Generating data-based kernels for regression

Datenbasierte Kernel für Regressionsschätzungen
Bachelor-Thesis von Nils Christopher Boeschen aus Limburg
Tag der Einreichung:

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Dipl.-Math. Moritz Schneider

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Knowledge Engineering Group

Fachbereich Elektrotechnik und Informationstechnik
Fachgebiet Regelungsmethoden und Robotik

Generating data-based kernels for regression
Datenbasierte Kernel für Regressionsschätzungen

Vorgelegte Bachelor-Thesis von Nils Christopher Boeschen aus Limburg

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Dipl.-Math. Moritz Schneider

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 06.02.2017

(Nils Boeschen)

Contents

1 Introduction

The increasing amount of available or collectable data had a great impact in many branches of science. Research in mathematics, computer science, electrical engineering and many other related fields has brought forth ambitious efforts in the search of methods to extract information from growing mountains of data.
One of the oldest and most desired goals of these approaches is to create an artificial model that can predict attributes of future (or otherwise incomplete) data points, in order to use this knowledge for informed decisions. If the predicted attributes are of numerical type, this problem is known as regressional analysis. From weather forecasts, controlling of industrial plants, to market analysis and production optimization, the application scenarios for accurate inference systems are numerous.

In this thesis, I will examine a novel system called "AnYa" ([Angelov and Yager, 2011]), which uses a supervised learning approach to construct a rule-based model from given data samples. AnYa was developed based on advances made in the field of computational intelligence and has the interesting feature of being almost free of the need to predefine parameters. Instead, most decisions made in the construction of the model rely on the real distribution of the underlying data, by using a kernel density function over the distances between samples.
Another key feature is online applicability, meaning that samples can be processed in a stream-like fashion.

This document is structured as follows: First, the foundations are presented in chapter 2. The AnYa system itself is described in its entirety in 3. In chapter 4, the performance of the implemented system is tested on various real-life data sets and compared to those of various traditional approaches used in machine learning and related fields. Additionally, several possible extensions or modifications of AnYa are explored and evaluated as well (chapter 5). Final conclusions are drawn in chapter 6.

2 Foundations

2.1 Fuzzy sets

The AnYa system that is examined later in this thesis is proposed by authors working on Fuzzy Rule-Based Systems. A very coarse overview of the general structure of these is given in section 2.2. In this section, the foundations of fuzzy logic that are necessary to understand AnYa will be explained.

Fortunately, the only concept of fuzzy logic that is really relevant to AnYa is fuzzy membership. In traditional logic, the *characteristic function* of a set X is defined as:

$$c_X(x) = \begin{cases} 1 \text{ if } x \in X \\ 0 \text{ if } x \notin X \end{cases}$$

This is a model of a definite yes-no decision on set inclusion. The returned values are "crisp", the set of possible returned values is $\{0, 1\}$. [Zadeh, 1965] first proposed the use of fuzzy sets, which are sets defined by a *membership function*, serving a similar purpose than the characteristic function. Membership is defined as:

$$\mu_X(x) = \lambda \qquad \text{where } \lambda \in [0, 1]$$

λ is called the *degree of membership*. The interval border values zero and one can be seen as equivalent to the two possible outputs of a characteristic function, whereas everything in between is a real-valued membership that is not expressible in traditional logic. A yes-no decision on set inclusion is transformed into an unsharp response, therefore the set is "fuzzy".

2.2 Fuzzy Rule-Based Systems

Rule-based Systems are commonly used in machine learning and computational intelligence. Generally, a Rule-Based System describes an Input-Output relation as a set of conditional IF-THEN rules, consisting of an antecedent (IF) and a consequent part (THEN) (also called "Body" and "Head"). The rules directly represent the knowledge of the system. This also means that if the system is constructed to learn, the rules are updated in the learning process. Fuzzy Rule-Based Systems are a subset of Rule-Based Systems that incorporate methods of fuzzy logic. The general structure of these systems is depicted in figure 2.1.

Fuzzification is the process of obtaining fuzzy degrees of membership from crisp input values. In Fuzzy Rule-Based Systems, the degrees of membership refer to the matching of each rules antecedent to the input.

Defuzzification is used to transfer the fuzzy membership back to crisp output values. The two

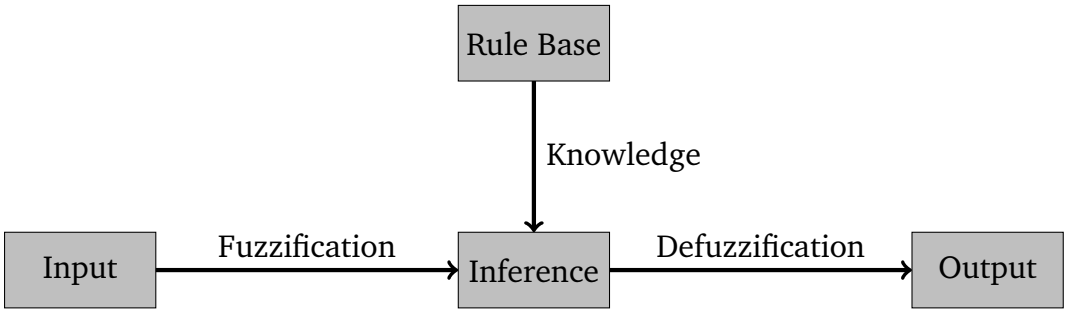


Figure 2.1.: The basic structure of a Fuzzy Rule-Based System.

most prevalent Fuzzy Rule-Based Systems used today are introduced in [Mamdani and Assilian, 1975] and [Takagi and Sugeno, 1985] respectively. Both use antecedents of the following type:

$$\text{Rule}_i : \text{IF } x_0 \text{ is } L_{i0} \text{ AND ... AND } x_n \text{ is } L_{in} \text{ THEN ...}$$

$$\text{where: } x_0, \ldots, x_n \text{ are input variables}$$
$$\text{and } L_{i0}, \ldots, L_{in} \text{ are linguistic terms}$$

Linguistic terms are expressions describing variable states of the input variables. For example, if an input variable is the temperature of water flowing through a pipe, possible linguistic terms for this input could be: *very cold, cold, lukewarm, hot, very hot*. This illustrates the inherent "fuzzyness" of rule membership and the reason for the need of fuzzification and defuzzification steps in Fuzzy Rule-Based Systems. These transformations are necessary if the system is to be used with both crisp input and output vectors.

The consequents of Mamdani-Type systems are of the same form as the antecedents:

$$\begin{aligned}\text{Rule}_i : \text{IF } x_0 \text{ is } L_{i0} \text{ AND ... AND } x_n \text{ is } L_{in} \\ \text{THEN } y_i \text{ is } L_i\end{aligned}$$

Here, L_i is also a linguistic term, which makes the necessary defuzzification apparent. Consequents of Takagi-Sugeno-Kang-Type systems are, broadly speaking, any real-valued function of the input variables. Generally they are chosen as constants or linear functions:

$$\begin{aligned}\text{Rule}_i : \text{IF } x_0 \text{ is } L_{i0} \text{ AND ... AND } x_n \text{ is } L_{in} \\ \text{THEN } y_i = \begin{bmatrix}1 & x_0 & \ldots & x_n\end{bmatrix} \pi_i\end{aligned}$$

$$\text{where } \pi_i \text{ is a parameter matrix}$$

This consequent type is also used in the particular AnYa implementation described in chapter 3.

The complete explanation of Mamdani- and Takagi-Sugeno-Kang-Type systems is omitted here, because the AnYa system is far less complex. The very coarse overview of antecedents and consequents given here is enough to observe differences and similarities with AnYa.

2.3 Recursive formula for a Cauchy-Type function

Kernels are mathematical concepts that have seen broad application in various machine learning tasks, most notably in support vector machines. They allow for high-dimensional computations to take place in lower dimensions and can serve as a measure of similarity for numerical and other data (see for example string kernels). AnYa uses a function that resembles a Cauchy-Type kernel in its antecedents (see section 3.3). A Cauchy kernel has the following general structure:

$$K(v,w) = \frac{1}{1 + \frac{\|v-w\|^2}{\sigma^2}}$$

where v, w are vectors,
σ^2 is the variance

AnYa operates on the accumulated distance between data samples, embedded in a Cauchy-Type kernel (having similar properties):

$$D^{(k)} = \frac{1}{1 + \frac{1}{k-1}\sum_{j=1}^{k-1} d(k,j)^2}$$

where k is the time step of the latest incoming data sample
and $d(k,j)$ is a distance between the data samples $x^{(k)}$ and $x^{(j)}$

Clearly, if the sample count gets large, the computation of this formula gets infeasible quickly. In this section, the recursive solution to the above formula will be derived (introduced in [Angelov and Filev, 2004a]), in which the update of D can be done in constant time (as opposed to $O(n)$, linear in the number of samples), all while using constant memory.

First, assume an euclidean distance measure:

$$d(p,q) = \sqrt{\sum_{i=1}^{l} \left(x_i^{(p)} - x_i^{(q)}\right)^2}$$

where x_i denotes the i-th component of sample x
and l is the dimension of the data samples

Substituting this in the formula and regrouping gives:

$$
\begin{aligned}
D^{(k)} &= \frac{1}{1+\frac{1}{k-1}\sum_{j=1}^{k-1}\sum_{i=1}^{l}\left(x_i^{(k)}-x_i^{(j)}\right)^2} \\
&= \frac{1}{1+\frac{1}{k-1}\sum_{j=1}^{k-1}\sum_{i=1}^{l}\left(x_i^{(k)^2}-2x_i^{(k)}x_i^{(j)}+x_i^{(j)^2}\right)} \\
&= \frac{1}{1+\frac{1}{k-1}\left(\sum_{j=1}^{k-1}\sum_{i=1}^{l}x_i^{(k)^2}-2\sum_{j=1}^{k-1}\sum_{i=1}^{l}x_i^{(k)}x_i^{(j)}+\sum_{j=1}^{k-1}\sum_{i=1}^{l}x_i^{(j)^2}\right)} \\
&= \frac{1}{1+\frac{1}{k-1}\left((k-1)x^{(k)^T}x^{(k)}-2\sum_{i=1}^{l}x_i^{(k)}\sum_{j=1}^{k-1}x_i^{(j)}+\sum_{j=1}^{k-1}x^{(j)^T}x^{(j)}\right)} \\
&= \frac{1}{1+\frac{1}{k-1}\left((k-1)x^{(k)^T}x^{(k)}-2x^{(k)^T}\sum_{j=1}^{k-1}x^{(j)}+\sum_{j=1}^{k-1}x^{(j)^T}x^{(j)}\right)} \\
&= \frac{1}{1+x^{(k)^T}x^{(k)}-\frac{2}{(k-1)}x^{(k)^T}\sum_{j=1}^{k-1}x^{(j)}+\frac{1}{(k-1)}\sum_{j=1}^{k-1}x^{(j)^T}x^{(j)}} \\
&= \frac{k-1}{(k-1)(1+x^{(k)^T}x^{(k)})-2x^{(k)^T}\sum_{j=1}^{k-1}x^{(j)}+\sum_{j=1}^{k-1}x^{(j)^T}x^{(j)}}
\end{aligned}
$$

The remaining sums in the denominator do not require iterating through all past samples and can be updated recursively after a new sample has been processed. This has the advantage that updates of D can be done in constant time and past samples do not have to be stored in memory (only the current value of the sum).
D is called *density* in AnYa and the explicit recursion and application is shown and discussed in section 3.3.

There are recursive variants for other distance measures, for example for the cosine distance ([Angelov and Zhou, 2008]), which are not discussed any further in this thesis.

2.4 Fuzzily-Weighted Recursive Least-Squares Algorithm

The parameters of the AnYa system are found by an algorithm called *Fuzzily-Weighted Recursive Least-Squares*, which is explained in this section. The basis of this approach is the Recursive Least-Squares Algorithm (RLS) ([Plackett, 1950], presumably already introduced by Gauss), which is a well-known method to find a suitable parameter matrix θ for linear relations between vectors (for example input and output vectors). The advantage over other approaches is the possibility to update a parameter matrix online based on the last seen sample.
A linear input-output relation can be defined as follows:

$$x^T\theta = y$$

where

$x \in \mathbb{R}^{p,1}$ is a single input vector $:= \begin{bmatrix} x_1 & \dots & x_p \end{bmatrix}^T$

$y \in \mathbb{R}^{1,q}$ is a single output vector $:= \begin{bmatrix} y_1 & \dots & y_q \end{bmatrix}$

$\theta \in \mathbb{R}^{p,q}$ is a parameter matrix

The general problem that needs to be solved can be written as a linear equation system dependent on all K read samples:

$$X^T\theta = Y$$

where

X is a matrix of all input vectors x

$$:= \begin{bmatrix} x_1^{(1)} & \dots & x_p^{(1)} \\ \vdots & \vdots & \vdots \\ x_1^{(K)} & \dots & x_p^{(K)} \end{bmatrix}^T$$

Y is a matrix of all output vectors y

$$:= \begin{bmatrix} y_1^{(1)} & \dots & y_q^{(1)} \\ \vdots & \vdots & \vdots \\ y_1^{(K)} & \dots & y_q^{(K)} \end{bmatrix}$$

The least-squares solution is given by:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{2}(Y - X^T\theta)^T(Y - X^T\theta)$$

If we differentiate the squared error in respect to θ and set it to zero to find the minimum of its parabola, we get the explicit form of the least-squares solution:

$$\frac{\partial \frac{1}{2}(Y - X^T\theta)^T(Y - X^T\theta)}{\partial \theta} = -X(Y - X^T\theta) \qquad | \stackrel{!}{=} 0$$

$$\Rightarrow -XY + XX^T\hat{\theta} = 0$$

$$XX^T\hat{\theta} = XY$$

$$\hat{\theta} = (XX^T)^{-1}XY$$

To get a recursive solution, we first split X and Y in old and new data:

$$X^{(k+1)} = \begin{bmatrix} X^{(k)} & \begin{matrix} x_1^{(k+1)} \\ \dots \\ x_p^{(k+1)} \end{matrix} \end{bmatrix} = \begin{bmatrix} X^{(k)} & x^{(k+1)} \end{bmatrix}$$

$$Y^{(k+1)} = \begin{bmatrix} \begin{matrix} Y^{(k)} \\ y_1^{(k+1)} \end{matrix} & \begin{matrix} \\ \dots \end{matrix} & \begin{matrix} \\ y_q^{(k+1)} \end{matrix} \end{bmatrix} = \begin{bmatrix} Y^{(k)} \\ y^{(k+1)} \end{bmatrix}$$

We rewrite the parameter matrix as a temporary solution at time step k and fill in X and Y at time step $k+1$:

$$\begin{aligned}
\theta^{(k)} &= (X^{(k)}X^{(k)^T})^{-1}X^{(k)}Y^{(k)} \\
\theta^{(k+1)} &= (X^{(k+1)}X^{(k+1)^T})^{-1}X^{(k+1)}Y^{(k+1)} \\
&= \left(\begin{bmatrix} X^{(k)} & x^{(k+1)} \end{bmatrix}\begin{bmatrix} X^{(k)^T} \\ x^{(k+1)^T} \end{bmatrix}\right)^{-1}\begin{bmatrix} X^{(k)} & x^{(k+1)} \end{bmatrix}\begin{bmatrix} Y^{(k)} \\ y^{(k+1)} \end{bmatrix} \\
&= \left(\underbrace{X^{(k)}X^{(k)^T}}_{:=C^{(k)^{-1}}} + x^{(k+1)}x^{(k+1)^T}\right)^{-1}\left(X^{(k)}\underbrace{Y^{(k)}}_{=X^{(k)^T}\theta^{(k)}} + x^{(k+1)}y^{(k+1)}\right) \\
&= \left(C^{(k)^{-1}} + x^{(k+1)}x^{(k+1)^T}\right)^{-1}\left(C^{(k)^{-1}}\theta^{(k)} + x^{(k+1)}y^{(k+1)}\right)
\end{aligned}$$

Using the matrix inversion lemma ([Woodbury, 1950]), the updates of θ and the covariance matrix C can be simplified significantly (see appendix chapter A):

$$\begin{aligned}
\theta^{(k+1)} &= \theta^{(k)} + \frac{C^{(k)}x^{(k+1)}}{1 + x^{(k+1)^T}C^{(k)}x^{(k+1)}}\left(y^{(k+1)} - (x^{(k+1)})^T\theta^{(k)}\right) \\
C^{(k+1)} &= C^{(k)} - \frac{C^{(k)}x^{(k+1)}x^{(k+1)^T}C^{(k)}}{1 + x^{(k+1)^T}C^{(k)}x^{(k+1)}}
\end{aligned}$$

The lemma allows to replace the matrix inversion in both updates with a scalar inversion, making the updates easier and more robust.
Using these two formulas, a given parameter matrix can be consecutively updated on arrival of new samples. At the start of the training process, θ can be initialized as a matrix containing only zeros, C as an identity matrix multiplied by a large positive number (the higher this number, the higher the confidence that θ is not all-zeros). Another possible initialization is to use the first pq samples to non-recursively find the unique $\theta^{(pq)}$ and $C^{(pq)}$, starting the recursive update at time step $pq+1$.

The Fuzzily-Weighted Recursive Least-Squares Algorithm is a special case of the standard setup, where the vector x is a collection of fuzzily-weighted input vectors. Assume the following setup:

$$\begin{aligned}
x, y &: \text{ input/output vectors (as before)} \\
\lambda_1, \ldots, \lambda_N; \sum_{i=1}^{N}\lambda_i = 1 &: \text{ fuzzy degrees of membership of a given input to } N \text{ rules} \\
y_i = x^T\pi_i &: \text{ output predicted by i-th rule} \\
y = \sum_{i=1}^{N}\lambda_i y_i &: \text{ output predicted by system}
\end{aligned}$$

This setup essentially describes the commonly used inference in Takagi-Sugeno-Kang-Type or AnYa-Type systems. It is not apparent that the recursive least-squares algorithm is applicable here, without rewriting the predicted output equation:

$$y = \sum_{i=1}^{N} \lambda_i y_i = \sum_{i=1}^{N} \lambda_i x^T \pi_i$$
$$= \underbrace{\begin{pmatrix} \lambda_1 x^T & \lambda_2 x^T & \dots & \lambda_N x^T \end{pmatrix}}_{\psi^T} \underbrace{\begin{pmatrix} \pi_1^T & \pi_2^T & \dots & \pi_N^T \end{pmatrix}^T}_{\theta}$$

This fits the linear relation equation $y = x^T\theta$, for which we deduced the recursive least-squares algorithm. Applying RLS to an equation $y = \psi^T\theta$, where ψ^T is a matrix of fuzzily-weighted input vectors has been named *Fuzzily-Weighted Recursive Least-Squares* (fwRLS) by [Angelov, 2010] (introduced in this form by [Angelov and Filev, 2004a]).

The main concept to take from this section is the method to recursively find parameters of linear relations and the possibility to extend this approach to fuzzy systems by rewriting the system output equation. It should be noted that the given example assumes a single parameter matrix for the whole system. Even though this approach is also used in the AnYa system described in chapter 3, other approaches are possible. These and other intricacies of the application of fwRLS (for example the effects of rule removal/creation) are discussed in detail in the respective sections.

3 The AnYa Fuzzy Rule-Based System

This chapter is about the Fuzzy Rule-Based inference system AnYa. The main reference will be [Angelov and Yager, 2011] (the name "AnYa" is derived from the authors last names). There are also many past publications of Prof. Angelov and others incorporated in this system, which are cited in the relevant sections. In the rest of this chapter, the following notations are used:

$n \in \mathbb{R}$	:	dimension of the input vector
$m \in \mathbb{R}$	:	dimension of the output vector
$x \in \mathbb{R}^{n,1}$	:	input vector
$y \in \mathbb{R}^{1,m}$	:	output vector
$x_e^T \in \mathbb{R}^{1,n+1}$	:	$\begin{pmatrix}1 & x_1 & \dots & x_n\end{pmatrix}$, extended input
$N \in \mathbb{N}$	:	the total number of rules in the system

3.1 Motivation

The antecedents of the Mamdani-Type and the Takagi-Sugeno-Kang-Type systems (depicted in section 2.2) have their advantages: expert knowledge is easy to incorporate and the conditions can be verbalized well. There is a lot of research done in search of ways to create and evolve rules for these type of systems. However, there are also various apparent problems. The antecedents are parametrized which makes the search for good values for these one of the main tasks and introduces further uncertainty. Also, even if the rules are expressive for humans, they can be too complex (because of the conjunctions). Simpler, similar performing methods may exist. AnYa proposes the use of recursive kernel-driven densities for its antecedents, which are free of parameters.

3.2 Basic structure

The AnYa system, as described in [Angelov and Yager, 2011], is mainly built on the idea of using a novel type of antecedent. Other than the ones briefly described in chapter 2.2, this one does not need predefined fuzzy membership functions, linguistic terms or logical connectors between antecedent terms. More importantly, the suggested approach does not rely on the estimation of antecedent parameters (it has none), therefore making design, implementation and learning process easier than in common Fuzzy Rule-Based Systems.
The structure of single rules is shown in figure 3.1.
"x is like $cloud_i$" describes the membership of the input vector x to the cloud $cloud_i$ (see section 3.3). Based on this information, every rule is associated with a percentual "firing level" λ_i. Finding these values is a key problem that is tackled by Fuzzy Rule-Based Systems. It will later be shown that the antecedent is in his structure not as complex as in other approaches (there

$$\text{Rule}_i : IF\ x \text{ is like } cloud_i$$
$$THEN\ y_i = x_e^T \pi_i$$

Figure 3.1.: The structure of rules in an AnYa system.

are no long conjunctions of single conditions).
The consequent is the same as in Takagi-Sugeno-Kang-Type systems, although any kind of consequent is applicable in AnYa systems. Here, $\pi_i \in \mathbb{R}^{n+1,m}$ is a matrix of parameters for the i-th rule. It is updated in the learning process (see section 3.4).
There are also different possibilities to define the general output of the system. A "Winner-Takes-All" approach ($y = y_i$ is the output of Rule_i with the highest firing level λ_i) is possible, but not practical for regression (the resulting function would not be very smooth). AnYa for regression uses the "Fuzzily-Weighted-Average" approach (also used in Takagi-Sugeno-Kang-Type systems):

$$y := \sum_{i=1}^{N} \lambda_i y_i$$

The AnYa system described in the form of a general Fuzzy Rule-Based System (see figure 2.1) is depicted in figure 3.2.

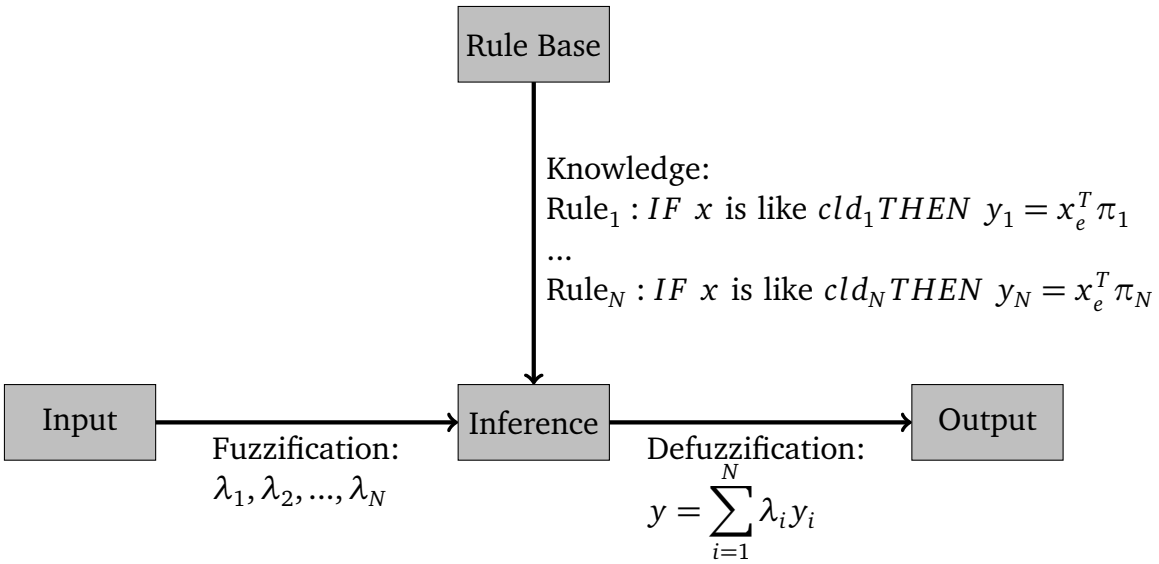


Figure 3.2.: A graphical representation of the AnYa system.

3.3 Antecedents

3.3.1 Local density

AnYa uses non parametric antecedents based on *clouds* of data. A cloud is an accumulation of samples that previously produced their highest firing level at a particular rule. Every rule is associated with a single cloud and every sample is a member of one cloud.
A straight-forward way to describe how well a new sample "fits" a cloud is to measure the distances to the samples that are already associated with the cloud. The *local density* is exactly this, defined as a kernel over the sample distances.

$$\gamma_i^{(k)} = K\left(\sum_{j=1}^{M_i} d_i(k,j)\right)$$

Figure 3.3.: The definition of *local density*

In the definition (figure 3.3), $K(...)$ is an appropriate kernel function, M_i is the number of samples already linked to the i-th cloud and $d_i(k,j)$ is an appropriate distance measure (euclidean, cosine, etc.) between the k-th sample (the currently observed sample) $x^{(k)}$ and the j-th sample of the i-th cloud. This is in fact enough to define a meaningful "firing level" λ for each cloud (figure 3.4).

$$\lambda_i^{(k)} = \frac{\gamma_i^{(k)}}{\sum_{j=1}^{N} \gamma_j^{(k)}}$$

Figure 3.4.: The definition of the degree of rule membership.

It is apparent that this value is bigger than or equal to zero (the distances are non-negative) and smaller than or equal to one (all λ together equal one). It can be regarded as the percentage of how well a sample fits the antecedent of a rule. This definition is also what makes the "fuzzyness" of the AnYa system, because a sample is member of every rule, but with different degree.

Regarding the kernel used for *local density*, Angelov and Yager suggest that a Cauchy kernel is well-suited. In section 2.3 it is derived how a Cauchy-Type kernel over euclidean distances can be recursively updated to compute the distance to all previous samples in constant time and without storing past samples. The formula shown in that section is the one used in AnYa, with the explicit recursion shown in figure 3.5. On arrival of a new sample, the local density is computed for every rules' cloud and the variables necessary for the recursive computation are updated only for the cloud that the sample is associated with (if it did not lead to the creation of a new rule).

The reason to use this exact approach might not be clear (the recursive accumulated distance can be incorporated in other ways). Mostly, it is because the resulting formula is meaningful as a measure of density, because the computed value gets closer to zero for increasing distances

$$\gamma_i^{(k)} = \frac{1}{1 + \frac{1}{M_i}\sum_{j=1}^{M_i} d_i(k,j)^2} = \frac{M_i}{M_i(x^{(k)^T}x^{(k)} + 1) - 2\alpha^{(k)} + \beta^{(k)}}$$

$$\text{with } \alpha^{(k)} = x^{(k)^T}\xi^{(k)}$$
$$\xi^{(k)} = \xi^{(k-1)} + x^{(k-1)} \qquad \xi^{(0)} = 0$$
$$\beta^{(k)} = \beta^{(k-1)} + x^{(k-1)^T}x^{(k-1)} \qquad \beta^{(0)} = 0$$

Figure 3.5.: The definition of local density based on a Cauchy-Type kernel and its recursive computation.

and closer to one if the distances get smaller. It is monotonic and well-defined even when all distances equal zero. Another advantage is that there is no need to approximate kernel bandwidths or variances this way, since this similarity measure is solely based on the real data samples in each cloud.

3.3.2 Global density

For rule creation, the *global density* is a defining value. It is not part of the antecedent, but uses the same approach. Global density is defined almost the same way as local density except it is a measure of similarity of **all** past samples and the current sample.

$$\Gamma^{(k)} = K\left(\sum_{j=1}^{k-1} d(k,j)\right)$$

Figure 3.6.: The definition of *global density*.

When using a Cauchy kernel with euclidean distances, global density is also recursively computable (figure 3.7).
The global density is defined on a vector concatenating input and output, which makes it possible to differentiate samples based not only on input but also on output vectors. Because the global density is only necessary for training (where the output is given), this is applicable.
If the local density is regarded as the similarity of a sample in regard to a cloud, then the global density is a measure of how different a sample is in comparison to all samples that are already processed. As such it is a valuable information on the overall distribution of samples and is used in the process of rule creation (see section 3.5).

$$\Gamma^{(k)} = \frac{1}{1 + \frac{1}{k-1}\sum_{j=1}^{k-1} d(k,j)^2} = \frac{k-1}{(k-1)({z^{(k)}}^T z^{(k)} + 1) - 2A^{(k)} + B^{(k)}}$$

$$\text{with } A^{(k)} = {z^{(k)}}^T \Xi^{(k)}$$

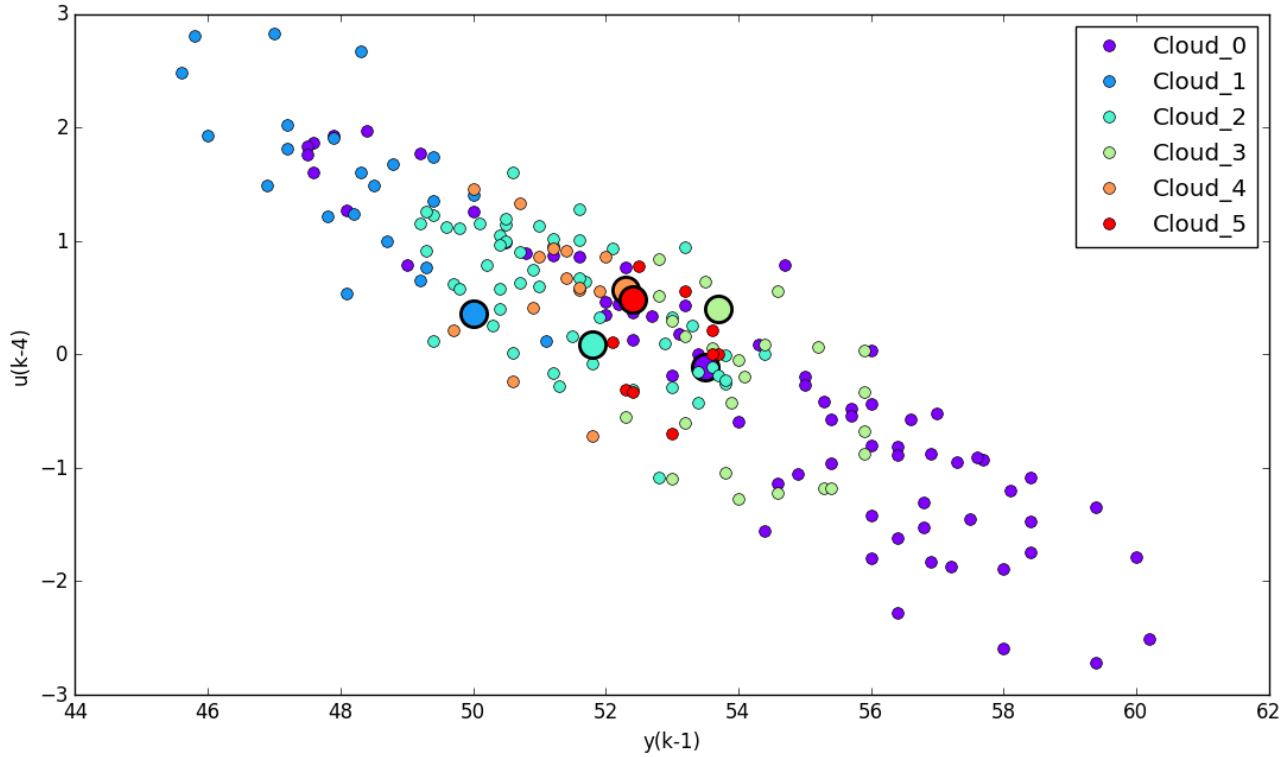
$$\Xi^{(k)} = \Xi^{(k-1)} + z^{(k-1)} \qquad \Xi^{(0)} = 0$$

$$B^{(k)} = B^{(k-1)} + {z^{(k-1)}}^T z^{(k-1)} \qquad B^{(0)} = 0$$

Figure 3.7.: The definition of global density based on a Cauchy-Type kernel and its recursive computation. $z^{(k)}$ is the vector resulting from vertically concatenating input $x^{(k)}$ and output ${y^{(k)}}^T$.

3.3.3 Illustration

To visualize the resulting partitioning of the input space using antecedents based on recursive local density, the AnYa system is trained on the Box-Jenkins data set ([Box and Jenkins, 1970]). The resulting clouds and an idea of the rules' antecedents is shown in figure 3.8. The creation of rules is explained in section 3.5. Another plot that better shows the partitioning is shown in figure 3.9. It was created by training AnYa on artificial data with randomly distributed input samples. The resulting clouds resemble a Voronoi diagram of the input space (partitioning by boundaries based on the farthest points that are closest to defined center points), but with overlapping boundaries. Both models were modified to save the samples they were fed. Normally the samples would not be saved and only the variables necessary for density calculation are updated.



IF x is like $cloud_0$, i.e. around [[53.5 -0.109]] THEN...
IF x is like $cloud_1$, i.e. around [[50.0 0.360]] THEN...
IF x is like $cloud_2$, i.e. around [[51.8 0.088]] THEN...
IF x is like $cloud_3$, i.e. around [[53.7 0.403]] THEN...
IF x is like $cloud_4$, i.e. around [[52.3 0.566]] THEN...
IF x is like $cloud_5$, i.e. around [[52.4 0.484]] THEN...

Figure 3.8.: Cloud membership of the first 196 samples of the Box-Jenkins data set and the extracted rule antecedents, illustrated by the initial sample of the respective cloud. Samples assigned to the same cloud have the same color. Enlarged are the initial points of all clouds.

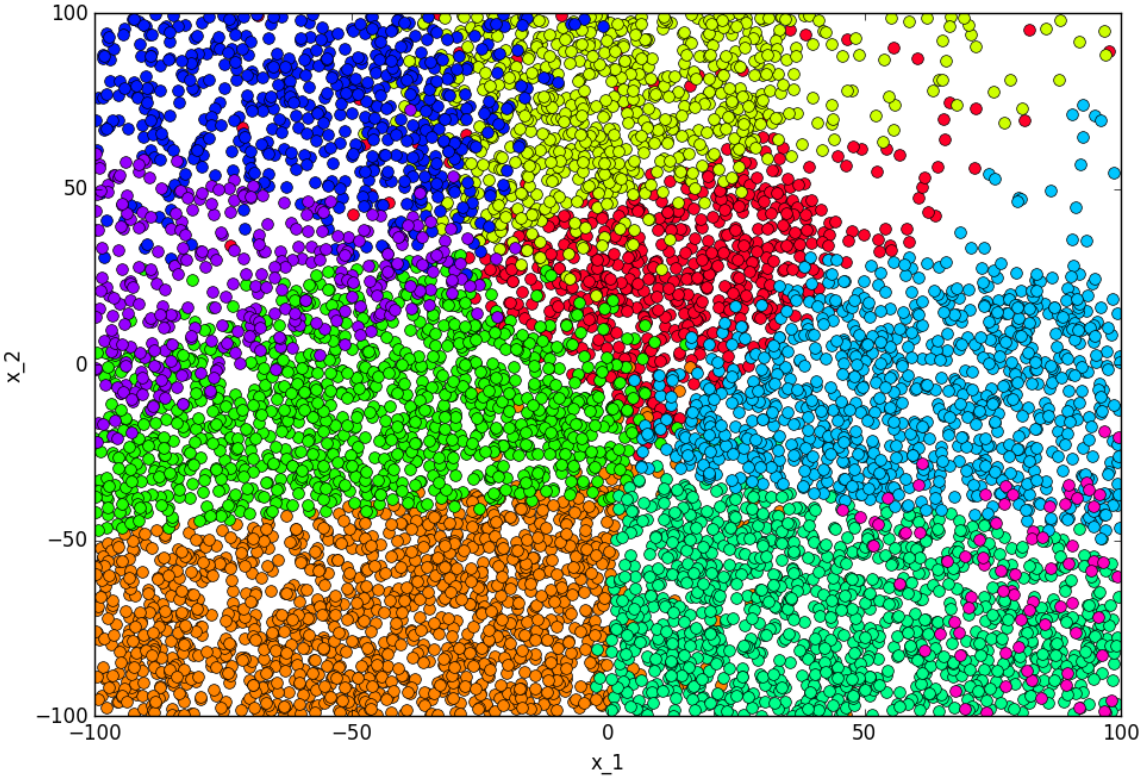


Figure 3.9.: A hypothetical example of a two dimensional input space consisting of 10000 samples distributed uniformly random in [-100,100]x[-100,100]. Again, samples of the same cloud are equally colored. The approximated function was $f(x_1, x_2) = x_1^2 + x_2^2$. The sparseness of samples (e.g. in the top-right corner) is due to recent removal of a rule and its associated samples (see section 3.6).

3.3.4 Properties of clouds

As illustrated in section 3.3.3 the data clouds built on local and global density have no specific shape that future samples are forced into. In comparison to circular, Gaussian and other membership functions, no parameters are necessary. There is also no loss of knowledge on the samples, because local density exactly represents the distance to other samples, whereas in other approaches, the exact sample values are lost because they just update a proposed (ideal) parametrized distribution.

3.4 Consequents

3.4.1 Structure

The consequents of AnYa rules can be of any type, but in many scenarios a linear consequent with parameter matrix $\pi \in R^{n+1,m}$ is appropriate. The output of the i-th rule is then defined as:

$$y_i = x_e^T \pi_i$$
where x_e^T is the extended input vector $x_e^T = \begin{pmatrix} 1 & x_1 & x_2 & \dots & x_n \end{pmatrix}$

The extension of the input is necessary to add constant values to the output. Although the AnYa system can be used for non-linear regression, the consequents are locally linear. In figure 3.10, an example of a rule consequent is depicted.

$$y_i = x_e^T \begin{pmatrix} 3 \\ 4 \\ 5 \end{pmatrix} = 3 + \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} 4 \\ 5 \end{pmatrix}$$

Figure 3.10.: A rule consequent example for input dimension $n = 2$ and output dimension $m = 1$.

3.4.2 Parameter Learning

There are many different ways to find parameters that are suitable for the overall output. Two generally different methods are individual and joint optimization. The authors of AnYa sometimes call these “local” and “global” optimization, which might easily be confused with local and global optima. When using individual optimization, one would observe the error of single rule outputs and try to minimize that value. This leads to small errors of single rules, but does not necessarily guarantee a small error in the output of the whole system, which is a fuzzily-weighted average over all rule outputs. When optimizing jointly, all π_i are stacked and regarded as a compound matrix of parameters. The matrix is then adjusted to produce a small system error. This however diminishes the expressiveness of single rules ([Yen et al., 1998]). Which

optimization to use could also be described as a decision between description and prediction. Individual optimization yields rules that better show how a function behaves in certain parts of the input space (the system is more descriptive), whereas joint optimization is possibly better suited to predict new outcomes of future inputs (the system is more predictive), because the system error is smaller. In the implementation for this thesis, the joint optimization approach is used, since it is more appropriate for producing results on regression problems that are comparable to those of other techniques.
When using joint optimization techniques, it is convenient to redefine the system output as show in figure 3.11.

$$
\begin{aligned}
y = \sum_{i=1}^{N} \lambda_i y_i &= \sum_{i=1}^{N} \lambda_i x_e^T \pi_i \\
&= \underbrace{\begin{pmatrix} \lambda_1 x_e^T & \lambda_2 x_e^T & \dots & \lambda_N x_e^T \end{pmatrix}}_{\psi^T} \underbrace{\begin{pmatrix} \pi_1^T & \pi_2^T & \dots & \pi_N^T \end{pmatrix}^T}_{\theta}
\end{aligned}
$$

Figure 3.11.: An alternative way to define the inference output.

This simplifies the inference process to building $\psi^T \in \mathbb{R}^{1,N(n+1)}$ by weighting the extended input using the (non-parametric) antecedents and computing a dot product with θ. θ is a matrix in $\mathbb{R}^{N(n+1),m}$ containing parameters that have to be learned. With these simplification, the definition of the "least squares" system error is straight-forward:

$$
\begin{aligned}
err(\theta) &= (Y - \psi^T \theta)^T (Y - \psi^T \theta) \\
&\text{where } Y \text{ is the correct output of the system.}
\end{aligned}
$$

The Fuzzily-Weighted Recursive Least-Squares algorithm introduced in section 2.4 can then be used to update the system parameters (figure 3.12). The slight difference from the original algorithm is due to a rewrite shown in appendix B.

In the definition, (k) denotes the time index in which the associated term is set (i.e. $\theta^{(k)}$ is the parameter matrix optimizing the inference output after k samples have been read). The correction factor in fwRLS contains the matrix $\psi^{(k)}$, which is depend on the rule firing levels λ. This is what makes the method "fuzzily-weighted". C is a covariance matrix and is initialized with variances Ω. Ω has to be positive (so the covariance stays positive definite) and can be seen as the confidence that $\theta^{(1)} \neq 0$. A bigger value therefore ensures that corrections are applied in a higher extent.

AnYa is well suited for online application, because the training samples can both be provided as a batch during initialization, as well as anytime during the operation of the system. More specifically, if an incoming sample contains an observed output, this sample can be used to update the parameter matrix θ. Otherwise, the system infers the output based on the latest θ and the current input.
There are some special cases concerning consequent updates that are caused by rule creation and rule maintenance, which are explained in the respective sections 3.5 and 3.6.

$$\theta^{(k)} = \theta^{(k-1)} + \overbrace{C^{(k)}\psi^{(k)}}^{\text{correction factor}} \left(\overbrace{Y^{(k)}}^{\text{new observation}} - \overbrace{\psi^{(k)^T}\theta^{(k-1)}}^{\text{prediction with old } \theta} \right)$$

$$\theta^{(1)} = 0$$

$$C^{(k)} = C^{(k-1)} - \frac{C^{(k-1)}\psi^{(k)}\psi^{(k)^T}C^{(k-1)}}{1 + \psi^{(k)^T}C^{(k-1)}\psi^{(k)}}$$

$$C^{(1)} = \Omega\mathbb{I}$$

$\mathbb{I}$ is the identity matrix in $\mathbb{R}^{N(n+1),N(n+1)}$
and Ω is a large positive number (e.g. 1000).

Figure 3.12.: The fuzzily-weighted Recursive Least Squares method (fwRLS) used in AnYa.

3.4.3 Illustration

The Box-Jenkins data set is used to illustrate the consequents of extracted rules (figure 3.13).

IF x is like $cloud_0$, i.e. around [[53.5 -0.109]]
THEN y_0 = 26.313 + 0.505 x_1 + -1.491 x_2

IF x is like $cloud_1$, i.e. around [[50.0 0.360]]
THEN y_1 = 33.915 + 0.344 x_1 + -1.412 x_2

IF x is like $cloud_2$, i.e. around [[51.8 0.088]]
THEN y_2 = 16.051 + 0.705 x_1 + -1.207 x_2

IF x is like $cloud_3$, i.e. around [[53.7 0.403]]
THEN y_3 = 18.202 + 0.655 x_1 + -1.151 x_2

IF x is like $cloud_4$, i.e. around [[52.3 0.566]]
THEN y_4 = 22.366 + 0.578 x_1 + -1.168 x_2

IF x is like $cloud_5$, i.e. around [[52.4 0.484]]
THEN y_5 = 22.868 + 0.573 x_1 + -1.019 x_2

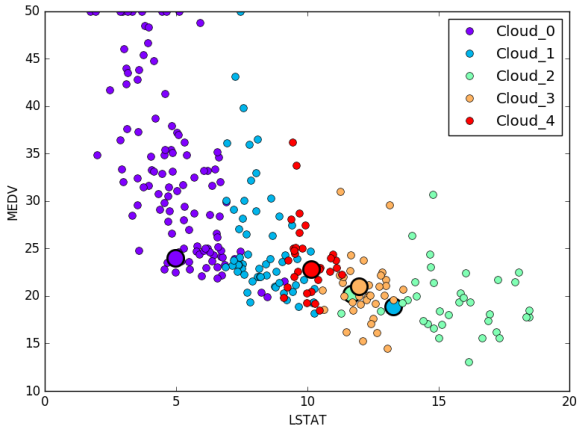
Figure 3.13.: The complete rules extracted from the first 196 samples of the Box-Jenkins data set using the input [[y(k-1) , u(k-4)]].

To visualize some resulting functions, AnYa was trained on the Boston housing data set and the Istanbul stock exchange data set ([Akbilgic et al., 2014]), both taken from the UCI Machine Learning Repository ([Lichman, 2013]).
For the housing data set, the input variable is the percent of lower-income citizens per town. The median of housing prices in the region was used as dependent variable (output). The resulting

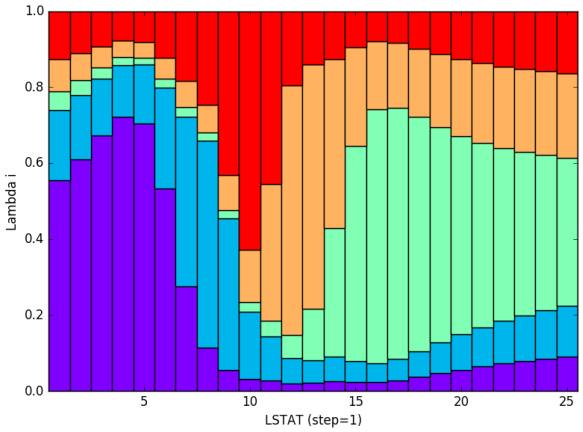
data clouds are depicted in figure 3.14 (a). (b) shows the memberships of $x \in [1, 2, \ldots, 25]$ to these clouds. The same x are used for inference and the returned values are plotted in (c). Additionally, the local functions (defined by π_i for rule i) are plotted (sub figure (d)), each in the interval of x, where rule i has the highest membership. This highlights how the parameters are not optimized based on local error. This should also visualize, how the linear consequents of each rule are combined into the function show in sub figure (b), by weighting the respective rule outputs by their membership (or firing level) λ_i. For example, if the input is $x = 25$, the membership values are (rounded):

$$\lambda_0 = 0.09 \quad \lambda_1 = 0.13 \quad \lambda_2 = 0.39 \quad \lambda_3 = 0.22 \quad \lambda_4 = 0.17$$

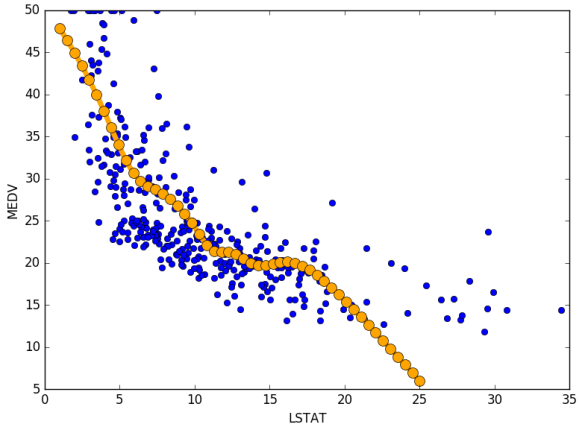
This results in the inferred value of about 6, shown in (b). Intuitively, it should be higher, i.e. λ_2 should be well above 40 percent. This is because the created clouds do not include samples in the interval $x \in [20, 35]$, because the rule including these in its cloud was removed in the training process (see section 3.6). Also, if they were not, the phenomenon occurring at the edges of the training sample space (see sub figure (b) from $x = 18$ to $x = 25$]), where the memberships slowly start to equalize would not have been totally prevented by this.



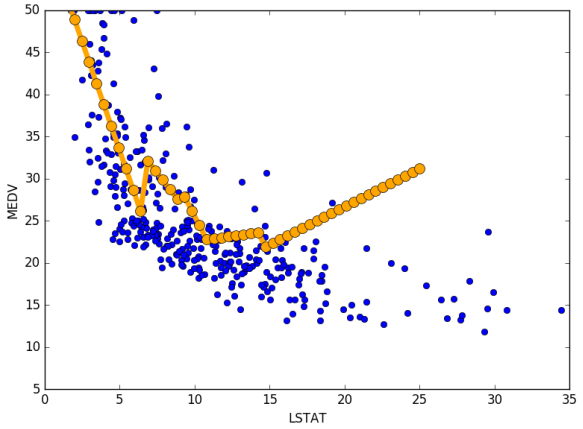
(a) Created clouds.



(b) Membership.



(c) Training data and obtained function.



(d) Visualization of local linear functions.

Figure 3.14.: The results of training on the housing data set. Note the scales.

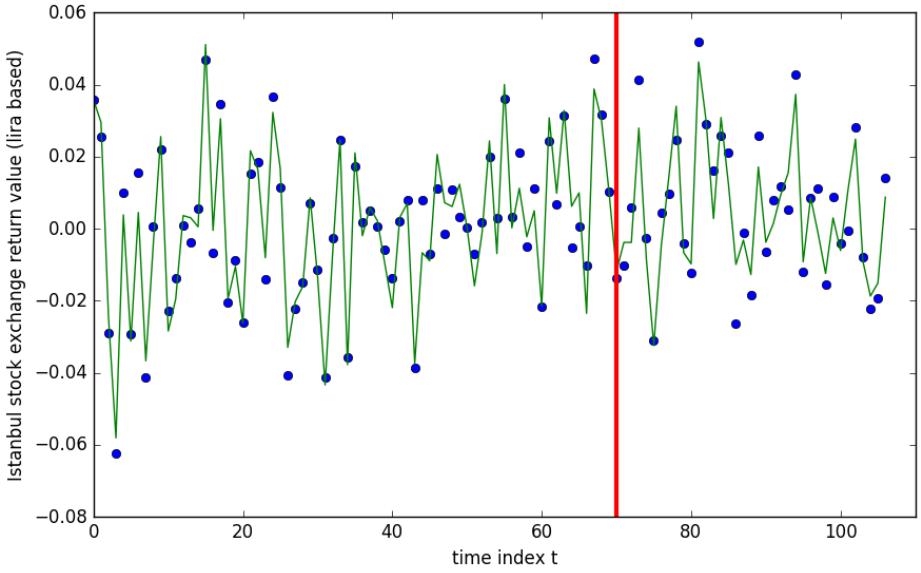


Figure 3.15.: The results of training and testing on the Istanbul stock exchange data set. The red line is the boundary between training and test data.

The Istanbul stock exchange data is sequential (one row per day, sorted by day). It is displayed as a time series of output values (return value of the stock exchange based on lira). The input is a 7-dimensional vector of return values of other stock exchanges and relevant market indices. The training and test data plus the predicted values are depicted in figure 3.15.

3.5 Rule Creation

The creation of new rules in a rule-based system is an important task, because the number of rules and their significance influence the performance and applicability of the model. Desirable is a small number of rules where no rule is irrelevant. In contrast to other rule-based systems, AnYa systems always evaluate every rule antecedent (compute the local density). This means a greater number of rules can also have a performance impact. Rule creation in AnYa is done both in the case of already existing rules (created by expert knowledge or previous studies etc.) and the case of an empty rule base. It should be mentioned that for some scenarios, the creation of additional rules is not recommended. For example, in a classification task, a fixed number of rules (one for each class) is reasonable, both in regard to performance and expressiveness.

In the case of an AnYa system with an evolving structure, there are two conditions that decide whether or not a new rule is created when a sample is processed (figure 3.16). The first condition tests if the new sample is a good generalization in regard to the overall distribution of samples. The second condition ensures that there is no cloud “near” the sample that produces a local density higher than e^{-1}. Note that this condition produces different results on differently scaled samples, since the formula for local density includes the average squared distances to other samples (which changes with the scale). In newer publications, there are other conditions used instead of the second one.

$$\forall i \in \{1,...,N\}: \qquad \Gamma^{(k)} > \Gamma_i \tag{1}$$

$$\forall i \in \{1,...,N\}: \qquad \gamma_i^{(k)} <= e^{-1} \tag{2}$$

Figure 3.16.: The two conditions necessary for rule creation. Γ_i is the global density at the initial sample of the i-th cloud.

If a new rule with a new cloud is created, there are several variables that need to be set to account for the extended system structure. First of all, the input that led to the rule creation is saved, along with its output. This is needed for the rule creation condition (see figure 3.16, condition one). This is also the only sample that is explicitly saved, all other associated samples just update the density related variables of the data cloud. This is essentially all that has to be done regarding the antecedents. There are no parameters to approximate and no focal points to declare.
The consequents, however, need more drastic adjustments. Generally, a rule's consequent is dependent on a parameter matrix π. In section 3.4, it is described why these matrices are regarded as one big matrix of system parameters and how the resulting matrix θ is found. In the definition of the recursive optimization procedure in figure 3.12, it can be seen that both θ and the covariance matrix C are to be updated.
θ is a matrix in $\mathbb{R}^{N(n+1),m}$. If a rule is created, N is incremented and a new matrix $\pi_{N+1} \in \mathbb{R}^{(n+1),m}$ is appended to it. Remembering the Fuzzily-Weighted Recursive Least Squares approach, one could initialize π_{N+1} the same as $\theta^{(1)}$, namely as a matrix containing only zeros. However, this method disregards all knowledge of the system that has been learned before. In some scenarios this might even be desirable, so an unwanted bias is not imposed on the learning process. In many cases, θ is used to initialize π_{N+1} in a meaningful way and reduce the system error ([Angelov and Filev, 2004a]). The idea is very straight-forward and is similar to other fuzzy concepts used in AnYa and traditional Fuzzy Rule-Based Systems: Create a new local parameter matrix as a weighted approximation based on the "firing levels" and the existing matrices (see figure 3.17).

$$\pi_{N+1} = \sum_{i=1}^{N} \lambda_i \pi_i$$

Figure 3.17.: The extension of the parameter matrix θ (expressed in local parameter matrices π_i)

π_i is the part of θ that is responsible for the output of the i-th rule (after all, θ is defined as the vertical concatenation of the parameter matrices of all N rules, see figure 3.11).

For the covariance matrix C, [Angelov and Filev, 2004a] suggest an extension that is more elaborate (figure 3.18). The extension part of the covariance is initialized in the normal way as a fitting identity matrix times Ω. The values of the old covariance matrix are still used in the

respective fields, but they do not encode an accurate representation of the covariance anymore. They are multiplied by $\rho = \frac{N^2+1}{N^2}$ to compensate. More specifically, this should approximate the effect that the new rule would have had on the covariance, had it been present right from the start. [Angelov and Filev, 2004b] showed that this would not have changed much of the existing covariances (even less so for increasing N). These observations are encoded in the updating factor ρ.

$$C^{(k+1)} = \begin{pmatrix} & & & 0 & \dots & 0 \\ & C^{(k)}\rho & & 0 & \dots & 0 \\ & & & 0 & \dots & 0 \\ 0 & 0 & 0 & \Omega & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \Omega \end{pmatrix}$$

where Ω is a large positive number

$$\rho = \frac{N^2+1}{N^2}$$

Figure 3.18.: The extension of the covariance matrix C.

3.6 Rule Maintenance

A system that depends on a fixed rule base is rarely the best choice for many types of problems (except e.g. classification). Preferably, the system should adapt to changes in the training samples and incorporate new knowledge in the rule base. This means new rules need to be created and old ones are to be deleted. How new rules are created in AnYa is discussed in section 3.5. Removing rules that are no longer needed should keep the rule base expressive for humans, performant and less prone to over-fitting. In this section, two heuristics that can be used online to determine how valuable single rules are for a system will be presented. Both are taken from [Angelov and Yager, 2012].

3.6.1 Utility

A very simplistic rule of thumb to describe rule usefulness is: If a rule is used often, it is valuable. Based on this idea, there are many ways to quantify the usage of single rules. *Utility* is a measure that depends on values that are already computed during system runtime, namely the "firing levels" λ. To recall, λ_i is a percentual (fuzzy) value of how well an input sample fits the antecedent of rule i and is also used to accumulate the outputs of all rules into a best fitting single output vector. Intuitively, a rule that produces small λ for nearly all input samples can not be useful for the system. Utility is defined as the mean of all "firing levels" that were produced for this rule during its lifetime (figure 3.19).

$$Util_i = \frac{1}{k - I_i + 1} \sum_{l=I_i}^{k} \lambda_i^{(l)}$$

where $I_i \in \{0, \ldots, k\}$ is the time step of creation of rule i

Figure 3.19.: The definition of rule utility.

For this simple measure, [Angelov and Yager, 2012] suggest an equally simple condition for rule removal: Remove all rules whose current utility is smaller than a constant ϵ (see figure 3.20). A visualization of the development of utility during runtime is given in figure 3.21.

remove rules $i, i \in \{1, ..., N\} : Util_i < \epsilon$
where $\epsilon = 0.1$ ([Angelov and Yager, 2012])

Figure 3.20.: The condition for rule removal based on utility.

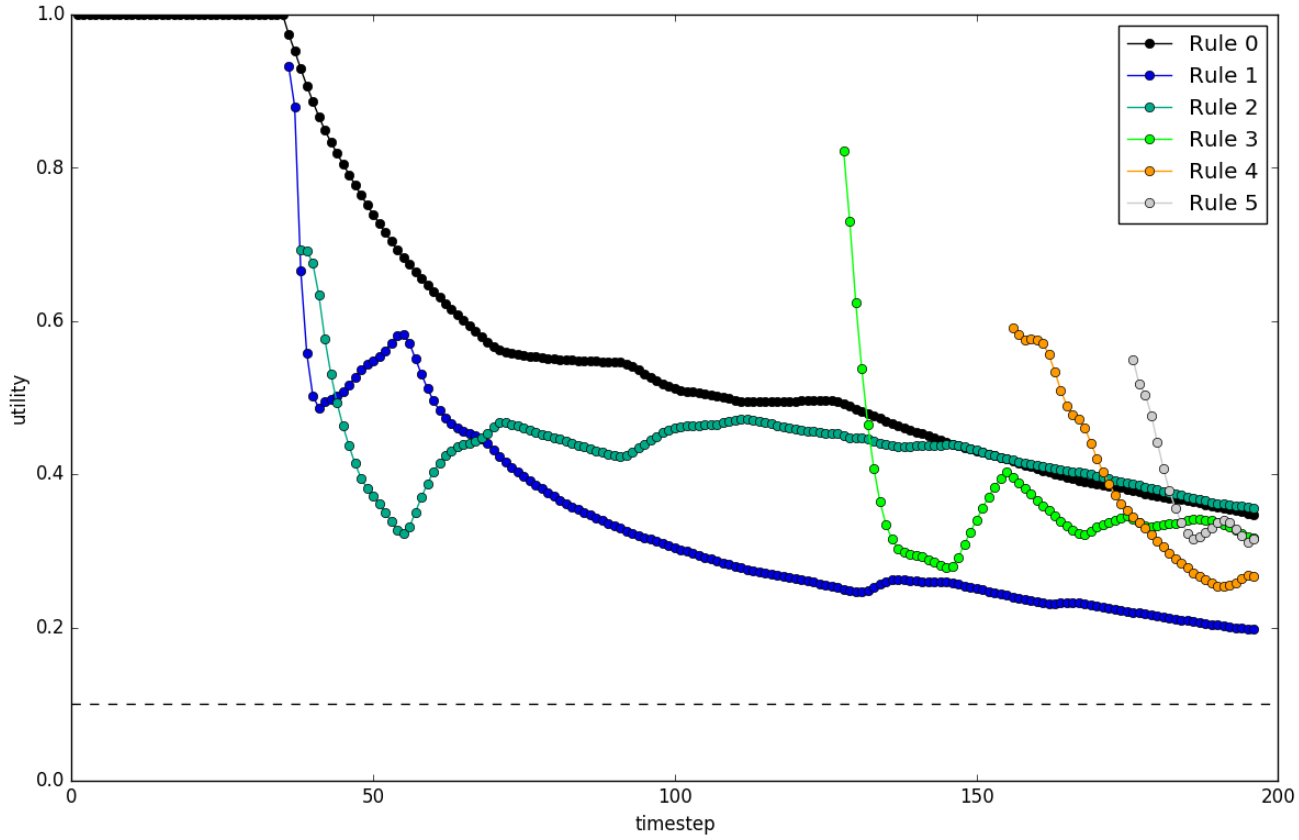


Figure 3.21.: The utility of all created rules during the training of AnYa on the 196 first samples of the Box-Jenkins data set ([Box and Jenkins, 1970]). The dashed line is the decision boundary $\epsilon = 0.1$, for removing rules.

3.6.2 Age

Age is a measure that is really only important for systems that perform dynamic control or predict processes that are time dependent in any kind. The idea is again easy to verbalize: If a rule was mostly useful for samples that were not recently read, its value in the current state of the system is diminished. This is expressed mathematically in the formula in figure 3.22. The subtrahend in the given formula is the mean time index of the samples associated with rule i. As such, it is not necessary to save all the time indices I_{ij} because the mean can be updated recursively before adding a sample (figure 3.23).

$$Age_i = k - \underbrace{\left(\frac{1}{M_i} \sum_{j=1}^{M_i} I_{ij} \right)}_{:= \bar{I}_{i\cdot}}$$

where :

k is the index of the latest time step

M_i is the number of samples associated with rule i

I_{ij} is the index of the time step in which the j-th sample of rule i was added

Figure 3.22.: The definition of rule age.

$$\bar{I}_{i\cdot}^{(k)} = \frac{M_i}{M_i + 1} \bar{I}_{i\cdot}^{(k-1)} + \frac{k}{M_i + 1}$$

Figure 3.23.: Recursive computation of the mean time index.

3.6.3 Rule Deletion

If a rule is deleted (because of minor utility or age), it is removed from the rule base. The matrices θ and C have to be updated after this removal. θ is the matrix containing the parameter matrices of all rules stacked vertically. This means that when rule i is removed, the rows with indices $\{i(n+1), i(n+1)+1, \ldots, i(n+1)+(n+1)\}$ have to be removed as well. This is best illustrated as the inverse operation of the extension of θ when creating a rule (see section 3.5). The same set of indices denotes the rows *and* columns of the covariance matrix C that are deleted.

3.7 Input Selection

The AnYa Fuzzy Rule-Based System is applicable in many different scenarios in machine learning and control. In many of these, it is not known beforehand, which components of the input are crucial for prediction, which are less important, which are redundant and which are not relevant at all. A high-dimensional input vector is often a cause for several problems (see e.g. [Tikka et al., 2005]):

1. The computational performance of the system is poor.
2. The system error is higher than it can be.
3. The created rules are inexpressive.

An example scenario in which these potential problems are noticeable is time series analysis (they apply in all models, of course). A standard objective in time series analysis is to find a function f that predicts a future state x or control u based on the last n_x states and n_u control variables:

$$x^{(k+1)}; u^{(k+1)} = f(x^{(k-n_x+1)}, \ldots, x^{(k)}; \quad u^{(k-n_u+1)}, \ldots, u^{(k)})$$

Assuming that the number and index of past significant states and controls is not known, a high-dimensional input vector is chosen. If the resulting system is used in control, each incoming sample has to be processed in a time that is far less than the sampling period of the sensors. The high dimension can lead to an increase in processing time that could make the system infeasible. It also impacts the parameter estimation and can produce a model that is over-fitted and performs worse in predicting future outputs than simpler models. Although this may not always be the case, many studies support this correlation (e.g. [Jang, 1996]). Rule antecedents get more complex too, because the input space that has to be partitioned is of high dimension. Complex rules are more difficult to evaluate by experts and are not as easy to convert to "human" knowledge (knowledge that is practically applicable and expressible by humans).

This makes input/feature selection necessary for many systems. There are various approaches to input selection today, of which most are done offline. In the tradition of Fuzzy Rule-Based Systems, one method suggests creating models for different numbers and arrangements of the input components ([Jang, 1996]). All models are then trained for one single time step and the best performing input vector is chosen for the online system. Other approaches for offline input selection have been proven to work well in practical studies involving artificial neural networks like Principal Component Analysis ([Zhang, 2007]) and genetic algorithms ([Zhang et al., 2005]). Even neural networks themselves are tools suitable for input selection ([Setiono and Liu, 1997]). [Loh, 2011] explores the use of decision and regression trees for the task.

For the implementation accompanying this thesis, a method is used that is simple in its idea, but has some advantages over traditional methods. It is described in [Angelov and Zhou, 2008] and depends on the Takagi-Sugeno-Kang-Type rule output (see figure 3.1). Closer inspections of the parameter matrices π_i show that they already contain information about the relevance of the input components:

$$x_e^T \pi_i = \begin{pmatrix} 1 & x_1 & \ldots & x_j & \ldots & x_n \end{pmatrix} \begin{pmatrix} \alpha^i_{0,1} & \ldots & \alpha^i_{0,m} \\ \vdots & & \vdots \\ \alpha^i_{j,1} & \ldots & \alpha^i_{j,m} \\ \vdots & & \vdots \\ \alpha^i_{n,1} & \ldots & \alpha^i_{n,m} \end{pmatrix}$$

$$= \begin{pmatrix} \ldots + \alpha^i_{j,1} x_j + \ldots \\ \vdots \\ \ldots + \alpha^i_{j,m} x_j + \ldots \end{pmatrix}^T$$

The j-th row of π_i holds the factors for the j-th input component (if we start counting the rows of π_i at zero, to compensate for the constant summands in the first row). This is enough information to define a simple metric for input relevance (figure 3.24). This approach is similar

$$S_{ij} = \sum_{l=1}^{k} \sum_{r=1}^{m} |\alpha_{j,r}^{i}| \qquad w_{ij} = \frac{S_{ij}}{\sum_{p=1}^{n} S_{ip}}$$

where:

$i \in \{1, ..., N\}$

$j \in \{1, ..., n\}$

k is the latest time step and is omitted in the sum for clarity

Figure 3.24.: The definition of input weight w_{ij}.

to the definition of rule utility in section 3.6; it uses information that is already available to generate new usable insight. A practical example of the progression of the generated values used to determine input relevance is shown in figure 3.25.
The downside of this method is that a comparison of the absolute parameter values really only makes sense if the input components all have the same scale to them. If that is not the case, standardization might be necessary to use this kind of input weighting. One possible way to do this online is shown in section 3.8.
[Angelov and Zhou, 2008] did a study on feature relevance metrics and came to the conclusion that S_{ij} is a good measure of input importance, but that for the removal of features, two cases have to be distinguished. Empirically, if there are less than or exactly ten input components, the contribution of a single component should be compared to the sum of all contributions. If there are more input features, the comparison should be between single contribution and maximum contribution. This is plausible, because a single feature's relevance is possibly diminished in a high-dimensional input space, if it is compared to the overall sum. Figure 3.26 shows the exact input removal conditions. If there is an input component that is not contributing in any cloud, it can be removed. However, the removed feature is never going to be considered again, as long as the system operates (if it is not added again manually). This means that for a system with known relevant features or for one that may experience a drift in input relevance over the runtime, input selection should be observed carefully or disabled by default.

The exact procedure to remove an input in this implementation of AnYa is an update of all dimensions of variables that are dependent on the length of the input vector ($= n$).

For inference, the matrices θ and C have to be updated. We saw that in every parameter matrix π, the j-th row is the one relevant for the j-th input. θ contains all π_i, so the rows $j, j + (n + 1), ...$ are removed. The covariance matrix C contains one row and one column for each parameter in each rule, therefore the indices $j, j + (n + 1), ...$ denote both the rows and the columns that are removed.
For density estimation, the j-th row in the last ξ and Ξ are removed (for local and global density, respectively).
Finally, n is decremented and the removed feature is disregarded in future samples.

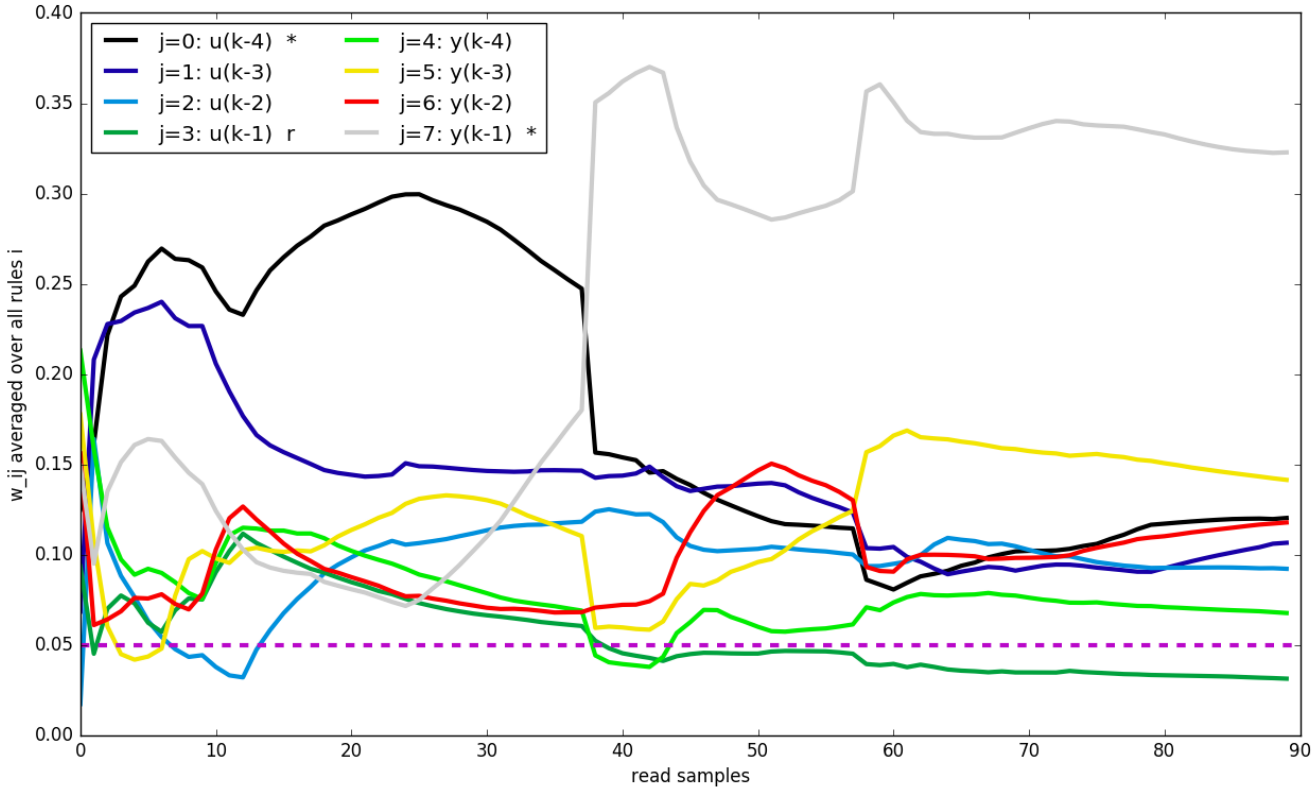


Figure 3.25.: The trend of average input weights w_{ij} for 8 input features during the training of an AnYa system on the first 90 samples of the standardized Box-Jenkins gas furnace data set [Box and Jenkins, 1970]). Samples consist of the input gas rate $u(k)$ and the output gas rate $y(k)$. The best model is known to depend solely on the inputs marked with "*". If input removal is enabled, inputs marked "r" are removed. The dashed line is the removal boundary $\epsilon = 0.05$.

IF (the input dimension is large, n $>$ 10)

remove inputs $j \in \{1,...,n\} : \forall i \in \{1,...,N\} :$ $\quad w_{ij} < \epsilon \max_{k,l} w_{kl}$

ELSE

remove inputs $j \in \{1,...,n\} : \forall i \in \{1,...,N\} :$ $\quad S_{ij} < \epsilon \sum_{p=1}^{n} S_{ip} \iff w_{ij} < \epsilon$

where:

ϵ is the minimum feature weight

($\epsilon \in [0.03, 0.05]$, [Angelov and Zhou, 2008])

Figure 3.26.: The condition that has to be met for a removal of an input component.

3.8 Standardization

In some scenarios, standardization of the sample data might be necessary, for example to use the particular method of input selection described in section 3.7 or to better express the regression coefficients without regard of the scale of the input variables. Also, if the sample contains data of very different scales, standardization might be crucial so the distance between samples remains meaningful. Consider a sample containing two variables with values in $[0, 0.1]$ and $[0, 1000]$ respectively. The latter variable will dominate the distance computation and the first one will be near irrelevant.
Standardization can be done by subtracting the mean and dividing by the standard deviation:

$$z_{st}^{(k)} = \frac{z^{(k)} - \bar{z}^{(k)}}{\sigma^{(k)}}$$

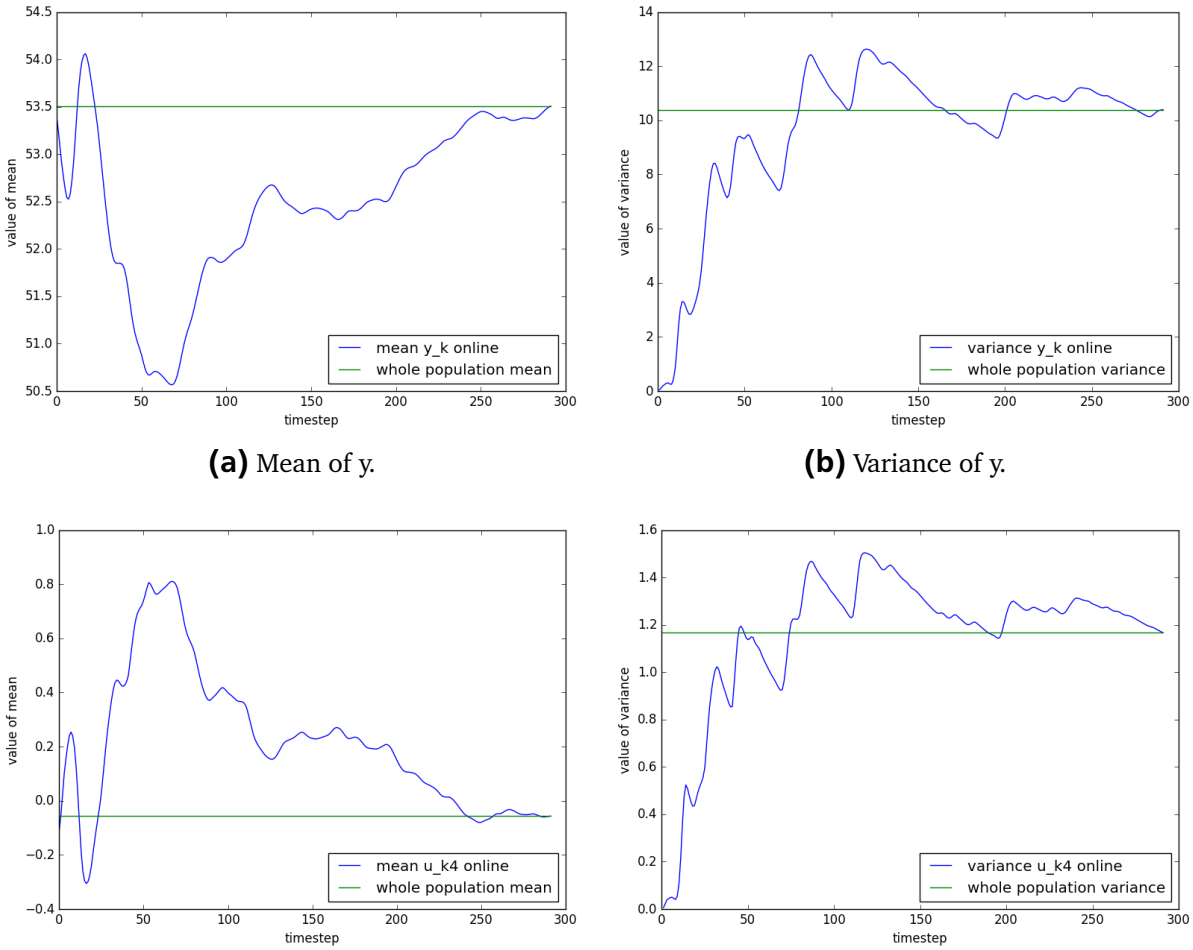
where z is the vector of concatenated input and output

This is also known as the "studentization" of a random variable. It is a measure to accomplish a zero-mean and unit variance in every component of z. A straight-forward method to recursively update mean and variance was introduced by [Angelov and Filev, 2005]:

$$\begin{aligned}
\bar{z}^{(k)} &= \frac{k-1}{k}\bar{z}^{(k-1)} + \frac{1}{k}z^{(k)} \\
\bar{z}^{(1)} &= z^{(1)} \\
\sigma^{2(k)} &= \frac{k-1}{k}\sigma^{2(k-1)} + \frac{1}{k-1}\left(z^{(k)} - \bar{z}^{(k)}\right)^2 \\
\sigma^{2(1)} &= 0
\end{aligned}$$

Note that z is a vector, so the mathematical operations have to be implemented accordingly. Also, the first standardized sample has the index $k = 2$, because dividing by zero standard deviation is not defined for $k = 1$. If these formulas produce wrong results, cancellation[1] may be the cause and more robust algorithms should be used (see for example [Welford, 1962]).
One problem with this kind of online standardization is that every sample gets standardized with a different mean and standard deviation, resulting in further noise in the data. The mean and standard deviation are accurate in every time step, but only for the read samples. The computed values are only an approximation of the mean and standard deviation of the whole population. These approximations are sometimes far off from the real values, resulting in a performance decreasing skew of the data. This is visualized in figure 3.27. If this is a problem that increases the error significantly, the standardization should be done offline or turned off completely.

[1] loss of precision in floating point represented numbers caused by subtracting values of marginal difference



(a) Mean of y.

(b) Variance of y.

(c) Mean of u.

(d) Variance of u.

Figure 3.27.: The comparison of the online approximation and the real (whole population) mean and variance of input variables u and y of the Box-Jenkins gas furnace data set. The mean and variance were computed using the robust online algorithm of [Welford, 1962].

3.9 Pseudocode

There are several different ways to define the order of operations in an AnYa system. One possible way is illustrated in figure 3.28 (training) and figure 3.29 (inference).

```
rules ← [ ]
k ← 0
function TRAIN(x,y)
    k ← k + 1
    if the sample is a good generalization
        and has small density in all clouds then                    ▷ fig 3.16
        new_rule ← new rule with cloud around x
        add new_rule to rules
    else
        best_rule ← rule with best fitting antecedent in rules      ▷ fig 3.4
        associate x with best_rule by updating local density        ▷ fig 3.5
    end if

    update global density                                           ▷ fig 3.7
    remove less valuable rules                                      ▷ fig 3.20
    update consequent parameters θ using (x, y)                     ▷ fig 3.12
    remove less valuable inputs                                     ▷ fig 3.26
end function
```

Figure 3.28.: Pseudocode for the training of an AnYa system.

```
function INFERENCE(x)
    for all rule i ∈ rules do
        λ_i ← the membership of x to the cloud belonging to rule i  ▷ fig 3.4
    end for
    ψ^T ← (λ_1 x_e^T   λ_2 x_e^T   ...   λ_N x_e^T)                 ▷ fig 3.11
    y ← ψ^T θ
    return y
end function
```

Figure 3.29.: Pseudocode for inference using an AnYa system.

3.10 Remarks

This section is a summary of the properties and concepts of the AnYa system from a high-level perspective.

The AnYa system can be regarded as a combination of supervised and unsupervised learning paradigms. The antecedents are based on the distance to other samples and resemble unsupervised clustering approaches. The k-means ([MacQueen et al., 1967]) and especially the x-means ([Pelleg et al., 2000]) clustering algorithms are similar in their ideas (but use the distance to the center of a cluster and not to all samples of it). Learning the consequents, of which the type is freely selectable, is an instance of a typical supervised learning task. They can be modeled as a linear relation between input and output. However, the creators of AnYa emphasize that the consequent can be adapted to any kind necessary.

Also worth mentioning is that the number of rules and the way they are combined into a single output is a crucial part influencing the system performance. A system limited to one rule is a simple linear least-squares regression.

An interesting side effect of using fuzzy rule memberships for both regression and classification is the information gained from the "firing levels" (or memberships) λ_i. Apart from being used for inference and providing the necessary information for associating a sample with a cloud, these can also serve as a measure of confidence. Imagine an AnYa system for multi-class classification where every rule is associated with one class label and the "winner-takes-all" approach is applied for inference. The λ_i can then be used to decide for each input if the system is very confident about its choice, or if there is not enough information / the input is ambiguous etc.

As an inference system created for control problems, AnYa has all properties necessary for online application (sometimes also called "stream learning"). Among others, the following four requirements are often sought after in these systems (taken from [Bifet et al., 2010]):

1. Process incoming samples one at a time and do it only once.
2. Use a limited amount of memory for all problem sizes.
3. Use a limited amount of time for sample processing.
4. Be able to perform inference at any time.

All of these are met by AnYa: When a sample arrives, every antecedent is evaluated. The evaluations are done in constant time, leaving the general complexity for this task at O(N), where N is the number of rules, which is a variable but very small number (typically never more than 10). The amount of used memory is also dependent on the rule count, but not on the number of samples (of which there is just one explicitly saved per rule). Also, the system is always in a state where predictions can be made, if at least one sample was processed. The update of the consequents is done using a recursive least-squares approach, which makes the parameter update a constant time operation. There are some accumulation variables that could overflow, which can be reset to prevent this.

Online and batch learning systems both have their obstacles and the application scenarios of both have (in general) little overlap. Systems dealing with possibly endless streams of data have seen more and more demand in recent years, mainly because of two factors. First, interesting data has been increasingly abundant and gets easier to come by (also in form of streams). This means that both the constraints of time and memory are stressed for the results to be up to date and correct. Secondly, the deployment of machine learning or control systems on embedded computers requires the adaptation to scenarios where the performance, memory or battery power is possibly limited or should at least be considered more thoroughly.
In the case of AnYa, one of the biggest drawbacks from online learning is that unstandardized data has a negative impact on the performance.

Of course, there are more limitations to this algorithm. As with all learners that rely heavily on the given instances, AnYa needs many samples with good coverage of the input space to make accurate predictions. Approaches that assume a preset distribution may be a better choice if the training set is small in its size and unbalanced.
Also, the quadratic distances that are computed for the local density of samples are prone to being skewed by outliers.

3.11 Similar approaches

Regression based on given instances and the distances between them is a very common approach. Generally, these are only called “instance-based”, if all seen instances are saved. Some authors use the term “memory-based” to better reflect this constraint.

One of the fundamental approaches in kernel regression is called the Nadaraya-Watson estimator, proposed by both [Nadaraya, 1964] and [Watson, 1964]. This instance-based estimator uses the provided training samples $(x_1, y_1), \ldots, (x_n, y_n)$ and a kernel K with bandwidth h to predict the output $f(x)$ for any given input x. The whole inference process can be summarized in one function:

$$f(x) = \frac{\sum_{i=1}^{n} \frac{1}{h} K(\frac{x - x_i}{h}) y_i}{\sum_{i=1}^{n} \frac{1}{h} K(\frac{x - x_i}{h})}$$

This function weighs every seen output y_i by the similarity of x_i to the query x. Choosing a good bandwidth h ensures that the $f(x)$ is not under- (h to small) or over-smoothed (h too big) and is the deciding factor of the system error. It is apparent, that the computational demands of this regression are high, both in regard to memory and processing time.
This estimator should appear familiar after reading about rule membership in AnYa, since the combination of local outputs is essentially the same, except the weighting is done for each training sample. This is an example of the interesting development that took place in the early studies of fuzzy engineering, kernel regression and other fields, where similar methods were developed in parallel by different authors.
An extension of this method is called LOESS ([Cleveland, 1979]) and uses varying bandwidths and robust estimates for the weights. The outputs are generalized to be any low-order polynomials (mostly linear of quadratic).

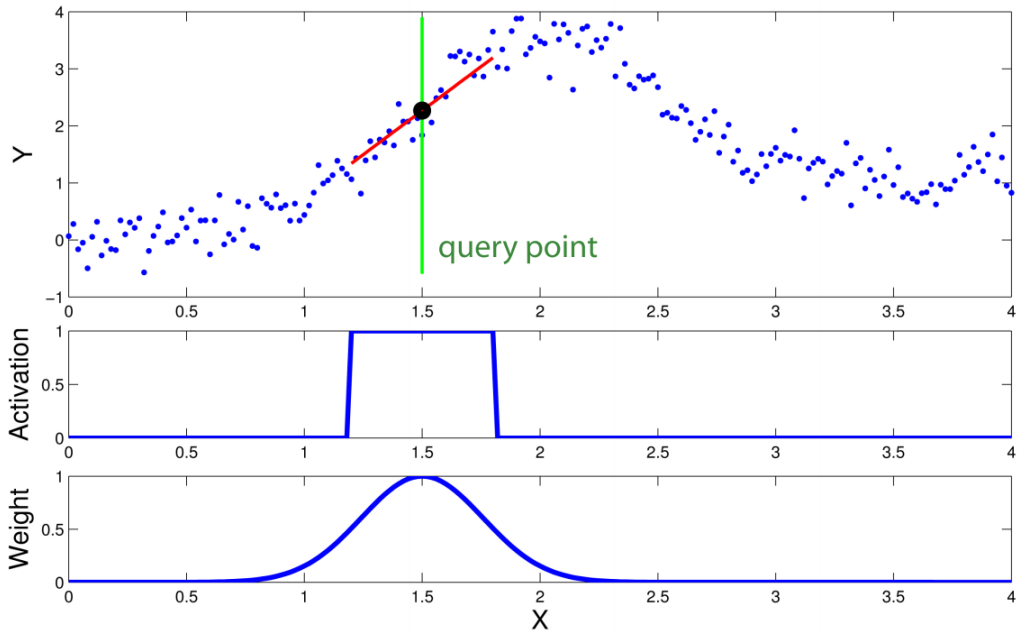


Figure 3.30.: Visualization of a local model created using Locally Weighted Regression. Taken from [Englert, 2012].

Another related method is called Locally Weighted Learning ([Atkeson et al., 1997]), more specifically Locally Weighted Regression. It uses a local linear model around a given query to predict the corresponding output (visualized in figure 3.30). Different kernels and distance measures can be applied. In its naive implementation, this approach builds local models for each query point and discards them after the prediction, all while using $O(n^2)$ time to build them (where n is the number of training samples). Incremental algorithms that update saved local models exist ([Vijayakumar and Schaal, 2000]). These resemble AnYa both in theoretical backgrounds and applicability (online, incremental learning).

4 Application of AnYa in the context of regression problems

AnYa has already been tested on several data sets for both classification and regression problems. However, the results were mostly compared to those of other fuzzy systems like ANFIS ([Jang, 1993]) or GeNFIS ([Jana et al., 1996]). To get a better understanding of the overall capabilities, I will instead present comparisons to several algorithms used traditionally in machine learning and stream mining applications.
The tests are split in two kinds of application scenarios. First, the performance in traditional batch learning tasks is evaluated. Results of common approaches are obtained from the open source machine learning framework Weka[1] ([Holmes et al., 1994]). Secondly, the online capabilities of AnYa were compared to those of algorithms implemented in the stream mining framework Moa[2] ([Bifet et al., 2010]).
There are two different kind of data sets used in the tests. The data sets of the first kind are comprised of numerical data with possibly multiple inputs and a single output. To predict a single output is the most prevalent regression task researched today. The second kind of data sets have possibly multiple input variables as well as multiple output values. These scenarios are sometimes called "multi-target" regressions and are not that commonly studied. Multi-target regression is not available in Weka (see Meka[3] for a Multi-label/Multi-target extension of Weka), but Moa implements some algorithms capable of this task.
The following performance measurements were used in the single-target case:

the mean absolute error:

$$MAE = \frac{1}{K}\sum_{k=1}^{K}|\widehat{y^{(k)}} - y^{(k)}|$$

the root mean squared error:

$$RMSE = \sqrt{\frac{1}{K}\sum_{k=1}^{K}\left(\widehat{y^{(k)}} - y^{(k)}\right)^2}$$

the root relative squared error:

$$RRSE = \sqrt{\frac{\sum_{k=1}^{K}\left(\widehat{y^{(k)}} - y^{(k)}\right)^2}{\sum_{k=1}^{K}\left(y^{(k)} - \overline{y}\right)^2}}$$

where : $\widehat{y^{(k)}}$ is the prediction for the k-th input
$y^{(k)}$ is the observation

[1] http://www.cs.waikato.ac.nz/ml/index.html
[2] http://moa.cms.waikato.ac.nz/
[3] http://meka.sourceforge.net/

These can essentially also be used for multi-dimensional outputs (if the operations are carried out per component). Additionally, the following measurements can be useful (y_j denoting the j-th component of vector y, m being the output dimension):

the average mean absolute error:

$$aMAE = \frac{1}{m}\sum_{j=1}^{m}\frac{1}{K}\sum_{k=1}^{K}|\widehat{y_j^{(k)}} - y_j^{(k)}|$$

the average root mean squared error:

$$aRMSE = \frac{1}{m}\sum_{j=1}^{m}\sqrt{\frac{1}{K}\sum_{k=1}^{K}\left(\widehat{y_j^{(k)}} - y_j^{(k)}\right)^2}$$

4.1 Performance in batch learning scenarios

4.1.1 Airfoil data set

This data set ([Brooks et al., 1989]) contains attributes of different airfoil designs and tests. The single output is the noise they produced in an wind tunnel. The input and output dimensions are therefore $n = 5$ and $m = 1$. The data is split into a training set (66% of all 1503 samples) and a test set, to make sure that both the Weka regressions and AnYa have exactly the same conditions. The mean and standard deviation of the training set is used to standardize both sets.
AnYa was compared to 5 algorithms. “ZeroR” always predicts the mean of seen outputs for any input. “LinReg” is a simple linear regression. “SVMReg” is a support vector machine regression. “kNN” is a k-nearest neighbor regression approach that was chosen because it is based on an idea inherently similar to AnYa (distance to known instances). “M5 Tree” is an improved version of the M5 model tree of Quinlan, which uses linear regression in its leaves (just like AnYa in its rules). The results are shown in table 4.1.

Algorithm	Airfoil noise (std.)			
	MAE	RMSE	RRSE	notes
AnYa	0.6806	0.8636	0.8231	4 rules
ZeroR	0.9071	1.0901	1.0000	-
LinReg	0.6754	0.8813	0.8085	-
SVMReg	0.6977	0.9149	0.8393	PolyKernel
kNNReg	**0.6333**	**0.7919**	**0.7265**	0< k <16
M5 Tree	0.6384	0.8490	0.7789	30 rules/leaves

Table 4.1.: The results of 6 algorithms predicting the standardized airfoil data set.

It seems like AnYa performed moderately well in comparison to the other approaches. Unfortunate is that the error would have at least been the same as the one using linear regression,

had only one rule be created. The error is still better than the one of the support vector regression, although that could very well be a parameter issue. The M5 tree regression performed noticeably better than AnYa, which is expected, since the design and structure are much more sophisticated. The k-nearest neighbor approach produced the best overall results, by doing a cross-validation of the best $k \in \{1, 2, ..., 15\}$, which is a computationally expensive operation.

4.1.2 Concrete data set

The concrete data set [Yeh, 1998] is a data base containing 1030 samples of different concrete mixtures and their capabilities to withstand compressive force. There are $n = 8$ input variables (including the ratio of cement and water and the age of the concrete) and a single ($m = 1$) output, the compressive strength in mega-pascal. The preprocessing steps are similar to the ones used for the airfoil data (826 samples for training, standardized) and the same algorithms are used for comparison. Additionally, the input selection mechanism of AnYa, described in detail in section 3.7, is enabled in one test run of the algorithm. Also included is an evaluation of the linear regression with the options "attributeSelectionMethod" set to use M5 and the option "eliminateColinearAttributes" set to true, to see if the results of both approaches are similar. Results are shown in table 4.2.

Algorithm	Concrete (std.)			
	MAE	RMSE	RRSE	notes
AnYa	0.4752	0.5982	0.8471	7 rules
AnYa_InputSelect	**0.3453**	**0.4565**	**0.6465**	(-age), 8 rules
ZeroR	0.5810	0.7363	1.00	-
LinReg	0.3927	0.4974	0.6756	-
LinReg_InputSelect	0.3938	0.4987	0.6772	(-fine aggr.)
SVMReg	0.4155	0.5070	0.6886	PolyKernel
kNNReg	0.6346	0.7492	1.0175	0< k <16
M5 Tree	0.6221	0.7569	1.0280	28 rules/leaves

Table 4.2.: The results of 7 algorithms predicting the standardized concrete data set. When input selection was used, the note "(-x)" means the input named x was removed.

Interestingly, the M5 Tree and the k-nearest neighbor regression, which performed best on the airfoil data, produced the highest errors on this data set. The linear regression got the smallest error when using all attributes, even better than the support vector machine regression. AnYa got mediocre error scores again, but got really good results when input selection is enabled. When the attribute "age" is removed, the error of the system is reduced drastically. This is a good example of why expert knowledge should be incorporated into the evaluation of inference systems, since the age of cement is known to have an undeniable impact on the compressive strength. Nevertheless, the errors on the test set are the smallest with this configuration (the data was also shuffled and the test run again, resulting in similar results). The linear regression implemented in Weka also removed one attribute, the "fine aggregate", if the input selection was enabled. This affected the performance negatively.

4.2 Performance in online learning scenarios

4.2.1 Indoor Temperature data set

This data set ([Zamora-Martínez et al., 2014]) is a collection of sensor data sampled in and around an domotic house (also known as "smart home"). Attributes include the exterior temperature, humidity, carbon dioxide concentration, wind speed and sun light intensity, uploaded every 15 minutes over a span of about 40 days. As dependent variables, the indoor temperatures in the dining room and the living room were chosen. The columns containing the date, the time and the day of the week were removed, leaving an input dimension of $n = 19$ and output dimension $m = 2$. The training set was chosen as the first 1824 samples (66 percent of all samples), the test set as the rest.

Four algorithms implemented in Moa were used on the same data as AnYa. "NoChange" just predicts the last output vector that the learner has been trained on. "TargetMean" predicts the mean of the seen sample outputs. So in this case of a holdout test set, the mean of the training set is predicted every time. "ISOUPTree" (Incremental Structured OUtput Prediction Tree, [Osojnik et al., 2015]) is an approach based on regression trees. "Perceptron" is an ensemble of linear single-target perceptron regressions capable of multi-target prediction. The results are shown in table 4.3.

Algorithm	Indoor temperature				
	MAE	aMAE	RMSE	aRMSE	notes
AnYa	0.82, 0.89	**0.855**	1.11, 1.17	**1.140**	4 rules
NoChange	1.85, 1.92	1.888	2.27, 2.36	2.316	-
TargetMean	2.00, 2.00	2.000	2.51, 2.51	2.509	-
ISOUPTree	1.26, 1.22	1.243	1.58, 1.52	1.547	-
Perceptron	1.57, 1.59	1.580	1.90, 1.93	1.916	-

Table 4.3.: The evaluation results of 5 algorithms on the indoor temperature data set.

AnYa got smaller errors than the tested algorithms. Optimizing the parameters for the ISOUP trees or the perceptron ensemble learner might change this outcome. Fortunately, AnYa performed significantly better than the two baseline learners (NoChange and TargetMean). Another information that can be extracted from these results is that most algorithms had more problems predicting the second output (the temperature in the living room).

4.2.2 Stock exchange data set

This data was collected by [Akbilgic et al., 2014] and contains the return values of various stock exchanges and market indicators. The Istanbul stock exchange return value was used as output variable ($n = 8, m = 1$). The data is again sequential and countains 534 samples.

For this data set, the evaluation was done *prequentially*, an approach that is very often used

for online learning systems. Prequential (predictive sequential) means that there are no separate training and test sets, but every sample is first used for testing and then passed to the system again as a training example. This method is known to give a pessimistic estimate of the system performance ([Gama et al., 2009]).

Three algorithms were used for comparison. "TargetMean" predicts the mean of the seen sample outputs. "AMRules" means "Adaptive Model Rules" ([Almeida et al., 2013]) and is an incremental approach that produces rules with linear output, just like AnYa. "FIMT" (fast incremental model trees with drift detection, [Ikonomovska et al., 2011]) makes use of an evolving model tree.

The results (table 4.4) suggest that AnYa performed better than any of the tested regressions. However, the parameters of AMRules and FIMT were kept in their default state, which would not be the case in a realistic application scenario. Better knowledge of the underlying algorithms and the situation circumstances could lead to parameter values that produce far better results. Still, AnYa produced a small rule base of only one rule to achieve good error values. As mentioned early, an AnYa system with only one rule is simply a linear regression that is updated on every incoming sample. That no other rules were created arises partly from the fact that the samples were not standardized (since they probably would not be in a real-time application, for example).
In figure 4.1, it can be seen that the error values begin to stabilize after only 50 read samples.

Algorithm	Stock exchange			
	MAE	RMSE	RRSE	notes
AnYa	**0.0043**	**0.0063**	**0.3906**	1 rule
TargetMean	0.0123	0.0166	1.0194	-
AMRules	0.0082	0.0123	0.7624	2 rules
FIMT	0.0128	0.0208	1.2830	2 leaves

Table 4.4.: The results of 4 algorithms evaluated prequentially on the stock exchange data set.

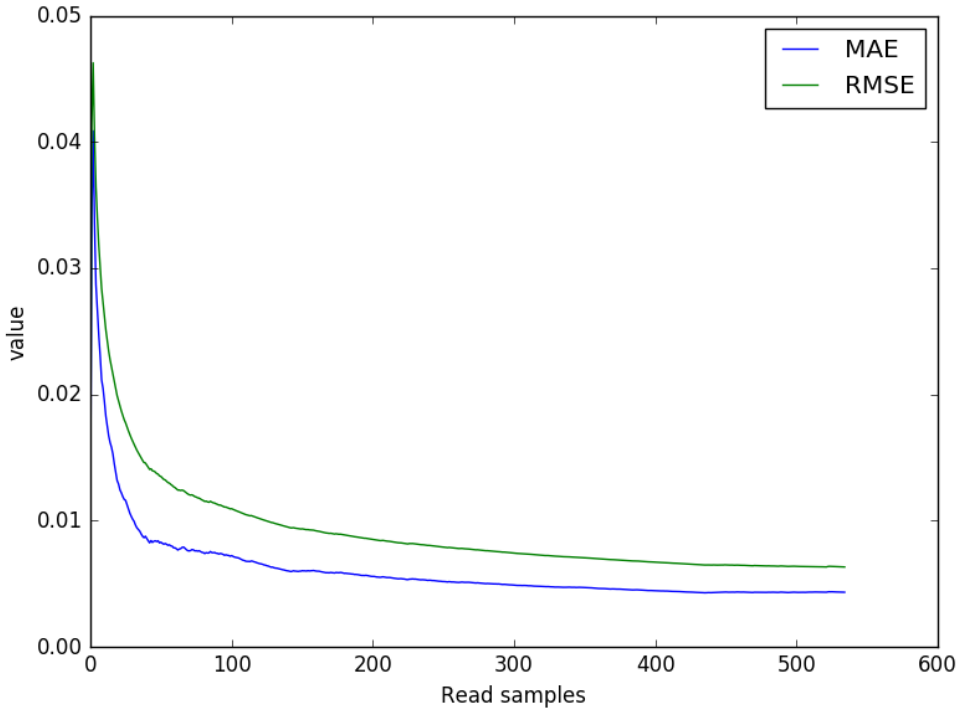


Figure 4.1.: The progression of error values on the prequential evaluation of AnYa on the stock exchange data set.

5 Proposed extensions for AnYa

This chapter is a collection of possible extensions and modifications of the AnYa system ([Angelov and Yager, 2011]). Every approach is motivated, implemented and tested. The results are summarized at the end of each section. The propositions are studied independent of one another.

5.1 Update of cloud-representing samples

The two conditions that decide over rule creation in AnYa are:

$$\forall i \in \{1, ..., N\}: \qquad \Gamma^{(k)} > \Gamma_i \tag{1}$$

$$\forall i \in \{1, ..., N\}: \qquad |\gamma_i^{(k)}| <= e^{-1} \tag{2}$$

The first condition can be verbalized as: "The global density at the current sample is bigger than the global density at the initial sample of the clouds of each rule.". This should ensure that new clouds are built on samples that are able to generalize the underlying population well. The initial sample can be seen as a representation of each cloud that is fixed over the whole lifetime of the corresponding rule. However, it is not a "center" of the associated samples of the clouds and is not even guaranteed to maintain a high local density for the cloud it created. The same is true for global density. This section will explore how this representing sample can be updated and what effects this has on model performance.

Let z_i^{repr} be the sample that led to the initialization of rule i and is used in condition (1). To compare densities of new samples with z_i^{repr} seems arbitrary at first, because there are no apparent reasons to choose the first over any other samples of each cloud. The one thing that is known for sure, is that at the time when z_i^{repr} was processed, it was a sample that had low membership to all clouds (condition (2)) and had smaller overall distances to the other samples than the initial samples of each cloud (condition (1)). When many samples have been processed afterwards, these conditions may no longer hold. It could be beneficial to update the representing sample used for further comparisons. The following possible updates have been tested (these are executed when a training sample is associated with a cloud):

1. Replace z_i^{repr} with every new sample that has a higher local density in this cloud.
2. Replace z_i^{repr} with every new sample that has a lower local density in this cloud.
3. Replace z_i^{repr} with every new sample that has a higher global density.
4. Replace z_i^{repr} with every new sample that has a lower local density.

For illustration and performance measure of the system, the Box-Jenkins gas furnace data set ([Box and Jenkins, 1970]) and the concrete data set ([Yeh, 1998]) were used. The first was chosen because it has two input variables ($n = 2$) and is thus good for visualizing the resulting clouds and the latter because it has a high-dimensional input ($n = 8$) and poses a hard regression

problem. In both data sets, the output is one-dimensional ($m = 1$). For the gas furnace data, the first two thirds of the available data was used as the training data, the rest as a test set. The first 826 samples of the concrete data set were used for training. The resulting clouds on the gas furnace data set are depicted in figure 5.1.

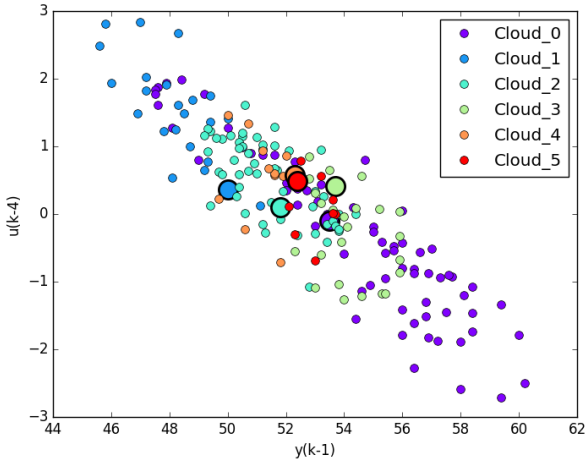
First, we observe the clouds formed by updating based on global density. Updating the representing sample with incoming samples with higher global density has the expected effect of grouping these in the center, where they have the smallest distance to all other samples. This leads to a small number of clouds/rules (because of condition (1)), which look fairly similar in their distribution than the ones in the standard version shown in the top plot. Updates done by lower global density produce a lot of clouds, which makes sense, because condition (1) gets ever easier to fulfill, when representing samples do not generalize the overall population well. This might make one expect a bad test set performance.
Updates based on local density are the ones we are interested in, when we want to find a representing sample that helps to understand the distribution of the particular cloud it belongs to. The obvious case (and probably the most intuitive of all updates) is to use the sample with the higher local density, resulting in clouds visualized in the top left plot. Intuitively, this result looks promising, because the clouds have the least overlap and the representing samples are almost in the center of each cloud. However, if this is reflected in the test results is to be seen. The updates done by lower local density produce the most clouds of all proposed updates. This is explained by the fact that samples with low local densities are found at the outer parts of each cloud, also having a low global density. Good test results are not expected because of bad generalization.

Update method	Gas furnace			Concrete		
	MAE	RMSE	#rules	MAE	RMSE	#rules
no updates (standard)	**0.4330**	**0.6728**	6	9.4175	11.7933	8
higher local density	0.4585	0.6958	5	21.8688	27.7492	14
lower local density	0.4925	0.7310	8	130.7981	145.7166	11
higher global density	0.4735	0.7129	5	**9.2475**	**10.9384**	7
lower global density	0.4535	0.6965	7	82.2893	95.4554	11

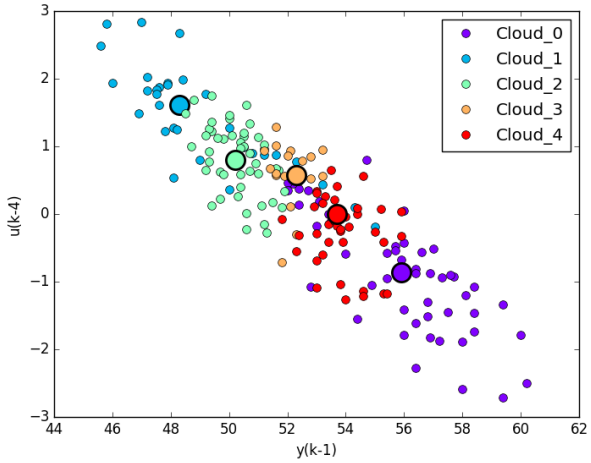
Table 5.1.: The test results of the possible updates on two data sets. The performance measures are explained in detail at the beginning of chapter 4.

Table 5.1 shows the results of the benchmarks. Generally, none of the updates lead to an increased performance. Updating by higher global density has little impact on the performance, but got slightly better scores on the concrete data set. This may be because the initial samples themselves had a high density and samples with higher global density lay nearby. Updates based on lower global/local density expectedly produce worse results; there is no real reasoning behind performing these. Updates by higher local density also perform worse unfortunately, even though they have the instinctive advantage of "describing" the distribution of cloud samples better. On the concrete data set, the number of rules is the highest with this method, which is probably an over-fitted rule base (explaining the bad results).
In conclusion, the explored extensions of the rule creation condition by updating the representing sample (instead of using the initial one) do not lead to an increased performance of the AnYa system. However, it might me a good idea to update a separately saved sample in each

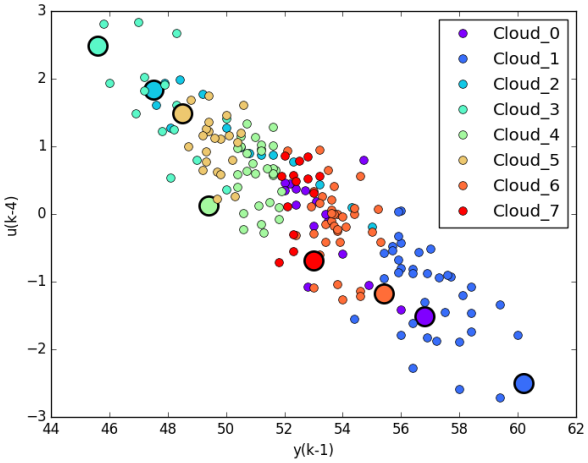
rule with samples of higher local density. Describing the antecedents of rules in a expressive way is a problem of AnYa systems and that sample could then be used for this, because it is the most centered of the data samples in the associated cloud.



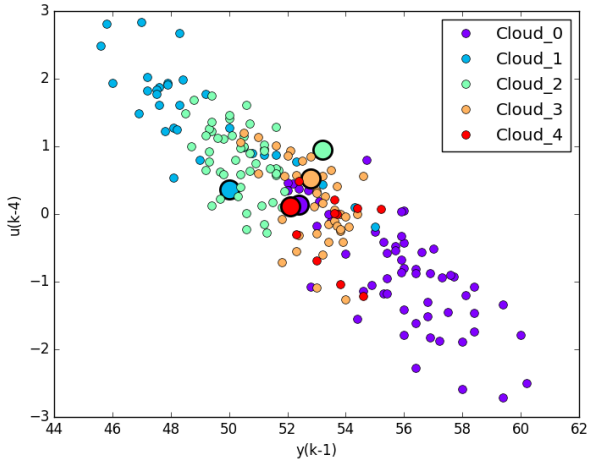
(a) No updates.



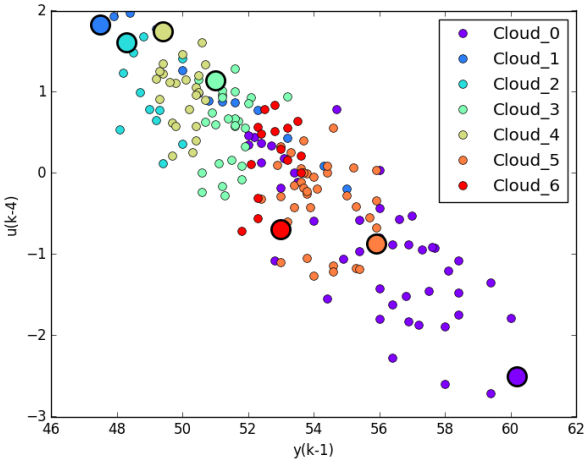
(b) Higher local density.



(c) Lower local density.



(d) Higher global density.



(e) Lower global density.

Figure 5.1.: The resulting clouds (of the gas furnace data set) with four different updates of the representing sample. Highlighted are the representing samples.

5.2 Penalizing rules with few associated samples

The utility $Util_i$ of a rule is defined as the mean firing level over its lifetime, yielding a value in $[0, 1]$. Rules with utility below a fixed value (here: 0.1) are removed. This can lead to clouds grouping around each other, because a sample in between two clouds has enough membership in both to justify keeping them. It would be preferable if the system had less clouds that differentiate seemingly disparate parts of the input space better.
Consider the following formula (time index k omitted on the left side):

$$M_i^{norm} = \frac{M_i}{k - I_i + 1}$$
$$\text{where } I_i \in \{0, \ldots, k\} \text{ is the time step of creation of rule } i$$

Here, $M_i^{norm} \in (0, 1]$ describes how many samples have been associated with the cloud of rule i in each time step since creation. Note that this is not the same as the age of a rule (see section 3.22), which is defined as the difference between the current time step and the mean of the time steps, in which samples were associated with rule i. M_i^{norm} is not dependent on *when* samples were added. With this value, we can penalize rules that have enough average membership (= utility) to not get removed, but are rarely the ones with the nearest cloud. One possible way to do this is:

$$Util_i^{new} = \frac{\sum_{l=I_i}^{k} \lambda_i^{(l)}}{k - I_i + 1} + \left(\frac{M_i}{k - I_i + 1} - 1 \right) p$$
$$= Util_i + \left(M_i^{norm} - 1 \right) p$$
$$\text{where } p \in \{0, 0.1, \ldots\} \text{ is a penalizing subtrahend}$$

In the above formula, the utility is not penalized if there has been one sample associated with the rule each time step since its creation ($M_i^{norm} = 1$). If the ratio between cloud members and lifetime gets worse (closer to zero), up to p is subtracted from the average membership, making it more likely for the rule to be removed. The resulting errors of this penalizing approach on part of the Boston housing data set (same setup as in section 3.14) are depicted in figure 5.2. The plot shows that a penalty of $p = 0.2$ produces the smallest error, reducing the MAE by a third. Looking at the created clouds and the learned approximation using this penalty value (figure 5.3), it seems that there are indeed less rules created, although they still have a good amount of overlap. Intuitively, the resulting function looks like a better approximation of the training data, in comparison to figure 3.14. This is reflected in the smaller error on the test set. The exact performance measures on the housing data set, as well as the errors on the Box-Jenkins gas furnace data set are shown in table 5.2. The results seem to imply that using $p \in [0.05, 0.3]$ can have a beneficial impact on system performance. Also visible is that in general, the rule count decreases for increasing penalties, since with this method, utility can only be decreased (possibly leading to the removal of a rule). This is not true for every single value (see $p = 0.2$ on the housing data set), because the changing system structure can produce new rules in return.
Further tests have shown that penalties $p > 0.3$ influence the system performance negatively. Also tested were other ways to penalize rules with few associated samples, for example a percentual penalty reducing the original utility to up to $pUtil_i$ with $p \in \{1, 0.9, \ldots\}$. These also

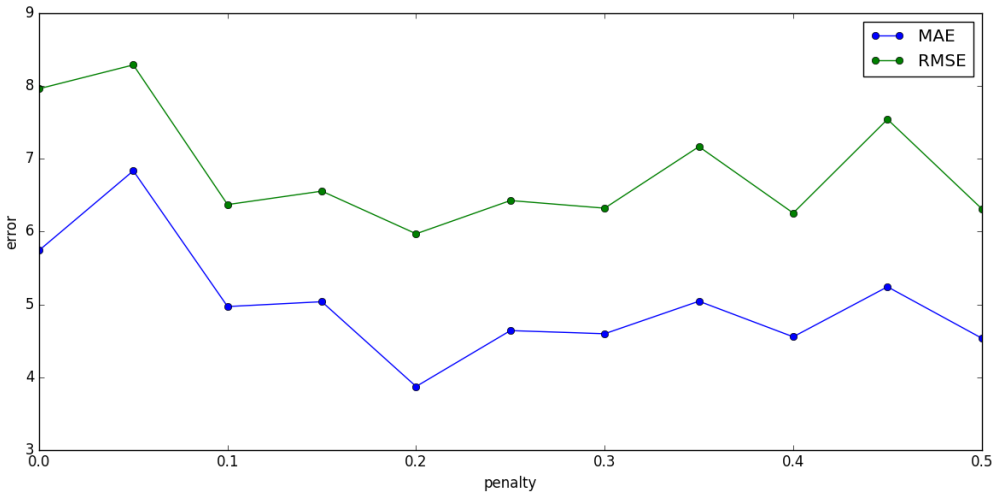


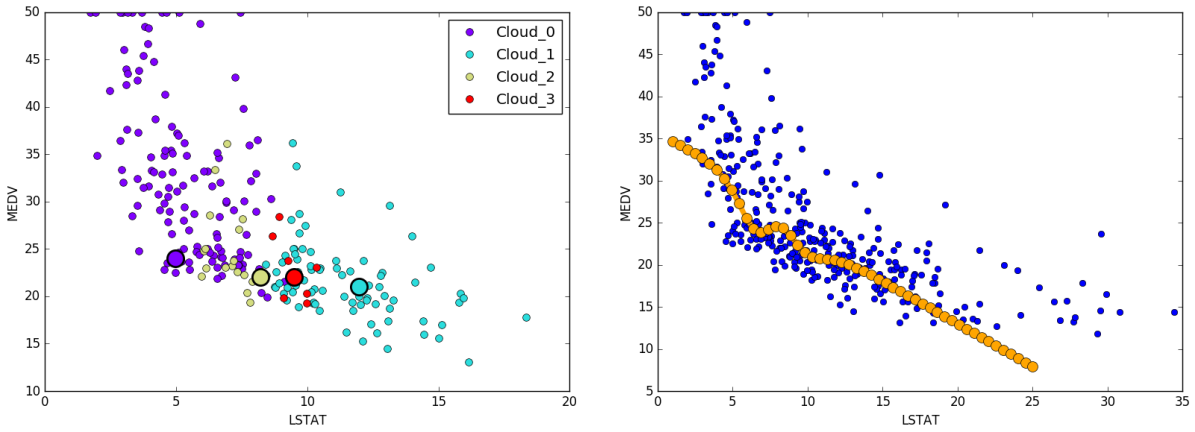
Figure 5.2.: The errors on the Boston housing data set using different penalties.

Penalty	Gas furnace			Housing		
	MAE	RMSE	#rules	MAE	RMSE	#rules
p=0 (standard)	0.4330	0.6728	6	5.7423	7.9648	5
p=0.05	0.4330	0.6728	6	6.8367	8.2921	5
p=0.1	0.4330	0.6728	6	4.9707	6.3724	4
p=0.15	0.4155	0.6546	5	5.0372	6.5571	3
p=0.2	**0.4129**	0.6525	5	**3.8716**	**5.9710**	4
p=0.25	0.4157	**0.6506**	4	4.6402	6.4278	2
p=0.3	0.4526	0.6915	4	4.5969	6.3230	2

Table 5.2.: The test results on two data sets using different penalty values.

produced worse results.

In this section it has been shown that rules, whose clouds are rarely closest to a sample but have enough average membership to not get removed, can be deleted to improve the performance of an AnYa system. The removal of these rules has been carried out by penalizing the utility by subtracting up to p from it (for the rules with the fewest members). When including this approach in an online application, one has to know a good value for p beforehand, have the possibility to run several systems with different values for it (and selecting the best one), or find a way to update it online. Then, it can be a fairly simple idea that reduces the error significantly (see the results of the method on the housing data set).



(a) Created clouds.

(b) Training data and obtained function.

Figure 5.3.: The results of training on the housing data set, using a penalty of $p = 0.2$. Highlighted points are initial samples of the respective clouds.

5.3 Alternative density function derived from the Gaussian kernel

AnYa uses a density function based on squared distances that resembles a Cauchy kernel. This is a reasonable choice, since the maximum of it is one and the minimum is zero with a monotonic descent in between. However, the Cauchy kernel is not the only kernel with these characteristics. The Gaussian kernel has the same properties and can also be used to construct a density function:

$$K(v,w) = exp\left(-\frac{\|v-w\|^2}{2\sigma^2}\right)$$

where v, w are vectors,

σ^2 is the variance

A density similar to the one used in AnYa could be (shown for global density):

$$D^{(k)} = exp\left(-\frac{1}{k-1}\sum_{j=1}^{k-1} d(k,j)^2\right)$$

The similarities of the (simplified) density functions derived from the Cauchy and the Gaussian kernel are pictured in figure 5.4 Finding a recursive formula for the new density function is the

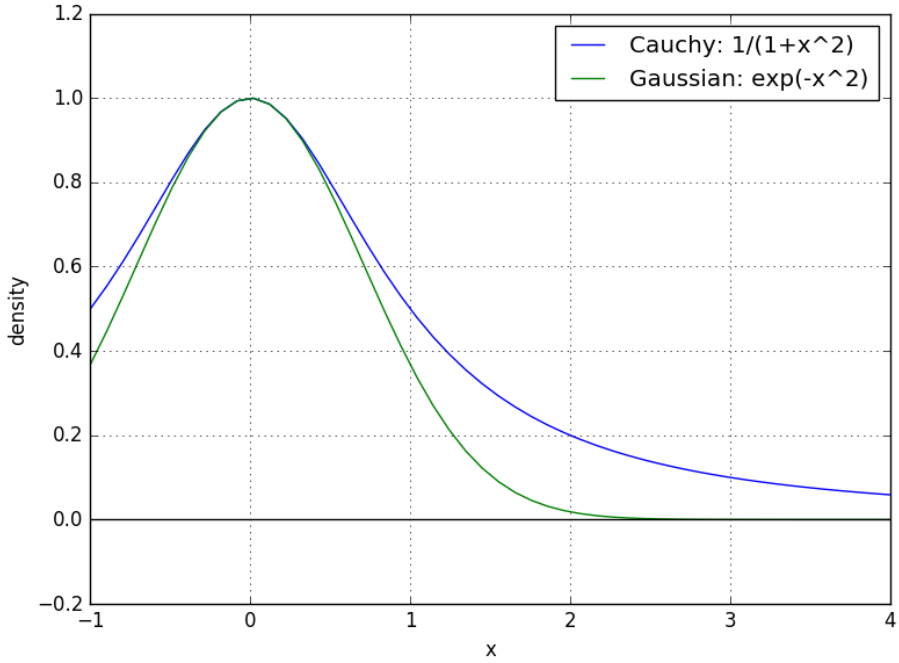


Figure 5.4.: Density functions based on the Cauchy and the Gaussian kernel.

same as before for the one based on a Cauchy kernel. The new density function for the global density is then:

$$\begin{aligned}
\Gamma^{(k)} &= exp\left(-\frac{1}{k-1}\sum_{j=1}^{k-1}\sum_{i=1}^{l}\left(x_i^{(k)} - x_i^{(j)}\right)^2\right) \\
&= \ldots \text{ (see 2.3)} \\
&= exp\left(-\frac{1}{k-1}\left((k-1)x^{(k)^T}x^{(k)} - 2x^{(k)^T}\sum_{j=1}^{k-1}x^{(j)} + \sum_{j=1}^{k-1}x^{(j)^T}x^{(j)}\right)\right) \\
&= exp\left(-\left((x^{(k)^T}x^{(k)} - \frac{2}{k-1}x^{(k)^T}\sum_{j=1}^{k-1}x^{(j)} + \frac{1}{k-1}\sum_{j=1}^{k-1}x^{(j)^T}x^{(j)}\right)\right)
\end{aligned}$$

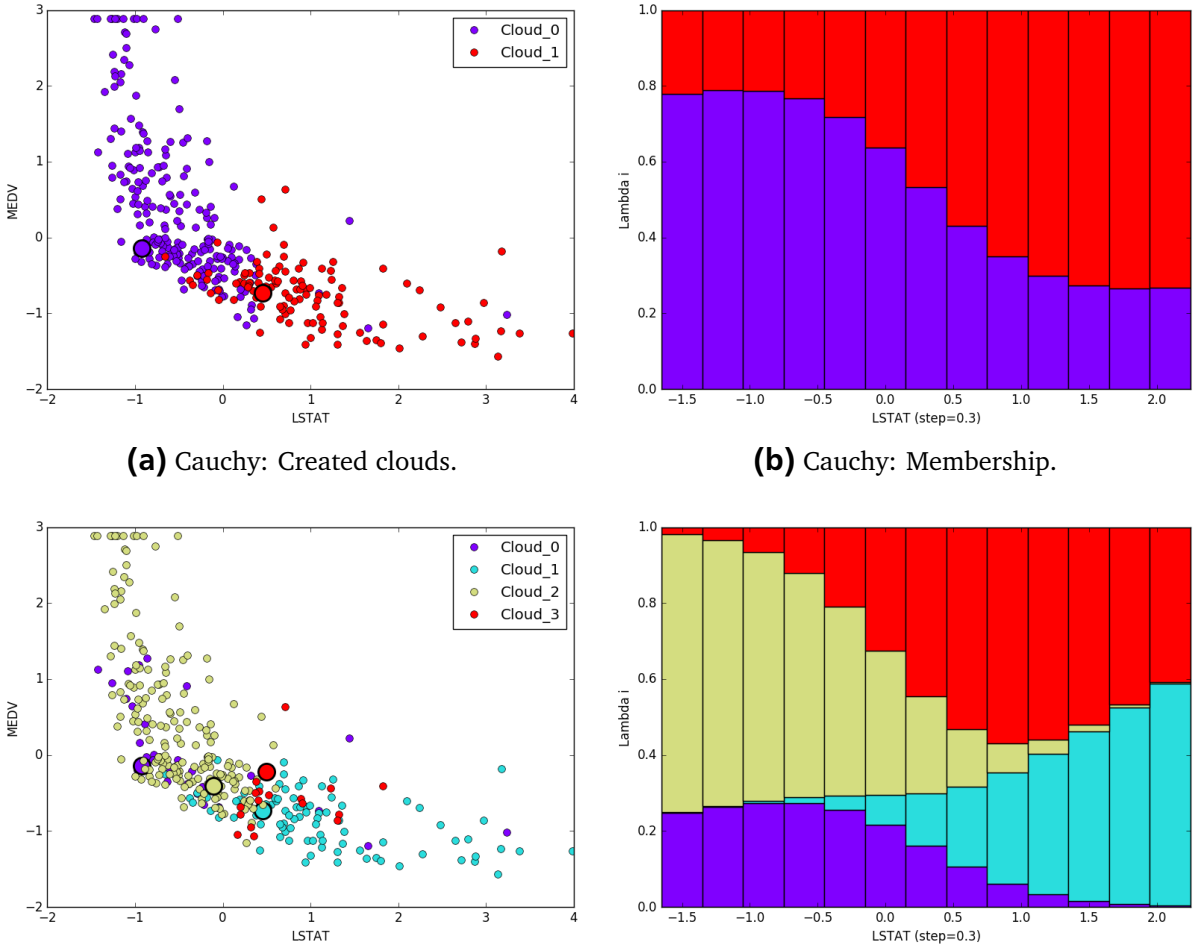
The recursion:

$$\Gamma^{(k)} = exp\left(-\left((x^{(k)^T}x^{(k)} - \frac{2}{k-1}\alpha^{(k)} + \frac{1}{k-1}\beta^{(k)}\right)\right)$$

$$\begin{aligned}
&\text{with } \alpha^{(k)} = x^{(k)^T}\xi^{(k)} \\
&\xi^{(k)} = \xi^{(k-1)} + x^{(k-1)} \qquad \xi^{(0)} = 0 \\
&\beta^{(k)} = \beta^{(k-1)} + x^{(k-1)^T}x^{(k-1)} \qquad \beta^{(0)} = 0
\end{aligned}$$

From the plot we can deduce that with the new function, samples will lose their density in a cloud faster than with the original one, as the average squared distance increases. This means,

that generally, more clouds for the same number of training samples will be created. The two density functions were tested on a standardized version of the subset of the Boston housing data used throughout this thesis. Standardization was necessary, because for the Gaussian-Type density, the argument passed to the exponentiation could overflow (when using the concrete data set). The test set was selected as one third of the available data and was standardized by the training set mean and standard deviation to give an unbiased performance estimate.



(a) Cauchy: Created clouds.

(b) Cauchy: Membership.

(c) Gaussian: Created clouds.

(d) Gaussian: Membership.

Figure 5.5.: The created clouds and memberships for two density function on the standardized housing dataset. Note the scales and axis limits.

First, observe the created clouds and memberships depicted in figure 5.5. Two things should be noticed here. One, that the Gaussian density did create two more cloud, which is not unexpected, since the creation condition making sure that samples that are candidates for a new cloud have little density in existing clouds gets easier to fulfill. And two, that the memberships show the difference between the relatively level memberships of the Cauchy density and the more differentiated Gaussian density-based ones, which increase and decrease more over the same distance. This can best be seen for λ_2, which is essentially zero for LSTAT > 1.75.
Here, I want to point out another difference of the functions, which is the inference on outliers (or generally any points at the edges of the data space). In the standard one, the memberships

of an outlier will approach $1/N$ for every one of the N rules. This can be seen clearly in figure 3.14 (b), where memberships start to equalize for LSTAT values greater 18. This is because the difference between the densities get neglectable when the distances to all clouds are large. The result is that outliers are an average of all linear functions evaluated at the sample input. For the new function, the exponentiation makes even small relative distances noticeable, resulting in an asymptotic membership of one for the "nearest" cloud. This can be guessed from figure 5.5 (d) (see LSTAT greater than 1.75). Inference on outliers is therefore almost the same as the linear output of the nearest cloud. A sketch proof of these claims is given in appendix C. Which of the possible behaviors is preferred is context- and data-dependent, as is the answer to the question whether this matters at all.

The errors on the housing data set and the concrete data set (same preprocessing) are shown in

Density function	Concrete (std.)			Housing (std.)		
	MAE	RMSE	#rules	MAE	RMSE	#rules
Cauchy-Type	0.4752	0.5982	7	**0.6129**	**0.7629**	2
Gaussian-Type	**0.4585**	**0.5704**	4	0.8141	0.9936	4

Table 5.3.: Results of both density functions on the concrete and housing data sets.

table 5.3. On the concrete data, the functions performed equally good, with the Gaussian-Type reaching a slightly better error and surprisingly even less rules. This is explained by the removal of 4 rules during the training phase. On the housing data, the Cauchy-Type density performed arguably better than the Gaussian one, all while using half the number of rules.

All in all, choosing the density function between Cauchy- and Gaussian-Type is not the defining task in the application of an AnYa inference system. Even if the produced errors are not that much different, the number of rules can be very unlike. What needs to be known beforehand, is if the inference on the edges of the possible input data space should be defined by the nearest cloud and its linear function, which can be an advantage over the averaging of the Cauchy-Type density. A great disadvantage of the Gaussian-Type density lies in the necessity to evaluate several exponential functions every iteration. This can be both a performance hit and, mainly, it makes this approach less robust and prone to overflows or similar errors. When the incoming data is guaranteed to be bounded or standardized this might not be an obstacle.

5.4 Alternative cloud-creation condition

This extension is motivated by the results of the modifications described in section 5.1. There it can be seen that when using the standard rule creation conditions...

$$\forall i \in \{1,...,N\}: \qquad \Gamma^{(k)} > \Gamma_i \tag{1}$$

$$\forall i \in \{1,...,N\}: \qquad \gamma_i^{(k)} <= e^{-1} \tag{2}$$

... the update of the initial sample, which is used in the first condition, has at best little effect on the performance. In this section, I will explore an alternative condition that will replace (2), which relies on one of the possible updates of the representing sample.

Condition (2) ensures that the first sample of every rule is not "near" any clouds of existing

rules. If the representing sample of each cloud is updated using the "higher local density" strategy, there are other possible ways to formulate similar conditions. We know that it is possible to get the value of the local density $= 1/(1 + d^2_{avg})$ for any sample x, where d^2_{avg} is the average of the squared distances to all samples of a cloud. With the described update of the representing sample, the local density at that point is one of the highest of all samples of the cloud, and the average squared distance is one of the lowest (so is the average distance, since the distances are positive). This is illustrated in figure 5.6.

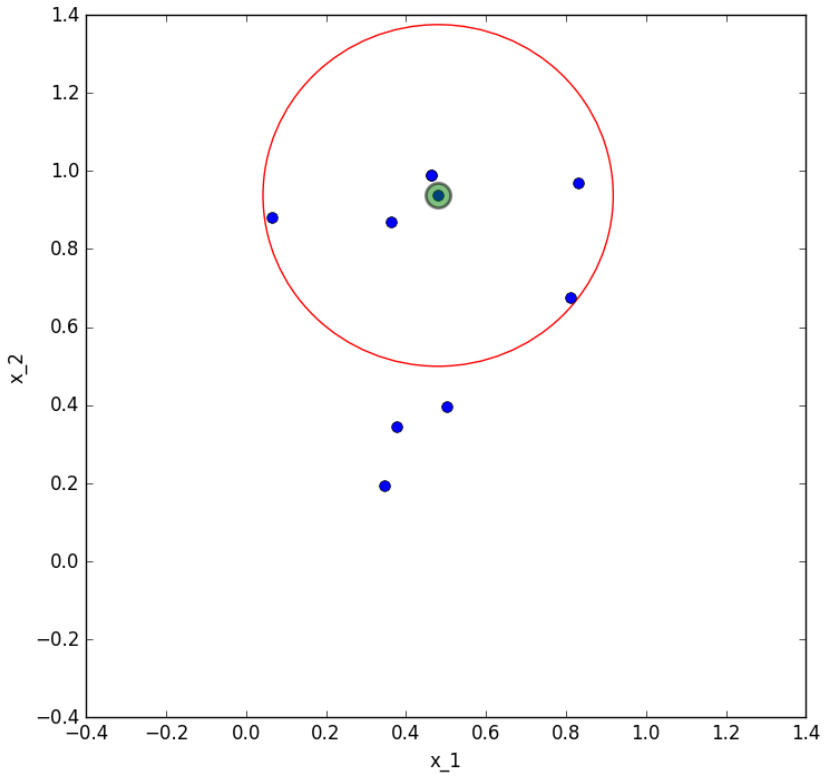


Figure 5.6.: A cloud where the representing sample was updated using the "higher local density" strategy. The highlighted green point is the current representing sample. The red circle is a visualization of the square root of the average squared distance to the other samples of the cloud.

Intuitively, if a sample should be considered for creating a new rule, it should at the very least be further away from the representing sample than the average distance to the other samples (marked red in the figure). Note that this radius is different (or it could be) for every cloud, unlike the fixed value used in the original condition (2). This could lead to the creation of various

clouds of different sizes, which may be a better model of the input space.
These thoughts can be used to formulate new rule creation conditions:

$$\forall i \in \{1, ..., N\}: \quad \Gamma^{(k)} > \Gamma_i^{repr} \tag{1}$$

$$\forall i \in \{1, ..., N\}: \quad d(x^{(k)}, x_i^{repr})^2 > d_{i\_avg}^2 c \tag{2}$$

$d(x^{(k)}, x_i^{repr})^2$ is the squared distance of the current sample
and the representing sample of the i-th cloud,
x_i^{repr} is chosen using the "higher local density" strategy,
$d_{i\_avg}^2$ is the average of the squared distances of the representing sample
to all the others from the i-th cloud,
and $c \approx 1$ is a positive constant

The constant c decides how much further away a sample has to be (regarding the squared distances). The first of the new conditions is still an indicator of how well a sample is a good generalization of the population, since x_i^{repr} is assumed to have a high local density and subsequently a relatively high global density.

These new conditions have been tested with different values of c on three already mentioned data sets: the concrete, housing, as well as the airfoil data set (all standardized). The results are shown in table 5.4.

value of c	Concrete (std.)			Housing (std.)		
	MAE	RMSE	#rules	MAE	RMSE	#rules
none, default conds	0.4752	0.5982	7	0.6128	0.7629	9
0.5	1.0264	1.3063	8	0.6090	0.8133	9
0.7	0.6504	0.8198	9	**0.4998**	**0.7274**	9
0.9	0.3745	0.4765	8	0.5181	0.7653	8
1.1	0.4226	0.5301	3	0.5630	0.7735	10
1.3	**0.3670**	**0.4672**	2	0.6154	0.7679	6
1.5	**0.3670**	**0.4672**	2	0.6190	0.7763	4

value of c	Airfoil (std.)		
	MAE	RMSE	#rules
none, default conds	0.6806	0.8636	4
0.5	0.7424	0.8994	7
0.7	0.7584	0.9316	7
0.9	0.6764	**0.8494**	4
1.1	**0.6605**	0.8559	4
1.3	0.6952	0.8807	3
1.5	0.6952	0.88067	3

Table 5.4.: Results of various values of c using the alternative rule creation conditions on the concrete, housing and airfoil data sets.

As expected, choosing a higher value for c generally leads to the creation of less rules. Also, the

evaluation showed that for each data set, there were at least some values for c that produce better results using the new over the standard conditions. For the housing and the concrete data set, the improvements were particularly good. Interesting is that sometimes the best results were found when using $c < 1$, meaning that, for example, the squared distance between current and representing sample just had to be greater than 0.7 times the average squared distances from the representing sample to the others. This could be a result of the outliers dominating the distance computation, when the real cloud (partition of the input space) should have less spread. In this case, the model benefits from an additional cloud in the vicinity.

Using the proposed new conditions can reduce the overall system error. Similar to the extension presented in section 5.2, this comes at the cost of introducing an additional parameter. However, the evaluation results suggest that even a non-optimal value for c that is chosen somewhere around 1 can lead to an error reduction in comparison to the original conditions.

6 Conclusion

This thesis presented the AnYa Fuzzy Rule-Based System with its non-parametric antecedents in a variant using locally linear rule outputs capable of online multi-target regression. Comparisons with several approaches traditionally used in machine learning applications have shown that AnYa is fit to produce well above baseline results in batch learning and good results for online learning scenarios.
Four extensions to the base system have been proposed, several of which can lead to decreased error measures.

AnYa was mainly developed with control application support in mind. Its small memory footprint and constant training/inference processing time also justify the use in other scenarios, namely stream mining or learning of big data sets.
Since the rule consequent can be adapted, other inference tasks like classifications can also be solved (and have been shown to perform well) with the same antecedents. How results of these compare to traditionally used algorithms should be evaluated further.
The kernel-driven similarity function does support all possible distance measures. If the chosen distance measure supports a recursive computation, then the small memory and processing power demands are preserved. The euclidean distance is one of the most used ones and the one implemented for this thesis. Future work should include benchmarks for other measures, for example the cosine distance, which is useful in the context of text mining.

One deciding factor for the quality of the predictions made by AnYa and similar approaches is the system structure. Even though each rules cloud may be a good cluster of similar samples on its own, the number, position and combination of these clouds is crucial for the systems performance, expressiveness and applicability. The conditions for creation and removal of rules proposed in the original publication are suboptimal for many data sets (mind the “no-free-lunch” theorem [Wolpert and Macready, 1997]), as can be seen in the extensions chapter. Further studies should be done in regard to these and alternatives that perform better on many data sets are likely to be found.

7 Tools and ressources

The implementation of an AnYa inference system, all benchmarks on it, the creation of all plots and the proposed extensions were realized using Python 3.5.2[1], in conjunction with NumPy[2] for numerical operations on matrices and Matplotlib[3] for plotting.
The performance was evaluated using data that is either taken from the Machine Learning Repository of the University of California, Irvine ([Lichman, 2013]) or whose source is cited in the respective section.
The frameworks Weka[4] ([Holmes et al., 1994]) and Moa[5] ([Bifet et al., 2010]) were used to obtain results of commonly used machine learning and stream mining algorithms.
For the replication of the results presented in this thesis, it should be noted that evaluation of the multi-target learners in Moa was done on the raw predictions and not in the framework itself, since the latest release at this time had problems with multi-target evaluation.

1 https://www.python.org/downloads/release/python-352/
2 http://www.numpy.org/
3 http://matplotlib.org/
4 http://www.cs.waikato.ac.nz/ml/index.html
5 http://moa.cms.waikato.ac.nz/

Appendices

A Simplification of RLS update equations

The relevant part of the matrix inversion lemma, called the Sherman-Morrison-Formula:

$$(A+uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1+v^TA^{-1}u}$$

where A is an invertible matrix and u, v are column vectors

Update of θ:

$$\begin{aligned}
\theta^{(k+1)} &= \left(C^{(k)^{-1}} + x^{(k+1)}x^{(k+1)^T}\right)^{-1}\left(C^{(k)^{-1}}\theta^{(k)} + x^{(k+1)}y^{(k+1)}\right) \\
&= \left(C^{(k)} - \frac{C^{(k)}x^{(k+1)}x^{(k+1)^T}C^{(k)}}{1+x^{(k+1)^T}C^{(k)}x^{(k+1)}}\right)\left(C^{(k)^{-1}}\theta^{(k)} + x^{(k+1)}y^{(k+1)}\right) \\
&= \theta^{(k)} + C^{(k)}x^{(k+1)}y^{(k+1)} - \frac{C^{(k)}x^{(k+1)}x^{(k+1)^T}\theta^{(k)}}{1+x^{(k+1)^T}C^{(k)}x^{(k+1)}} - \frac{C^{(k)}x^{(k+1)}x^{(k+1)^T}C^{(k)}x^{(k+1)}y^{(k+1)}}{1+x^{(k+1)^T}C^{(k)}x^{(k+1)}} \\
&= \theta^{(k)} + \frac{C^{(k)}x^{(k+1)}y^{(k+1)} + C^{(k)}x^{(k+1)}x^{(k+1)^T}C^{(k)}x^{(k+1)}y^{(k+1)}}{1+x^{(k+1)^T}C^{(k)}x^{(k+1)}} \\
&\quad + \frac{-C^{(k)}x^{(k+1)}x^{(k+1)^T}\theta^{(k)} - C^{(k)}x^{(k+1)}x^{(k+1)^T}C^{(k)}x^{(k+1)}y^{(k+1)}}{1+x^{(k+1)^T}C^{(k)}x^{(k+1)}} \\
&= \theta^{(k)} + \frac{C^{(k)}x^{(k+1)}}{1+x^{(k+1)^T}C^{(k)}x^{(k+1)}}\left(y^{(k+1)} - (x^{(k+1)})^T\theta^{(k)}\right)
\end{aligned}$$

Update of C:

$$\begin{aligned}
C^{(k)} &= \left(X^{(k)}X^{(k)^T}\right)^{-1} \\
C^{(k+1)} &= \left(\begin{bmatrix}X^{(k)} & x^{(k+1)}\end{bmatrix}\begin{bmatrix}X^{(k)^T} \\ x^{(k+1)^T}\end{bmatrix}\right)^{-1} \\
&= \left(C^{(k)^{-1}} + x^{(k+1)}x^{(k+1)^T}\right)^{-1} \\
&= C^{(k)} - \frac{C^{(k)}x^{(k+1)}x^{(k+1)^T}C^{(k)}}{1+x^{(k+1)^T}C^{(k)}x^{(k+1)}}
\end{aligned}$$

B Alternative update equation used in AnYa

$$C^{(k+1)}x^{(k+1)} = C^{(k)}x^{(k+1)} - \frac{C^{(k)}x^{(k+1)}x^{(k+1)^T}C^{(k)}x^{(k+1)}}{\underbrace{1 + x^{(k+1)^T}C^{(k)}x^{(k+1)}}_{:=s}}$$

$$\begin{aligned}
&= \frac{C^{(k)}x^{(k+1)}s}{s} - \frac{C^{(k)}x^{(k+1)}(s-1)}{s} \\
&= \frac{C^{(k)}x^{(k+1)}(s-(s-1))}{s} \\
&= \frac{C^{(k)}x^{(k+1)}}{1 + x^{(k+1)^T}C^{(k)}x^{(k+1)}}
\end{aligned}$$

This allows to write an alternative update equation:

$$\theta^{(k+1)} = \theta^{(k)} + C^{(k+1)}x^{(k+1)}\left(y^{(k+1)} - (x^{(k+1)})^T\theta^{(k)}\right)$$

C Proof sketch of outlier membership for Gaussian and Cauchy-based density

Suppose there are N rules/clouds with just one sample each. Let d_i be the distance of a sample x to the sample of the i-th cloud and cloud 1 be the nearest one. Let all $d_i, i \in \{2,\ldots,N\}$ equal $d_1 + c_i, c_i \in R^+$ (meaning there are further away). Then the following limit exists for the Cauchy-Type density function:

$$\begin{aligned}
\lim_{d_1\to\infty} \frac{\frac{1}{1+d_1^2}}{\sum_{i=1}^{N}\frac{1}{1+d_i^2}} &= \lim_{d_1\to\infty} \frac{1}{1 + \sum_{i=2}^{N}\frac{1+d_1^2}{1+(d_1+c_i)^2}} \\
&= \frac{1}{1+\sum_{i=2}^{N}1} = \frac{1}{N}
\end{aligned}$$

And for the one of Gaussian-Type:

$$\begin{aligned}
\lim_{d_1\to\infty} \frac{exp(-d_1^2)}{\sum_{i=1}^{N}exp(-d_i^2)} &= \lim_{d_1\to\infty} \frac{exp(-d_1^2)}{\sum_{i=1}^{N}exp(-d_i^2)} \\
&= \lim_{d_1\to\infty} \frac{1}{1 + \sum_{i=2}^{N}\frac{exp(d_1^2)}{exp((d_1+c_i)^2)}} \\
&= \lim_{d_1\to\infty} \frac{1}{1 + \sum_{i=2}^{N}\frac{exp(d_1^2)}{exp(d_1^2)exp(2d_1c_i)exp(c_i^2)}} \\
&= \frac{1}{1+\sum_{i=2}^{N}0} = 1
\end{aligned}$$

These limits visualize why the membership approaches $1/N$ in the first case (for the nearest cloud, can also be shown for the others) and 1 for the second.

Bibliography

[Akbilgic et al., 2014] Akbilgic, O., Bozdogan, H., and Balaban, M. E. (2014). A novel Hybrid RBF Neural Networks model as a forecaster. *Statistics and Computing*, 24(3):365–375.

[Almeida et al., 2013] Almeida, E., Ferreira, C., and Gama, J. (2013). Adaptive model rules from data streams. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 480–492. Springer.

[Angelov, 2010] Angelov, P. (2010). Evolving takagi-sugeno fuzzy systems from streaming data (etsþ). *Evolving intelligent systems: Methodology and applications*, 12:21.

[Angelov and Filev, 2005] Angelov, P. and Filev, D. (2005). Simpl_ets: a simplified method for learning evolving takagi-sugeno fuzzy models. In *The 14th IEEE International Conference on Fuzzy Systems, 2005. FUZZ'05.*, pages 1068–1073. IEEE.

[Angelov and Yager, 2011] Angelov, P. and Yager, R. (2011). Simplified fuzzy rule-based systems using non-parametric antecedents and relative data density. In *Evolving and Adaptive Intelligent Systems (EAIS), 2011 IEEE Workshop on*, pages 62–69. IEEE.

[Angelov and Yager, 2012] Angelov, P. and Yager, R. (2012). A new type of simplified fuzzy rule-based system. *International Journal of General Systems*, 41(2):163–185.

[Angelov and Zhou, 2008] Angelov, P. and Zhou, X. (2008). On line learning fuzzy rule-based system structure from data streams. In *Fuzzy Systems, 2008. FUZZ-IEEE 2008.(IEEE World Congress on Computational Intelligence). IEEE International Conference on*, pages 915–922. IEEE.

[Angelov and Filev, 2004a] Angelov, P. P. and Filev, D. P. (2004a). An approach to online identification of takagi-sugeno fuzzy models. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(1):484–498.

[Angelov and Filev, 2004b] Angelov, P. P. and Filev, D. P. (2004b). Flexible models with evolving structure. *International Journal of Intelligent Systems*, 19(4):327–340.

[Atkeson et al., 1997] Atkeson, C. G., Moore, A. W., and Schaal, S. (1997). Locally weighted learning for control. In *Lazy learning*, pages 75–113. Springer.

[Bifet et al., 2010] Bifet, A., Holmes, G., Kirkby, R., and Pfahringer, B. (2010). Moa: Massive online analysis. *Journal of Machine Learning Research*, 11(May):1601–1604.

[Box and Jenkins, 1970] Box, G. E. P. and Jenkins, G. M. (1970). The gas furnace data set from box and jenkins' book on time series analysis (series j). `http://openmv.net/info/gas-furnace`. Accessed: 2016-08-03.

[Brooks et al., 1989] Brooks, T. F., Pope, D. S., and Marcolini, M. A. (1989). *Airfoil self-noise and prediction*, volume 1218. National Aeronautics and Space Administration, Office of Management, Scientific and Technical Information Division.

[Cleveland, 1979] Cleveland, W. S. (1979). Robust locally weighted regression and smoothing scatterplots. *Journal of the American statistical association*, 74(368):829–836.

[Englert, 2012] Englert, P. (2012). Locally weighted learning.

[Gama et al., 2009] Gama, J., Sebastião, R., and Rodrigues, P. P. (2009). Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 329–338. ACM.

[Holmes et al., 1994] Holmes, G., Donkin, A., and Witten, I. H. (1994). Weka: A machine learning workbench. In *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, pages 357–361. IEEE.

[Ikonomovska et al., 2011] Ikonomovska, E., Gama, J., and Džeroski, S. (2011). Learning model trees from evolving data streams. *Data mining and knowledge discovery*, 23(1):128–168.

[Jana et al., 1996] Jana, A., Yang, P., Auslander, D., and Dave, R. (1996). Real time neuro-fuzzy control of a nonlinear dynamic system. In *Fuzzy Information Processing Society, 1996. NAFIPS., 1996 Biennial Conference of the North American*, pages 210–214. IEEE.

[Jang, 1993] Jang, J.-S. (1993). Anfis: adaptive-network-based fuzzy inference system. *IEEE transactions on systems, man, and cybernetics*, 23(3):665–685.

[Jang, 1996] Jang, J.-S. R. (1996). Input selection for anfis learning. In *Proceedings of the fifth IEEE international conference on fuzzy systems*, volume 2, pages 1493–1499. Citeseer.

[Lichman, 2013] Lichman, M. (2013). UCI machine learning repository.

[Loh, 2011] Loh, W.-Y. (2011). Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):14–23.

[MacQueen et al., 1967] MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

[Mamdani and Assilian, 1975] Mamdani, E. H. and Assilian, S. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. *International journal of man-machine studies*, 7(1):1–13.

[Nadaraya, 1964] Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142.

[Osojnik et al., 2015] Osojnik, A., Panov, P., and Džeroski, S. (2015). Multi-label classification via multi-target regression on data streams. In *International Conference on Discovery Science*, pages 170–185. Springer.

[Pelleg et al., 2000] Pelleg, D., Moore, A. W., et al. (2000). X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML*, volume 1.

[Plackett, 1950] Plackett, R. L. (1950). Some theorems in least squares. *Biometrika*, 37(1/2):149–157.

[Setiono and Liu, 1997] Setiono, R. and Liu, H. (1997). Neural-network feature selector. *IEEE transactions on neural networks*, 8(3):654–662.

[Takagi and Sugeno, 1985] Takagi, T. and Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control. *IEEE transactions on systems, man, and cybernetics*, (1):116–132.

[Tikka et al., 2005] Tikka, J., Hollmén, J., and Lendasse, A. (2005). Input selection for long-term prediction of time series. In *International Work-Conference on Artificial Neural Networks*, pages 1002–1009. Springer.

[Vijayakumar and Schaal, 2000] Vijayakumar, S. and Schaal, S. (2000). Locally weighted projection regression: An o (n) algorithm for incremental real time learning in high dimensional space. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, volume 1, pages 288–293.

[Watson, 1964] Watson, G. S. (1964). Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 359–372.

[Welford, 1962] Welford, B. (1962). Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420.

[Wolpert and Macready, 1997] Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82.

[Woodbury, 1950] Woodbury, M. A. (1950). Inverting modified matrices. *Memorandum report*, 42:106.

[Yeh, 1998] Yeh, I.-C. (1998). Modeling of strength of high-performance concrete using artificial neural networks. `https://archive.ics.uci.edu/ml/machine-learning-databases/concrete/compressive/`. Accessed: 2016-08-03.

[Yen et al., 1998] Yen, J., Wang, L., and Gillespie, C. W. (1998). Improving the interpretability of tsk fuzzy models by combining global learning and local learning. *IEEE Transactions on Fuzzy Systems*, 6(4):530–537.

[Zadeh, 1965] Zadeh, L. A. (1965). Fuzzy sets. *Information and control*, 8(3):338–353.

[Zamora-Martínez et al., 2014] Zamora-Martínez, F., Romeu, P., Botella-Rocamora, P., and Pardo, J. (2014). On-line learning of indoor temperature forecasting models towards energy efficiency. *Energy and Buildings*, 83:162–172.

[Zhang et al., 2005] Zhang, Y., Li, H., Hou, A., and Havel, J. (2005). Artificial neural networks based on genetic input selection for quantification in overlapped capillary electrophoresis peaks. *Talanta*, 65(1):118–128.

[Zhang, 2007] Zhang, Y. X. (2007). Artificial neural networks based on principal component analysis input selection for clinical pattern recognition analysis. *Talanta*, 73(1):68–75.