, linearly parameterized model structures like polynomials are chosen, because in practice this often leads to well generalized fits, but other model structures are also possible. It is not advisable to use a high order polynomial in the case of high dimensional data, as it requires the computation of many parameters and high dimensional spaces tend to be sparsely occupied. A low order polynomial is sufficient in most applications.

The third step is, to estimate the parameters $\boldsymbol{\theta}$ of the chosen model, based on the valid neighborhood region. The regressors $\boldsymbol{x}_i$ correspond to the underlying model structure, e.g. a second order polynomial in two dimensional space has the regression vector

$$\boldsymbol{x}_i^T = \begin{bmatrix} 1 & u_1(i) & u_2(i) & u_1(i)u_2(i) & u_1^2(i) & u_2^2(i) \end{bmatrix}. \quad (4)$$

It is beneficial to store the data points $\boldsymbol{x}_i \in \mathbb{N}$ in a matrix equation. This leads to the standard least squares (LS) formulation:

$$\boldsymbol{X}\,\boldsymbol{\theta} = \boldsymbol{y}\,, \quad (5)$$

where $\boldsymbol{X}$ is the regression matrix whose ith row is $\boldsymbol{x}_i^T$ and $\boldsymbol{y}$ whose ith element is y_i, representing the ith data point. This equation can be solved efficiently with LS, see [13]. The result achieved is the estimated parameter vector $\hat{\boldsymbol{\theta}}$. The LS solution is

$$\hat{\boldsymbol{\theta}} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}\,. \quad (6)$$

As the last step, the model output $\hat{y}$ for the query $\boldsymbol{q}$ is calculated. With the estimated parameters of the model the generalized solution for the query is

$$\hat{y}(\boldsymbol{q}) = \boldsymbol{q}^T\,\hat{\boldsymbol{\theta}}\,. \quad (7)$$

III. Confidence and Prediction Intervals

After a model is estimated, it is of great interest to know how valid this model is. A common way to judge the quality of a model is the concept of error bars. In order to compute the error bars of an estimated model, some information about the accuracy of the estimated parameters is needed. As shown in [13] [14] it is possible to describe the accuracy of the estimated parameters by its covariance matrix

$$\mathrm{cov}\{\hat{\boldsymbol{\theta}}\} \;=\; \mathrm{E}\left\{\left(\hat{\boldsymbol{\theta}} - \mathrm{E}\{\hat{\boldsymbol{\theta}}\}\right)\left(\hat{\boldsymbol{\theta}} - \mathrm{E}\{\hat{\boldsymbol{\theta}}\}\right)^{\mathrm{T}}\right\}. \quad (8)$$

When $\mathrm{E}\{\boldsymbol{X}\} = \boldsymbol{X}$, i.e., $\boldsymbol{X}$ is deterministic and, if white noise of variance σ^2 is assumed, it can be shown that

$$\mathrm{cov}\{\hat{\boldsymbol{\theta}}\} \;=\; \sigma^2\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}. \quad (9)$$

As one is interested in the absolute values of $\mathrm{cov}\{\hat{\boldsymbol{\theta}}\}$, an estimate of σ^2 is required. The noise variance is usually unknown, but it can be estimated from the residuals. One simple unbiased estimator of σ^2 is [15]

$$\hat{\sigma}^2 \;=\; \frac{\boldsymbol{e}^{\mathrm{T}}\boldsymbol{e}}{N-n}\,, \quad (10)$$

where $\boldsymbol{e} = \boldsymbol{y} - \hat{\boldsymbol{y}}$. The denominator in the above formula represents the degrees of freedom of the residuals, that is, the number of data samples N minus the number of parameters n. It is wise to use these estimates carefully, since they are based on the assumption of additive white measurement noise and a correctly assumed model structure. These assumptions can be quite unrealistic. Especially, considerable errors due to a structural mismatch between process and model can be expected for almost any application.

As described by (9), the parameters of a model can be estimated only with a certain variance, given finite and noisy data. The estimated model output $\hat{\boldsymbol{y}}$ is

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\,\hat{\boldsymbol{\theta}}\,. \quad (11)$$

The parameter covariance matrix $\mathrm{cov}\{\hat{\boldsymbol{\theta}}\}$ determines the accuracy of the model output for a given input:

$$\mathrm{cov}\{\hat{\boldsymbol{y}}\} = \mathrm{E}\left\{(\hat{\boldsymbol{y}} - \mathrm{E}\{\hat{\boldsymbol{y}}\})(\hat{\boldsymbol{y}} - \mathrm{E}\{\hat{\boldsymbol{y}}\})^{\mathrm{T}}\right\} \quad (12a)$$
$$= E\left\{\left(\boldsymbol{X}\left(\hat{\boldsymbol{\theta}} - \mathrm{E}\{\hat{\boldsymbol{\theta}}\}\right)\right)\left(\boldsymbol{X}\left(\hat{\boldsymbol{\theta}} - \mathrm{E}\{\hat{\boldsymbol{\theta}}\}\right)\right)^{\mathrm{T}}\right\} \quad (12b)$$
$$= \boldsymbol{X}\,\mathrm{E}\left\{\left(\hat{\boldsymbol{\theta}} - \mathrm{E}\{\hat{\boldsymbol{\theta}}\}\right)\left(\hat{\boldsymbol{\theta}} - \mathrm{E}\{\hat{\boldsymbol{\theta}}\}\right)^{\mathrm{T}}\right\}\boldsymbol{X}^{\mathrm{T}}. \quad (12c)$$

Thus, the covariance matrix of the model output $\hat{\boldsymbol{y}}$ is

$$\begin{aligned} \operatorname{cov}\{\hat{\boldsymbol{y}}\} &= \boldsymbol{X}\operatorname{cov}\{\hat{\boldsymbol{\theta}}\}\boldsymbol{X}^{\mathrm{T}} && (13a)\\ &= \sigma^2\boldsymbol{X}\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}, && (13b)\end{aligned}$$

under the assumption $\mathrm{E}\{\boldsymbol{X}\} = \boldsymbol{X}$. Since the diagonal entries of $\operatorname{cov}\{\hat{\boldsymbol{y}}\}$ represent the variances of the model output for each data sample in $\boldsymbol{X}$, the error bars can be defined as $\hat{\boldsymbol{y}}$ plus and minus the standard deviation of the estimated output, that is,

$$|\hat{\boldsymbol{y}} - \boldsymbol{y}| = \hat{\boldsymbol{y}} \pm \sqrt{\operatorname{diag}\left(\operatorname{cov}\{\hat{\boldsymbol{y}}\}\right)}\,. \quad (14)$$

It is important to notice that the equations (12)-(14) show the confidence intervals based on the variance of the estimated parameters. If the model fits the data well, the confidence intervals become narrower, as more data points are added and the parameters are estimated more reliable. To calculate the prediction intervals, which include a constant percentage of the data points of $\mathbb{N}$ around the approximated model, the noise variance of the data must be included. The equation becomes

$$|\hat{\boldsymbol{y}} - \boldsymbol{y}|_\sigma = \hat{\boldsymbol{y}} \pm \sqrt{\operatorname{diag}\left(\sigma^2 + \operatorname{cov}\{\hat{\boldsymbol{y}}\}\right)}. \quad (15)$$

Furthermore, one does not want a prediction interval with just $\pm\sigma$ around the estimated model. These prediction intervals would only include about 68% of all data points. Thus, a factor to enlarge the prediction intervals is needed. In addition, $\mathbb{N}$ may include only a low number of data points, so one has to keep in mind, that a small number of data points is possible. As a normally distributed population is assumed, a compensation for a small sample size is necessary. One solution to this problem is the usage of Student's t-distribution, to compensate for low numbers of N. The prediction interval with a significance level α for a query results in

$$|\hat{\boldsymbol{y}} - \boldsymbol{y}|_\alpha = \hat{\boldsymbol{y}} \pm \sqrt{\operatorname{diag}\left(\sigma^2 + \operatorname{cov}\{\hat{\boldsymbol{y}}\}\right)} \cdot t_{1-\frac{\alpha}{2}}\,, \quad (16)$$

where t is Student's t-distribution for $N - n$ degrees of freedom. In the following examples $\alpha = 0.05$ is used.

IV. Confident nearest neighbors

As noted above, prediction intervals can be used to make a statement about the probability that an estimated model is a good description of nearby data points. This property shall now be exploited to find a reasonable neighborhood $\mathbb{N}$. The basic idea hereby is an incrementally growing neighborhood $\mathbb{N}$ [16]. After each iteration all data points within $\mathbb{N}$ are checked, if they are engulfed by the prediction intervals of an approximated model. A simple algorithm is

1) Calculate the distance d of all data points of $\mathbb{M}$ to the query and sort them by increasing distance.
2) Initialize the neighborhood $\mathbb{N}_0$ of the model with a minimum number of nearest neighbors. To initialize the neighborhood $\mathbb{N}_0$ at least one more data point than parameters to estimate is necessary: $\mathbb{N}_0 = \mathbb{N}(n+1)$.
3) Estimate the parameters $\hat{\boldsymbol{\theta}}$ of the model and calculate the prediction intervals for $\mathbb{N}_i$.
4) Check, if all data points of $\mathbb{N}_i$ are within the prediction intervals.
   a) If more than α% of the data points of $\mathbb{N}_i$ are outside of the prediction intervals, $\mathbb{N}_{i-1}$ is the desired neighborhood.
   b) If less than α% of the data points of $\mathbb{N}_i$ are outside of the prediction intervals add the next nearest neighbor to $\mathbb{N}_{i+1}$ and go to 3).

A step size larger than one data point per iteration is also possible and further improvements can be made, if the computed information about the noise is used to approximate the model. But those are open topics for further research.

A. Results

The algorithm for reliable nearest neighbors shows several appealing features. First of all, it inherits the flexible neighborhood region of the k nearest neighbor approach. This is a mayor advantage in contrast to a kernel [8] based method. Imagine a constant kernel bandwidth for the neighborhood of a query. In a sparsely occupied data space it is comprised of few data points, while in a dense data space it contains many data points. The knn approach is resistant against different densities of data points in one data set. The k nearest neighbors is a constant value that does not change. Whether the data is dense or not, $\mathbb{N}$ contains the same number of data points. If the data points are very compact, the spatial extent of $\mathbb{N}$ is small, if the density is low, it is rather large. Thus, the algorithm is by this inherited feature adaptive to the density of the data.

Secondly, as the prediction intervals are narrow bands around the approximated model, a change of the gradient of the respective function leads to data points, which are not within the confidence intervals any more. As an example where this feature is easily observable, we use the function

$$y = \frac{1}{1.1 - u}\,, \quad (17)$$

see Fig. 2a, with 50 equal distanced data points with low noise ($\sigma^2 = 10^{-5}$) and a first order polynomial for modeling. We expect the algorithm to decrease the size of $\mathbb{N}$ as the gradient of $y(u)$ increases, because the error between model and data increases. The resulting size of the neighborhood computed with the described algorithm is depicted in Fig. 2b. In the left part of Fig. 2a the change of the gradient is low and the chosen model fits the data points. The resulting neighborhood depicted in Fig. 2b with about 20 data points is rather large. While one proceeds to higher values of u the change of gradient ascends and the chosen model is increasingly worse at following the data. This means that for a decreasing size of $\mathbb{N}$ more data points leave the prediction intervals and the algorithm stops. Thus, the algorithm adjusts the size of $\mathbb{N}$ to the slope of the gradient. Note, that the variance of noise was not announced to the algorithm and hence was approximated according to (10). The result is shown in $\hat{y}(u)$. If the true variance of the noise is given to the algorithm, its performance improves

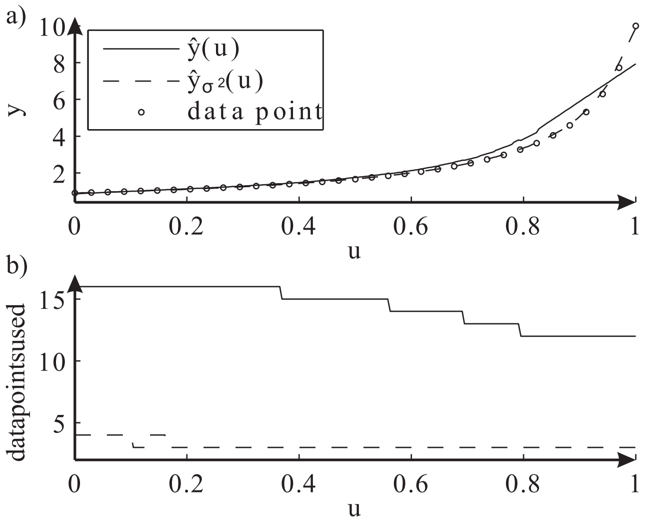

Fig. 2. a) Sample one-dimensional function b) Resulting size of $\mathbb{N}$.

dramatically. This behavior is shown in $\hat{y}_{\sigma^2}(u)$

Thirdly, the prediction intervals depend directly on the noise of the data. This is an example of a sine wavelike function

$$y(u) = \sin(2\,\pi\,u) + 2\,u \tag{18}$$

with additive white noise of variance $\sigma^2 = 4 \cdot 10^{-5}$ and 100 equally distanced data points, see Fig. 3. The model used

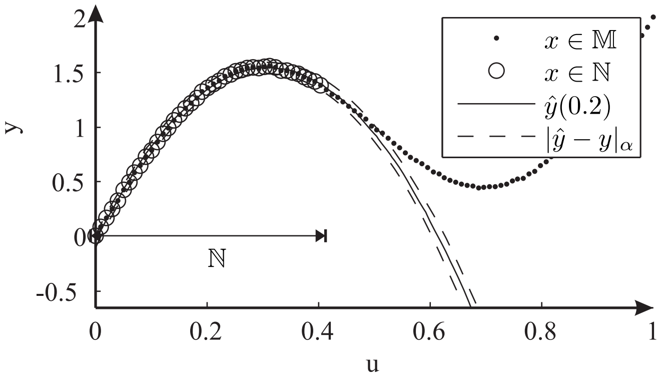

Fig. 3. Resulting model, prediction intervals and $\mathbb{N}$ for $q(u = 0.2)$ and low noise.

for the query $q(u = 0.2)$ is a second order polynomial. As the noise is low, the prediction intervals are narrow. This leads to a narrow $\mathbb{N}(u \approx 0 \ldots 0.4)$. The algorithm is aware of data points, which do not fit the assumed model and stops the growth of $\mathbb{N}$. If the noise is more noticeable with $\sigma^2 = 4 \cdot 10^{-2}$, see Fig. 4, the prediction intervals include a much larger space between them. This shows, that the algorithm is less strict about the deviation of data points from the model, which leads to a grown $\mathbb{N}(u \approx 0 \ldots 0.6)$. The neighborhood is about 30% larger, as more data points are within the prediction intervals. The neighborhoods shown in Fig. 3 and Fig. 4 are the last $\mathbb{N}_i$ the algorithm accepts

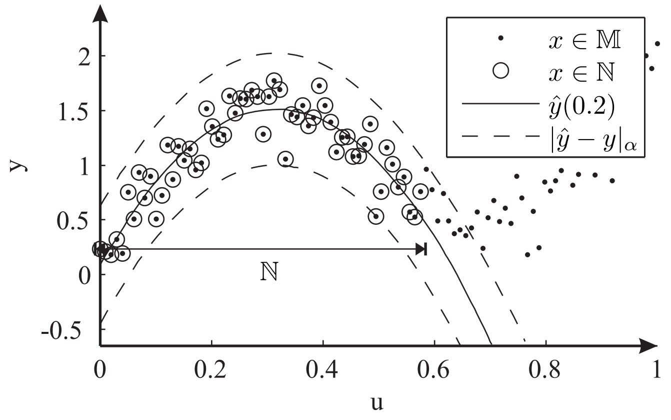

Fig. 4. Resulting model, prediction intervals and $\mathbb{N}$ for $q(u = 0.2)$ and stronger noise

as valid before stopping. As shown in these examples, the algorithm is able to adjusts the size of the valid neighborhood to the noise within the data.

V. Confident Nearest Neighbors with Kernel Function

Without a kernel function the algorithm described above is sensible to migration of data, as it will enlarge $\mathbb{N}$, if the gradient of the underlying function changes. An example of a worst case scenario is a query in the middle $q(u = 0.5)$ of function (18). The Goal is, to approximate it, without any knowledge about the noise. As the gradient slowly changes on both ends in different directions the approximation of the noise becomes increasingly wrong. Directly linked to the approximated variance of the noise, the prediction intervals grow. In the end, too many data points form the neighborhood before the algorithm stops, as shown in Fig. 5. While

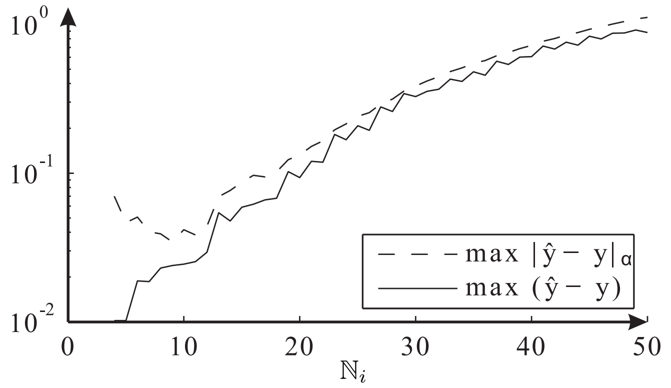

Fig. 5. Estimation error and prediction intervals in each iteration for $q(u = 0.5)$ and low noise

it is possible to manually avert this behavior by adjusting α, or define a maximal size of $\mathbb{N}$, this is far from ideal, because the need for manual intervention is an unappealing attribute. Note that this behavior can occur on any data set.

A robust solution to this problem can be achieved. The estimated model for $\mathbb{N}$ can be stabilized by forcing the estimated model to depend mainly on the neighbors close to the query. An approach to solve this is the application of a weighting function, also known as kernel function, which makes sure that the data points near the query are more important to the estimation of θ, see [8]. One common weighting or kernel function is the Gaussian curve which can be described by

$$q_i = e^{\left(-\frac{d_i^2}{2}\right)} \quad i = 1 \dots N \tag{19}$$

The general solution to the weighted least squares optimization problem (6) is

$$\hat{\boldsymbol{\theta}}_Q = \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{Q} \, \boldsymbol{X}\right)^{-1} \boldsymbol{X}^{\mathrm{T}} \boldsymbol{Q} \, \boldsymbol{y} . \tag{20}$$

where $\boldsymbol{Q}$ is the weighting matrix. In the approach presented here the weighting matrix has a diagonal structure. The matrix entry Q_{ii} is given by q_i. Similar to (11) the estimate of the model with weighted parameters is

$$\hat{\boldsymbol{y}}_Q = \boldsymbol{X} \, \hat{\boldsymbol{\theta}}_Q . \tag{21}$$

As shown in [13], the estimated weighted covariance matrix of the parameters $\mathrm{cov}\{\hat{\boldsymbol{\theta}}_Q\}$ follows

$$\mathrm{cov}\{\hat{\boldsymbol{\theta}}_Q\} = \sigma_Q^2 \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{Q} \boldsymbol{X}\right)^{-1} \boldsymbol{X}^{\mathrm{T}} \boldsymbol{Q} \boldsymbol{Q} \boldsymbol{X} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{Q} \boldsymbol{X}\right)^{-1} , \tag{22}$$

with the weighted estimation of σ^2

$$\hat{\sigma}_Q^2 = \frac{\boldsymbol{e}^{\mathrm{T}} \boldsymbol{Q} \boldsymbol{e}}{\left(\sum\limits_{i=1}^{N} q_i\right) - n} . \tag{23}$$

The covariance matrix of $\hat{\boldsymbol{y}}_Q$ changes accordingly to

$$\begin{aligned} \mathrm{cov}\{\hat{\boldsymbol{y}}_Q\} &= \boldsymbol{X} \, \mathrm{cov}\{\hat{\boldsymbol{\theta}}_Q\} \, \boldsymbol{X}^{\mathrm{T}} & (24) \\ &= \sigma_Q^2 \, \boldsymbol{X} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{Q} \boldsymbol{X}\right)^{-1} \boldsymbol{X}^{\mathrm{T}} \boldsymbol{Q} \, \boldsymbol{Q} \boldsymbol{X} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{Q} \boldsymbol{X}\right)^{-1} \boldsymbol{X}^{\mathrm{T}} . & (25) \end{aligned}$$

Finally, the new weighted prediction intervals are given by

$$|\hat{\boldsymbol{y}}_Q - \boldsymbol{y}|_\alpha = \hat{\boldsymbol{y}}_Q \pm \sqrt{\mathrm{diag}\left(\sigma_Q^2 + \mathrm{cov}\{\hat{\boldsymbol{y}}_Q\}\right)} \times t_{1-\frac{\alpha}{2}} . \tag{26}$$

By introducing a kernel function one is now challenged to find a good scaling factor for the distance to weight a data point, which is another bandwidth problem. But in this case one has an advantage: There is an already defined neighborhood. By design the data points near the query shall have a larger impact upon the model for the query. To do so, we scale the Gaussian kernel (19) in each dimension p with a factor c_j. That gives us

$$q_i^* = e^{-\frac{d_i^{*2}}{2}} , \tag{27}$$

with

$$d_i^* = \sqrt{\sum_{j=1}^{p} \left(\frac{d_{ij}}{c_j}\right)^2} . \tag{28}$$

Now a compensation factor c_j is defined, that guarantees that at the median $\overline{d}$ of the distance of all data point from the query, a data points weight is 0.5, see Fig. 6. For a $\mathbb{N}$ of dimension p and the distance function (3) the weighting function (27) takes the form

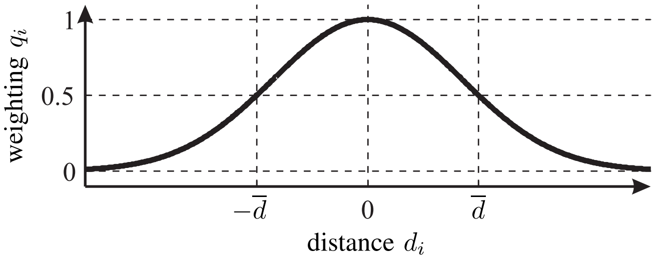

Fig. 6. Aspired weighting over distance

$$0.5 = e^{-\frac{D^2}{2}} , \tag{29}$$

where

$$D = \sqrt{\sum_{j=1}^{p} \left(\frac{\overline{d_j}}{c_j}\right)^2} . \tag{30}$$

The application of the natural logarithm and $q = 0.5$ gives us

$$\log(0.5) = -\frac{1}{2} \sum_{j=1}^{p} \left(\frac{\overline{d_1}}{c_1}\right)^2 \tag{31}$$

As we aim for an equal weighting for all dimensions, that is

$$\frac{\overline{d_1}}{c_1} = \frac{\overline{d_2}}{c_2} = \dots = \frac{\overline{d_p}}{c_p} \tag{32}$$

equation (31) becomes

$$\log(0.5) = -\frac{1}{2} \cdot p \cdot \frac{\overline{d_j}^2}{c_j^2} \tag{33}$$

Finally, if we solve for c_j we get

$$c_j = \sqrt{\frac{p \cdot \overline{d_j}^2}{-2 \cdot \log(0.5)}} . \tag{34}$$

Note, that a distinction between different compensating factors for different dimensions is only necessary, if the inputs are not normalized to the same upper and lower limits. If they are normalized, the equations (27), (28) and (34) become

$$q_{in} = e^{-\frac{d_i^2}{2}} , \tag{35a}$$

$$d_{in} = \left|\frac{d_i}{c}\right| , \tag{35b}$$

$$c_{in} = \sqrt{\frac{p \cdot \overline{d}^2}{-2 \cdot \log(0.5)}} . \tag{35c}$$

With this compensating factor one can now apply a scaled kernel function upon the data points of $\mathbb{N}$ without the need for a second bandwidth estimation. The algorithm described at the beginning of Sec. IV needs to be adjusted as follows:

1) Calculate the distance d of all data points of $\mathbb{M}$ to the query and sort them by increasing distance.
2) Initialize the neighborhood $\mathbb{N}_0$ of the model with a minimum number of nearest neighbors.
3) Calculate the compensating factors c_j and weight the data points with the scaled kernel function (27). Estimate the parameters $\hat{\boldsymbol{\theta}}_Q$ of the model and calculate the weighted prediction intervals $|\hat{\boldsymbol{y}}_Q - \boldsymbol{y}|_\alpha$ for $\mathbb{N}_i$.
4) Check, if all data points of $\mathbb{N}_i$ are within the weighted prediction intervals.
    a) If more than $\alpha\%$ of the data points of $\mathbb{N}_i$ are outside of the weighted prediction intervals, $\mathbb{N}_{i-1}$ is the desired neighborhood.
    b) If less than $\alpha\%$ of the data points of $\mathbb{N}_i$ are outside of the weighted prediction intervals, add the next nearest neighbor to $\mathbb{N}_{i+1}$ and go to 3).

A. Results

With the improved algorithm one can now re-estimate the worst case scenario described above. While we use exactly the same data the new algorithm outperforms the old one and stops at a much more reasonable size for the neighborhood, as shown in Fig. 7. By applying a kernel function on the data points used to estimate the model, it becomes resistant to a slow increase of the gradient for growing $\mathbb{N}$, as the estimated model for the query depends for the most part on the data points, which are nearest to the query.

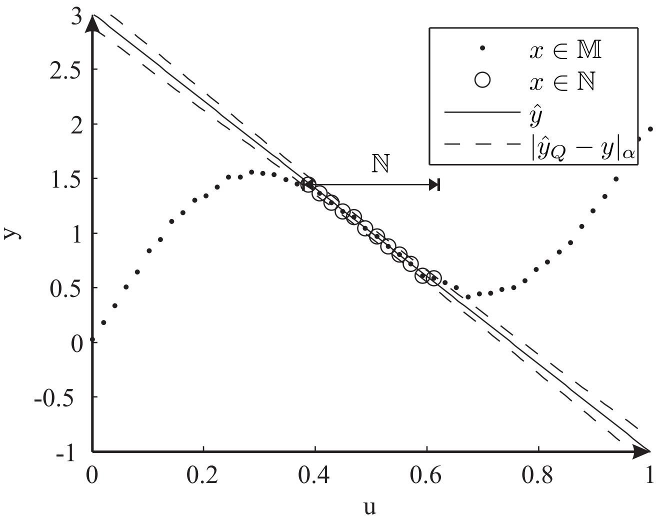

Fig. 7. Resulting model for a worst case scenario with reliable nearest neighbors with kernel function, $q(u = 0.5)$

VI. Conclusion

In this paper, we propose a novel algorithm to answer the bandwidth problem for lazy learning techniques, without the need for computational demanding leave-one-out cross-validation. For learning problems with large or evolving training sets, this can offer significant computational savings. The quality of the model improves drastically, if previous knowledge about the process is present, such as information about noise or what model structure fits the data well. Even without previous knowledge the algorithm is able to determine a reasonable neighborhood region or bandwidth, for a chosen model on a given data set.

References

[1] D. Aha, “Lazy learning,” *Artificial Intelligence Review*, vol. 11, no. 1-5, pp. 7–10, 1997.

[2] G. Cybenko, “Just-in-Time Learning and Estimation,” *Identification, adaptation, learning: the science of learning models from data*, p. 423, 1996.

[3] A. Stenman, F. Gustafsson, and L. Ljung, “Just in time models for dynamical systems,” in *IEEE Conference on Decision and Control*, vol. 1, 1996, pp. 1115–1120.

[4] A. Stenman, “Model on demand: Algorithms, analysis and applications,” *Linkoping Studies in Science and Technology - Dissertations -*, 1999.

[5] D. Aha, D. Kibler, and M. Albert, “Instance-based learning algorithms,” *Machine learning*, vol. 6, no. 1, pp. 37–66, 1991.

[6] F. Fdez-Riverola, E. Iglesias, F. Díaz, J. Méndez, and J. Corchado, “Applying lazy learning algorithms to tackle concept drift in spam filtering,” *Expert Systems with Applications*, vol. 33, no. 1, pp. 36–48, 2007.

[7] J. Fan and I. Gijbels, *Local polynomial modelling and its applications*. Chapman & Hall/CRC, 1996.

[8] C. Atkeson, A. Moore, and S. Schaal, “Locally weighted learning,” *Artificial intelligence review*, vol. 11, no. 1, pp. 11–73, 1997.

[9] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, “An optimal algorithm for approximate nearest neighbor searching fixed dimensions,” *J. ACM*, vol. 45, no. 6, pp. 891–923, 1998.

[10] M. Kearns and D. Ron, “Algorithmic stability and sanity-check bounds for leave-one-out cross-validation,” *Neural Computation*, vol. 11, no. 6, pp. 1427–1453, 1999.

[11] J. Wang and J. Zucker, “Solving the multiple-instance problem: A lazy learning approach,” in *In Proc. 17th International Conf. on Machine Learning*. Morgan Kaufmann, 2000, pp. 1119–1126.

[12] G. Bontempi, M. Birattari, and H. Bersini, “Lazy learning for local modelling and control design,” *International Journal of Control*, vol. 72, no. 7, pp. 643–658, 1999.

[13] O. Nelles, *Nonlinear System Identification*. Berlin, Germany: Springer, 2001.

[14] N. Draper and H. Smith, *Applied regression analysis*. Wiley New York, 1981.

[15] T. Söderström and P. Stoica, *System Identification*. New York: Prentice Hall, 1989.

[16] S. Jakubek and N. Keuth, “Optimierte Neuro-Fuzzy-Modelle für Auslegungsprozesse und Simulation im Automotive-Bereich (Improved Neuro-Fuzzy Models for Design Processes and Simulation in Automotive Applications),” *at-Automatisierungstechnik*, vol. 53, no. 9, pp. 425–433, 2005.