TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Page

Page

LIST OF ABREVIATIONS

AdaBoost	Adaptive Boosting
ART	Adaptive Resonance Theory
ART-2A	Adaptive Resonance Theory for analog patterns
ARTIST	ART-2A driven rule management in Takagi-Sugeno Fuzzy model
ASC	Applied Soft Computing
CCL	Competitive Consequent Learning
CI	Confidence Interval
DNA	Deoxyribonucleic acid (dataset of)
ED	Euclidean Distance
EML	Elastic Memory Learning
EML+	Elastic Memory Learning +
ETS	École de Technologie Supérieure
ETS	Evolving Takagi-Sugeno
ETS+	Evolving Takagi-Sugeno +
HS	Handwritten Symbols
IFM	Incremental Fuzzy Model
ICDAR	International Conference on Document Analysis and Recognition
IEEE	The Institute of Electrical and Electronics Engineers
IJCNN	International Joint Conference on Neural Networks

IL	Incremental Learning
IS	Incremental Similarity
ISSPA	Information Science, Signal Processing and their Applications
JMLC	International Journal of Machine Learning and Cybernetics
K-means	machine learning model for clustering
K-NN	K Nearest Neighbors
MD	Mahalanobis Distance
MNIST	Mixed National Institute of Standards and Technology database
MOA	Massive Online Analysis
NSERC	Natural Sciences and Engineering Research Council
GD	Gradient Descent
OHS	Online Handwritten Symbols
ORF	Online Random Forest
PCA	Principal Component Analysis
PCR	Principal Component Regression
RLS	Recursive Least Squares
RGPIN	Discovery Grant Program
RR	Recognition Rate
SGD	Stochastic Gradient Descent
SI	Satelite Images

SO-ARTIST	Self-Organized ARTIST
SOM	Self-Organized Maps
SOTA	State-Of-The-Art
SSHRC	Social Sciences and Humanities Research Council
SV	Support Vector
SVD	Singular Value Decomposition
SVM	Support Vector Machines
TS	Takagi-Sugeno
UCI	University of California
USPS	United States Postal Service (digits from)
VQ	Vector Quantization

INTRODUCTION

In this chapter we introduce the online learning problem and its various settings that influence the type of solution chosen. Further, we pose a number of problems and challenges of online learning that as well serve as our motivation for our work.

0.1 Context of the thesis

Online learning is a part of machine learning that unlike offline learning performs the learning and recognition steps simultaneously, at every iteration, see Figure 0.1. Thus, the recognition is not delayed by the learning process to be finished. Many online learning algorithms are based on incremental learning, however this implication is not mutual. Incremental learning is based on recursive updates, where the model does not store the past data, but rather keeps a set of parameters describing the processed data. Thus, most of the incremental learning methods are parametric. However, there are semi-incremental methods that store some portion of the data, either due to the nature of these models by being non-parametric or due to the proper functionality. We will describe some of these methods in Chapter 1. Besides incremental learning methods, there are online adaptive ones that do not rely on recursive updates, but rather adapt the parameters to the environment.

When we approach a problem with online learning solution, we face stability-plasticity dilemma. This tells us how much we want the model to rely on the old information and be more reserved in the predictions, i.e. how much we want to avoid adapting to an unimportant impulses for stability vs. being more flexible and adapting to the very new data, i.e. learning the important impulses for plasticity. We can see a slight difference in incremental and adaptive methods. For incremental learning the aim is to remember all the past data and converge to the true state of the environment, and tend to aim more towards the stability (unless the parameters responsible for the convergence are set otherwise). Adaptive methods do not necessarily rely on the past

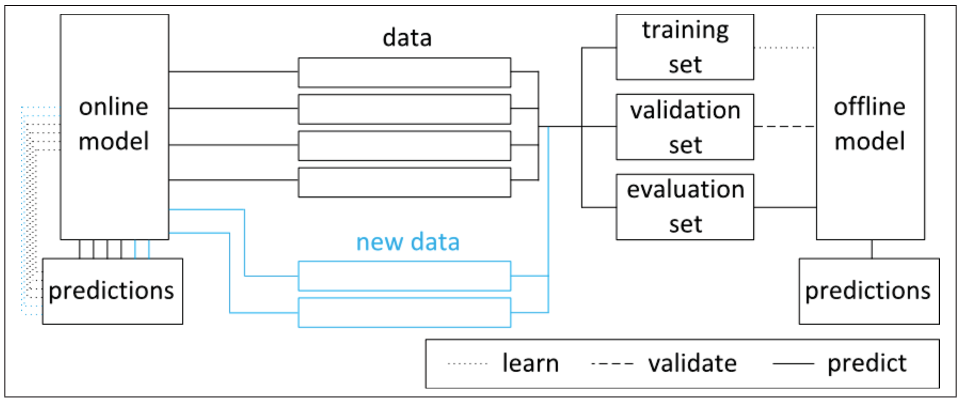

Figure 0.1 Offline learning vs Online learning. In online learning the data is processed as a sequence, whereas in offline learning in a bulk. In new data occurs, they are treated as another data in a sequence for online learning. In offline learning, new data need to be added to the old data and the whole model needs to be retrained.

data, but rather adapt to the changes in a dynamic way expecting the state of the environment to be more current, aiming rather for the plasticity part of the problem. Based on the learning approach, we may divide the models into four groups from which in this work we will focus on the first two.

- Online incremental learning. The learning and recognition are performed simultaneously at each iteration. The model learns a set of parameters that describe the processed data. While these models might seem to be more stable than plastic compared to adaptive ones, one needs to pay attention to the parameters related to the stability-plasticity, e.g. the learning rate. Once the data is processed it is not stored anymore. These models have the advantage of low memory cost and fast processing, however they can lack the accuracy. Usually, the more robust model is, the more computationally costly it is.

- Online adaptive learning. The learning and recognition are performed simultaneously at each iteration. The model adapts to the current state of the environment rather than describing the expected value of the world by parameters. Thus, these models, if not treated carefully, might lack stability while focusing on plasticity. Once the data is processed it is not stored anymore. Similarly, these models have the advantage of low memory and processing costs. However, besides the possibility of lower accuracy they might pay for the adaptive nature of the learning and be prone to the low stability.

- Offline incremental learning. The learning and recognition are performed in two stages. The learning is performed in a recursive manner, and the already processed data is lost. The free parameters of the model are fixed by cross-validation. While these models are preferred for their low memory and processing costs, they can lack in the accuracy as they do not have access to the whole batch of data and might not see the true structures within. However, in some cases, incremental learning can avoid falling into local minima, especially when the data in the sequence are uniformly distributed. Since only after the whole learning process is finished, the recognition takes a place, if new change in the environment occurs the model needs to be updated.

- Offline batch learning. The learning and recognition are performed in two stages. During the learning process all the data is available and used as a bunch to derive parameters. Further free parameters are then fixed using cross-validation. These models tend to have higher accuracy, but also higher memory and computational costs. They are not able to perform recursive updates, and thus f new situation in the environment occurs they need to be learned again, with both old and new data.

In pattern recognition we can find many applications of online learning. Especially with the growth of data and the aim towards user-friendliness of the systems. Our main motivation from the application point of view is recognition of handwritten gestures. Here, the gestures are not given at the beginning, but they appear within a sequence of occurrences. Another interesting application of online learning is the prediction of the internet data, whose size is not only big, requiring processing with low computational cost, but they appear at very dense frequencies. In Chapter 1 we will discuss various online learning methods that can be applied for various problems, keeping in mind that each model fits best to only a certain class of applications, as there is not one hammer for every nail.

0.2 Problem statement

In our work, we focus on online learning to deal with requirements of dynamic nature of the environment. Such requirements include that the data is not known ahead and does not appear at once, but in a sequence of occasions. Thus, we need to be able to accommodate such changes and to be able to process data in a sequences rather than one bulk. Since the data is not known, the system cannot be told by the teacher when the classes will appear and what form will they take. Our knowledge of the problem is limited to the type of the application only. At the same time, the system needs to be able to learn in a systematic way without forgetting past knowledge, but also adequately adapting to the new state of the environment. In many applications, the cost of the learning is important as well as the accuracy of the recognition and the aim is to find the balance in these two. We address these problems in detail in the following subsections.

0.2.1 Dynamic changes in the environment, learning from scratch and on the fly

In many applications, especially with the raise of the use of internet, the environment that the machine intends to find description in is changing dynamically and many new data occur in it, see Figure 0.2. To keep the predictions about the states of the environment up-to-date, it is best to adapt to all these changes once they come to happen. One option is to wait for sufficient amount of data and then use these along with the old data to retrain the model of the environment. This is the approach of traditional offline models that require to learn the whole model from the beginning. This requires significant amount of time for the delay of the system to be used for predictions. A better option seems to be to update the model every time there is a change in the state of the environment and to be able to accommodate new structures of the data in the model. Many known offline models require, by definition, initialization of the number of the structures in the data and use methods such as cross-validation to determine these in case they are not known. This seems to be a problem for many online models, since the number of the classes needs to be set ahead and thus there is no possibility for learning on the fly.

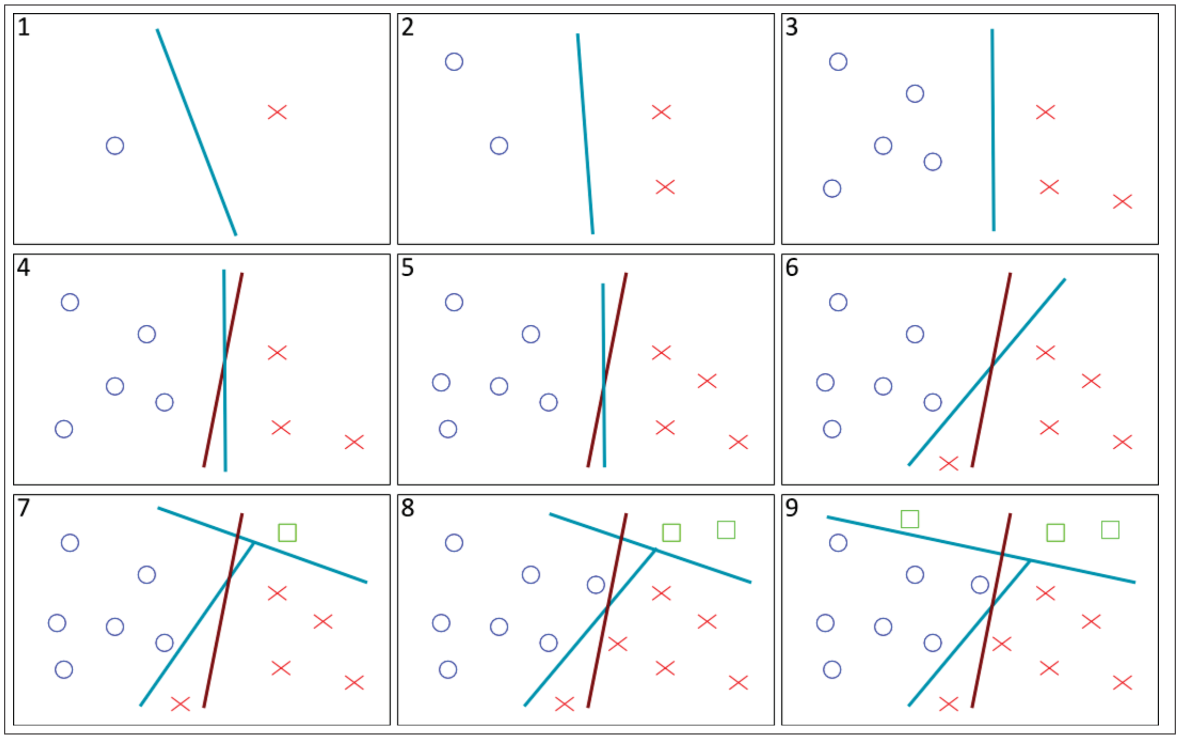

Figure 0.2 The problem of dynamic changes in the environment. In image 1 first samples are introduced to the system, where the online model (blue decision boundaries) learns from scratch, however, offline model (red decision boundary) need to wait to gather data to create the decision boundary. As the data amount appear, the online model adapts, but offline model is already set from image 4. In image 6 we note a miss-classification due to the static decision boundary in offline model, however online model adjusts its decision boundaries accordingly. In images 7-9 we see a new class to appear, where the offline model cannot cope with this new structure in the data, whereas the online model adapts to such change.

0.2.2 Accuracy vs. Processing time

For many applications, the use of online learning is purposed to be real-time, when the response or the prediction of the system to the environment needs to be fast. Furthermore, the use of incremental learning for offline learning is due to the saving of the processing time, when classical batch learning is too slow. Thus, in online and also incremental learning models, the machine needs to rely on simple updates that are fast, but also result into accurate predictions. In parametric learning a distribution of the data is assumed, such that this distribution is described by the parameters. Usually, an expected value, such as the mean is used accompanied with a variance or a covariance. By comparing these two we can see, that in univariate distributions, the mean and variance ignore covariances among the data as well as the variances for each dimension separately. There are methods, such as Naive Bayes, that take into account the variances within each dimension, but still lack the covariances. In some application this might not be an issue and the models based on univariate distributions can benefit from the low computational cost. But in many other applications a covariances between variables carry information that can be crucial for the whole learning and prediction process. On the other hand, introducing covariance brings increased computational cost, which can be detrimental for the real-time processing.

0.2.3 Forgetting

Since online learning relies on small updates and at each time step uses only sample of size one, the system cannot look at the data as a whole. Thus, instead of extracting the statistics from the whole sample, we assume that by these small updates we will converge to the expected value. Usually we need to set a rate at which the system learns or adapts to the new input. This rate tells us how agile the learning is. Setting this is an issue of stability-plasticity dilemma. We need to be able to decide, whether we want to rely on all data already known, making more stable decisions, or whether we want to be agile and flexible by adapting to the new state of the environment. In other words, ideally we want to avoid learning an unimportant event that brings only noise to the data, but at the same time be able to adapt to an important event. This dilemma goes hand in hand with the problem of forgetting. In online learning, the data stream continues and the old samples are usually not revisited. In multi-class problems, there are two basic strategies how to transform two-class models into multi-class establishment, and this is by keeping the true class as the positive class and either letting all the other classes as negative classes (one vs all) or doing pairwise models by choosing always only one other class as the negative class, and doing this for all possible combinations (one vs one). Either way, if one class is not used for a longer period of time, it becomes sparse compared to the other class/classes and basically is overrun by the negative examples, see Figure 0.3. This shifts the decision boundary so that the positive class is not longer recognizable.

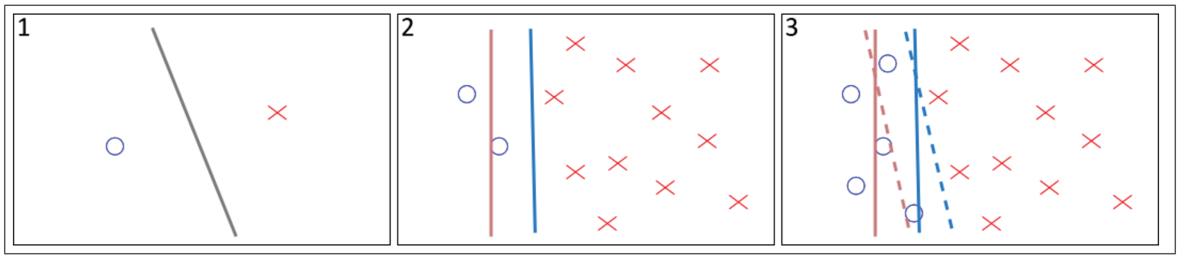

Figure 0.3 The problem of forgetting. In the image 1 we can see a decision boundary when starting from scratch. After some time, image 2, one class is not used for a longer period of time and the decision boundary, red, is shifting towards the forgotten class. As we can see in the image 3, the new decision boundary, dashed, is not changing dramatically and still causes confusion. The blue decision boundaries are from a model that is immune to the forgetting, making it fast to adapt to new examples while not being confused.

0.2.4 Parameter-free modeling

One of our motivations is to be able to build user-friendly model, that is capable of adapting to new situations and to respond to them adequately, with as little human/users involvement as possible. This results into the need of being able to adapt to data that has not been seen ahead and our knowledge of the environment is restricted to the task the system is coping with. For example, in handwritten gesture recognition all we know is that the data are going to be handwritten gestures, but we do not know how they will look like, how many classes there will be or when they will be added to the system. We cannot say for sure how fast the system needs to learn, because we do not know how the environment will change in the future. We cannot rely on cross-validation, because we cannot rely on previously learned model. Thus, it is essential to avoid free/hyper parameters and let the model to adapt to the environment itself, using only the knowledge that will not change in time (i.e. the application). Many models rely on hyper parameters and the expertise of the teacher to be able to set these parameters. But this does not fit to the nature of online learning, and thus models that are able to adapt themselves are in need for many online learning based applications.

0.2.5 Missing data

So far, we have been focusing on the overall learning capabilities resulting in a stable system. However, an important part of the process is also the initial state, where the system needs to learn from very few examples, see Figure 0.4. Such setup can, and usually does, result in over-fitting and a high variance. In offline learning setup we can discover the variance on the testing/evaluation stage, after the model is trained. However, in online learning setup we do not have the testing data, our testing data is all the data. Thus, we can see the over-fitting in a poorer performance, when the model is not robust enough to predict the new example caused by the low generalization abilities. Thus, at each time t, all the data until this time instance are the training data and the next data point is the testing data point. In the learning on the fly and learning from scratch setups, we usually have only one example per class to begin with, and the classes are not initialized at the beginning, they appear during the learning/recognition process. Therefore, when we learn from only one example, the next example which is not the very same one, is not going to fit well to the model that the system has built for its class.

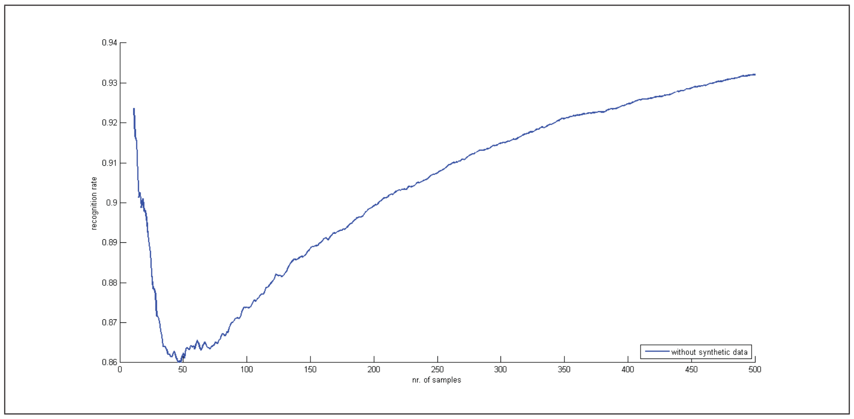

Figure 0.4 The problem of missing data. At the beginning of the learning/recognition process occurs the lack of data resulting into high variance visible by the drop in recognition rate in the figure.

0.3 Contributions

In this work we tackle several problems stated in previous sections. In general, we solve the online learning problem that involves different aspects of this kind of machine learning. First of all, in online learning, the data need to be processed in a sequence, and thus a model needs to be able to perform simple and one-example based updates. We accomplish this goal by proposing and developing few novel models successfully published in international journal and conferences.

With our models we successfully tackle the learning on the fly restriction, when the classes need to be added during the processing of the data. The models that we introduce do allow addition of new classes on the fly as well as addition of new structures. This is important aspect of the online learning in case when the classes are not known ahead and we simply cannot initialize the model using knowledge we do not posses.

In our later work we tackle accuracy vs processing time problem. We successfully develop an incremental similarity measurement that is capable of avoiding high computational cost while remaining at high accuracy level. We apply this method to all our models and improved the overall results.

Since in online learning the data are processed in a sequence, it is not easy to avoid the forgetting of unused classes. In the stream, in real-time recognition, the classes do not occur uniformly distributed and we cannot influence the distribution of the classes in the stream. Thus, some classes can become very sparse in some periods of time, resulting into their forgetting by the model. In our research we successfully develop a strategy for optimization that is immune to forgetting and we apply this into our models as well as to other state of the art models.

Another requirement that we will tackle is the problem of free parameters. These parameters require expert fixing before the prediction process. In offline learning a cross-validation is used for this purpose. However in online learning there is no room for cross-validation and the only other options are to learn the parameters, avoid the parameters or assume the parameters. To avoid the parameters require specific models that do not need too many parameters to store the information extracted from the data. In our approach we choose both learning and avoiding the parameters and we successfully develop a self-organized model. This model is capable of organization of its structures based on the current and past data as well as adapt all necessary parameters.

Lastly, one of our main goals was the learning from scratch which results from no initialization requirement for online learning. Many techniques require this step such that the models can start with already some knowledge and lead to accurate results. We successfully tackle this requirement by introducing models that are capable of starting from scratch. At the same time, we tackle the problem of high variance that results from this requirement, by introducing the framework for generation of synthetic data. For this purpose we explore various aspects of this approach, e.g. the length of the generation of the synthetic data and their volume. The aim of this research is to find the best setting when the generation of synthetic data improves the generalization ability of the model, but at the same time does not cause extra processing expenses.

Our solutions are applied for various applications, using high amount of machine learning benchmarks. The main application that we focus on in this work are the handwritten gestures. We successfully use our models for this application and solve specific corresponding problems.

0.4 Outline of the thesis

In this work we discuss the online learning, problems that result from it and the solutions that we bring to solve these problems. Thus this work is structured as follows:

- In **Literature review** we discuss the basic state of the art methods that solve the online learning restriction. However, these models still do not solve all the problems that we have posed in the Introduction, leading to the Methodology section. We describe each group of methods and explain which problem still remains when using it. More detailed literature review can be found in the chapters concerning the journal publications.
- **General Methodology** describes the methodology of our work, putting all our publications together. At first we pose our specific objectives of this work that we follow in the methodology section in order to solve it. In the methodology section we describe our solutions on a higher level such that the reader will get the main idea of our work and how we tackle the objectives posed in this work. More detailed methodology is accessible in the following chapters related to our journal publications.

- **Journal publications** are four chapters dedicated to our journal publications. Here we describe our incremental similarity metric, the elastic memory learning for forgetting, the self-organized model and the generation of synthetic data.
- In **General Discussion** we gather all the problems and objectives of our thesis with corresponding solutions. We discuss the results of our work according to our objectives along with the advantages and disadvantages of our works.
- **Conclusion and Recommendations** wraps up our work with the focus on each of our main contributions and pose problems for future works.

CHAPTER 1

LITERATURE REVIEW

In this Chapter we briefly describe and discuss machine learning methods tackling online learning with the emphasis on problems stated in Section 0.2. The online learning is a special case of machine learning when the learning and prediction occur at each iteration and the data are thus processed in a sequence. In this chapter we divide online learning methods into five groups according to their nature, and list a number of representative models for each.

1.1 Online learning methods

Similarly to all machine learning methods, online learning based ones can be viewed from different aspects, not only by the way they learn (incremental vs. adaptive), but also by the way they describe the data and the environment. To address these differences, we list few models from selected groups of machine learning methods that are capable of online learning and briefly explain their functionality.

1.1.1 Online linear methods

Linear models for classification as well as for regression expect linearity within the data, i.e. the data are either linearly separable or the data can be modeled using a linear combination of some parameter. In this section we will describe solutions for finding the parameters for both, classification and regression.

In the case of linear regression, hypothesis expects the data to be modeled by a line, i.e. all the data points lie roughly on a line. However, by inserting polynomial features we can break this assumptions, allowing to model the data with polynomial curves and bringing more complexity to the solution. The hypothesis for linear regression calculates a linear combination of parameters given the model and data 1.1.

$$h(x|\pi_1,\pi_2,...,\pi_n) = \pi_0 + \pi_1 x_1 + \pi_2 x_2 + ... + \pi_n x_n \tag{1.1}$$

$$h(x|\Pi) = \begin{pmatrix} \pi_0 \\ \pi_1 \\ \pi_2 \\ \vdots \\ \pi_n \end{pmatrix}^T \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \Pi^T X \tag{1.2}$$

Then, the cost function is usually a mean squared error between prediction and the actual state 1.3.

$$J(\Pi|X) = J(\Pi|x^1,x^2,...x^m) = \frac{1}{2m}\sum_{i=1}^{m}\left(h(x^i|\Pi) - y^i\right)^2 \tag{1.3}$$

For the case of classification, the use of regression can be detrimental for various reasons. A way to apply linear regression for classification problems is to model the two-class problem, or a set of two-class problems. Each sub-problem than models both two classes as a line or a polynomial and sets a threshold to determine the true class based on the output of the hypothesis. In the case of multiple classes, and thus multiple two-class models, we can simply choose the highest value to be the true class. Thus, the aim is to model true class to result into 1 and all the other classes to result into 0. Normally, a simple way how to set the threshold is to set it to the middle value, 0.5. However, in case of outliers, the model, line or polynomial, will be shifted towards these, making the threshold not straightforward to adjust. Also, the outputs are not bounded within the interval $[0,1]$.

Thus, in general, different hypothesis is used for classification problems, i.e. the logistic regression. Here, a logit function is applied to the original hypothesis 1.4, transforming it so that the output lies within the interval $[0,1]$ and the model now represents a decision boundary between the true and false class.

$$h(x|\Pi) = \frac{1}{1+e^{-\Pi^T X}} \tag{1.4}$$

Then, to get the final decision, we apply the threshold θ to the hypothesis 1.5.

$$y = \begin{cases} 1 & h(x|\Pi) > \theta \\ 0 & othewrve \end{cases} \tag{1.5}$$

The cost function takes into account the miss-classification, such that in case of correct classification the cost will be 0 and in case of miss-classification the cost will go to infinity 1.6.

$$J(\Pi|X) = \begin{cases} -\log h(X|\Pi) & y = 1 \\ -\log(1-h(X|\Pi)) & y = 0 \end{cases} \tag{1.6}$$

We can then rewrite the cost function as a cross-entropy 1.7.

$$J(\Pi|X) = -\sum_{i=1}^{m} \left[y^i \log h(X|\Pi) + (1-y^t)\log(1-h(X|\Pi)) \right] \tag{1.7}$$

To find the solution for parameters Π, we need to optimize the cost function. Given the cost function, i.e. 1.3 for regression and 1.7 for classification, we want to find parameters Π that minimize this cost function. In case of offline learning the cost function is optimized on the whole sample X, i.e. in a batch mode. In online learning the optimization needs to be able to perform recursive updates, such that it is capable of processing data as a stream. In the following, we will describe two basic approaches for both linear and logistic regression, i.e. Stochastic Gradient Descent (SGD) and Recursive Least Squares (RLS).

1.1.1.1 Stochastic gradient descent

To find the best parameters Π 1.9 we minimize the cost function $J\left(\Pi|X\right)$ by putting its first derivative to zero. In traditional gradient descent (GD) (Snyman, 2005), we optimize over all X 1.8, obtaining an update 1.11 for the parameters Π 1.12.

$$X = \left\{ \begin{array}{cccc} \boldsymbol{x}^1 & \boldsymbol{x}^2 & \cdots & \boldsymbol{x}^m \end{array} \right\} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1^1 & x_1^2 & \cdots & x_1^m \\ x_2^1 & x_2^2 & \cdots & x_2^m \\ \vdots & \vdots & \ddots & \vdots \\ x_n^1 & x_n^2 & \cdots & x_n^m \end{pmatrix} \tag{1.8}$$

$$\Pi = \begin{pmatrix} \boldsymbol{\pi}^0 \\ \boldsymbol{\pi}^1 \\ \boldsymbol{\pi}^2 \\ \vdots \\ \boldsymbol{\pi}^n \end{pmatrix} \tag{1.9}$$

$$Y = \left\{ \begin{array}{cccc} y^1 & y^2 & \cdots & y^m \end{array} \right\} \tag{1.10}$$

$$\frac{\partial J\left(\Pi|X\right)}{\partial \Pi} = X\left(h\left(X|\Pi\right) - Y\right)^T \tag{1.11}$$

$$\Pi = \Pi - \alpha \frac{\partial J\left(\Pi|X\right)}{\partial \Pi} = \Pi - \alpha X\left(h\left(X|\Pi\right) - Y\right)^T \tag{1.12}$$

In case of linear regression, we replace the hypothesis $h\left(X|\Pi\right)$ using equation 1.1 and in case of logistic regression using equation 1.4. To obtain the best parameters, we iteratively update parameters Π, where each iteration optimizes over all data points X. We initialize the parameters Π to small values, and the iteration stops when the cost function is below a threshold. The parameter α denotes the increment used for updates and can be understood as a learning rate set within the interval $[0,1]$. To determine this parameter along with the stop criterion and the decision threshold , we use cross-validation.

For stochastic gradient descent (Bottou, 2010) we do not optimize over all data points X, but for each data point separately, and incrementally update parameters Π. Thus, instead of X we only use x^i for $i = 1...m$, resulting into new updating formula 1.13.

$$\Pi_{i+1} = \Pi_i - \alpha \frac{\partial J\left(\Pi_i | \boldsymbol{x}^i\right)}{\partial \Pi_i} = \Pi_i - \alpha \boldsymbol{x}^i \left(h\left(\boldsymbol{x}^i | \Pi_i\right) - y^i\right)^T \tag{1.13}$$

As for GD, α denotes the learning rate. The algorithm however slightly changes. Instead of one update at each iteration, we perform m updates. We repeat these updates for a number of iterations to result into the proper parameter vector Π.

The advantage of stochastic gradient descent over traditional gradient descent methods lies in its small incremental updates. To obtain the derivative on data, when m is too big, the complexity grows significantly. However, in the case of SGD, the m is not longer a part of the dot product in the update, but rather an iteration limit. Also, in traditional GD, the algorithm can fall in a local minima and since the cost function will grow in case of trying to explore beyond the local minima, the algorithm will terminate resulting into a not best solutions. In SGD, the updates are more 'chaotic' since they are based on single data points, but this allows it to avoid local minima increasing the chances of finding global minima.

For our application, SGD is an interesting solution for linear and logistic regression, however still requires fixing some parameters (e.g. α), using cross-validation or using a knowledge about data. In addition, to be able to perform real-time learning and recognition, we cannot keep the data and allow iterations to find the best parameters, i.e. two loops of iterations - over all data (inner loop) and until convergence it obtained (outer loop).

1.1.1.2 Recursive Least Squares

Another way for learning the parameters Π is by minimizing the squared cost function 1.14 resulting into Recursive Least Squares optimization (Ljung, 1999). Here we define the equations to derive the parameters Π 1.15 using Gain matrix C^{-1} 1.18.

$$J(\Pi|X) = J\left(\Pi|\boldsymbol{x}^1, \boldsymbol{x}^2, ..., \boldsymbol{x}^m\right) = \sum_{i=1}^{m}\left(h\left(\boldsymbol{x}^i|\Pi\right) - y^i\right)^2 \tag{1.14}$$

$$\Pi_t = \sum_{i=1}^{t}\boldsymbol{x}^i y^i \left(\sum_{i=1}^{t}\boldsymbol{x}^i\left(\boldsymbol{x}^i\right)^T\right)^{-1} = M_t C_t^{-1} \tag{1.15}$$

$$M_t = \sum_{i=1}^{t}\boldsymbol{x}^i y^i = \sum_{i=1}^{t-1}\boldsymbol{x}^i y^i + \boldsymbol{x}^t y^t = M_{t-1} + \boldsymbol{x}^t y^t \tag{1.16}$$

$$C_t = \sum_{i=1}^{t}\boldsymbol{x}^i\left(\boldsymbol{x}^i\right)^T = \sum_{i=1}^{t-1}\boldsymbol{x}^i\left(\boldsymbol{x}^i\right)^T + \boldsymbol{x}^t\left(\boldsymbol{x}^t\right)^T = C_{t-1} + \boldsymbol{x}^t\left(\boldsymbol{x}^t\right)^T \tag{1.17}$$

$$C_t^{-1} = C_{t-1}^{-1} - \frac{C_{t-1}^{-1}\boldsymbol{x}^t\left(\boldsymbol{x}^t\right)^T C_{t-1}^{-1}}{1 + \left(\boldsymbol{x}^t\right)^T C_{t-1}^{-1}\boldsymbol{x}^t} \tag{1.18}$$

$$\Pi_t = M_t C_t^{-1} = \left(M_{t-1} + \boldsymbol{x}^t y^t\right) C_t^{-1} = \left(\Pi_{t-1} C_{t-1} + \boldsymbol{x}^t y^t\right) C_t^{-1} = \tag{1.19}$$

$$= \left(\Pi_{t-1}\left(C_t - \boldsymbol{x}^t\left(\boldsymbol{x}^t\right)^T\right) + \boldsymbol{x}^t y^t\right) C_t^{-1} = \Pi_{t-1}\left(C_t^{-1}\left(C_t - \boldsymbol{x}^t\left(\boldsymbol{x}^t\right)^T\right)\right) + \tag{1.20}$$

$$+ C_t^{-1}\boldsymbol{x}^t y^t = \Pi_{t-1}\left(1 - C_t^{-1}\boldsymbol{x}^t\left(\boldsymbol{x}^t\right)^T\right) + C_t^{-1}\boldsymbol{x}^t y^t = \tag{1.21}$$

$$= \Pi_{t-1} - C_t^{-1}\boldsymbol{x}^t\left(\Pi_{t-1}\boldsymbol{x}^t - y^t\right) \tag{1.22}$$

$$\Pi_t = \Pi_{t-1} + C_t^{-1}\boldsymbol{x}^t\left(y^t - \Pi_{t-1}\boldsymbol{x}^t\right) \tag{1.23}$$

Thus, as each new data point occurs, a new Gain matrix C_t^{-1} is derived 1.18 and used for the update of the parameters Π 1.23. This method does not need to iterate until convergence and thus can be used for real-time learning and prediction. Further, in this work we will use this method for optimizing parameters of linear models for Fuzzy rules, where these updates will need to be fuzzyfied.

1.1.2 Online clustering methods

In previous subsection as well as in latter subsections we talk about supervised learning, where we can use the true label to find the best parameters Π to model the data or the decision boundary for linear and logistic regressions. In this subsection, we describe some unsupervised approaches, where the labels are not known, thus we lack the teacher, and the aim is to find structures within the data that we believe are the classes. Thus, clustering approaches are suited for classification problems, since in regression, the dependent variable is continuous.

1.1.2.1 Online k-means

In k-means algorithm (MacQueen *et al.*, 1967), the aim is to find k structures within the data. These k structures are represented by a vector that is a mean of the data points belonging to each structure - cluster 1.24, where $i = 1...k$ is the index of the cluster defined by m data points.

$$\mu_i = \frac{1}{m} \sum_{j=1}^{m} x^j \tag{1.24}$$

$$w = \arg\min_i \left\| \mu_i - x^j \right\|^2 \tag{1.25}$$

K-means is based on iterative updates of cluster and parameters, composing of two steps which are repeated until convergence of the squared error from equation 1.25. To initialize the first k means, we randomly choose k data points from our data collection.

a. For each data point find the closest mean μ_w 1.25

b. Update each mean μ_i according to the assigned data points 1.24

In online version of k-means (Beringer *et al.*, 2006) we try to obtain the means by doing stochastic gradient descent and thus we change the cost function to 1.26 with an update obtained from the partial derivative with respect to the means 1.27, where this update is added to only one from all k means, with respect to the choice 1.25.

$$J\left(\mu_i|x^j\right) = \frac{1}{2}\left|\left|\mu_i - x^j\right|\right|^2 \tag{1.26}$$

$$\frac{\partial J\left(\mu_i|x^i\right)}{\partial \mu_i} = \left(\mu_i - x^j\right) \tag{1.27}$$

This results into an updating formula for $i = 1 \dots k$ means 1.28.

$$\mu_i^t = \mu_i^{t-1} - \alpha \frac{\partial J\left(\mu_i|x^i\right)}{\partial \mu_i} = \mu_i^{t-1} - \alpha\left(\mu_i - x^j\right) \tag{1.28}$$

Unlike in traditional k-means where each iteration all the data points are assigned and the means are derived from the batch data, in online version the means are derived using small updates in a sequential manner. Similarly to traditional k-means, this is repeated until convergence. Thus, as in the case of stochastic gradient descent for linear and logistic regressions, it is not as well suited for real-time processing.

1.1.2.2 Adaptive Resonance Theory

Adaptive Resonance Theory (ART) (Carpenter *et al.*, 2010) is an incremental clustering method using a three-layer (i.e. recognition and comparison layers and a reset module) competitive neural network, see Figure 1.1. The number of neurons in the recognition layer equals the number of dimensions in the feature space, and the number of neurons in the comparison layer equals to the number of already known structures. The size of the comparison layer is incrementally updated as well as the weights on the edges between the layers. In recognition layer, each data point is recognized by all known structures and the strength of the recognition is sent to the comparison layer. The weights on the edges between the recognition and comparison layers are divided into bottom-up (recognition layer to comparison layer) and top-down (comparison layer to recognition layer).

In ART, each data point is firstly sent to the recognition layer and matched to all known categories, from which the winning category is chosen. If the data point belongs to the winning category, a resonance should occur, which is handled by the reset module. If there is a resonance, an update is performed according to the winning category. If there is no resonance, the search for the proper class continues, until a specified depth of the search is reached. If the search for the correct category is not successful, it is a trigger for adding a new category to the system, i.e. a new node to the comparison layer. Within the reset module, the resonance is given by a threshold ρ known as the vigilance parameter from an interval $[0,1]$.

To determine the similarity of the data point to the known categories, a cosine distance is derived 1.29, see Figure 1.2. For this purpose, both input data and bottom-up weights w need to be normalized, making the result of cosine distance within an interval $[0,1]$. The more similar the data point is to the category i, the closer is the result to 1.

$$\beta_i = \frac{\left(x^j\right)^T \cdot w_i}{||x^j||\,||w_i||} \tag{1.29}$$

To update the weights w, a learning rate λ is used 1.30, denoting how significant the update is to be. The learning rate varies within interval $[0,1]$,where if set close to 0, the increment will be less significant, and conversely in case that the learning rate is set closer to 1, the significance of the new data point is higher and the weights move faster towards it.

$$w_i^t = \frac{(1-\lambda)\,w_i^{t-1} + \lambda x^j}{\left\|(1-\lambda)\,w_i^{t-1} + \lambda x^j\right\|} \tag{1.30}$$

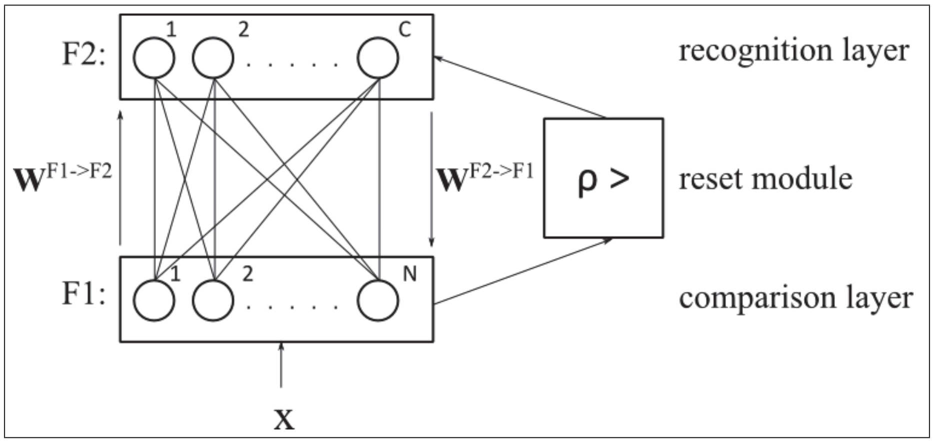

Figure 1.1 Adaptive Resonance Theory model.

1.1.3 Online dimensionality reduction

The curse of dimensionality can be detrimental to the machine learning solution. While in some cases, the problem lies in the number of dimensions being higher than the number of data points, in other cases the high dimensionality can result into over-fitting, and thus high variance as well as computational complexity of the learning and recognition processes. In both cases, feature selection techniques are used, where a smaller number of features is chosen to be the ones representative enough. One group of techniques dealing with feature selection is dimensionality reduction techniques and in this section we describe Principal Component Analysis, where the principal components are updated in an incremental manner.

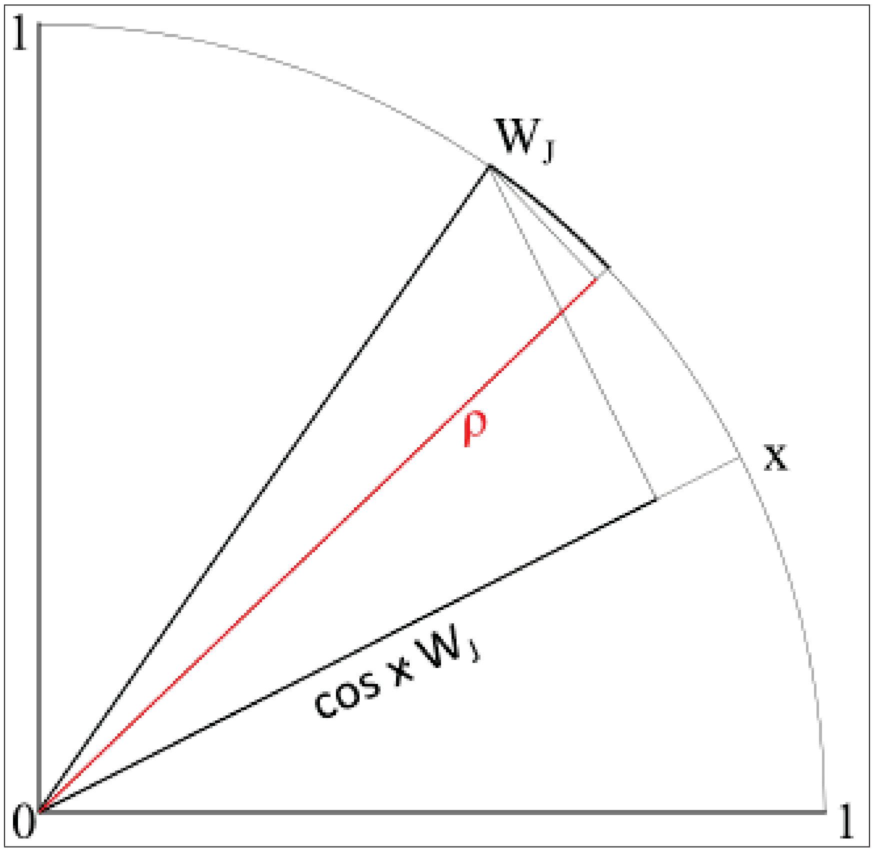

Figure 1.2 Cosine distance of normalized weight vector W_j and input x. We can see, that the further the input x is from the centroid W_j, as compared to the vigilance ρ, the smaller is the projection of W_j onto x, i.e. $\cos xW_j$, which represents their dot product. Then the red line represents the projection of W onto a vector on the boundary of the distances from it, resulting into the vigilance parameter ρ.

1.1.3.1 Incremental Principal Component Analysis

Principal Component Analysis (Hotelling, 1933) is an unsupervised projection method aiming to minimize the loss of information after mapping of the data X to a new space Z and maximize the variance within the data. The $W$are the principal components, that the data X is projected onto, where after this projection the data Z are obtained such that on the first dimension of W the data have the highest variance, on second principal component they have the second highest variance, etc.

$$Z = W^T X \tag{1.31}$$

To center the data on the origin the mean normalization 1.32 is applied, where m is the mean of the data X before the projection.

$$Z = W^T (X - m) \tag{1.32}$$

One way of obtaining Z is by using Singular Value Decomposition, which assumes to find the matrix W such that it results into an uncorrelated Z. Thus, we aim to find eigenvectors of covariance matrix S of the sample X . Having a matrix C of normalized eigenvectors of S, where $C^T C = I$, we can derive the decomposition of S 1.33 with D being a diagonal matrix of eigenvalues λ.

$$S = SCC^T = S(c_1, ..., c_d)C^T = (\lambda_1 c_1, ..., \lambda_d c_d)C^T = CDC^T \tag{1.33}$$

To obtain the matrix of eigenvalues we use equation 1.34.

$$D = C^T SC \tag{1.34}$$

From 1.31 and knowing that the covariance of Z is $W^T SW$ we can set W to C from 1.34.

To reduce the number of dimensions in the new data we pick the k top eigenvalues and associated eigenvectors, where the dimension of data is being reduced from d to k. The proportion of variance kept within the reduced data can be derived as a fraction of the kept eigenvalues and all eigenvalues from the data 1.35.

$$\frac{\lambda_1 + ... + \lambda_k}{\lambda_1 + ... + \lambda_k + ... + \lambda_d} \tag{1.35}$$

In online version of PCA (Li, 2004), an online version of SVD needs to be used. Here, the eigenvectors and eigenvalues are updated in an incremental manner, so as to accommodate the streamed nature of the data. One of the solutions is to firstly project the new data point onto the subspace 1.36 to obtain the residual r from the reconstruction 1.37.

$$z^t = \left(W^t\right)^T \left(x^t - m\right) \tag{1.36}$$

$$r = x - (Wa + m) \tag{1.37}$$

Then the residual r s used as an update for the eigenmatrix W.

1.1.4 Online kernel models

The kernel methods are based on projection of data to a new higher space, where they can be linearly separable. The core of the models are Support Vector Machines (SVM), where the kernels are used in the cost function, to avoid a dot product in the new space (i.e. the kernel trick) and thus to use more complex decision boundaries than simple linear separators. In this section we discuss the incremental version of SVM, where the support vectors are updated in an incremental manner.

1.1.4.1 Incremental Support Vector Machine

Support Vector Machines (Chapelle *et al.*, 1999) are used for classification problems, where a linear decision boundary is placed between the positive class and the negative class. In case of multi-class problem, there are two strategies: one vs all and one vs one. The former defines k number of decision boundaries, where k is the number of classes. Each decision boundary divides the samples from the corresponding class as positive examples from samples from all the other classes as negative examples. In the latter one vs one case, there is one decision boundary for each pair of classes, increasing the number of the separators significantly to $c(c-1)/2$. Thus, one vs all approach is the most common one due to its lower complexity.

Unlike in traditional classification solutions, where the positive class results to 1 and the negative class to 0, in SVMs, the aim is to find such decision boundary, for which all data points of the true class lie at a distance greater or equal than 1 and all negative examples at a distance also greater or equal to1, but in an opposite direction, thus -1 1.38. All the data points lying on the margin are then the support vectors, defining the decision boundary.

$$w^T x^t + w_0 \begin{cases} \geq +1 & for y^t = +1 \\ \leq -1 & for y^t = -1 \end{cases} \tag{1.38}$$

The formula 1.38 can be rewritten as 1.39 for which we want to maximize the margin by minimizing the $\|w\|$ 1.40.

$$y^t \left(w^T x^t + w_0 \right) \geq +1 \tag{1.39}$$

$$\min \frac{1}{2} \|w\|^2 \, \text{subject to} \, y^t \left(w^T x^t + w_0 \right) \geq +1 \tag{1.40}$$

Using Lagrange multipliers α we obtain a cost function 1.41 that we want to minimize with respect to w and maximize with respect to $\alpha^i \geq 0$.

$$L_p = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{t} \alpha^i y^i \left(w^T x^i + w_0 \right) + \sum_{i=1}^{t} \alpha^i \tag{1.41}$$

We can solve the partial derivative of 1.41 to find w and plug this back into the equation to obtain the dual 1.42 that we maximize with respect to α subject to $\sum_{i=1}^{t} \alpha^i y^i = 0$ and $\alpha^i \geq 0$.

$$L_p = -\frac{1}{2}\sum_{i=1}^{t}\sum_{j=1}^{s} \alpha^i \alpha^j y^i y^j \left(x^i\right)^T x^j + \sum_{i=1}^{t} \alpha^i \tag{1.42}$$

$$L_p = -\frac{1}{2}\sum_{i=1}^{t}\sum_{j=1}^{s} \alpha^i \alpha^j y^i y^j K\left(x^i, x^j\right) + \sum_{i=1}^{t} \alpha^i \tag{1.43}$$

The dot product from 1.42 can be replaced by a kernel $K\left(x^i, x\right)$ 1.43, e.g. polynomial, radial-basis function, sigmoid, etc. After solving this we are left with only a number of $\alpha^i > 0$ and corresponding data points x^i. These are the support vectors that lie on the margin, thus their distance from the decision boundary equals 1, where the decision boundary w is a weighted sum of them 1.44.

$$w = \sum_{i=1}^{t} \alpha^i y^i x^i \tag{1.44}$$

Using soft margin hyperplane as a decision boundary, we allow some data points to fall within the margin at a distance ξ from the margin, ξ being a slack variable. The data points that lie within the margin along with the data points that lie on the margin then become the support vectors.

In incremental version of SVM we need to be able to update the support vectors to accommodate all known examples (Diehl *et al.*, 2003; Tax, 2003). First, a new example x^t is being assigned to either not correctly classified, correctly classified within the margin and correctly classified and far enough from the margin. If the new data point is classified correctly, the model stays unchanged. However, if the new data point falls within the margin or is not correctly classified, the samples are updated and a new support vectors derived. Thus, unlike in offline version where the support vectors are derived from the whole data set, here only some data are used to derive the support vectors, i.e. only when there is a need for new SVs.

1.1.5 Online ensemble models

In this section we discuss some of the ensemble models that are widely used in many applications, and their online versions. Ensemble models are composed of number of classifiers, that are often described as weak classifiers, since they are experts only on a subset of the whole problem, and thus usually do not perform on the whole data as well as the complete ensemble, or a mixture. From the methods we have chosen ones based on bagging, boosting and fuzzy logic, where their structure and parameters are being updated in a sequence. Namely, we describe Online AdaBoost based on online boosting, Evolving Fuzzy models based on fuzzy logic and Online Random Forests based on bagging. All these methods tackle the ensemble nature differently. In AdaBoost methods, new classifier is set to be an expert on samples that previous classifiers are confused on. In Random Forests, each weak classifier is an expert on a random subset of the data, with replacement. In Neuro-Fuzzy models, each classifier is an expert on a fuzzy structure from which the data points are similar to each other in a specified way.

1.1.5.1 Online AdaBoost

AdaBoost is based on a boosting, where the model is composed of a number of weak classifiers (Freund *et al.*, 1995). Each such weak classifier is an expert on a random subset of the whole sample and derives a local inference. The final inference of the model is derived as a weighted sum of all these local inferences. The weight is assigned based on the strength of the weak classifier. AdaBoost stands for adaptive boosting. It is an implementation of boosting, where the weighting can be assigned to classifiers, samples or features. The chosen entities are weighted according to their importance. In case of classification, AdaBoost in general creates a group of classifiers that are trained on weighted samples. These weights are derived based on their miss-classification. Thus, if a sample is correctly classified by some weak classifier, it will have only a half of the total weight oppositely to the miss-classified ones.

In online version of AdaBoost, the random sampling with replacement as well as adaptation of the weights need to be performed in an online manner (Oza, 2005; Grabner *et al.*, 2006). To mimic the random sampling with replacement, a number from distribution Poisson(λ) is drown, where by default λ=1. This number indicates the number of times that some example is fed to the corresponding classifier. Parameter λ is updated according to the importance of this data point, where if it is miss-classified, it is given half weight for λ and the remaining data are given the other half of the weight. Thus, this weight is calculated from the total error of this data point and each time it is fed to the classifier (which is Poisson(λ) times), this parameter is adapting according to its meaningfulness. If some data point is fed to the classifier for example three times, its weight is decreasing when it is not misleading and increasing oppositely. The error of such data point is derived using a hard margin, which makes the original AdaBoost a binary classifier. However, there are methods adapting it to multi-class problems. One of the remaining hyper parameters is the number of weak classifiers, for which we need to have some knowledge about the data. As well as for online bagging, it is not straightforward to automatically add new classifiers without storing the past data, as the random sampling with replacement would be broken.

1.1.5.2 Evolving Neuro-Fuzzy models

Neuro-fuzzy models are ensemble models, where the weak classifiers are rather local fuzzy rules based on a fuzzy clusters within the data. Each such cluster *i* has a form of a rule *R* where the inference *y* is a soft inference for every class j, known as the possibility of every class within the area of expertise of the corresponding rule being the class j 1.45.

$$R_i : IF\ x\ IS\ A_i\ THEN\ \beta_i \left\{ y_i^j \right\}_{j=1\ldots c} \tag{1.45}$$

Each rule can be divided into two parts, an antecedent describing the similarity of example x to prototype A_i that is given by rule R_i, i.e. $IF\, x ISA_i$ and a consequent part describing the consequence y_i^j of the rule based on the similarity from antecedent part β_i, i.e. $THEN\, \beta_i \left\{y_i^j\right\}_{j=1...c}$. The β is also referred as a firing degree of the rule that works as a weight for the consequent part of the rule. There are two main groups of Neuro-Fuzzy models based on the type of consequences they output, Mamdani and Takagi-Sugeno.

1.1.5.2.1 Mamdani model

In case of Mamdani models (Mamdani *et al.*, 1975), the consequence of rules in a singleton value stating the class of the data point based on its area of expertise 1.46.

$$R_i : IF\ x\, IS\, A_i\, THEN\ \beta_i B_i \tag{1.46}$$

The computational complexity of these models is thus lower than the next group of fuzzy models, however their performance tends to be lower as well. Therefore it depends on the application and the importance of the low computational cost compared to the performance in terms of accuracy, precision, etc.

1.1.5.2.2 Takagi-Sugeno model

Unlike in Mamdani models, the consequence of Takagi-Sugeno models (Takagi *et al.*, 1985) is derived for each class, where the inference y is either singleton consequence or based on a linear model 1.47. As well as for Mamdani, singleton consequences achieve lower computational complexity, however lack the performance compared to the linear consequence. Thus, the linear consequences 1.48 are used more widely and we focus on those in this work.

$$y_i = \begin{cases} s_i & singleton \\ \pi_i^T x + b_i & linear \end{cases} \tag{1.47}$$

$$R_i : IF\, x IS A_i\, THEN\, \beta_i \left\{ \left(\pi_i^j\right)^T x + b_i^j \right\}_{j=1\ldots c} \tag{1.48}$$

The π_i^j is a vector of consequent parameters for rule i and class j with an offset b_i, weighted by a firing degree of this rule β_i.

We can imagine these models as partially linear, where each class is defined by its corresponding linear model, that is broken down by the clustering - the fuzzy rules, making this model non-linear. Since the rules give inference about every class, they are suitable for applications, where there tends to be variance within one class, i.e. there might be substructures within one class. Such class can be then defined within a number of fuzzy rules, where each rule is an expert on that specific subclass.

To derive the membership function, that serves as a similarity, often a distance metrics are used and various models vary on this aspect, accompanied with a clustering approach for the generation of rules.

In evolving Neuro-Fuzzy models, the rules are generated in an online or incremental manner, and thus an incremental clustering is required. Along with this, recursive updates to the parameters of the membership function are necessary to be able to follow the trend in the data points assigned to each specific rule. Lastly, the consequent part needs to be updated in an online manned as well.

Since the antecedent part varies among all models, each evolving fuzzy model uses different strategy for generating the rules and updating the membership function. We describe the fuzzy methods in the Methodology section more closely. The consequent part is often solved by using Recursive Least Squares that we have discussed previously in section 1.1.1.2.

These models are very powerful and capable of online learning, where we do not need to store the past data and do not need to iterate until convergence. However, most of the state of the art models require some setting of hyper-parameters, either by cross-validation or by obtaining some knowledge about the data. Thus, in this thesis we push the capabilities of fuzzy models further by adapting all the parameters to accomplish the objectives of this thesis (2.1).

1.1.5.3 Online Random Forest

Random Forests are based on bagging, where the model is an ensemble of a number of weak classifiers (Breiman, 2001). Each such weak classifier is trained on a random subset of the data and features, where the random subset is sampled with replacement from the whole data set. Then, each weak classifier gives an inference, its opinion about the class, either hard or soft, and these opinions are averaged over all the weak classifiers. Unlike in boosting, each weak classifier has the same weight and does not take into account the miss-classified data.

In the case of Random Forests weak classifiers are decision trees. Decision trees lack from the high variance, since they tend to over-fit. To tackle this, one option is to bring a noise into the data, and in case of bagging it is in a form of not seeing the whole picture - the bootstrap sampling. This makes the decision trees weak classifiers, suitable for the ensemble model based on boosting. Thus, a Random Forest is an ensemble of decision trees.

In decision trees, the aim is to find the best features that lower the uniformity in the data - improve the classification (in case of classification). Thus, each node performs a split on data belonging to it based on the gain it will get after the splitting, while the root of the tree contains all the possible data. The uniformity of the data within the node is most commonly measured by the entropy 1.49. The aim is to achieve a low entropy. In case that there are uniformly distributed classes within a node, i.e. no class is dominant, the entropy is 1 which means that there is chaotic decision in such node. In case that there is only one class present, entropy is 0.

$$-\sum_i p_i \log p_i \tag{1.49}$$

To make a split, we compare all the possible splits on the features and find the best threshold such that the information gain on the node is the highest possible 1.50. $H(X)$ is the entropy of the parent node to be split and $H(Y|X)$ is the conditional entropy, the complete entropy for both new nodes $Y = \{y_1 = 1, y_2 = 0\}$. Then the $I(X,Y)$ is the information gain explained by the splitting of the node X, i.e. how much information we gain from the new split compared to the information obtained within parent node X.

$$I(X,Y) = H(X) - H(Y|X) \tag{1.50}$$

In Random Forests, to fight the over-fitting nature of decision trees, the features for each split are chosen by random, so only a specific number of features is chosen. This results into Randomized trees, and making weaker trees compared to the original Decision trees, however with higher generalization ability, and thus more suitable for boosting.

In Online Random Forests, the Randomized trees are replaced by Extremely randomized trees, where not only the features are chosen by random for each split, but also the thresholds on which the splits are performed (Leistner *et al.*, 2009). Then, the simulation of bootstrap is achieved by Poisson(λ) distribution similarly to Online Boosting, where the λ is often set to 1. To be able to use Random Forest for sequence data, we need to be able to grow the trees incrementally. Thus, besides the sampling, an online decision making about the split is necessary. Thus, we need to decide when to make a split based not only on the information gain, but also on the size of the node. The number of the trees needs to be set ahead, because similarly to boosting, it is not as straightforward to grow new trees without the knowledge of the past data. Although, since Online Random Forests need to have access to the past data in order to make new splits and choose the features, adding of new trees in the case of need might be simulated.

CHAPTER 2

GENERAL METHODOLOGY

2.1 Objective of the research

The main objective of the thesis is to develop and by experimental testing to confirm a system based on online learning with the emphasis on learning from scratch, learning on the fly and user-friendliness. To address particular problems raised in previous section, we set four sub-objectives to be tackled in this work.

2.1.1 Sub-objective 1: Develop online learning model tackling learning on the fly and from scratch along with the accuracy vs. processing time problem

Our problem of unknown data and user-friendliness results into unknown times of addition of new classes to the system and no initialization of the system. We can call these problems as learning on the fly and learning from scratch, that are important to address while we choose the model for this work. At the same time, such a choice needs to address the processing time vs. accuracy problem as well, in order to work in a real-time setting. Thus, we aim to build a model capable of

- make predictions from restricted amount of labeled data,
- easily update in order to accommodate new classes,
- grow in order to adapt to handle the increasing knowledge of the world,
- make fast and simple updates.

Following all these condition the development of a suitable model for our task will then result in a robust and flexible model suitable for the tasks when the environment can change while the prediction capabilities of the model are in use and it is not preferable to learn the model from the beginning every time such situation occurs. At the same time, such model will not require initialization, and as such follow the user-friendly requirement, e.g. in the case of handwritten gestures recognition, a user will not be obliged to sample multiple strokes in an uncomfortable process, but will only set a new class. At the end, such model will be able to work in a real-time setting, i.e. allowing real-time responses and predictions, when the application is required to do so.

2.1.2 Sub-objective 2: Tackle forgetting of unused classes for online learning models and apply

In online learning, the data that are processed in a sequence are not necessarily uniformly distributed with respect to their classes. Thus, it is necessary to tackle the problem of forgetting of unused classes that become sparse in the local models responsible for their detection. We will develop a method that will be immune to the forgetting and be applicable to the whole solution in our problem.

2.1.3 Sub-objective 3: Develop online learning model able to work without free parameters

The use of free/hyper parameters is very common for offline learning methods. However, in online learning setting we cannot wait for cross-validation, which would mean to use batch data to perform it. This is in very contrast with the nature of online learning. Another option is to have an expert to set the free parameters based on the knowledge of the data. However, such experts are not common as well as the data are not always known ahead. Thus, we will tackle the problem of unknown data by finding a way to let the model adapt its free parameters based on the evolution in the data and the state of the environment.

2.1.4 Sub-objective 4: Tackle the problem of missing data

In online learning, when the data are processed in a sequence, at the beginning there is no data at all. Thus, the model starts with one example at a time. This results into low generalization ability of the model accompanied by a high variance, when the new data points are not recognized well, because the model over-fits the old data. A solution for this problem is to get more data. In this work we will find a suitable solution to generate more data for this period of time where the small size of the data causes high variance. We will explore the setting of such methods and its application to online learning such that the whole process will be still capable of real-time performance.

2.2 Methodology

In this work, we focus on online incremental learning with specific requirements as stated in the objectives. However there are two significant aspects of our work, i.e. online and incremental learning. In many applications, incremental learning is used as an offline learning, mainly to deal with the long processing time. The data is processes by a size-restricted blocks, often one data point at a time, which brings simple updates to a use. Unlike in batch learning, where often matrices are needed to be processed and plugged into formulas, incremental learning tends to work rather with vectors. In many applications benefiting from incremental learning, the data is known ahead, and thus the learning can be repeated until some convergence. Thus, even though the data is not accessible all at once during one single processing, it can be re-accessed after the processing, i.e. learning is finished. In online learning, the data is not known ahead, and thus we cannot run the learning for several times. Many models of online learning do not need to follow the incremental learning restriction, as they are allowed to store the already known data in a memory and access them later for processing. Since there is only one single processing altogether, they can access the data within a single and the only one learning process.

Our work is originally motivated by the handwritten gesture recognition for natural human-computer interactions. Such an application requires real-time learning that is online, since the data is unknown. To enable the real-time processing, as well as a possible deployment on regular computers, we need to avoid storing and re-accessing the memory of past data. Hence, incremental learning is a suitable solution for such problem. In online incremental learning based models, the data point is accesses exactly one time, and that is the time of its introduction.

After it is processed, i.e. the model has adjusted its parameters to cope with similar data points, it is forgotten and never accesses again. Unlike in offline incremental learning, the future data is unknown and there are no further processing of the data, which yields a hardened task for the model to adjust its parameters as if they would be after convergence, while bringing the possibility of more dynamic model capable of adjusting to new and current situations. Contrary to online learning that is not incremental, there is no storage of past data required, yielding a not being able to look at a bigger picture in the sense of data, while at the same time avoiding the high requirements for the memory.

In handwriting, the styles between writers vary which results into multiple sub-classes within each class of symbols. For example in alphabet, one letter can be represented by lowercase or uppercase, and even the font/writing styles bring dissimilarity inside of one class and similarities for letters from different classes. By allowing more users to participate in the interaction with the same platform we allow more styles for one class. If we were to build one model for each class, we would end up with overlapping models, where the results for some data points would be similar. Therefore, it is essential to use modeling that allows within-class variance and can cope with such behavior.

The Neuro-Fuzzy models are based on dividing the feature space into a group of sub-problems, where each such problem is handled by a fuzzy rule. Therefore, even though there is a partition, it is not absolutely clear, as all data influences the learning of all the rules, which can be compared to a soft margin, i.e. each data point belongs to each rule to some extent. Such a modeling allows to use simple models for each rule, and in our work these are linear models. The combined result of all rules can be then understood as a partially linear model, since the combination of the results is nonlinear. In the case of regression, one can imagine a high polynomial curve that can be segmented into a number of lines and then combined back into a curve. Originally, such a polynomial curve would be described by a number of parameters making the optimization problem hard, however by describing it by straight lines, the hard problem is divided into a number of simpler problems. Neuro-Fuzzy models introduce an interesting structure suitable for our main motivation, i.e. the handwritten symbol recognition, where each symbol can be expresses by more users and therefore in more ways. This means that one class of symbols can be handled by a number of rules and a number of classes can be handled by one rule. This partition of classes within the rules then becomes fuzzy, where each data point of certain class participates on the results for this class even though it is not within the same rule, i.e. it is not highly similar. We can say, that there is a possibility that each class belongs to each rule, i.e. within each rule all classes are possible to some extent.

In this work we focus on a group within the Neuro-Fuzzy models (Figure 2.1), i.e. the Evolving Neuro-Fuzzy models, where the rules are created on the fly. Such a setup enables the learning from scratch, which is very suitable for our aim of user-friendliness, where the users themselves can set the symbols that they want to use. However, to be able to fully apply these models for our specific setup, we need to adjust some structural parameters of the models.

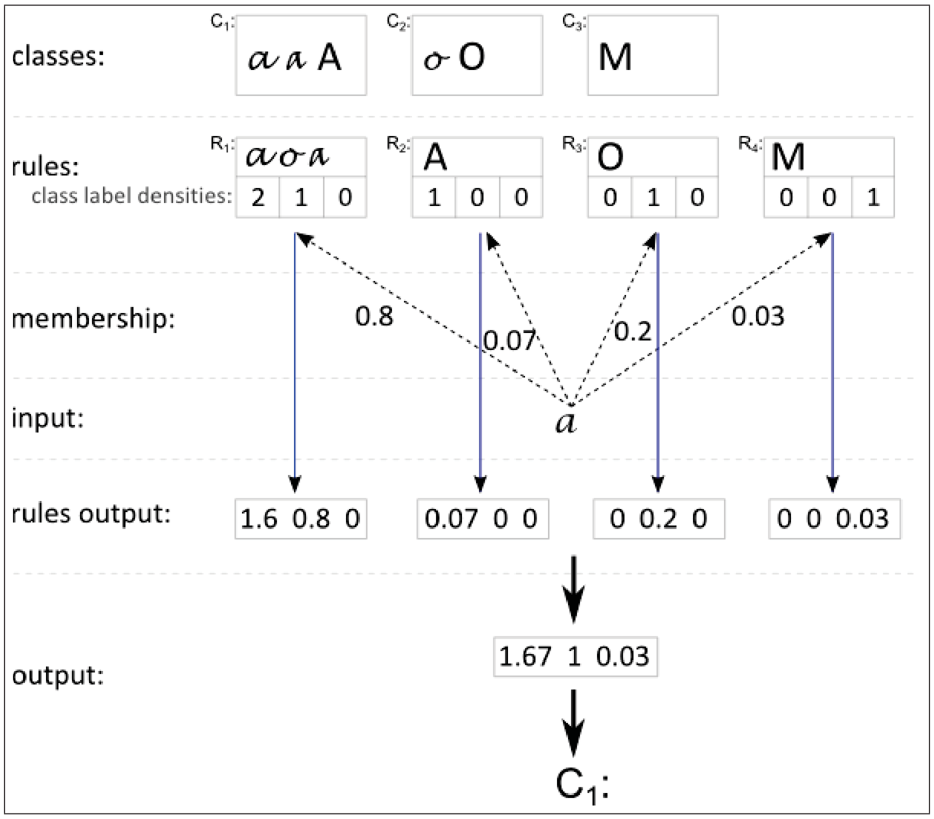

Figure 2.1 Takagi-Sugeno Neuro-Fuzzy model.

In the search for the proper model we have developed a number of models that are contained in the methodology section. All these models are able to answer specific objectives of this theses and have gradually led to the accomplishment of all our objectives posed by this work.

We apply our solutions to various tasks, however from which handwritten gesture recognition stands out the most. For this specific application we were able to bring solutions for our last objective - the missing data, since this problem is bounded to the specific application.

2.2.1 Incremental Similarity

To avoid restricting the users in the number of classes they use along with the number of users themselves, we need to apply solutions for the real-time processing without affecting the ability to learn and recognize these classes. Thus we have developed a similarity measure that is necessary for Neuro-Fuzzy models and the weight of the rules used in the combined result of the model (Režnáková *et al.*, 2016a). This similarity measure uses incremental updates and is suitable for online learning. It keeps only a vectors in the memory which avoids the use of matrices in the formula which yields the improved processing-time.

2.2.2 Elastic Memory Learning

Since the Evolving Neuro-Fuzzy models are transformed from offline Neuro-Fuzzy models, the learning is prone to forgetting. In offline batch learning we do not need to worry about the order of the samples, we simply access them all at once and can see the 'whole picture'. The model is capable of processing the data several times until convergence and each time fix the parameters for all the data at once. In online incremental learning this is not possible, since we cannot access the past data and we cannot see the 'whole picture'. Thus, the performance of the model may vary for different order of the data points. Usually when evaluating the model, the data is randomly sorted for each test in the evaluation protocol. This results into uniform distribution of the classes within the stream of the data, if there are not sparse classes. However if such sparsity occurs, i.e. some class is not used for a specific time, the learning still continues and overwrites the parameters of this class. This is the result of the collaborative learning in fuzzy models, where each data point is used to adjust the parameters of each rule. In our main application, the uniform distribution of classes is unlikely, and we need to change our learning along with the evaluation protocol to cope with this problem. Therefore we have developed the Elastic Memory Learning (Režnáková *et al.*, 2016b) that uses a competitive learning where each data point needs to prove its importance in order to be used for the learning of a specific rule (Figure 2.2).

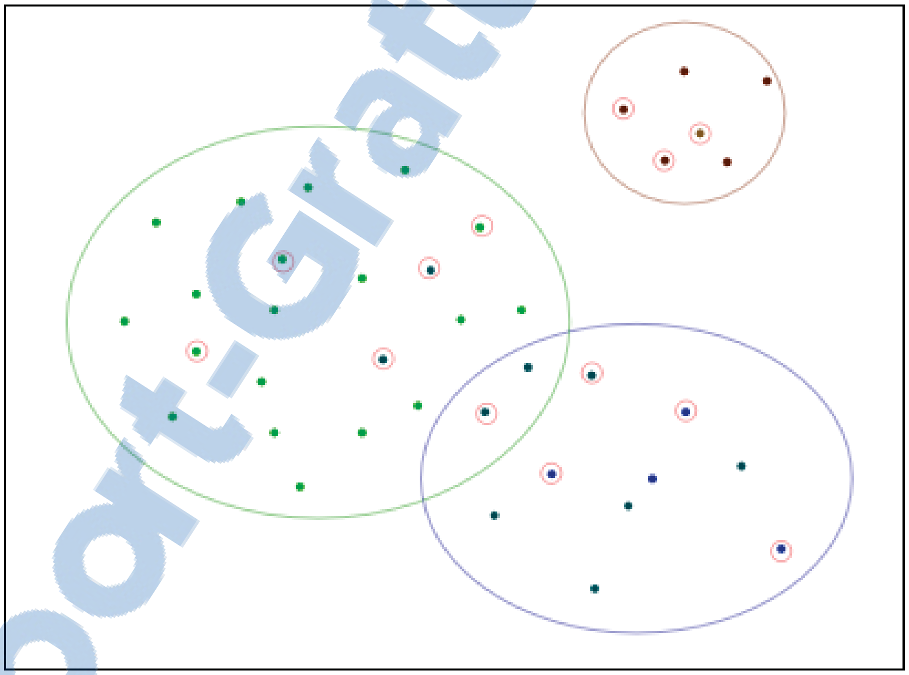
Figure 2.2 Competitive learning. Points of different classes (colors) compete within their rules to be used for the learning (red circled points).

2.2.3 Self-Organized Incremental Model

The evolving nature of the model results into rules being properly created for the data known at a specific time with the knowledge from that time. However, the partition of the space can become improper and the rules created at different conditions can overlap or even lie within each other. Moreover, the number of the rules is not restricted which can lead to increased processing time which is in contradiction with the real-time nature of our task. Thus it is important to use a model that is capable of re-organizing its structure in an automated way, where each time all the rules will be proper for the specific conditions of that time. This led us to the self-organized Neuro-Fuzzy model (Režnáková *et al.*, 2015a), with the introduction of the conditions and mechanisms for the merging (Figure 2.3), splitting (Figure 2.4) and discarding (Figure 2.5) of the rules. This is also important for the stability-plasticity dilemma, where the model remembers the past, but also adjusts to the new.

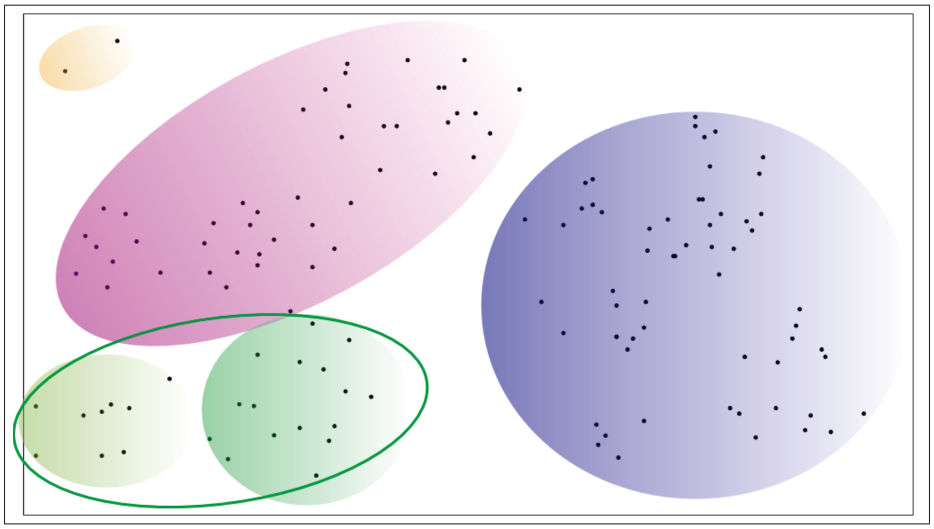

Figure 2.3 Merging of the rules that are close enough and similar enough to each other.

By introducing the self-organized mechanism as well as by being in an online environment, we cannot rely on the cross-validation to fix the parameters. Not only that we simply do not have time to perform it which would result into not real-time processing, we do not have the data anymore due to the incremental nature and our model is able to change itself at any time. Neuro-Fuzzy models as well as their Evolving versions still use some hyper-parameters that need to be fixed. One option is to simply fix them to some default value, however this is not always that easy. Thus, we have avoided the use of hyper-parameters in the model by allowing them to adjust based on the behavior of the data and keeping in mind the stability-plasticity dilemma.

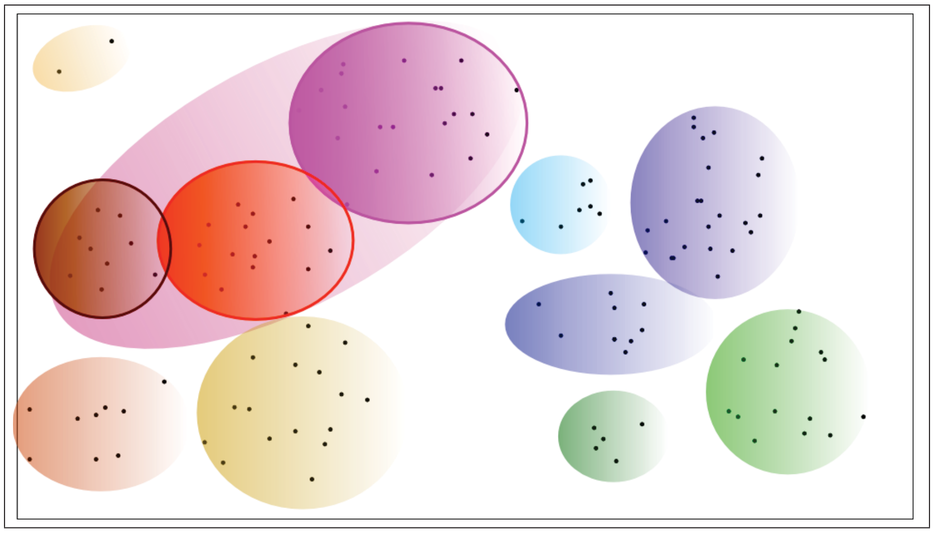
Figure 2.4 Splitting of the rule into dissimilar rules.

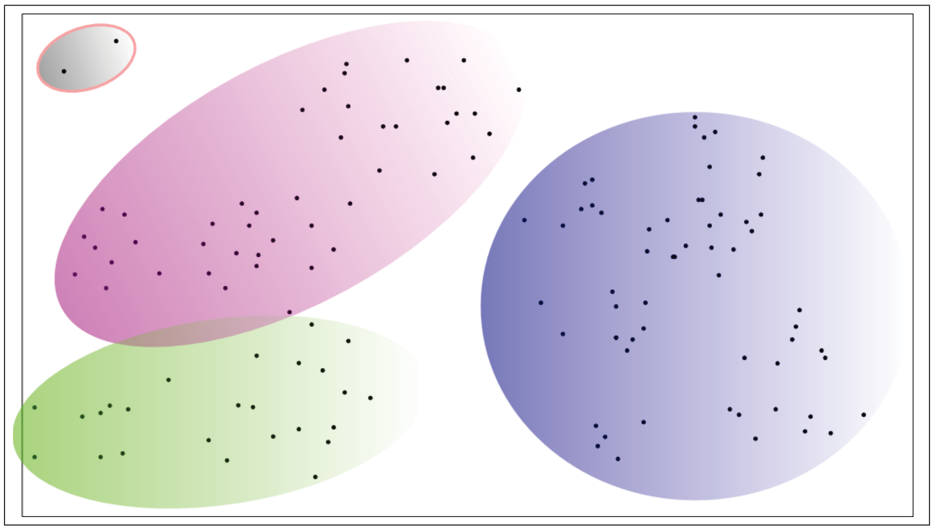
Figure 2.5 Discarding of the rules that are too small and old.

2.2.4 The problem of missing data

To be able to fully tackle all the problems stated in this work and fulfill all the objectives, we needed to solve the generalization problem of online models when starting from scratch. Learning from scratch opens the question if it is possible to learn from one or only a few examples properly, such that the model will be able to deal with the data to come. If we introduce only a small group of examples, and these examples will most probably be very similar to each other in our main application, the model is prone to over-fitting. Thus, the learning curve we can usually see for online learning contains a drop at the beginning of the learning process. One straight-forward solution is to introduce more data with some variation to increase the generalization ability of the model. In our work we have applied the kinematic theory in order to generate synthetic samples and integrate this process into the online learning process, while taking into account the real-time processing (Režnáková *et al.*, 2016c). This addition has led to improved initial learning results, avoiding the drop and increasing the generalization. This allows the model to be fully functional from the beginning with no restriction to wait for more real data.

CHAPTER 3

INCREMENTAL SIMILARITY FOR REAL-TIME ON-LINE INCREMENTAL LEARNING SYSTEMS

Marta Režnáková[1], LukasTencer[1], Mohamed Cheriet[1]

[1] Synchromedia Lab, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

Paper published in *Pattern Recognition Letters*, April 2016.

3.1 Abstract

The expectation of higher accuracy in recognition systems brings the problem of higher complexity. In this paper, we introduce a novel Incremental Similarity (IS) that maintains high accuracy while preserving low complexity. We apply IS to on-line and incremental learning tasks, where the need of low complexity is significant. Using IS enables the system to directly compute with the samples themselves and update only few parameters in an incremental manner. We empirically prove its efficiency on several evolving models and show, that by using IS they achieve competitive results and outperform the baseline models. We also consider the problem of incremental learning used to handle fast growing datasets. We present a very detailed comparison for not only evolving models, but also for the well-known batch models, showing the robustness of our proposal. We perform the evaluation on various classification problems to show the wide application of evolving models and our proposed IS.

3.2 Introduction

In many areas of research, the growth of the amount of data is inevitable. This is very positive for machine learning and pattern recognition, where the lack of data often results in low accuracy. However, the bigger the data is, the more time we need to process it and to train our models. Furthermore, if new data arrives, all of them need to be re-processed so that all the information is included. This may cause a delay in the usage of the system. Thus, in last few years we have been noticing more attempts for incremental learning of the models aimed at absorbing all the necessary information and at the same time lowering the burden of huge datasets.

In incremental learning, the model is incrementally built (learned) each time new data arrives. Once the information on this new data is stored, the original data is often discarded and thus the system does not have access to the original data after. Thus, the model works faster than offline batch techniques. However, beside all the benefits, there are also challenges that ask for solutions and this paper aims to solve some of them described in the following.

- Processing time vs. accuracy challenge asks how far we want to go with the complexity of our model in order to achieve better performance.
- At the beginning of the learning process, we can struggle with a lack of data, missing important information regarding the variances within classes. This occurs especially when incremental learning techniques are applied to real-time recognition and dynamic tasks, in which the recognition does not wait for the whole learning process to be finished. Using incremental techniques can make the process faster, however it should not be done to the detriment of high recognition capabilities.

Incremental Similarity introduced in this paper allows to learn from scratch, where even a small number of samples gives reasonable estimate of the class. At the same time, with the increasing number of samples, the learning time is kept constant, learning only few non-matrix parameters and describing the samples themselves.

In the following sections, we investigate through incremental learning (IL) modeling approaches (Section 3.3), then give a detailed description of Incremental Similarity (IS) and learning of its parameters (Section 3.4). We then apply IS to the baseline incremental and batch approaches to develop new models (Section 3.5). At the end of this paper, we evaluate the performance of these models for various applications (Sections 3.6 and 3.7).

3.3 Related works

In this paper we focus on a classification task, in which the aim is mainly to classify handwritten symbols. We also focus on online incremental learning techniques, where the learning and classification are done on one sample level and the model is learned by small increments.

This section briefly summarizes several groups of incremental or otherwise online techniques, out of which some focus on the clustering part only and others use clustering as one of their layers followed by other classification or regression layers.

There are many works that focus on online, incremental or adaptive learning. Usually, researchers choose an offline method and adapt it to work in an online manner. There are several attempts to transfer offline clustering methods, such as K-means or K-NN into incremental or online learning based. From these we mention (Beringer *et al.*, 2006) in which the authors propose online version of K-means, (Forster *et al.*, 2010), (Yu *et al.*, 2009) for incremental K-NN, or (Lughofer, 2008b) for incremental vector quantization (VQ).

In many applications, SVM based methods are widely used. Thus, there have also been attempts for incremental learning variations, such as in (Carozza *et al.*, 2000), (Wang *et al.*, 2007), (Wang *et al.*, 2008) for incremental and online SVM in regression and (Bordes, 2005), (Tax, 2003) for online SVM.

There have been several attempts for incremental subspace methods, especially for images, such as in (Li, 2004), (Ross *et al.*, 2007), (Song *et al.*, 2008) proposing incremental PCA and (Yao *et al.*, 2010) with online PCA.

In (Oza, 2005) the authors propose an online version for bagging and boosting algorithms, coming up with AdaBoost (adaptive boosting). Another versions of AdaBoost has been proposed in (Grabner *et al.*, 2006) used for vision problems, (Saffari *et al.*, 2010) and (Ditzler *et al.*, 2010) for multi-class online boosting.

In this paper we focus mostly on evolving or adaptive neuro-fuzzy models usually based on Takagi-Sugeno fuzzy model (Takagi *et al.*, 1985), or less on Mamdani model (Mamdani *et al.*, 1975). They vary in the clustering part that is essential for online purposes, as in offline, where the division of the known space is easier and more straight-forward. In (Angelov *et al.*, 2004) the authors propose to solve the clustering part using Recursive Mountain Clustering for space division and the creation of rules, which they combined with univariate normal distribution (antecedent part) and Recursive Least Squares (consequent part). This research was updated in (Almaksour *et al.*, 2010) using Mahalanobis distance for antecedent part (similarity to rules - clusters). In (Carse *et al.*, 1996), (Gomez *et al.*, 2002) the authors utilize genetic algorithms for the rule control and generation. In (Kasabov, 2001) the authors solve the problem of evolution of fuzzy rules by using connectionist systems. In (Režnáková *et al.*, 2012) we proposed to use incremental distance and clustering, followed by (Režnáková *et al.*, 2013) where we proposed to use ART-2A clustering (Carpenter *et al.*, 1991b) for the rule generation and handling. In (Lughofer, 2008a) authors propose to use incremental VQ for the space division.

To our knowledge, the works cited in (Angelov *et al.*, 2004; Almaksour *et al.*, 2010; Režnáková *et al.*, 2012, 2013) are closest to the proposition presented in our study. However, the extensive comparison and empirical proofing was not performed in any of them.

3.4 Incremental similarity

To address some of the challenges of on-line learning, i.e. the complexity vs. precision, learning from scratch and learning on the fly, we introduce the Incremental Similarity (IS). We apply this to a number of baseline models to figure as a similarity measure. The more similar is sample x to a group of samples in a set a, the higher value the similarity measure takes. Thus, all the similarity measures in this work will be adjusted, if necessary, accordingly (3.1) with d being the distance. This results in the range of similarities to be [0, 1], where 1 is the lowest distance and highest similarity of sample x to the set a.

$$\frac{1}{1+d} \tag{3.1}$$

In this work we show the effectiveness of IS by a comparison to the Euclidean (ED) and the Mahalanobis (MD) distances that we both explain later in this section. The baseline models we apply all these similarity measures to, along with the detailed description of the structures within these models that the similarities are used at, are described in Section 3.5, namely Takagi-Sugeno based (similarity measures are used for antecedent learning) and K-means (similarity measures are used for distance from k means).

3.4.1 Euclidean and Mahalanobis distances

In this section, we describe two basic distance measurements ED and MD, taking into account the nature of the membership function (the firing degree) – the more similar the sample is to the baseline of the rule, the higher firing degree this rule has. Thus, we calculate the reversed squared ED (3.2) and the reversed MD (3.3) as an opposite to the univariate and multivariate distributions. Here μ_{t_i} is the mean at time t_i for rule i, i.e. the number of samples already introduced to the rule i and S_{t_i} is the covariance matrix at time t_i for rule i.

$$\beta_i = \frac{1}{1 + \left(\boldsymbol{\mu}_{t_i} - \boldsymbol{x}\right)^T \left(\boldsymbol{\mu}_{t_i} - \boldsymbol{x}\right)} \tag{3.2}$$

$$\beta_i = \frac{1}{1 + \left(\boldsymbol{\mu}_{t_i} - \boldsymbol{x}\right)^T S_{t_i}^{-1} \left(\boldsymbol{\mu}_{t_i} - \boldsymbol{x}\right)} \tag{3.3}$$

The update of the parameters μ_{t_i} and $S_{t_i}^{-1}$ can be derived from the univariate and multivariate normal distributions. For univariate normal distribution we have the probability of samples $\boldsymbol{x}_1 ... \boldsymbol{x}_N$, $P(\boldsymbol{x}_1 ... \boldsymbol{x}_N)$ further referred as P.

$$P = \prod_i \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{ -\frac{1}{2} \frac{(\boldsymbol{x}_i - \boldsymbol{\mu})^2}{\sigma^2} \right\} \tag{3.4}$$

To find the parameter μ that minimizes the error we set the derivation of the logarithm of the function to zero. This will lead to an updating formula for μ (3.7).

$$\log P = \sum_i \left[-\frac{1}{2}\log 2\pi - \log\sigma - \frac{1}{2}\frac{(\boldsymbol{x}_i - \boldsymbol{\mu})^2}{\sigma^2} \right] \tag{3.5}$$

$$\frac{\partial}{\partial \boldsymbol{\mu}} \log P = \sum_i \frac{\boldsymbol{x}_i - \boldsymbol{\mu}}{\sigma^2} = 0 \Rightarrow \sum_i [\boldsymbol{x}_i - \boldsymbol{\mu}] = 0 \tag{3.6}$$

$$\boldsymbol{\mu} = \frac{1}{N}\sum_i \boldsymbol{x}_i \tag{3.7}$$

Then the recursive formula can be derived as in (3.8), where t_i is the time and it updates according to (3.9).

$$\boldsymbol{\mu}_{t_i} = \frac{1}{t_i}\left((t_i - 1)\boldsymbol{\mu}_{t_i - 1} + \boldsymbol{x}_{t_i}\right); \boldsymbol{\mu}_1 = \boldsymbol{x}_1 \tag{3.8}$$

$$t_i = t_i + 1 \tag{3.9}$$

For multivariate normal distribution, the derivation of μ_{t_i} is similar to univariate distribution. The covariance matrix update is derived as follows resulting into (3.15).

$$P = \prod_i \frac{1}{2\pi^{\frac{d}{2}}} |S|^{-\frac{1}{2}} \exp\left\{ -\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{\mu}) S^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu})^T \right\} \tag{3.10}$$

$$\log P = \sum_i \left[-\frac{d}{2}\log 2\pi - \frac{1}{2}\log|S| - \frac{1}{2}D \right] \tag{3.11}$$

$$D = (\boldsymbol{x}_i - \boldsymbol{\mu}) S^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu})^T \tag{3.12}$$

$$\frac{\partial}{\partial S}\log P = \sum_i \left[-\frac{1}{2}S^{-1} + \frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{\mu}) S^{-2} (\boldsymbol{x}_i - \boldsymbol{\mu})^T \right] = 0 \tag{3.13}$$

$$\sum_i \left[-S + (\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T\right] = 0 \tag{3.14}$$

$$S = \frac{1}{N}\sum(\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T \tag{3.15}$$

To be able to recursively update directly the inverse matrix, we derive (3.19), where again t_i is the time and is updated according to (3.9).

$$S_{t_i}^{-1} = \frac{1}{\frac{t_i-1}{t_i}S_{t_i-1} + \frac{1}{t_i}\left(\boldsymbol{x}_{t_i} - \boldsymbol{\mu}\right)\left(\boldsymbol{x}_{t_i} - \boldsymbol{\mu}\right)^T} \tag{3.16}$$

$$= \frac{1}{\frac{t_i-1}{t_i}S_{t_i-1}\left[1 + \frac{1}{t_i-1}\left(\boldsymbol{x}_{t_i} - \boldsymbol{\mu}\right)^T S_{t_i-1}^{-1}\left(\boldsymbol{x}_{t_i} - \boldsymbol{\mu}\right)\right]} \tag{3.17}$$

$$= \frac{1}{\frac{t_i-1}{t_i}S_{t_i-1}} + \frac{-\frac{t}{(t-1)^2}S_{t_i-1}^{-1}\left(\boldsymbol{x}_{t_i} - \boldsymbol{\mu}\right)\left[S_{t_i-1}^{-1}\left(\boldsymbol{x}_{t_i} - \boldsymbol{\mu}\right)\right]^T}{\left[1 + \frac{1}{t_i-1}\left(\boldsymbol{x}_{t_i} - \boldsymbol{\mu}\right)^T S_{t_i-1}^{-1}\left(\boldsymbol{x}_{t_i} - \boldsymbol{\mu}\right)^T\right]} \tag{3.18}$$

$$S_{t_i}^{-1} = \frac{t_i}{t_i - 1}S_{t_i-1}^{-1} - \frac{t_i}{t_i - 1}C \tag{3.19}$$

$$C = \frac{S_{t_i-1}^{-1}\left(\boldsymbol{x}_{t_i} - \boldsymbol{\mu}\right)\left(S_{t_i-1}^{-1}\left(\boldsymbol{x}_{t_i} - \boldsymbol{\mu}\right)\right)^T}{t_i - 1 + \left(\boldsymbol{x}_{t_i} - \boldsymbol{\mu}\right)^T S_{t_i-1}^{-1}\left(\boldsymbol{x}_{t_i} - \boldsymbol{\mu}\right)} \tag{3.20}$$

3.4.2 Our novel Incremental Similarity measurement

In this paper, we propose the use of Incremental Similarity (IS) in several models, with a satisfying trade-off between the computational cost and accuracy. The main reason for searching for such trade-off is the real-time setting of many on-line and incremental learning systems. Here, the computational cost is one of the crucial factors for the decision about the learning and recognition model. In our work we focus on real-time on-line learning with starting from scratch. Our method offers a simple updating formula that does not rely on covariance matrix and at the same time is able to deliver high accuracy. Rather that the distance from mean and covariances, it describes the distance from all points in the rule i (3.21).

$$\beta_i = \frac{1}{1+\frac{1}{t_i}\sum_{j=1}^{t_i}\left(\boldsymbol{x}-\boldsymbol{x}_j\right)^2} \tag{3.21}$$

In order to accommodate incremental learning into the model, we need to bring up a recursive model for learning the membership function that can be easily derived from the original non-recursive one. This is achieved by substituting $\boldsymbol{x}^2$, $\sum_{j=1}^{t_i}\boldsymbol{x}_j$ and $\sum_{j=1}^{t_i}\boldsymbol{x}_j^2$ using parameters α_i, $\boldsymbol{\eta}_i$ and γ_i (3.26), along with update formulas for these parameters in an incremental manner. Here, none of the samples will be forgotten or changed, but in these parameters, the true values of past samples are stored. Thus, every time new sample arrives to enrich the rule, updating formulas are triggered so that for next sample with unknown competency for a rule, membership β_i is derived (3.25).

$$\beta_i = \frac{1}{1+\frac{1}{t_i}\sum_{j=1}^{t_i}\left(\boldsymbol{x}^2-2\boldsymbol{x}\boldsymbol{x}_j+\boldsymbol{x}_j^2\right)} \tag{3.22}$$

$$= \frac{1}{1+\boldsymbol{x}^2-\frac{1}{t_i}2\boldsymbol{x}\sum_{j=1}^{t_i}\boldsymbol{x}_j+\frac{1}{t_i}\sum_{j=1}^{t_i}\boldsymbol{x}_j^2} \tag{3.23}$$

$$\beta_i = \frac{t_i}{t_i+t_i\boldsymbol{x}^2-2\boldsymbol{x}\sum_{j=1}^{t_i}\boldsymbol{x}_j+\sum_{j=1}^{t_i}\boldsymbol{x}_j^2} \tag{3.24}$$

$$\beta_i = \frac{t_i}{t_i+t_i\alpha_i-2\boldsymbol{x}\boldsymbol{\eta}_i+\gamma_i} \tag{3.25}$$

$$\begin{aligned} &a)\, t_i = t_{i-1}+1 \qquad &&c)\, \boldsymbol{\eta}_i = \boldsymbol{\eta}_{i-1}+\boldsymbol{x};\ \boldsymbol{\eta}_0 = 0 \\ &b)\, \alpha_i = \boldsymbol{x}^2 &&d)\, \gamma_i = \gamma_{i-1}+\alpha_i;\ \gamma_0 = 0 \end{aligned} \tag{3.26}$$

Using this novel technique we tackle the challenge of the lack of data at the beginning of the learning process as well as the processing time vs. accuracy challenge. Since our proposed IS is not dependent on a high number of samples to create relevant statistics (as would be the case in Normal distributions), it can describe small amount of data better.

3.5 Models

In this section, we describe 4 TS fuzzy models and K-means that we combine with ED, MD and IS. As we show in Section 3.6, the combination with IS leads to superior results compared to the models solely. Each Neuro-Fuzzy model is composed of a number of fuzzy rules in a form of $IF\ x\ IS\ a\ THEN\ b$. Here, the $x\ IS\ a$ is the antecedent part of the rule based on the similarity or the firing degree of the rule, denoting how much x is similar to a, x being a single input sample, a being a set of samples within the rule already known (not necessarily of one class) and b being the consequence of the rule. In a classification part, each rule gives an opinion for every class recorded in the system in a form of value within range [0, 1]. Then, these values are weighted by the membership function and averaged. The winning class is assigned to the class for which the weighted averaged opinion was the highest, i.e. the most confident. In the case of K-means, we choose the class that is most present (according to labels) within each cluster. More detailed description is given in Section 3.5.5.

3.5.1 ETS

In (Angelov *et al.*, 2004) the authors introduce a novel evolving TS (ETS) fuzzy model. The generation and management of rules is handled by Recursive mountain clustering, in which each new sample is assigned a potential that is compared to updated potentials of known centers of rules (tops of the mountains). These centers are updated within the algorithm. These updates are based on the samples, which are added to the centers or those which replace the centers based on distance conditions. In addition, each time a new class is added to the system, a new rule is created for it as well. In the original paper, the antecedent part is derived as univariate normal density function, however it has been replaced by ED in current paper. For the consequent part, authors propose to use local RLS, in which the optimization is handled for each rule locally, rather than globally for whole system.

3.5.2 ETS+

In (Almaksour *et al.*, 2010) the authors introduce the ETS+ model inspired by ETS. They also use Recursive mountain clustering for rule management, however with some slight adjustments. Similarly to ETS also in ETS+ is the addition of new rules influenced by the arrival of new classes. Firstly the multivariate normal along with Couchy distributions were proposed for handling the antecedent part, but MD yielded better results and is used for comparison in this paper. Similarly to ETS, authors use local RLS for the consequent part.

3.5.3 Incremental fuzzy model (IFM)

In (Režnáková *et al.*, 2012) the authors propose the use of IS for TS fuzzy models combined with very simple clustering based on the performance of the winning rule for each sample. Similarly to other approaches, it takes into account the arrival of new classes which influence the creation of new rules. It is based on the belief that on one hand each rule can, and does contain samples from more classes, but at the same time it should not be too confused in distinguishing them using consequent learning. Consequent learning itself is also based on local RLS.

3.5.4 ARTIST: ART-2A driven rule management in TS model

In (Režnáková *et al.*, 2013), the authors propose to use the ART-2A neural network (Carpenter *et al.*, 1991b) for cluster management. The generation of new rules is left solely on the incremental clustering without the influence of the arrival new classes. Similarities within the antecedent part are understood as possibilities rather as probabilities. This means, that one sample can be fully possible for all (giving 1 for every rule) or none of the rules (giving 0 for all of them). Then, for the consequent part, Competitive Consequent Learning (CCL) is used to avoid the class forgetting in incremental learning.

3.5.5 K-means

K-means clustering algorithm is based on k clusters (means) where the distance to each of them is measured by the Euclidean formula. At each step, all points are assigned to the closest mean and all means are recalculated. Then, the distance between the old and the new means is used as a stop rule for the whole process. This process iterates until the stop criterion is met.

Since we focus on incremental learning techniques, we do not update the centroids at the end of the assigning, but during this process. For this purpose we use the same updating formulas as in Section 3.4 (ED, MD and IS). Thus, for the K-means combined with MD, on the top of mean values μ_i we add the update of covariance matrices S_i and parameters α_i, $\eta_i$and γ_i for K-means combined with IS. In our evaluation we put the main focus on determining the distance and learning the distance parameters, not the latter phases of the method.

Since the original K-means is unsupervised clustering and our problems are supervised, we need to change it in this manner. Thus, during the assignment of samples to clusters and the recalculation of parameters, we also assign labels to each cluster. Then, we use a majority vote to determine the winning label in the cluster. In results we will notice that in order to make this method work precisely, many clusters are needed, as they must divide the space very carefully. We also need to point out, that we use K-means only to demonstrate the differences between distance measurements in this paper.

3.6 Results

In this section we show a detailed comparison of the distance measurements described in Section 3 to show the potential of Incremental Similarity (IS) for incremental learning purposes and as an interesting trade-off between using only mean and possibly variance and using covariance matrix. For the evaluation of our IS in comparison to different baseline metrics, we use five datasets described in Table 3.1[1]. In this section we show the results on only three of all datasets, with the rest displayed online[2].

Table 3.1 Datasets

	# classes	# samples	# dimensions
Digits	10	10992	16
OHS-I	17	13600	21
OHS-II	20	1923	50
Segment	7	2310	19
Mushrooms	2	8124	22

For each dataset, we display the results in 4 tables (A-D), first two for evolving fuzzy models and last two for K-means. All the results are then composed of the following entities: table A - accuracy for the whole dataset / time required to learn and recognize the whole dataset / time per one rule required to process the whole dataset; table B - average time required to learn parameters for the last 100 instances / time required to derive distance for the last 100 instances; table C - accuracy for the whole dataset / time required to learn and recognize the whole dataset; table D - average time required to learn parameters for one instance / time required to derive distance for one instance during assignment to clusters / time required to derive distance for one instance during classification.

For each dataset we separate the results for the Neuro-Fuzzy based models (tables A and B) and K-means (tables C and D). For K-means we find the best parameters k for each combination (based on accuracy) using by cross-validation within the interval <c, c+5, ..., 195, 200>, where c represents the number of classes for each dataset. We select 2/3 of dataset as a learning base and 1/3 as a testing base. The complete results are shown in Figures 3.1 (for accuracy) and 3.2

[1] Digits, Mushrooms and Segments can be found at UCI Machine Learning Repository; Online Handwritten Symbols I (OHS-I) can be found at http://www.synchromedia.ca/web/ets/gesturedataset; Online Handwritten Symbols II (OHS-II) can be found at http://www.irisa.fr/

[2] Complete results are published at http://www.synchromedia.ca/membres/marta/IncrementalSimilarityResults.html

(for processing time). Note, that whereas for accuracy, the best results are the highest ones, for processing time the aim are the lowest values.

In Table 3.2 we can notice that most precise for the Digits dataset are models using the MD achieving 91% recognition rate, followed by the ED achieving 88.9% and the IS with 87.7% recognition rate. Out of all these models, ARTIST+MD seems to be the most accurate and ETS+ED the fastest. As for the processing time, the most successful is the IS with 716.4/8.1 (sec) on average, followed by the ED with 739/8.4 (sec) and the MS with 957.9/10.35 (sec). The best processing time is achieved by ETS+ED (total processing time) and ETS++IS (time per rule).

We need to note, that these times, especially the time for processing the whole dataset, are influenced by many factors, such as the number of rules with exponential influence on the results, algorithms used for derivation of vector and matrix multiplications, etc.

In Table 3.3 we compare the processing time of parameters learning and distance derivation. We can notice that processing times of derivation of the distances are similar, as the formulas are comparably complex, especially for the ED and IS. However, for the updating formulas, the best results are achieved with the IS. This is no surprise and we will encounter similar result for K-means also. It is caused by the updating formulas themselves, where for the IS we simply perform a summation while for the ED we multiply and divide. This situation can be improved by making the updates simpler.

Table 3.4 shows the results for lowest, highest and best K-means for each distance for a fair comparison. We can notice that in all cases, IS with K-means outperforms both other distances, although its accuracy is comparable with the results for the K-means+ED. As for the processing time, the lowest in general is achieved by the ED, followed by the IS and the MD.

In Table 3.5 it is noticeable that for the learning of parameters, K-means+IS achieves the best processing time. However, we need to mention again, that although we used the well-known recursive formula for updating mean for ED, it is not needed due to the usage of means at the end of this learning process. If we use the same strategy as for the IS and at the end divide the summed samples by their total amount, the learning time would be smaller than for the IS, as that executes one more operation. Such a difference in updating will also change the formula for deriving the distance itself (as we need to derive the mean vector as well). For the formula updating used in this work, the ED achieves the best processing time, followed by the IS and the MD.

Table 3.2 Digits A - whole dataset results, Evolving fuzzy models (%/sec/sec)

	ED	MD	IS
ETS	85.4/**45**/4.9	**87.1**/55.2/5.4	85.2/49.5/**4.8**
ETS+	84.9/**49.2**/4.8	**87.1**/56.4/5.4	86.0/50.4/**4.1**
IFM	91.0/**768/9.6**	**93.9**/1044/11.4	89.7/810/10.2
ARTIST	94.2/2094/14.4	**95.9**/2676/19.2	89.9/**1956/13.4**

Table 3.3 Digits B - last 100 samples results, Evolving fuzzy models (ms/ms)

	ED	MD	IS
ETS	11.2/4.4	25.4/2.2	**6.2/2.1**
ETS+	8.1/3.6	21.7/6.5	**7.4/2.5**
IFM	37.1/13.5	188.2/23.1	**24.2/8.3**
ARTIST	**6.8**/17.2	126.1/**1.5**	15.1/18.2

Table 3.4 Digits C - whole dataset results, K-means (%/sec)

k	ED	MD	IS
10 (lowest)	72.9/**9.6**	29.9/49.2	**73.1**/13.4
185 (best ED)	**97.2/86.4**	91.8/294	**97.2**/314
190 (best IS)	97.0/**93.6**	92.6/309.6	**97.3**/312.1
195 (best MD)	97.1/**97.8**	93.1/348.6	**97.3**/302.4
200 (highest)	97.0/**90.6**	92.0/413.4	**97.3**/334.8

For the Online handwritten symbols I dataset (in Table 3.6), the most precise method proved to be the IS, achieving 98% rate, followed by the ED achieving 97.98% rate and the MD achieving 97.3% rate. The lowest processing time is achieved with models using the ED with 812.7/21.3

Table 3.5 Digits D - per one step results, K-means (ms/ms/ms)

k	ED	MD	IS
10	0.15/**0.003/0.003**	0.28/0.009/0.009	**0.07**/0.007/0.008
185	0.16/**0.003/0.003**	0.39/0.009/0.009	**0.06**/0.007/0.006
190	0.15/**0.003/0.003**	0.41/0.009/0.009	**0.06**/0.007/0.006
195	0.17/**0.003/0.003**	0.38/0.009/0.008	**0.06**/0.006/0.007
200	0.16/**0.003/0.003**	0.42/0.009/0.01	**0.05**/0.006/0.007

(sec), the IS with 927.85/22.8 (sec) and the MD with 1039.65/29.7 (sec). From all the above, the combination judged the most accurate is the ETS+IS and the fastest are ETS+ED (for total processing time) and IFM+ED also (for time per rule). Using our proposed setup we are able to increase the precision and at the same time reduce the processing time.

In Table 3.7 we notice that the learning time of both the ED and the IS are comparable, with the IS being slightly better, followed by the MD.

In Table 3.8 we notice, that similarly to the Digits dataset, also for Online symbols I, using the IS leads to the best accuracy, closely followed by the ED and then by the MD. As for the processing time, using ED achieves the best performance.

In Table 3.9 we can notice that in terms of parameter learning, the IS leads, followed by the ED and the MD. As for the derivation of the distance itself, ED outperforms the others, followed by the IS and then the MD.

Table 3.6 Online handwritten symbols I A - whole dataset results, Evolving fuzzy models (%/sec/sec)

	ED	MD	IS
ETS	97.8/**342/20.4**	97.8/543/31.8	**98.0**/468/25.6
ETS+	98.2/**448.8**/26.4	97.9/555.6/32.4	**98.4**/464.4/**25.2**
IFM	97.9/**1758/12.1**	**98.2**/1986/16.2	98.1/2101/15.1
ARTIST	**98.0**/702/26.4	95.2/1074/38.4	**98.0/678/25.2**

Table 3.7 Online handwritten symbols I B - the last 100 samples results, Evolving fuzzy models (ms/ms)

	ED	MD	IS
ETS	39.1/15.2	110/11.3	**28.1/9.2**
ETS+	35.5/14.7	101/25.1	**29.4/9.1**
IFM	41.5/14.6	198/25.2	**33.2/9.2**
ARTIST	**6.1**/25.2	81.1/17.4	16.1/**16.1**

Table 3.8 Online handwritten symbols I C - whole dataset results, K-means (%/sec)

k	ED	MD	IS
17 (lowest)	**54.8/13.8**	45.4/97.8	51.1/52.8
185 (2nd MD)	90.1/**138.6**	89.4/889.2	**91.0**/576.6
190 (2nd ED)	90.5/**133.8**	89.1/906	**92.1**/547.2
195 (2nd IS)	90.5/**141.6**	88.7/824.4	**92.1**/552
200 (best)	91.2/**132.6**	90.1/707.4	**92.3**/487.8

For Online Handwritten symbols II (Table 3.10), the most accurate models are those using the IS achieving 91.2% rate, followed by the ED with 91.15% rate and the MD with 91.0% rate. From all of the above models, ARTIST+ED seems to be the most accurate. The best performance time is achieved by the models using the ED with 207.75/3.2 (sec) followed by the IS with 217.55/3.45 (sec) and the MD with 341.9/5.3 (sec). From the results, the best processing time from all of the combinations is achieved by ETS+ED (for total processing time) and both IFM+ED and IFM+IS (for time per rule).

Table 3.9 Online handwritten symbols I D - per one step results, K-means (ms/ms/ms)

k	ED	MD	IS
17	0.21/**0.005/0.005**	0.93/0.014/0.012	**0.12**/0.009/0.009
185	0.26/**0.004/0.003**	1.41/0.015/0.015	**0.1**/0.008/0.008
190	0.25/**0.004/0.004**	1.37/0.014/0.015	**0.11**/0.008/0.008
195	0.25/**0.004/0.004**	1.39/0.015/0.015	**0.11**/0.008/0.008
200	0.24/**0.004/0.003**	1.44/0.015/0.014	**0.11**/0.008/0.009

For parameter learning for the Online symbols II dataset (in Table 3.11), the ED and the IS achieve comparable results for average update on last 100 samples. The derivation of the distance is the fastest for the IS, followed by the ED and the MD.

For Online symbols II, K-means achieves the best results using the IS, while the ED achieves similarly good results, both followed by the MD (Table 3.12). As for the performance time, the fastest proved to be the ED, followed by the MD performing comparably well as the IS.

In Table 3.13 we can notice that the IS leads on the learning performance time, followed by the ED and the MD. As for the derivation time, the ED achieves the best results, closely followed by the IS and then the MD.

Table 3.10 Online handwritten symbols II A - whole dataset results, Evolving fuzzy models (%/sec/sec)

	ED	MD	IS
ETS	91.2/**69.6/3.6**	91.1/123.6/6.1	**91.4**/81.2/4.2
ETS+	90.9/**86.4/4.2**	91.0/138.6/6.6	**91.3**/87/**4.2**
IFM	90.4/**228/1.8**	90.4/355.2/3.2	**91.1/228/1.8**
ARTIST	**92.1/447/3.1**	91.6/750.3/5.4	91.1/474/3.6

Table 3.11 Online handwritten symbols II B - the last 100 samples results, Evolving fuzzy models (ms/ms)

	ED	MD	IS
ETS	39.5/15.1	101.2/11.5	**27.4/10.1**
ETS+	33.6/14.9	149.1/26.5	**14.4/8.1**
IFM	**17.1**/14.8	197/23.1	24.1/**7.2**
ARTIST	**6.9**/23.7	144.1/**12.9**	29.1/21.3

3.7 Conclusion and discussion

In this paper we have presented a novel Incremental Similarity (IS) measurement for incremental learning models. We evaluated our technique using 5 baseline models and achieved better

Table 3.12 Online handwritten symbols II C - whole dataset results, K-means (%/sec)

k	ED	MD	IS
21 (lowest)	61.8/**1.8**	52.7/2.4	**62.6**/2.1
180 (best IS)	87.4/**6.3**	82.7/14.4	**87.8**/16.8
195 (best MD)	86.9/**7.2**	84.0/16.2	**87.6**/18.9
200 (best ED)	**87.9/7.2**	83.8/14.4	87.6/17.8

Table 3.13 Online handwritten symbols II D - per one step results, K-means (ms/ms/ms)

k	ED	MD	IS
21	0.02/**0.004/0.004**	0.16/0.021/0.023	**0.007**/0.008/0.007
180	0.03/**0.003/0.004**	0.17/0.025/0.025	**0.005**/0.007/0.006
195	0.04/**0.003/0.003**	0.26/0.03/0.027	**0.009**/0.007/0.007
200	0.04/**0.003/0.004**	0.33/0.028/0.026	**0.009**/0.007/0.007

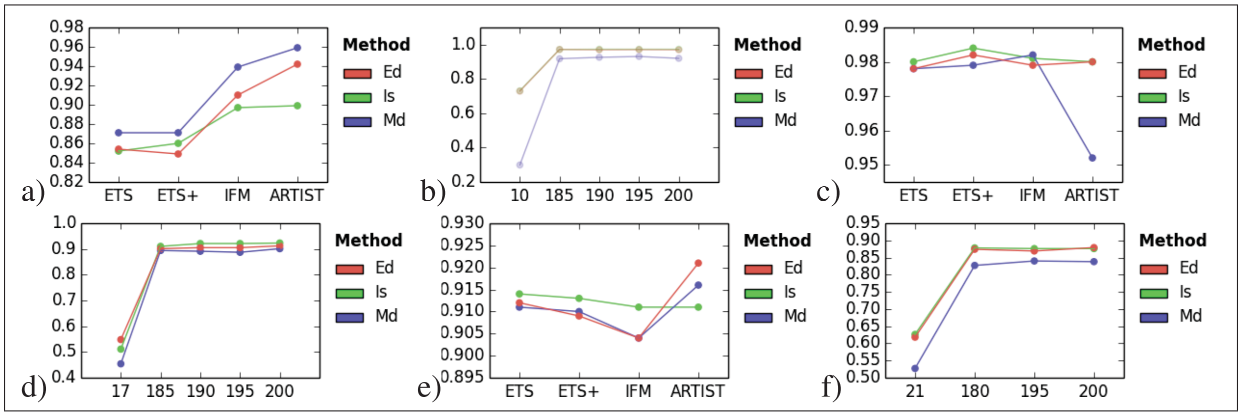

Figure 3.1 Recognition rate a), c) and e) for various Neuro-Fuzzy models, b), d) and f) for k-means and various k, on Digits (a), b)), OHS-I (c), d)) and OHS-II (e), f)) datasets.

results compared to the baseline techniques. This evaluation was performed on 5 datasets that varied in the number of classes, features and sample size. In all of the applications IS achieved a good trade-off between high accuracy and low computational cost. In some rare cases when our technique did not achieve the best accuracy, this slight reduction in accuracy was exchanged for a significant reduction in processing speed.

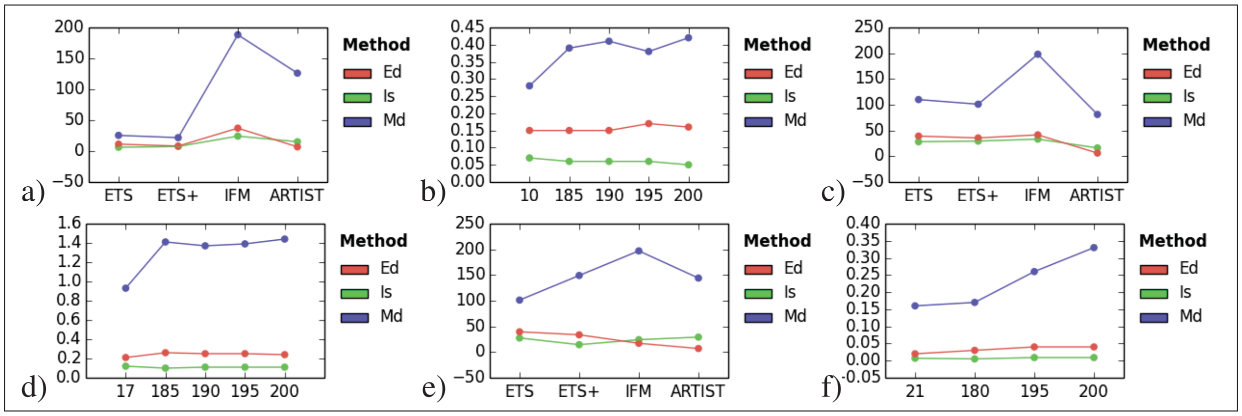

Figure 3.2 Computational complexity a), c) and e) for various Neuro-Fuzzy models, b), d) and f) for k-means and various k, on Digits (a), b)), OHS-I (c), d)) and OHS-II (e), f)) datasets.

Generally we can say that IS achieves better results than the any of the baseline techniques. This means, that not only is it highly accurate, but also it is adaptable to various distributions of data. This is very intriguing for incremental learning in which data is unknown beforehand and we cannot predict their distribution. In some specific cases for K-means method, the IS is outperformed by the MD. In our opinion, this is caused by the nature of K-means, where it iterates until it does not meet the stop criterion. Thus, the longer it takes to find the best means, the higher performance time it results in.

In our future work we want to explore applications of IS to a wider range of models that do not necessarily include fuzzy models only.

Acknowledgement

This work was supported by the Social Sciences and Humanities Research Council of Canada (SSHRC) as well as the Natural Sciences and Engineering Research Council of Canada (NSERC).

CHAPTER 4

ELASTIC MEMORY LEARNING FOR FUZZY INFERENCE MODELS

Marta Režnáková[1], LukasTencer[1], Mohamed Cheriet[1]

[1] Synchromedia Lab, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

Paper submitted for publication, October 2016.

4.1 Abstract

Abstract In this paper, we present a novel approach for solving the consequent part of Neuro-Fuzzy modeling with an emphasis on the forgetting factor in the multi-class learning problem. Our solution is based on Recursive Least Squares (RLS) for online and incremental learning applications, where the data stream is not necessarily uniformly distributed over time. Such a setup can lead to forgetting of specific classes that have not been used for a period of time. In this work, we present a reasoned and detailed description of Elastic Memory Learning (EML) and EML with the use of Confidence Interval (EML+) to avoid unnecessary treatment of the forgetting factor. We present the experimental results and evaluation of our novel methods in order to show their usefulness not only against forgetting of unused classes, but also for dealing with the lowered recognition rate after all classes have been learned. We note that by using EML, forgetting is significantly eliminated and the recognition rate is slightly affected as well, while EML+ puts more emphasis on keeping the recognition rate higher than the forgetting. Thus, this paper presents two methods that significantly eliminate the forgetting factor for incremental learning with a different focus on its importance, i.e. high recognition rate vs high immunity to forgetting; both of these methods perform significantly better than RLS for these aspects.

4.2 Introduction

Neuro-Fuzzy models divide the feature space into a number of fuzzy rules that are responsible for the local inferences (4.1). To derive the final inference of the model as a whole these local inferences are weighted and combined into a global inference. The main components needed to make an inference are the antecedent ($IF\ x\ IS\ A$) and consequent ($THEN\,\{y_i^j\}_{j=1..c}$) parts. In the case of online learning, these use sets of parameters learned from the data already known to make a prediction about the data not yet known. In this work we focus on the consequent part, which is responsible for the inference of each rule and the learning of its parameters. Each time a new datum occurs, its fuzzy possibility (inference y_i^j) for each class j within each rule i is derived. This information is then used for the update/learning of the parameters. Therefore, the learning of the consequent parameters is performed for all classes within each rule for each data point. However, such an approach can result in forgetting of certain class. A class that has not been used for a long period of time is surpassed by classes that are more frequent in the learning process. In most benchmarks for online learning with balanced classes, the data sets are shuffled so that such behavior is not noticeable, i.e. during the learning process all classes occur with a similar frequency and no blocks are created. However, in a real-world environment, this is not a universal behavior and especially in applications such as gesture-based interaction in which certain classes/gestures are far more frequent than others. Moreover, in cases with unbalanced classes, this behavior is usual even with the shuffled data. In our work we focus on real-time incremental learning of handwritten gestures, and thus the forgetting factor is very likely to be present in a real-world environment.

$$R_i : IF\ x\ IS\ A\ THEN\ Y_i = \{y_i^j\}_{j=1..c} \tag{4.1}$$

In this paper, we introduce two techniques to deal with the forgetting of unused classes. Each of these techniques, Elastic Memory Learning (EML) and EML+, is an extension of Recursive Least Squares (RLS) with a different sensitivity towards the trade-off between immunity from forgetting and the generally high recognition rate. From our previous experiments we concluded that for some data sets high immunity from forgetting results in a slightly decreased recognition rate when no forgetting occurs. This finding motivated our further research to find a better balance for the solution for forgetting of unused classes and the generally high performance. Both these methods focus on the consequent part and are applied to a number of evolving neuro-fuzzy models. As will be seen in the results section, both solution perform comparably, but with each being superior to the other in specific applications. Thus, we leave the choice of method to be driven by the application.

In the next section we introduce the relevant works used for this research. In order to introduce the reader to the whole work, we describe the RLS optimization that is often used for consequent learning in incremental or online Takagi-Sugeno models in section 4.4. We use this method as a baseline for TS models in this paper. Sections 4.5 and 4.6 are dedicated to our novel models, EML and EML+. It is demonstrated in the results (section 4.7), that by using our EML and EML+ both forgetting and the recovery recognition rate have been improved so as to be able to compete with and even overcome the recognition rate using the original RLS. In section 4.8, by way of conclusion, discuss our approaches and the specific results.

4.3 Related works

Many applications of machine learning need to process data in a sequence rather than a bulk which has resulted into a wide spectrum of online learning based models (Gama (2010); Moamar-Mouchaweh, Sayed; Lughofer (2012); Angelov, Plamen; Filev, Dimitar P.; Kasabov (2010)). In this paper we focus on neuro-fuzzy modeling and its application on several tasks within classification problems with an highlight in online handwritten symbols recognition. Therefore in this section we introduce several techniques tackling online and incremental learning with a stronger focus on neuro-fuzzy modeling.

The online learning models can be separated into several groups by the framework they use. Often, an inspiration for online models is offline models that have been successfully applied to similar tasks. There have been a number of studies of online Bordes (2005); Tax (2003) and incremental Carozza *et al.* (2000); Wang *et al.* (2007, 2008); Gua *et al.* (2015) Support Vector Machines (SVM) in which the support vectors (SVs) are updated as the size of the data increases. To add a new SV or change the former SVs, a set of conditions needs to be satisfied, and such a decision is made for every new data point.

In order to adapt the boosting method to work in an online setting, a bootsrap is performed using mostly Poisson distribution that guides the addition of each new data point into each known weak classifier Oza (2005). In this work, an online version of boosting is proposed as well. In it, a parameter is adapted on the basis of the error of the data point, and its weight is indicated in the learning process. From this process, an AdaBoost (adaptive boosting) algorithm is proposed. Other versions of AdaBoost have been proposed by Grabner *et al.* (2006) for vision problems, by Saffari *et al.* (2010) and Ditzler *et al.* (2010) for multi-class online boosting and by Wen *et al.* (2015) for vehicle classification. Online Random forests were proposed in Leistner *et al.* (2009). In the ensemble, the boosted decision trees are replaced by extremely randomized trees and are grown in an incremental manner following the online boosting strategy. A few conditions are added on the splitting of an existing leaf of a tree or the tree structure in order to better understand the space they subsequently divide into small portions that are easy to classify.

Incremental learning of the subspace was proposed in Kasabov *et al.* (2005) and applied to feature extraction, Li (2004) with an application on face recognition, Ross *et al.* (2007) for visual tracking, Yao *et al.* (2010) for hand gesture recognition for incremental PCA, and incremental Principal Component Regression (PCR) in Hall *et al.* (1998), compared to incremental fuzzy models in Cernuda *et al.* (2012). In Cheng *et al.* (2016) the authors used temporal t-stochastic neighbor embedding for incremental learning of silhouettes.

Other models that can be mentioned are online K-Means Beringer *et al.* (2006); incremental K-NN Forster *et al.* (2010); Yu *et al.* (2009); Vector Quantization Lughofer (2008b); Adaptive Resonance Theory proposed in Carpenter *et al.* (2010, 2017); ART-2A proposed in Carpenter *et al.* (1991b) and Recursive Mountain Clustering Angelov, Plamen; Filev, Dimitar P.; Kasabov (2010).

The main focus of this paper is on evolving or adaptive neuro-fuzzy models. They are composed of a number of fuzzy rules that can be separated into two main parts, i.e. antecedent and consequent. To derive a final decision, a similarity to the antecedent part is derived (the IF part of the rule) and is followed by the consequence (the THEN part of the rule). These models are usually dependent upon two basic models: the Takagi-Sugeno fuzzy model Takagi *et al.* (1985) and the Mamdani model Mamdani *et al.* (1975).

The neuro-fuzzy models normally vary in the clustering part, which is used for deriving the fuzzy rules of the model, i.e. the whole structure. Thus, several clustering methods have been applied to the evolving neuro-fuzzy models, for example genetic algorithms in Carse *et al.* (1996); Gomez *et al.* (2002); Kasabov (2001); the Recursive Mountain Clustering used in Angelov *et al.* (2004), Almaksour *et al.* (2010); connectionist systems in Kasabov (2001); incremental clustering in Režnáková *et al.* (2012), ART-2A in Režnáková *et al.* (2013); VQ further developed in Lughofer (2008a) and later expanded in Lughofer *et al.* (2011), where the authors propose new elimination techniques for rule redundancies, including merging of the rules. In Lughofer *et al.* (2015) the authors propose a new way for merging the rules. In Lughofer *et al.* (2013b) the authors introduce a new perspective on the learning of the consequent part using the all-pairs approach. A study of the differences within models using different clustering methods was done in Babuska (1998). A comprehensive survey of this topic was made by Lughofer (2011, 2013).

In online learning, the optimization techniques need to be adapted accordingly to be able to work with a stream of data instead of having to deal with the whole dataset at once. In this study we focus on RLS optimization Ljung (1999); Angelov *et al.* (2008), which is a recursive adaptation of Least Squares optimization. We use it as a baseline in our experimental setting and apply our work, described later in this paper, to various evolving neuro-fuzzy models. This optimization is linear, making the whole model only partially linear since the combination of the rules is not linear. However, there are some studies on non-linear fuzzy modeling as well Babuška *et al.* (2003); Tan *et al.* (2000). Many incremental online learning methods can be easily applied using MOA (Massive Online Analysis, Bifet *et al.* (2010)).

4.4 Recursive Least Squares for TS fuzzy models

Each neuro-fuzzy model is composed of two main components, the antecedent part and the consequent part. In this paper we focus on the consequent part because it is the main contributor to the learning process and the making of the inference. For this purpose, a set of local parameters is used, $\Pi_i = \left\{\boldsymbol{\pi}_i^j\right\}_{j=1\ldots c}$; $i = 1\ldots r$ for r rules and c classes. The error to be minimized is derived as a squared difference of an actual result $\beta_{i,k}\boldsymbol{x}_k\Pi_i = \left\{\beta_{i,k}\boldsymbol{x}_k\boldsymbol{\pi}_i^j\right\}_{j=1\ldots c}$ and an expected result $Y_{i,k}^{'} = \left\{a^j\right\}_{j=1\ldots c}$ (4.2). Here, $a^j = 0$ for the inactive class and $a^j = 1$ for the active class. $\beta_{i,k}$ is a scalar, the firing degree of rule i for sample $\boldsymbol{x}_k$ at time k.

$$E = \sum_{k=1}^{t} ||\beta_{i,k}\boldsymbol{x}_k\Pi_i - Y_{i,k}^{'}||^2 \tag{4.2}$$

To minimize this error, Least Squares optimization is used, where the aim is to find parameters Π_i to fit all samples $\boldsymbol{x}_k$. To apply this to online learning, recursive updates are in need. Thus, $\Pi_{i,t}$ and $C_{i,t}$ are parameters and convariance matrix for rule i at time t. Here, $k = 1\ldots t$ are all the updates form the beginning fo the leanring until the current time t. Then, parameters at time $t-1$ are derived using all samples $\boldsymbol{x}_k$; $k = 1\ldots t-1$.

$$\Pi_{i,t} = \sum_{k=1}^{t} \beta_{i,k}\boldsymbol{x}_k Y_{i,k}^{'} \left(\sum_{k=1}^{t} \beta_{i,k}\boldsymbol{x}_k \beta_{i,k}\boldsymbol{x}_k^T\right)^{-1} = M_{i,t}C_{i,t}^{-1} \tag{4.3}$$

$$M_{i,t} = \sum_{k=1}^{t} \beta_{i,k}\boldsymbol{x}_k Y_{i,k}^{'} = \sum_{k=1}^{t-1} \beta_{i,k}\boldsymbol{x}_k Y_{i,k}^{'} + \beta_{i,t}\boldsymbol{x}_t Y_{i,t} = M_{i,t-1} + \beta_{i,t}\boldsymbol{x}_t Y_{i,t}^{'} \tag{4.4}$$

$$C_{i,t}=\sum_{k=1}^{t}\beta_{i,k}\boldsymbol{x}_k\beta_{i,k}\boldsymbol{x}_k^T=\sum_{k=1}^{t-1}\beta_{i,k}\boldsymbol{x}_k\beta_{i,k}\boldsymbol{x}_k^T+\beta_{i,t}\boldsymbol{x}_t\beta_{i,t}\boldsymbol{x}_t^T=C_{i,t-1}+\beta_{i,t}\boldsymbol{x}_t\beta_{i,t}\boldsymbol{x}_t^T \tag{4.5}$$

From equations (4.3)-(4.5) we derive the final updating formula for the covariance matrix $C_{i,t}$ (4.6) for rule i at time t.

$$C_{i,t}^{-1}=C_{i,t-1}^{-1}-\frac{C_{i,t-1}^{-1}\beta_{i,t}\boldsymbol{x}_t\beta_{i,t}\boldsymbol{x}_t^T C_{i,t-1}^{-1}}{1+\beta_{i,t}\boldsymbol{x}_t^T C_{i,t-1}^{-1}\beta_{i,t}\boldsymbol{x}_t} \tag{4.6}$$

From equation (4.7) the updating formula for parameters $\Pi_{i,t}$ is derived (4.8), where i is the index of the corresponding rule and t is the time, which corresponds to the number of samples already presented. Then, $\Pi_{i,t-1}$ and $C_{i,t}^{-1}$ are the parameters and covariance matric from the previous step and are used to derive the current result $\Pi_{i,t-1}\beta_{i,t}\boldsymbol{x}_t$.

$$\Pi_{i,t}=M_{i,t}C_{i,t}^{-1}=\Pi_{i,t-1}-C_{i,t}^{-1}\beta_{i,t}\boldsymbol{x}_t\left(\Pi_{i,t-1}\beta_{i,t}\boldsymbol{x}_t-Y_{i,t}^{'}\right) \tag{4.7}$$

$$\Pi_{i,t}=\Pi_{i,t-1}+C_{i,t-1}^{-1}\beta_{i,t}\boldsymbol{x}_t\left(Y_{i,t}^{'}-\Pi_{i,t-1}\beta_{i,t}\boldsymbol{x}_t\right) \tag{4.8}$$

Then, for each rule i and each class j there are local parameters $\boldsymbol{\pi}_{i,t}^{j}$ with an updating formula as shown in (4.9), given that local inference for the corresponding rule and class is given by (4.10) and $\varepsilon_{i,t}^{j}$ being a correction parameter further described in section 4.5 bellow.

$$\boldsymbol{\pi}_{i,t}^{j}=\boldsymbol{\pi}_{i,t-1}^{j}+C_{i,t-1}^{-1}\beta_{i,t}\boldsymbol{x}_t\left(\varepsilon_{i,t}^{j}-\boldsymbol{\pi}_{i,t-1}^{j}\beta_{i,t}\boldsymbol{x}_t\right) \tag{4.9}$$

$$y_{i,t-1}^{j}=\boldsymbol{\pi}_{i,t-1}^{j}\beta_{i,t}\boldsymbol{x}_t \tag{4.10}$$

The final inference for class j is then given by (4.11) and the decision is made by (4.12).

$$Y_t^j = \sum_{i=1}^{r} y_{i,t}^j \tag{4.11}$$

$$j_{winner} = \arg\max_j Y_t^j \tag{4.12}$$

4.5 EML: Elastic Memory Learning

As described in the previous section, for consequent learning we use local RLS. Here, on the basis of the result, each rule is updated by the weight of its firing degree to the sample. Responsible for this update is a correction parameter ε_i^j that is set to either 0 or 1, according to the inference Y_i^j, given rule i and class j. Setting this parameter to 0 leads to adjusting the responsible parameters π at time t for input sample x_t to infer 0, setting ε to 1 likewise, i.e. to correct parameter π. Every time, the parameters for the true class are corrected to 1 and all the others to 0.

However, this leads to a suspicion that if one class is unused for a long time, by updating its parameters within each rule to 0 for each sample, the ability of each rule to recognize such an unused class is forgotten. The correction to 0 is reasonable in cases when the responsible parameters π are very misleading and thus need to be corrected. However, parameters that do not mislead and perform poorly, i.e. infer values close to 0, for the true class and at the same time do not correspond to the true class are corrected unnecessarily. When such classes exist that occur only rarely during some periods of time, which for online learning is possible scenario, and infer low values for the other classes in the data stream, the values of their corresponding parameters are being corrected. This happens with the very rare correction when parameter ε is set to 1. When such a class reappears in the data stream, it has most likely already been forgotten and will be misclassified.

To prevent this from occurring, the correction to 0 is performed only if necessary, i.e. when results y_i^j for such a class j are very misleading, considering that $j*$ is the true class.

$$\varepsilon_{i,t}^{j} = \begin{cases} 1 & j = j^{*} \\ 0 & y_{i,t}^{j} > y_{i,t}^{j*} \\ \boldsymbol{\pi}_{i,t-1}^{j}\beta_{i,t}\boldsymbol{x}_t & otherwise \end{cases} \tag{4.13}$$

We set the parameter $\varepsilon_{i,t}^{j}$ to 1 if class j is the desired one. Setting $\varepsilon_{i,t}^{j}$ to 0 triggers when the parameters corresponding to the classes other than the true class result in an inference greater than the inference for the true class. Lastly, $\varepsilon_{i,t}^{j}$ is set to $\pi_{i,t-1}^{j}\beta_{i,t}x_t$ in all other cases, changing the formula (4.9) to (4.14) and leaving parameters $\pi_{i,t}^{j}$ unchanged.

$$\pi_{i,t}^{j} = \pi_{i,t-1}^{j} \tag{4.14}$$

As has been demonstrated in Režnáková *et al.* (2013), this kind of a competitiveness in updating the consequent parameters leads to better results even if some of the classes are not used for long periods of time; the model is prevented from forgetting them. This ability is very important especially when the distribution of classes of incoming samples is not necessarily uniform.

4.6 EML+

In EML, the correction learning, i.e. setting parameters to output 0, is performed only on parameters with higher inference than those of the winning class and their corresponding classes are not the true class. However, this can affect the learning of classes whose parameters of some rules result in reasonably high possibility, but at the same time are ignored by the correction learning since they are not the highest possibilities within their rules. In this section we introduce Elastic Memory Learning+ (EML+) in which the correction learning is triggered by the confidence interval (CI) of each rule rather than by only the top results(4.15). Thus, if the inference y_i^j of rule i and class j falls within the CI, the rule is confident enough that this class can be the winning class, even though it is mistaken, then we should allow the model to update the parameters. This equals saying that all such parameters that do not belong to the true class result in misleading possibilities and, as such, they mismatch the classes and need to be treated accordingly. From our point of view, such a treatment is the correction learning, whereby the correction parameter ε_i^j for rule i and class j is set to 0.

$$CI = \pm 1.96 \sqrt{\frac{y_i^{j^{winner}} \left(1 - y_i^{j^{winner}}\right)}{t_i}} \tag{4.15}$$

In what follows we set the correction parameter $\varepsilon_{i,t}$ for the parameters updating the formula in (4.9) to the values shown in (4.16). Here, $\boldsymbol{\pi}_{i,t}^j$ are parameters of rule i at time t for class j, where $i = 1..r$, with r being the number of all rules, and $j = 1...c$, with c being the number of all classes.

$$\varepsilon_{i,t}^j = \begin{cases} 1 & j = j^* \\ 0 & j\, is\, in\, CI\, of\, winner\, and\, j! = j* \\ \boldsymbol{\pi}_{i,t-1}^j \beta_{i,t} \boldsymbol{x}_t & otherwise \end{cases} \tag{4.16}$$

Parameters $\boldsymbol{\pi}_{i,t}^j$ of the correct class $j = j*$ are learned with respect to the expected result being 1 for the current sample $\boldsymbol{x}_t$, i.e. $\varepsilon_{i,t}^j = 1$. Parameters $\boldsymbol{\pi}_{i,t}^j$ of all the classes that are within the confidence interval and at the same time do not belong to the correct class are learned with respect to the expected result being 0 for the current sample, i.e. $\varepsilon_{i,t} = 0$. All the other parameters belonging to classes whose parameters do not fall within the confidence interval and are not the true class are ignored. This results in downsampling the majority class while keeping only the samples crucial for learning the correct decision boundary.

We will see in the results section that such a treatment of parameters can avoid the lowered recognition rate after the unused classes are re-incorporated into the learning process with a similar frequency as the rest of the classes, while at the same time they can still be immune to the forgetting of unused classes.

4.7 Results

In our setup we have applied our methods for consequent learning to a number of evolving neuro-fuzzy models. In ARTIST Režnáková *et al.* (2013), an Adaptive Resonance Theory was used for the generation of rules, while keeping the addition of a rule solely on the clustering approach. In IFM Režnáková *et al.* (2012), incremental clustering based on the erroneous outputs of rules was used. Both of these models use incremental similarity as the firing degree. In ETS Angelov *et al.* (2004) the authors applied Recursive Mountain Clustering for the generation of rules, while using univariate distribution for membership functions then combined into the firing degree. In ETS+ Almaksour *et al.* (2010) the authors extended ETS by using Mahalanobis distance as the firing degree. Both of these methods apply two ways of adding new rules to the model, i.e. the result of clustering or addition of a new class to the system. All of the models were altered to use RLS, EML and EML+.

When the distribution of classes among all the data is not uniform, neither is their arrival at the learning process. Thus, the possibility arises of building blocks of not only adding one class (which actually helps to learn such class), but also blocks of a forgotten class. Here, such a class is not used for some time. In this section we explore the forgetting of one class within fuzzy models using original RLS and our novel competitive techniques, EML and EML+. For the evaluation we have used four models and the results are grouped accordingly. For the evaluation we have used four models and the results are grouped accordingly. The results are shown for 8 distinct datasets[1], described in Table 4.1. Our main motivation is the recognition

[1] Digits, Mushrooms, DNA, Segments, SatImage and USPS can be found at UCI Machine Learning Repository;
Handwritten Gestures I can be found at http://www.synchromedia.ca/web/ets/gesturedataset;
Handwritten Gestures II can be found at http://www.irisa.fr/;

of handwritten gestures, corresponding to the HS I and HS II datasets, but in order to show the performance on various applications.

For each database and for each model, three results are shown: the recognition rate of forgotten class without learning (i.e. only the recognition, no incremental updates); recognition rate of all the rest of the dataset (along with the forgotten class) with learning (i.e. with incremental updates); and the recognition rate on the last 100 samples with learning. The evaluation benchmark is given for all the datasets differing in the number of samples used for each phase, depending on the total number of samples per class. Each run contains four phases, of which the results are shown for the last two phases:

a. learn all classes (50 - 300 samples per class)

b. forget one random class and learn all the rest of the classes (100 - 300 samples per class)

c. phase 3: check the recognition rate of the forgotten class, but do not update the parameters so that we can check the true forgetting for all samples (100 - 300 samples per class)

d. phase 4: learn all classes (the rest of the dataset) while keeping updating the parameters.

Table 4.1 Datasets

Dataset	# of classes	# of dimensions	# of samples
Digits	10	16	10992
HS I	17	20	13600
HS II	21	50	1923
Mush	2	22	8124
Segment	7	19	2310
DNA	3	180	2000
SI	6	36	4435
USPS	10	256	7291

Table 4.2 shows that both methods, EML and EML+, significantly improve the forgetting of unused classes in most of the datasets. On average, EML improves the immunity for forgetting

from a recognition rate of ~30% to ~61.5%, while on average maintaining the further learning from a ~88.3% to a ~89% recognition rate. Within individual results it can be noted that for Mushrooms, DNA and Sat Images, the further recognition rate (phase 4) is slightly decreased. Using EML+, the immunity for forgetting is improved to ~60%, while maintaining the recognition rate in the last phase at ~90.6%. Although the variation between EML and EML+ results is on average only slight, we can see that EML performs better at the forgetting phase, while EML+ maintains the general learning at a higher level. We can also note, that EML outperforms EML+ for Digits, HS II and DNA datasets, while EML+ outperforms EML for HS I, Mushrooms, Segment and USPS datasets. For Sat Images EML+ outperforms EML for phase 4 learning, but contrary for phase 3. This result is not expected from the theoretical point of view but supports the no free lunch theorem H. *et al.* (1997).

Table 4.2 Comparison of forgetting for ARTIST. The recognition rate of the forgotten class without learning/recognition rate of the rest of the dataset (along with the forgotten class) with learning/recognition rate on the last 100 samples with learning. The results are shown in % of recognition rate.

	RLS	EML	EML+
Digits	19.9 / 80.4 / 82.0	60.03 / 88.4 / 89.6	65.0 / 85.1 / 85.0
HS I	37.7 / 97.9 / 97.7	92.9 / 98.7 / 99.3	98.7 / 99.2 / 99.6
HS II	51.6 / 95.8 / 96.7	79.6 / 96.0 / 96.8	85.3 / 95.7 / 96.3
Mush	64.5 / 92.9 / 92.7	73.1 / 91.0 / 90.5	69.0 / 94.5 / 97.8
Segment	37.1 / 73.1 / 72.8	53.4 / 75.3 / 75.8	73.3 / 85.0 / 85.0
DNA	0.0 / 90.8 / 95.1	8.25 / 90.0 / 96.5	2.3 / 89.9 / 96.0
SI	2.5 / 83.7 / 90.4	3.3 / 80.6 / 94.0	3.3 / 82.5 / 92.0
USPS	25.2 / 92.1 / 92.7	68.7 / 92.2 / 93.3	66.5 / 93.4 / 93.5

For the second model ETS (Table 4.3), on average RLS achieves ~50% recognition rate, while raising the recognition rate to ~90% at phase 4. Using EML, the recognition rate at the phase focusing on the forgetting factor increases to ~81.4% and in the final phase decreases to ~83% on average. Although the results from the final phase show significant change, we can further see that for the last 100 samples, the final learning improves to 85% recognition rate. This result is still not satisfying, and we can note that the model achieves such results especially for

Mushrooms, DNA, Sat Images and USPS datasets. The EML+ for the ETS model achieves ~84.5% recognition rate in phase 3 and ~90% recognition rate in phase 4, on average. We can note that, again, EML outperforms EML+ at the immunity for forgetting by a slight margin; however, EML+ performs better in the last phase of the learning and maintains the recognition rate of RLS. We can also note that EML tends to perform better for the Digits and Segment datasets, while EML+ is superior for the HS I, HS II, Mushrooms, DNA, Sat Image and USPS datasets.

Table 4.3 Comparison of forgetting for ETS. The recognition rate of the forgotten class without learning/recognition rate of the rest of the dataset (along with the forgotten class) with learning/recognition rate on the last 100 samples with learning. The results are shown in % of recognition rate.

	RLS	EML	EML+
Digits	40.5 / 83.9 / 84.7	81.2 / 86.2 / 84.2	71.7 / 82.1 / 79.0
HS I	24.1 / 97.4 / 98.4	90.9 / 93.2 / 94.6	88.1 / 98.6 / 98.7
HS II	73.2 / 95.1 / 95.5	91.6 / 93.0 / 92.7	89.2 / 94.9 / 95.3
Mush	65.8 / 94.7 / 94.7	70.4 / 84.8 / 94.5	85.3 / 94.5 / 97.5
Segment	40.9 / 85.2 / 85.6	70.4 / 85.1 / 85.4	82.7 / 84.7 / 83.9
DNA	69.5 / 90.7 / 94.2	77.1 / 79.6 / 80.0	83.4 / 91.0 / 93.8
SI	31.7 / 81.1 / 87.3	82.4 / 63.2 / 66.8	89.8 / 82.4 / 88.7
USPS	53.8 / 91.4 / 91.1	87.3 / 77.5 / 79.0	87.1 / 92.6 / 93.2

In Table 4.4 we apply our methods to the ETS+ model. Here, the baseline RLS method achieves ~53% recognition rate in phase 3 and ~90% recognition in phase 4, on average. With EML we were able to improve the results from phase 3; the forgetting factor rose to ~75% recognition rate, while the recognition rate achieved in phase 4 was ~83.5% on average. For the last 100 samples with learning, the model was able to improve to ~85% recognition rate on average. We can see that the drop in phase 4 is generally significant for the HS I, DNA, Sat Images and USPS datasets, although the compensation in phase 3 is very significant as well. Using EML+ we achieved ~81% recognition rate in phase 3 and maintained the original recognition rate at ~90% on average by using RLS. This shows that EML+ is more suitable for the ETS model on the whole, for in both phases it tends to outperform EML.

Table 4.4 Comparison of forgetting for ETS+. The recognition rate of the forgotten class without learning/recognition rate of the rest of the dataset (along with the forgotten class) with learning/- recognition rate on the last 100 samples with learning. The results are shown in % of recognition rate.

	RLS	EML	EML+
Digits	20.5 / 84.9 / 84.7	79.8 / 85.4 / 87.3	87.2 / 86.1 / 90.0
HS I	72.4 / 96.8 / 98.3	78.6 / 84.0 / 84.2	87.5 / 98.2 / 99.0
HS II	78.0 / 95.2 / 96.7	83.6 / 91.6 / 93.1	80.8 / 95.5 / 96.0
Mush	64.8 / 94.2 / 93.9	80.0 / 93.1 / 93.9	82.7 / 94.2 / 95.3
Segment	48.0 / 84.4 / 83.4	53.8 / 80.0 / 82.3	57.5 / 79.7 / 84.5
DNA	75.1 / 91.2 / 93.3	75.7 / 84.6 / 86.6	83.3 / 88.9 / 90.0
SI	16.8 / 80.1 / 87.0	70.8 / 68.9 / 73.5	87.2 / 82.2 / 86.5
USPS	45.9 / 91.2 / 92.4	79.2 / 80.6 / 79.4	80.2 / 91.9 / 90.5

In our last set of results in Table 4.5, we can see that RLS achieves ~24% recognition rate in phase 3 and ~91% recognition rate in phase 4, on average. Using EML we were able to improve the results to ~53% recognition rate in phase 3 and ~94% recognition rate in phase 4, on average. For the majority of the datasets both results are superior when using our EML compared to the original RLS. Our second method, EML+, achieves ~51% recognition rate in phase 3 and ~93% recognition rate in phase 4, on average. It can be observed that EML tends to slightly outperform EML+, especially for the Digits, Mushrooms and Segment datasets. For DNA and Sat Images, the results are comparable; and for HS I, HS II and USPS, EML+ slightly outperforms EML.

In most of our results, it can be noticed that when using the original RLS, the accuracy of the forgotten class does not exceed 50%; and in some cases it is close to being completely forgotten. With both our techniques, forgetting is significantly suppressed while the recognition afterwards is comparable to or even better than for the original models. However, as mentioned earlier, in some cases even when considering the improvement, the recognition rate of the forgotten class is still not fully satisfying. This might be caused by the very low number of samples that the forgotten class was learned on in previous phases. This needs to be investigated in future research to improve the proposed framework further. Moreover, in a few cases even

Table 4.5 Comparison of forgetting for IFM. The recognition rate of the forgotten class without learning/recognition rate of the rest of the dataset (along with the forgotten class) with learning/recognition rate on the last 100 samples with learning. The results are shown in % of recognition rate.

	RLS	EML	EML+
Digits	17.6 / 88.7 / 86.8	60.3 / 92.2 / 92.4	58.7 / 91.1 / 91.6
HS I	0.3 / 97.7 / 98.5	58.9 / 99.0 / 99.0	69.4 / 99.0 / 99.2
HS II	38.0 / 95.4 / 95.7	74.8 / 95.7 / 96.9	75.1 / 96.1 / 96.8
Mush	74.9 / 95.9 / 96.9	77.0 / 96.5 / 97.8	76.9 / 95.9 / 96.8
Segment	15.5 / 84.7 / 87.0	58.2 / 94.0 / 94.6	21.6 / 88.4 / 93.2
DNA	37.3 / 91.1 / 94.6	51.9 / 92.2 / 93.1	59.8 / 91.8 / 93.4
SI	0.3 / 85.8 / 90.5	9.5 / 85.9 / 90.9	5.0 / 86.6 / 92.2
USPS	11.0 / 91.1 / 92.5	31.2 / 93.7 / 94.7	44.3 / 93.6 / 95.6

though the forgetting is suppressed significantly, the results are still low, leaving more room for future work and research.

4.8 Conclusion and discussion

In this paper we have presented two novel methods (EML and EML+) to suppress the forgetting of unused classes in an online incremental learning setup. Since the classes of data in a real-time online stream are not necessarily uniformly distributed, they can create a block of absence. During this absence, an unused class is not learned and the model is prone to forget it. Our methods use competitive updates in traditional RLS that focus on the importance of the data points in the stream during the learning process. This shrinks the blocks of absence of a class while keeping the important information available for the learning process.

We compared our techniques using four baseline models and achieved better results than the baseline techniques. We discuss several options for incremental learning, some of which have been used for online or incremental handwriting recognition. As we wanted to show the usability of the proposed models not only for handwriting recognition, but also for various other problems, we chose to test a spectrum of datasets. We evaluated results on 8 datasets that varied in the number of classes, features and sample sizes. In all of the applications our techniques achieved a good trade-off between the immunity for forgetting of unused classes and the general accuracy of the model.

From the results we can see that both of our methods are superior to the original RLS in regard to the immunity to forgetting of the unused classes, and they both are capable of maintaining the general accuracy of the models. The general accuracy is very important for the model to be able to achieve good results even when there are no blocks of absence of classes, and our methods do not influence the learning process of all the other classes. In some rare cases, however, the general accuracy was lower than in the original models. We believe that this is due to the nature of the data and the importance of each sample for the learning process. The learning process of all classes is slightly slowed down compared to a case in which all classes are learned at the same time. Especially in EML, the learning is more competitive than in EML+. This follows our assumption, since by using more samples in EML+ we were able to improve the general accuracy of the model. Further we point out that if we focus on the last samples' accuracy, the results improve as well. This corresponds to our theory that worse results are caused by lack of data due to the competitive nature of the model. On the other hand, the more competitive the learning is, the more it is, in general, immune to forgetting. Again, there are some rare cases when this is not completely true; but since in both EML and EML+ the learning is competitive, we can see that using such learning is superior to learning all samples, as in RLS. We can note that in all cases our methods surpassed RLS and in the vast majority of the results the improvement was very significant.

Given these findings we can state that our approaches outperform baseline techniques significantly in cases where some classes are used only on rare occasions compared to the rest of the classes. This gain in performance is reduced when distribution over classes is more or less uniform. However, this reduction is not critical and both methods lead to comparable or even superior results when there are no blocks of absence of classes present. Nevertheless, the main application of our models is on handwritten gestures recognition. Consequently, we might suppose that the distribution of handwritten gestures of users in a real-world setting is not uniform and that several gestures are used only on rare occasions.

In our future work we want to explore the problem of forgetting in cases where even though our methods did indeed improve the original results, the accuracy during the forgetting phase was still not acceptable. It might seem that for these datasets either the models or the competitive learning process itself is not quite suitable. As a part of this research we want to investigate the application of our EML and EML+ to optimization techniques other than RLS.

Acknowledgment

The authors thank NSERC RGPIN 138344 & 2014-04649 and SSHRC 412-2010-1007 of Canada for their financial support.

CHAPTER 5

SO-ARTIST: SELF-ORGANIZED ART-2A INSPIRED CLUSTERING FOR ONLINE TAKAGI-SUGENO FUZZY MODELS

Marta Režnáková[1], LukasTencer[1], Mohamed Cheriet[1]

[1] Synchromedia Lab, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

Paper published in *Applied Soft Computing*, June 2016.

5.1 Abstract

In this paper we introduce a novel online self-organized clustering method based on the ART-2A network for Takagi-Sugeno fuzzy models. To accomplish the self-organization, we introduce an automatic decision algorithm along with solutions for merging and splitting of rules as well as the parameters they operate with, such as our novel incremental distance measurement and competitive recursive least squares. We emphasize the leaning algorithm's having an impact for initial as well as long-term learning capabilities. We also emphasize the challenge for online learning, where examples are incoming in real-time and thus are unknown before they can be learned. Therefore, we solve parameter fixing by introducing a parameter free method. We show the performance of our method on various machine learning benchmarks as a highly accurate and low time-consuming method capable of adapting to different databases without the need for fixing any of its parameters according to the database.

5.2 Introduction

Across many fields, such as machine learning, pattern recognition, and document analysis, the need for online learning has become important though challenging. Often, new data are available and to keep up to date the classical pre-learning methods (i.e. offline learning) require long-term processes to perform recognition. On the one hand, incremental learning might seem to be an option, where each new example is learned as an increment to a previously learned model. However, we focus on online learning, where the model is learned incrementally or adapted to actual situations, but at the same time achieves real-time performance. This means that the whole learning process is linked to recognition, so the steps are performed at the same time, in contrast to offline learning. In addition, we emphasize the no-database-knowledge problem, where the model starts from scratch, not knowing the number of classes or their appearance. Thus, this kind of learning needs to provide a strong recognition rate at the very beginning of the recognition process as well as overall.

Our main motivation for this research is applications such as handwritten gesture (symbol) recognition for interactive devices, where the user sets his own gestures and the system adapts to his personal needs. In these cases, we do not have any information about the database or its content. That information is collected over time as the user's writing style evolves. The recognition begins immediately after a few samples per class are introduced (generally 1-5), and all classes can be introduced one after another during the process, forgotten in time or recalled after not being used for a long time. These are some of the challenges of online learning, where the distribution of classes is not given and the model has to be elastic enough to cope with them.

In this paper we introduce a novel variant for online learning when knowledge of the database is not required. The contributions in this paper are

- Self-organized clustering based on the ART-2A neural network (Carpenter *et al.* (1991b)), where its parameters are learned automatically and online;
- Similarity, dissimilarity and error measurement for fuzzy rules;
- Merging and splitting of parameters for Incremental distance (antecedent part) and parameters for CCL/RLS (consequent part).

This work leverages on our previous work, ARTIST (Režnáková *et al.* (2013)), in which we introduced the integration of the ART-2A network into the generation of fuzzy rules for a TS fuzzy model. In this paper, we propose a novel method in which the rules are organized by themselves and thus not solely by the ART-2A network. Along with automatic generation, we introduce the merging, splitting and discarding of the rules. In addition, we introduce the automatic learning of a vigilance parameter and a learning rate that for the ART network originally need to be set by user. Thus, this paper presents a novel self-organized method in which the learning starts from scratch with no knowledge of a dataset size or the number of classes, and time classes are introduced. Classes can be added on the fly, with rule organization on the fly, and in an automatic way. All these major features make this work's base-line drastically different from and complementary to the previous work, ARTIST.

We evaluate our proposed method on standard datasets in a comparison with our previous model (ARTIST), in which the rules are generated automatically without any self-organization by using fixed parameters. This comparison is performed in contrast to finding the best parameters by using a grid method, showing that our method is capable of finding optimal parameters. At the same time, by this comparison we show that our method is capable of finding the correct organization (fuzzy rules distribution) as a supervised solution.

This paper is divided into 6 sections. In section 5.3 we show the state-of-the-art methods for handling the TS fuzzy rules, self-organized clustering methods, and incremental real-time models. Within section 5.4 we briefly summarize our previous model, ARTIST, which is based on ART-2A based generation of rules for the Takagi-Sugeno (TS) fuzzy model, along with incremental distance measurement and competitive consequent learning (CCL). In section 5.5 we explain our novel self-organized (SO) ARTIST in detail, focusing on our contributions. We conclude this section with an algorithm summarizing the whole novel concept of online learning when starting from scratch with no parameters to set according to the database used. Finally, in section 5.6 we show the results of our novel method on several machine learning datasets containing handwritten symbols, which are the main motivation of our research. We end this paper with conclusion and possible future work.

5.3 Related works

In our work, we search for a sufficient online learning-based model in which the number of classes is unknown. Thus, we search for models using online learning, when starting from scratch and with no pre-definitions. Moreover, we assume no knowledge of the database, because of the online nature of our problem. If a database is known, we can simply use offline learning. Hence, such a task is very challenging, and in this section we briefly introduce several attempts to solve this learning problem.

Generally, there are more methods for online or incremental learning (Gama (2010); Moamar-Mouchaweh, Sayed; Lughofer (2012); Angelov, Plamen; Filev, Dimitar P.; Kasabov (2010)). Often, authors turn an offline model into an online one using just a few adjustments. The most common approaches are among kernel-based methods, subspace methods, tree structures or bagging/boosting-based approaches. Diehl *et al.* (2003); Carozza *et al.* (2000) propose to take the number of Support Vectors as not stable, making their generation incremental. Bordes (2005); Tax (2003); Wang *et al.* (2007, 2008) update the SVs in an online manner, with incrementally incoming samples in real time. There are several approaches for incremental Principal Component Analysis (PCA), where the subspace is learned in an incremental manner; they include Kasabov *et al.* (2005) used for feature extraction, Li (2004) for face recognition, Ross *et al.* (2007) for visual tracking, Yao *et al.* (2010) for hand gesture recognition, and incremental Principal Component Regression (PCR) in Hall *et al.* (1998), compared to incremental fuzzy models in Cernuda *et al.* (2012). Grabner *et al.* (2006); Oza (2005) propose an online version of the AdaBoost algorithm. However, these methods are not naturally adjusted for multiclass problems. Thus, Ditzler *et al.* (2010); Saffari *et al.* (2010) propose a multiclass solution for AdaBoost-based methods. However, the number of classes is still required. Leistner *et al.* (2009) transfer a Random Forest method introduced by Breiman (2001) into an online model; the divisions of the space are created in an incremental manner on the basis of the conditions of branches of the tree being updated online. The incremental decision trees were originally proposed in Utgoff (1989). Reddy *et al.* (2009) adapt an action-reaction tree into an online model to cope with samples incoming in an online manner. All these methods nonethless require some knowledge about the database, in the sense of knowing the number of classes, number of SVs, size of the database, or whether they are naturally designed for multiclass problems. Hence, such methods are not capable of fully adaptive and automatic learning from scratch.

In our research, we focus on fuzzy models and their capabilities for such online learning. There are two main streams for fuzzy models: Mamdani introduced by Mamdani *et al.* (1975) and Takagi-Sugeno introduced by Takagi *et al.* (1985). Their differences are based on the consequences they output. Babuška *et al.* (2003); Tan *et al.* (2000) explored the use of fuzzy models for nonlinear problems. In fuzzy models, the clustering solution states how the rules are treated. Thus, the methods can also vary according to this parameter. The work on this has been done by Babuska (1998). There are several works using different approaches, such as Ahn *et al.* (2010) for nonlinear dynamic fuzzy models and Cordón, Oscar; Herrera, Francisco; Hoffmann, F.; Magdalena (2001); Cordon *et al.* (2004); Yuan *et al.* (1996) using genetic algorithms for generation of the rules. However, these attempts lack the ability for online learning, where not only the antecedent part (clustering), but also the consequent part (reasoning) need to be adjusted accordingly.

For an online or incremental adaptation of fuzzy models, several works have made proposals. A comprehensive survey on this topic was done by Lughofer (2011). Angelov *et al.* (2004); Angelov (2008) introduced an incremental mountain clustering algorithm for generation of rules for TS fuzzy models. Here, the densities of already existing rules are compared to new possible rules for deciding the creation of a new rule. Almaksour *et al.* (2010) continued this work using multivariate distributions for the antecedent part, achieving higher accuracy of recognition. Carse *et al.* (1996); Gomez *et al.* (2002); Kasabov (2001) use a genetic algorithm for the evolution of fuzzy rules. Lughofer (2008a) proposed using incremental vector quantization for the organization of rules. This work was later expanded in Lughofer *et al.* (2011), where the authors propose new elimination techniques for rule redundancies, including merging of rules. However, they still expect a free parameter (threshold) to be used and adjusted. In Lemos *et al.* (2013) the authors propose an evolving fuzzy system with dynamic organization of the rules based on threshold values. In Rong *et al.* (2010) the authors propose an evolving fuzzy system by the addition and removal of fuzzy rules, but using threshold values to determine these actions. Režnáková *et al.* (2012) proposed the use of incremental density measurement for the antecedent part to achieve a recognition rate comparable to a multivariate distribution, but to keep low computational cost. These methods use Recursive Least Square (RLS, Ljung (1999); Angelov *et al.* (2008)) optimization to adjust the consequent parameters. In our previous work Režnáková *et al.* (2013) we adjusted these calculations by using Competitive Consequent Learning (CCL), avoiding the forgetting factor in incremental learning approaches.

In this research we assume the clustering part as very crucial for fuzzy models, especially when the learning is online and the rules are generated on the fly. Thus, we explore clustering approaches that may be incorporated into fuzzy models. Carpenter *et al.* (2010); Gail A. Carpenter, Stephen Grossberga (1991) introduced the Adaptive Resonance Theory (ART), where the number of classes is unknown and starting from scratch is possible. However, this strategy is applicable only for binary inputs. Thus, in Carpenter *et al.* (1991a,b) the authors proposed an adaptation of the ART network to work for analog signals. The comparison of the ART and ART-2A networks was done by Frank *et al.* (1998). There are several approaches Anagnostopoulos *et al.* (2000); Glotin *et al.* (2009); Hsu *et al.* (2008) that adjust the ART network for different situations. Other approaches towards online or incremental clustering were proposed by Forster *et al.* (2010); Yu *et al.* (2009), introducing an incremental k Nearest Neighbors (kNN). However, a certain number of neighbors need to be kept in order to derive the distance among them. Thus, only a base is adjusted in considering incrementally incoming samples. Zhang *et al.* (2004) proposed an online clustering using Singular Value Decomposition (SVD). Lughofer (2008b) proposed incremental Vector Quantization (VQ), Beringer *et al.* (2006) proposed an incremental k-means algorithm and Lühr *et al.* (2009) introduced an incremental graph-based clustering. Another interesting group is the Self-Organized Maps (SOM) proposed by Caridakis *et al.* (2010); Kohonen (1990); Lin *et al.* (1991); Vesanto *et al.* (2000), where the organization of clusters is automated, but with a fixed number of clusters. Then in Deng *et al.* (2003) the authors proposed evolving SOM where the number of classes is not set in advance, with clusters evolving in time. In many of these approaches, some of the conditions that need to satisfied are not accomplished. Either the number of classes needs to be known, some initialization is required, or storage of part of the dataset or parameter fixing is necessary. Many incremental online learning methods can be easily applied using MOA (Massive Online Analysis Bifet *et al.* (2010)).

As can be noticed, there are no methods capable of learning from scratch or adjusting automatically without requirements for the knowledge of the database in any sense. Hence, in this paper we introduce a novel method satisfying these requirements. In the next section we briefly recall the ARTIST concept when using the ART-2A clustering method for generation of rules. However, the vigilance and learning rate parameters still need to be set. Thus, we propose a Self-Organized ARTIST that will solve this constraint and show the possibilities of a clustering method for TS fuzzy models.

5.4 ARTIST: ART-2A driven generation of rules for TS fuzzy models

In this work, fuzzy rules are generated using the ART-2A incremental clustering method, in which the knowledge of the number of classes is not required. The rules are those proposed by Takagi *et al.* (1985). Here in (5.1) (Angelov *et al.* (2008)), the antecedent part is denoted as IS and the consequent part by $THEN$. The antecedent part describes similarity and is partially responsible for the rule organization and generation. The consequent part is responsible for the reasoning and inference of the model as well as for a decision about the erroneous of the model.

$$R_i: \ if\ \boldsymbol{x}\ IS\ A\ THEN\ y = \boldsymbol{x\pi} + b \tag{5.1}$$

The result $y = \boldsymbol{x\pi} + b$ in our solution is expressed as $y_i = \beta_i \boldsymbol{x}^T \Pi_i$, where β_i is a firing degree (membership function) of a rule R_i derived from the antecedent part, y_i is the inference of R_i for all classes; and $\Pi_i = \left\{ \boldsymbol{\pi}_i^j \right\}_{j=1\dots c}$ is a set of the consequent parameters of R_i for each class

j. Then, the final result of the system is derived as a weighted average of inferences y_i from each rule (5.2), where r is the number of all rules.

$$Y = \frac{1}{\sum_{i=1}^{r} \beta_i} \sum_{i=1}^{r} \beta_i \boldsymbol{x}^T \Pi_i \cong \sum_{i=1}^{r} \beta_i \boldsymbol{x}^T \Pi_i \tag{5.2}$$

The result of the system for one concrete class is represented as (5.3).

$$Y^j = \sum_{i=1}^{r} \beta_i \boldsymbol{x}^T \boldsymbol{\pi}_i^j \tag{5.3}$$

$$j' = arg \max_j Y^j \tag{5.4}$$

To assign the winning class j', the maximum of all results Y^j is derived (5.4). If the winning class j' does not equal the ground truth j^*, an error in decision will occur.

5.4.1 Rule organization

For the organization of rules, ARTIST uses the ART-2A network because of its capability of incremental learning, no class or its number initialization and its readability. In the model, the ART network is responsible only for the generation of rules, not for the firing degree or for the reasoning. The rule generation is not influenced by the information about arrival of a new class to the system, but is subsumed completely under the ART logic. Such an approach is to prevent the creation of unimportant rules, since in fuzzy models all classes are partially incorporated into all rules. Thus, if some rule is close enough to a new class, the latter can represent it, and the system will not create confusion by forcibly making a new rule with the same firing degree. This does not mean that the system will not create new rules for newly appearing classes. It will do this for classes that do not overlap with others. We consider fuzzy logic and see inferences of rules as possibilities. By not creating a separate overlapping rule for a new class and keeping this class within the other rule, we stipulate that in that part of the feature space close to the class and rule, both of these classes are possible. If we create a new rule for a new class despite their sharing of the same or overlapping parts of the feature space, the same effect occurs: in this part of the feature space both of the classes are possible. The only difference is that in the second case, we will get this information from two rules, and one of them can be seen as redundant.

This means that the rule generation is completely unsupervised and does not take the information about the new class appearance into account. This is an issue especially for learning on the fly approaches, where the classes are not introduced at the beginning of the learning process but emerge randomly during the learning process. Further in our experiments, this is the strategy we use for evaluation of our model and the appearance of the classes is random.

In the ART network, the system searches for a resonance or reset signal. Thus, the concept consists of three layers: comparison layer, recognition layer and reset module. In the comparison layer, the incoming sample I is filtered using function $f(\cdot)$ (Carpenter *et al.* (1991b)); it is normalized and compared to all known clusters using cosine distance; and a match is searched for. Then, a choice is made by selecting the best match over all the committed nodes (clusters) and compared to the vigilance parameter in the reset module (5.6). If the choice satisfies ρ, resonance occurs and the search is over. If choice does not satisfy ρ, reset is done, the chosen node is set as uncommitted, a new search for resonance continues, and a new choice is selected (Carpenter *et al.* (1991b)). If no resonance is possible, a new node (cluster) is added to the system. After resonance or addition of a new node, competitive learning of the weight vector ω takes place (5.7). For this to occur, a learning rate λ (free parameter) is used, which denotes how much of the weight vector should be learned. There is only one global learning rate in the system.

$$\boldsymbol{x} = \frac{f(I)}{\|f(I)\|} \tag{5.5}$$

$$\boldsymbol{x}^T \omega_i > \rho \tag{5.6}$$

$$\omega = \frac{\lambda \boldsymbol{x} + (1-\lambda)\,\omega}{\|\lambda \boldsymbol{x} + (1-\lambda)\,\omega\|} \tag{5.7}$$

The complete architecture of the system can be seen in Figure 5.1. Here, the firing degree β_i of a rule i is represented by an antecedent part explained in next subsection. This is followed by the learning of the consequent parameters. There, the antecedent parts' membership (firing degree) influences the weight of the decision of the consequent part. More details on the ART-2A network can be seen in Carpenter *et al.* (1991b).

5.4.2 Antecedent part

The antecedent part describes the properties of similar samples that distinguish them from other groups of samples. In fuzzy models, these properties are linked to each rule by the term *sample x IS property A*, where '*IS*' is the link and *A* is the property describing one rule. In the example, we can see this link as *car IS red* for one rule and *car IS blue* for another rule.

Usually, the antecedent part is described as a Gaussian distribution, either univariate or multivariate. As properties we can see a feature vector, where each rule will then be assigned a group of samples on the basis of the similarities among them. Applying univariate distribution, one can achieve low computational cost, using only variance scalar for update (in the case of incremental learning) and calculation, rather than covariance matrix with high dimension, in the case of many properties (features). However, multivariate distribution achieves higher accuracy in recognition because it is better in predicting the similarities among samples.

In ARTIST, we use a density calculation (5.8) with a satisfying trade-off between computational cost and accuracy. Here, β_i describes, how much *sample* $\boldsymbol{x}$ *IS properties i*, based on all the other samples $\boldsymbol{x}_j$ from rule i. Furthermore, in order to fulfill the fuzziness of the model, membership functions are not normalized, so they stand as possibilities rather than probabilities Režnáková *et al.* (2013).

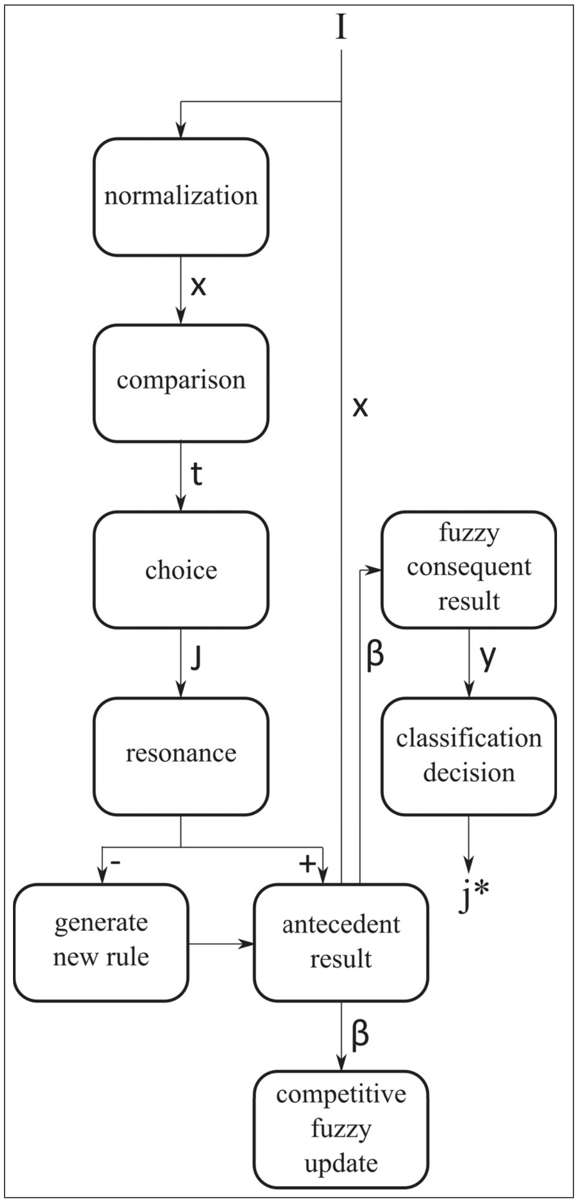

Figure 5.1 ARTIST

$$\beta_i = \frac{1}{1+\sum_{j=1}^{t_i}\left(\boldsymbol{x}-\boldsymbol{x}_j\right)^2} \tag{5.8}$$

In order to accommodate incremental learning into the model, we need to introduce a recursive model for learning the membership function, as described in the following subsection.

5.4.2.1 Incremental similarity measurement

For incremental learning, the recursive formula can be easily derived from the original one, resulting in (5.9). By substituting $\boldsymbol{x}^2$, $\sum_{j=1}^{t_i}\boldsymbol{x}_j$ and $\sum_{j=1}^{t_i}\boldsymbol{x}_j^2$ using parameters α_i, η_i and γ_i we can produce recursive formula (5.10) with update formulas for these parameters (5.11) in an incremental manner. Here, none of the samples will be forgotten or changed; but in these parameters, the true values of past samples are stored. Thus, every time a new sample arrives to enrich the rule, updating formulas are triggered so that for the next sample with an unknown competency for a rule, membership β_i is derived.

$$\beta_i = \frac{t_i}{t_i + t_i\boldsymbol{x}^2 - 2\boldsymbol{x}\sum_{j=1}^{t_i}\boldsymbol{x}_j + \sum_{j=1}^{t_i}\boldsymbol{x}_j^2} \tag{5.9}$$

$$\beta_i = \frac{t_i}{t_i + t_i\alpha_i - 2\boldsymbol{x}\boldsymbol{\eta}_i + \gamma_i} \tag{5.10}$$

$$\begin{array}{ll} a)\, t_i = t_{i-1} + 1 & c)\, \boldsymbol{\eta}_i = \boldsymbol{\eta}_{i-1} + \boldsymbol{x};\ \boldsymbol{\eta}_0 = 0 \\ b)\, \alpha_i = \boldsymbol{x}^2 & d)\, \gamma_i = \gamma_{i-1} + \alpha_i;\ \gamma_0 = 0 \end{array} \tag{5.11}$$

5.4.3 Consequent part

To enable rules that make an inference, we use a set of parameters that can also be seen as weights for all characteristics from our input vector I augmented by a bias. To learn these parameters, we follow the use of RLS (Recursive Least Squares), where the error we want to optimize is a squared difference of an actual result $\beta_i\boldsymbol{x}_i\Pi$ and an expected result Y_i (5.13). This kind of optimization leads to resulting formulas for updating a covariance matrix C_i (5.17) and a set of parameters Π_i (5.22) for rule i. In our work we use local optimization of the consequent parameters of fuzzy rules and thus local RLS (Angelov *et al.* (2008)).

$$\boldsymbol{x}_i = \begin{pmatrix} 1 \\ I_i \end{pmatrix} \tag{5.12}$$

$$E = \sum_{i=1}^{t} ||\beta_i \boldsymbol{x}_i \Pi - Y_i||^2 \tag{5.13}$$

$$\Pi_t = \sum_{i=1}^{t} \beta_i \boldsymbol{x}_i Y_i \left(\sum_{i=1}^{t} \beta_i \boldsymbol{x}_i \beta_i \boldsymbol{x}_i^T \right)^{-1} = M_t C_t^{-1} \tag{5.14}$$

$$M_t = \sum_{i=1}^{t} \beta_i \boldsymbol{x}_i Y_i = \sum_{i=1}^{t-1} \beta_i \boldsymbol{x}_i Y_i + \beta_t \boldsymbol{x}_t Y_t = M_{t-1} + \beta_t \boldsymbol{x}_t Y_t \tag{5.15}$$

$$C_t = \sum_{i=1}^{t} \beta_i \boldsymbol{x}_i \beta_i \boldsymbol{x}_i^T = \sum_{i=1}^{t-1} \beta_i \boldsymbol{x}_i \beta_i \boldsymbol{x}_i^T + \beta_t \boldsymbol{x}_t \beta_t \boldsymbol{x}_t^T = C_{t-1} + \beta_t \boldsymbol{x}_t \beta_t \boldsymbol{x}_t^T \tag{5.16}$$

$$C_t^{-1} = C_{t-1}^{-1} - \frac{C_{t-1}^{-1} \beta_t \boldsymbol{x}_t \beta_t \boldsymbol{x}_t^T C_{t-1}^{-1}}{1 + \beta_t \boldsymbol{x}_t^T C_{t-1}^{-1} \beta_t \boldsymbol{x}_t} \tag{5.17}$$

$$\Pi_t = M_t C_t^{-1} = (M_{t-1} + \beta_t \boldsymbol{x}_t Y_t) C_t^{-1} = (\Pi_{t-1} C_{t-1} + \beta_t \boldsymbol{x}_t Y_t) C_t^{-1} = \tag{5.18}$$

$$= \left(\Pi_{t-1} \left(C_t - \beta_t \boldsymbol{x}_t \beta_t \boldsymbol{x}_t^T\right) + \beta_t \boldsymbol{x}_t Y_t\right) C_t^{-1} = \Pi_{t-1} \left(C_t^{-1} \left(C_t - \beta_t \boldsymbol{x}_t \beta_t \boldsymbol{x}_t^T\right)\right) + \tag{5.19}$$

$$+ C_t^{-1} \beta_t \boldsymbol{x}_t Y_t = \Pi_{t-1} \left(1 - C_t^{-1} \beta_t \boldsymbol{x}_t \beta_t \boldsymbol{x}_t^T\right) + C_t^{-1} \beta_t \boldsymbol{x}_t Y_t = \tag{5.20}$$

$$= \Pi_{t-1} - C_t^{-1} \beta_t \boldsymbol{x}_t \left(\Pi_{t-1} \beta_t \boldsymbol{x}_t - Y_t\right) \tag{5.21}$$

$$\Pi_t = \Pi_{t-1} + C_{t-1}^{-1} \beta_t \boldsymbol{x}_t \left(Y_t - \Pi_{t-1} \beta_t \boldsymbol{x}_t\right) \tag{5.22}$$

5.4.3.1 Competitive Recursive Least Squares

For consequent learning we use RLS, where on the basis of result, each rule is updated by the weight of its membership function to the sample, to either 0 or 1 for each class. This leads to a suspicion that if one class is unused for a longer time, by updating its parameters within each rule to 0 for each sample, the ability of each rule to recognize such an unused class is reduced. To prevent this, we proposed a solution in which the update to 0 is performed only if necessary, i.e. when results y_j for such class j are very misleading, considering that j^* is the true class (5.23).

$$\boldsymbol{\pi}_{i,t}^{j} = \boldsymbol{\pi}_{i,t-1}^{j} + C_{i,t-1}^{-1}\beta_{i,t}\boldsymbol{x}_t\left(\varepsilon_i - \boldsymbol{\pi}_{i,t-1}^{j}\beta_{i,t}\boldsymbol{x}_t\right)$$

$$\varepsilon_i = \begin{cases} 1 & j = j^* \\ 0 & y_{i,j} > y_{i,j^*} \\ \boldsymbol{\pi}_{i,t-1}^{j}\beta_{i,t}\boldsymbol{x}_t & otherwise \end{cases} \tag{5.23}$$

As we can see in Režnáková *et al.* (2013), such competitiveness among updating consequent parameters leads to better results even if some class is unused for a longer time, preventing the model from forgetting it. This ability is very important especially when the distribution of classes of incoming samples is not necessarily uniform.

5.5 SO-ARTIST: Self-Organized ART-2A based management of rules for TS fuzzy models

In this section we propose a novel method for organizing the fuzzy rules of the Takagi-Sugeno fuzzy model. In our previous method (Režnáková *et al.* (2013)) we proposed the use of ART-2A incremental clustering for this purpose, as described in section 5.4. However, such clustering requires fixation of two parameters regarding the database used. At the same time, we have empirically found that especially for incremental learning, some clusters are not generated correctly, which means that they increase the error rate of the model. This is caused by an insufficient number of examples, especially at the beginning of the learning process. Thus, we propose a self-organized mechanism. To bring the division of a space that fits the requirements for each situation at each time it is required to make an assumption about the state of the world.

In this section, we introduce novel solutions for rules merging and splitting. Each subsection is divided into solutions for the antecedent, consequent and rule properties parts, as for fuzzy models. Before each such subsection, we introduce the novel ideas behind decisions about merging and splitting of the rules, such as decisions about similarities and, conversely, dissimilarities between rules. Furthermore, we propose an automatic derivation of the vigilance parameter and learning rate for each cluster from ART-based incremental clustering methods. We end this section with a description of the complete mechanism behind the novel automatic self-organization of rules, using automatically adapted parameters.

In Lughofer *et al.* (2011) the authors also propose a similarity measure in order to determine the merging of rules. However, their strategy is based solely on cluster shapes and positions and thus focuses on the antecedent part. They also expect the distributions of data to be Gaussian, which is not so in our case. At the same time they compare their similarity measures with a threshold value that is a free parameter. This work has been extended in Lughofer (2012), where the decision about merging and splitting rules is done in a more parameter-free way, but the authors focus only on the geometric nature of fuzzy rules. Unlike them, in this work we present the combined and consequent nature of fuzzy rules for the merging process. The authors of Lemos *et al.* (2013) also use merging of rules that are defined as Gaussian distributions. However, they also need a threshold value to compare with the similarity of the clusters, whereas our method does not. As we will see in this section, our method focuses not only on the antecedent part, but also on the consequent part, for we believe that redundancy of similar rules does not only rely on their position, but also on their opinion. We find it very important to look at this because if two rules are overlapping or close enough but output different information, merging them will cause loss of the information since the merged rule will possibly output uniform-like unsure results. This might not be the case of samples close to the center of the merged rule, but would be for samples close to the borders. Moreover, considering the incremental manner of the learning, it can happen that these rules will evolve into non-overlapping clusters in the same way as they evolved into overlapping clusters.

5.5.1 Merging the rules

In our approach, merging the rules is driven by the similarities and distance between them. In other words, the clusters are merged when they tend to perform very similar inferences and their distance is within the range of the vigilance parameter of the ART network. This assumption is based on the possible match of more clusters (see section 5.3); thus, more clusters that are close enough to one sample are able to satisfy the vigilance parameter. This may occur, when two clusters are close enough to each other to satisfy the vigilance parameter as well. This assumption will follow through the whole self-organization, so that self-adapting vigilance will not interfere with the division of the space.

Our novel similarity and rule distances are proposed in the next subsections, followed by our proposition for merging membership parameters (antecedent part), CCL parameters (consequent part) and rule parameters. The whole concept and intelligence behind our novel method is explained in detail and summarized in section 5.5.3.

5.5.1.1 Similarity measurement

The similarity between rules is based on the distance between their results, similarly to Režnáková *et al.* (2012). Since ideally these are normalized (0 for incorrect classes and 1 for correct class), we can use cosine distance as in ART-2A clustering. However, since the fuzzy inferences describe possibilities and not probabilities, we normalize these results in order to achieve distance of such range. Such a solution is correct even for comparison of two different vectors with equal direction, since such results have the same meaning. In our assumptions, we express the inferences as opinions about examples and their similarity to classes. Thus, when rule i believes that example $\boldsymbol{x}$ is class c, it is similar, if not the same, as opinion of rule j that believes example $\boldsymbol{x}$ is also class c. Hence, when the distribution of results for each class is the same, only with a different magnitude, we assume these results to be similar.

For similarity itself we use a similarity matrix in which each field represents the distance of results for two specific rules (5.25). Since such a matrix is symmetrical, we only need half of it (excluding the diagonal entries equal to 1). Such distance varies between 0 and 1. The closer the general inferences are, the higher the results they will achieve. In other words, the similarity matrix is composed of values that indicate how similar the rules are to each other by their opinions about the examples from all the classes.

$$inf_i = \frac{\boldsymbol{x}^T \Pi_i}{\|\boldsymbol{x}^T \Pi_i\|} \tag{5.24}$$

$$\begin{pmatrix} 1 & & & & & \\ \left(inf_2 \cdot inf_1^T\right) & 1 & & & & \\ \left(inf_3 \cdot inf_1^T\right) & \left(inf_3 \cdot inf_2^T\right) & 1 & & & \\ \vdots & & & \ddots & & \\ \vdots & & & & 1 & \\ \left(inf_r \cdot inf_1^T\right) & \left(inf_r \cdot inf_2^T\right) & \cdots & & \left(inf_r \cdot inf_{r-1}^T\right) & 1 \end{pmatrix} \tag{5.25}$$

5.5.1.2 Rule distance measurement

The distance between two clusters can be expressed as a cosine distance, using weighted vectors ω for each rule, since these are used to determine the resonance and thus the vigilance parameter. We assume that if two rules are close enough to each other to satisfy the vigilance parameter, then a sample belonging to one of them might also belong to another, and even more if the rules are similar. Conversely, if a sample matches one rule, it should not belong to another rule that does not make a good assumption about this sample.

After representing the cosine distance between rules j and k (5.26), we get the inputs for our matrix of distances between all clusters (5.27).

$$\delta(J,K) = \boldsymbol{\omega}_1^T \cdot \boldsymbol{\omega}_2 \tag{5.26}$$

$$\begin{pmatrix} 1 & & & & & \\ \delta(2,1) & 1 & & & & \\ \delta(3,1) & \delta(3,2) & 1 & & & \\ \delta(4,1) & \delta(4,2) & \delta(4,3) & 1 & & \\ \vdots & & & & \ddots & \\ \delta(r,1) & \delta(r,2) & \delta(r,3) & \cdots & \delta(r,r-1) & 1 \end{pmatrix} \tag{5.27}$$

5.5.1.3 Merging membership parameters

Membership β is represented as a density measurement, where this density is updated in an incremental manner. The density of merged clusters can be derived from the original formula by taking all the samples from both clusters as if they were in one. Then on the basis of (5.28) we can derive the final merging formulas for parameters α, γ and $\boldsymbol{\eta}$ (5.33).

$$\beta_{1\wedge 2} = \frac{1}{1+\frac{1}{t_1+t_2}\left(\sum_{j=1}^{t_1}\left(\boldsymbol{x}-\boldsymbol{x}_j\right)^2+\sum_{k=1}^{t_2}\left(\boldsymbol{x}-\boldsymbol{x}_k\right)^2\right)} \tag{5.28}$$

$$\beta_{1\wedge 2} = \tag{5.29}$$

$$= \frac{(t_1+t_2)}{(t_1+t_2)+(t_1+t_2)\boldsymbol{x}^2-2\boldsymbol{x}\left(\sum_{j=1}^{t_1}\boldsymbol{x}_j+\sum_{k=1}^{t_2}\boldsymbol{x}_k\right)+\left(\sum_{j=1}^{t_1}\boldsymbol{x}_j^2+\sum_{k=1}^{t_2}\boldsymbol{x}_k^2\right)} = \tag{5.30}$$

$$= \frac{(t_1+t_2)}{(t_1+t_2)+(t_1+t_2)(\alpha_1+\alpha_2)-2\boldsymbol{x}(\boldsymbol{\eta}_1+\boldsymbol{\eta}_2)+(\gamma_1+\gamma_2)} = \tag{5.31}$$

$$= \frac{t_{1\wedge 2}}{t_{1\wedge 2}+t_{1\wedge 2}\alpha_{1\wedge 2}-2\boldsymbol{x}\boldsymbol{\eta}_{1\wedge 2}+\gamma_{1\wedge 2}} \tag{5.32}$$

$$\begin{aligned} &a)\, t_{1\wedge 2} = t_1 + t_2 \qquad &c)\, \boldsymbol{\eta}_{1\wedge 2} = \boldsymbol{\eta}_1 + \boldsymbol{\eta}_2 \\ &b)\, \alpha_{1\wedge 2} = \alpha_1 + \alpha_2 \qquad &d)\, \gamma_{1\wedge 2} = \gamma_1 + \gamma_2 \end{aligned} \tag{5.33}$$

Furthermore, we remember these three parameters for each class contained in the assumed rule (and not necessarily, but most probably, for every class). These parameters are used when splitting the antecedent part of the rule and thus need to be merged as well while the rules are being merged. We define these as vectors $t_{C,i}$, $\boldsymbol{\eta}_{C,i}$ and $\gamma_{C,i}$, whose dimensions equal the number of classes absorbed by rule i. Thus, for example $t_{C,i}(j)$ is the number of samples inserted into rule i belonging to the class j. When merging these parameters, for each dimension we perform the same operations as in the formulas (5.33).

5.5.1.4 Merging CCL parameters

The merging of consequent parameters can be seen as solving the merged error of two recognition modules, in our case two different rules. The final error can then be expressed as a sum of these two error rates, as in (5.40). In this section, we propose the formulas for deriving a merged gain matrix C and parameters Π that minimize errors of system E. Then, the error of the first merged rule is E_1 with minimizing parameters Π_{t_1} (5.37).

$$E_1 = \sum_{j=1}^{t_1} ||\beta_j \boldsymbol{x}_j \Pi_{t_1} - Y_j||^2 \tag{5.34}$$

$$\frac{\partial}{\partial \Pi_{t_1}} E_1 = 0 \tag{5.35}$$

$$2\sum_{j=1}^{t_1} \beta_j \boldsymbol{x}_j \left(\beta_j \boldsymbol{x}_j \Pi_{t_1} - Y_j\right) = 2\left(\sum_{j=1}^{t_1} \beta_j \boldsymbol{x}_j \beta_j \boldsymbol{x}_j^T \Pi_{t_1} - \sum_{j=1}^{t_1} \beta_j \boldsymbol{x}_j Y_j\right) = 0 \tag{5.36}$$

$$\Pi_{t_1} = \sum_{j=1}^{t_1} \beta_j \boldsymbol{x}_j Y_j \left(\sum_{j=1}^{t_1} \beta_j \boldsymbol{x}_j \beta_j \boldsymbol{x}_j^T\right)^{-1} = M_{t_1} C_{t_1}^{-1} \tag{5.37}$$

The error of second merged rule is E_2 with minimizing parameters Π_{t_2} (5.39).

$$E_2 = \sum_{k=1}^{t_2} ||\beta_k \boldsymbol{x}_k \Pi_{t_2} - Y_k||^2 \tag{5.38}$$

$$\Pi_{t_2} = \sum_{k=1}^{t_2} \beta_k \boldsymbol{x}_k Y_k \left(\sum_{k=1}^{t_2} \beta_k \boldsymbol{x}_k \beta_k \boldsymbol{x}_k^T \right)^{-1} = M_{t_2} C_{t_2}^{-1} \tag{5.39}$$

The merged error can be then expressed as a summation of both partial errors (5.40).

$$E_{1\wedge 2} = \sum_{j=1}^{t_1} ||\beta_j \boldsymbol{x}_j \Pi_{t_1} - Y_j||^2 + \sum_{k=1}^{t_2} ||\beta_k \boldsymbol{x}_k \Pi_{t_2} - Y_k||^2 = \sum_{i=1}^{t} ||\beta_i \boldsymbol{x}_i \Pi_t - Y_i||^2 \tag{5.40}$$

$$\frac{\partial}{\partial \Pi_t} E_{1\wedge 2} = \frac{\partial}{\partial \Pi_{t_1}} E_1 + \frac{\partial}{\partial \Pi_{t_2}} E_2 = 0 \tag{5.41}$$

$$\sum_{j=1}^{t_1} \beta_j \boldsymbol{x}_j \beta_j \boldsymbol{x}_j \Pi_{t_1} - \sum_{j=1}^{t_1} \beta_j \boldsymbol{x}_j Y_j + \sum_{k=1}^{t_2} \beta_k \boldsymbol{x}_k \beta_k \boldsymbol{x}_k \Pi_{t_2} - \sum_{k=1}^{t_2} \beta_k \boldsymbol{x}_k Y_k = 0 \tag{5.42}$$

$$\Pi_{t_1} \sum_{j=1}^{t_1} \beta_j \boldsymbol{x}_j \beta_j \boldsymbol{x}_j + \Pi_{t_2} \sum_{k=1}^{t_2} \beta_k \boldsymbol{x}_k \beta_k \boldsymbol{x}_k - \sum_{i=1}^{t=t_1+t_2} \beta_i \boldsymbol{x}_i Y_i = 0 \tag{5.43}$$

As we notice from (5.37)(5.39) we can plug these solutions into (5.43). This allows us to derive the formulas for merged matrix C and parameters Π.

We know that the merged error should equal the error of rule E when feeding it with all parameters. Thus, we can also derive the merged parameters by minimizing it for this case.

$$E = \sum_{i=1}^{t=t_1+t_2} ||\beta_i \boldsymbol{x}_i \Pi_t - Y_i||^2 \tag{5.44}$$

$$\frac{\partial}{\partial \Pi_t} E = 0 \tag{5.45}$$

$$\Pi_t \sum_{i=1}^{t=t_1+t_2} \beta_i \boldsymbol{x}_i \beta_i \boldsymbol{x}_i^T - \sum_{i=1}^{t=t_1+t_2} \beta_i \boldsymbol{x}_i Y_i = 0 \tag{5.46}$$

By plugging this solution into (5.43), we derive the merged matrix C and parameters Π_t (5.48) and (5.50).

$$C_t = \sum_{i=1}^{t=t_1+t_2} \beta_i \boldsymbol{x}_i \beta_i \boldsymbol{x}_i^T = \sum_{j=1}^{t_1} \beta_j \boldsymbol{x}_j \beta_j \boldsymbol{x}_j^T + \sum_{k=1}^{t_2} \beta_k \boldsymbol{x}_k \beta_k \boldsymbol{x}_k^T \tag{5.47}$$

$$C_t = C_{t_1} + C_{t_2} \tag{5.48}$$

$$\Pi_t \sum_{i=1}^{t=t_1+t_2} \beta_i \boldsymbol{x}_i \beta_i \boldsymbol{x}_i^T = \sum_{i=1}^{t=t_1+t_2} \beta_i \boldsymbol{x}_i Y_i = \Pi_{t_1} \sum_{j=1}^{t_1} \beta_j \boldsymbol{x}_j \beta_j \boldsymbol{x}_j^T + \Pi_{t_2} \sum_{k=1}^{t_2} \beta_k \boldsymbol{x}_k \beta_k \boldsymbol{x}_k^T \tag{5.49}$$

$$\Pi_t C_t = \Pi_{t_1} C_{t_1} + \Pi_{t_2} C_{t_2} \tag{5.50}$$

These merged consequent parameters can be further used in the same way as the original ones from the previous rules. However, we need to remember that the model stores an inverted matrix C, and as such needs to be updated. Thus, from formulas (5.48) for matrix C and (5.50) for parameters Π, we derive the updating formulas for both of these properties, (5.52)(5.56) where $\vartheta = (1..1)^T$, I is identity matrix and ψ are weights for each set of parameters.

$$C_t^{-1} = (C_{t_1} + C_{t_2})^{-1} = \left(C_{t_1} C_{t_2} \left(C_{t_1}^{-1} + C_{t_2}^{-1}\right)\right)^{-1} \tag{5.51}$$

$$\begin{gathered} C_t^{-1} = \tfrac{C_{t_1}^{-1} C_{t_2}^{-1}}{C_{t_1}^{-1} + C_{t_2}^{-1}} \\ C_t^{-1} = \frac{C_{t_1}^{-1} C_{t_2}^{-1} C_{t_1}^{-1} \left(\vartheta^T \vartheta\right) I}{2\vartheta^T \left(\left(C_{t_1}^{-1} C_{t_1}^{-1} + C_{t_1}^{-1} C_{t_2}^{-1}\right) I\right) \vartheta} + \frac{C_{t_2}^{-1} C_{t_1}^{-1} C_{t_2}^{-1} \left(\vartheta^T \vartheta\right) I}{2\vartheta^T \left(\left(C_{t_1}^{-1} C_{t_2}^{-1} + C_{t_2}^{-1} C_{t_2}^{-1}\right) I\right) \vartheta} \end{gathered} \tag{5.52}$$

$$\Pi_t^{-1} C_t^{-1} = (\Pi_{t_1} C_{t_1} + \Pi_{t_2} C_{t_2})^{-1} = \frac{\Pi_{t_1}^{-1} C_{t_1}^{-1} \Pi_{t_2}^{-1} C_{t_2}^{-1}}{\Pi_{t_1}^{-1} C_{t_1}^{-1} + \Pi_{t_2}^{-1} C_{t_2}^{-1}} \tag{5.53}$$

$$\frac{\Pi_t}{C_t^{-1}} = \frac{\Pi_{t_1}^{-1}C_{t_1}^{-1} + \Pi_{t_2}^{-1}C_{t_2}^{-1}}{\Pi_{t_1}^{-1}C_{t_1}^{-1}\Pi_{t_2}^{-1}C_{t_2}^{-1}} = \frac{\Pi_{t_1}\Pi_{t_2}\left(\frac{\Pi_{t_2}C_{t_1}^{-1}+\Pi_{t_1}C_{t_2}^{-1}}{\Pi_{t_1}\Pi_{t_2}}\right)}{C_{t_1}^{-1}C_{t_2}^{-1}} \tag{5.54}$$

$$\Pi_t = C_t^{-1}\frac{\Pi_{t_2}C_{t_1}^{-1} + \Pi_{t_1}C_{t_2}^{-1}}{C_{t_1}^{-1}C_{t_2}^{-1}} = \frac{\Pi_{t_2}C_{t_1}^{-1} + \Pi_{t_1}C_{t_2}^{-1}}{C_{t_1}^{-1} + C_{t_2}^{-1}} \tag{5.55}$$

$$\begin{gathered} \Pi_t = \frac{C_{t_2}^{-1}}{C_{t_1}^{-1}+C_{t_2}^{-1}}\Pi_{t_1} + \frac{C_{t_1}^{-1}}{C_{t_1}^{-1}+C_{t_2}^{-1}}\Pi_{t_2} = \psi_1\Pi_{t_1} + \psi_2\Pi_{t_2} \\ \psi_1 = \frac{\vartheta^T C_{t_2}^{-1}\vartheta}{\vartheta^T\left(C_{t_1}^{-1}+C_{t_2}^{-1}\right)\vartheta} \\ \psi_2 = \frac{\vartheta^T C_{t_1}^{-1}\vartheta}{\vartheta^T\left(C_{t_1}^{-1}+C_{t_2}^{-1}\right)\vartheta} \end{gathered} \tag{5.56}$$

The formulas for merging consequent parameters then are (5.52) and (5.56) for merging the gain matrix and the parameters themselves. We can notice that the gain matrices of both rules function as weights of the parameters themselves.

In our solution, we use CCL. Despite the competitive learning process, we can merge such parameters in the same way as the original consequent parameters (not using CCL). This is because we merge only similar rules, where the similarity is given by the similarity in results. When two results are similar enough, the learning process is similar as well, and thus merging does not affect the competitiveness in CCL.

5.5.1.5 Merging rule parameters

For the organization of rules, to define their similarities and dissimilarities, we define these parameters for rule i:

- ω_i as learned weight vector for rules within the ART network;
- giv_i as general inference vector (average inference of rule i, more detailed description in section 5.5.2.1);
- tg_i as number of updates for similarity and dissimilarity measurements.

As mentioned above, when merging two rules we need to merge all parameters describing these rules. Thus, we also need to merge vectors *giv*. This merging is performed equally for each rule, considering the number of updates of each *giv* (5.57). As in rest of this section, we refer to rules one and two as 1 and 2, using these references as indexes of parameters. The resulting reference of merged rule (index) is then denoted as $1 \wedge 2$.

$$giv_{1\wedge 2} = \frac{1}{tg_{1\wedge 2}} \left(tg_1 giv_1 + tg_2 giv_2 \right) \tag{5.57}$$

$$tg_{1\wedge 2} = tg_1 + tg_2 \tag{5.58}$$

Furthermore, we also need to merge the weight vector ω for deriving the match function for the ART network. We derive its merged value as a mean value of original weights of two clusters, weighted by the number of samples introduced to each of them (5.59).

$$\boldsymbol{\omega}_{1\wedge 2} = \frac{1}{t_{1\wedge 2}} \left(t_1 \boldsymbol{\omega}_1 + t_2 \boldsymbol{\omega}_2 \right) \tag{5.59}$$

5.5.2 Splitting and discarding the rules

For splitting and discarding the rules, we measure several factors, such as dissimilarity, erroneousness, age and distance of the classes. When performing splitting, we search for underperforming rules whose samples of different classes are so far apart that they do not satisfy the vigilance.

When discarding rules, we also search for underperforming and dissimilar rules. Moreover, their underperformance (general error) needs to be at the peak of all rules. Furthermore, for a rule to be discarded it needs to be unused (not updated) for a long time so that it is significantly behind compared to the other rules. Since our model is based on fuzzy logic, and hence each rule is updated by each class, when discarding a useless rule, because of these facts, we do not need to worry about forgetting some of the classes. We admit that discarding rules is a very subjective matter. However, from our experience and considering incremental learning, we can see that especially at the beginning of the process with the lack of data, some rules are created too hastily and are unimportant for further use.

More detailed insights into the splitting and discarding decisions are described in section 5.5.3. Here, we focus on formulas for splitting parameters of the rules.

5.5.2.1 Dissimilarity, error and age measurement

As mentioned in section 5.5.1.1, when searching for similarity, we exclude the diagonal from our similarity matrix. However, we cannot compare the same results (giving 1 as the answer). Thus, we store a general inference vector describing the general average inference of the rule that we can compare with actual inference of the same rule. Such a comparison describes how much the results of one rule are similar to its previous results. We can translate this as if a concrete rule is consistent in its results. Such information thus describes if the rule stands as a good contribution to the overall results, stating similar results or, in other words, if a rule makes sense.

The ideal general inference vector is a zero vector with 1 for the true class k. This is thus an initial form of general inference vector $giv_{i,0}$, when no results have yet been presented to the system and we know only the true class (5.60).

$$giv_{i,0} = [a_1 \ldots a_{k-1}, a_k, a_{k+1} \ldots a_C] \tag{5.60}$$

$$a_k = 1;\ a_i = 0;\ i = 1 \ldots C;\ i \neq k \tag{5.61}$$

For such a vector, we cannot simply learn all the inferences, since they vary from one ground truth to another. The inference for one ground truth (the true class) should not be similar to the inference of a different ground truth. A correct rule needs to stand behind its opinion. It cannot assign the same class to every sample in the system (unless there is only one known class). The vector itself is learned, so each inference iv_i^j (inference vector of rule i for class j) is presented in the resulting general inference vector $giv_{i,(tg_i+1)}$ by its strength, the firing degree of the rule (5.62). Vector iv is preprocessed, so that it ranges within [0,1] and is normalized so that it is a unit vector. Here, tg_i represents the total number of previous updates of vector giv, where each dimension of this vector represents the total number of updates for one specific class. Then, giv_i^j is a single parameter describing the average weighted opinion of rule i for known examples of class j. Thus, for one example of class j, only one dimension of parameters giv_i and tg_i is updated.

$$giv^j_{i,\left(tg_i^j+1\right)} = \frac{\left(tg_i^j giv^j_{i,tg_i^j} + \beta_i iv_i^j\right)}{tg^j_{i,tg_i^j} + \beta_i} \tag{5.62}$$

$$tg^j_{i,\left(tg_i^j+1\right)} = tg^j_{i,tg_i^j} + \beta_i \tag{5.63}$$

$$i = 1 \ldots r \tag{5.64}$$

$$j = 1 \ldots c \tag{5.65}$$

However, the comparison of stored general inference to the complete actual inference vector is not sufficient, since this gives only a current opinion about a current class, unlike the general vector's giving an opinion about every class. Thus, we compare these only considering the current class, stating how much the opinion about class j is similar to the general opinion about class j (5.66). These values are within the range [0,1], where 0 stands for absolutely dissimilar inferences and 1 for absolutely similar ones.

$$\begin{pmatrix} \left(1-|giv_1^j-\beta_1 y_1^j|\right) & & & \\ & \left(1-|giv_2^j-\beta_2 y_2^j|\right) & & \\ & & \ddots & \\ & & & \left(1-|giv_r^j-\beta_r y_r^j|\right) \end{pmatrix} \tag{5.66}$$

As a second splitting or discarding condition, we consider an error rate weighted by the influence to overall result (5.68). Thus, if a rule makes an inference which is wrong, its influence on the overall result should not be significant. This means that the clustering creates good clusters and all parameters and rules can stay untouched. However, if the significance grows high, this means that some rules are meant to describe the incoming example but are wrong, and therefore their error should be weighted higher. We define the error as a confusion when the true class is j^*. Here, t is the lifetime of rule i.

$$e_i = \frac{1-(y_{j^*}-\max_{j=1\ldots C,\, j\neq j^*} y_j)}{2} \tag{5.67}$$

$$e_{i,t+1} = \frac{t_i e_{i,t} + \beta_i \frac{1-(y_{j^*}-\max_{j=1\ldots C,\, j\neq j^*} y_j)}{2}}{t_i+1} \tag{5.68}$$

A third parameter, age, is considered for discarding the rules. We want to make sure that the erroneous rule is both old and more unused than any other rule. Thus, we define the age of rule i as a ratio between its lifetime t and the time it has not been updated nt. The more the rule has not been updated compared to its lifetime, the higher its age is (5.69). A rule is considered old if its age it higher than the sum of the ages of all the other rules. This condition is not easy to accomplish; it can be done only if a rule is not updated for a long time.

$$age_i = \frac{nt_i}{t_i} \tag{5.69}$$

5.5.2.2 Class distance measurement

For class distance measurement, we use a similar approach to that for rule distance. However, instead of comparison of vectors ω_i representing each rule i, we take the mean values of samples from rule i belonging to each class j (5.70). Such an entity is easily derivable from parameters $t_{C,i}$ and $\eta_{C,i}$ (section 5.5.1.3).

$$\rho(J,K) = \left(\tfrac{1}{t_{C,J}}\eta_{C,J}\right)^T \cdot \left(\tfrac{1}{t_{C,K}}\eta_{C,K}\right)$$

$$\begin{pmatrix} 1 & & & & & \\ \rho(2,1) & 1 & & & & \\ \rho(3,1) & \rho(3,2) & 1 & & & \\ \rho(4,1) & \rho(4,2) & \rho(4,3) & 1 & & \\ \vdots & & & & \ddots & \\ \rho(r,1) & \rho(r,2) & \rho(r,3) & \cdots & \rho(r,r-1) & 1 \end{pmatrix} \tag{5.70}$$

This distance is then checked for satisfying the vigilance parameter and it is determined whether or not a rule should be split. This measure is important because the vigilance parameter of the ART network is adapted automatically. Then, for low vigilance, small number of clusters is created. But with growing vigilance, these clusters are too big and, potentially, the classes are too far apart. Hence, we need to split such a rule. Further studies on the quality of new rules after splitting has been done in Lughofer (2012).

5.5.2.3 Splitting membership parameters

As mentioned in section 5.5.1.3, for splitting membership parameters (antecedent part), we store separate parameters $t_{C,i}$, $\boldsymbol{\eta}_{C,i}$ and $\gamma_{C,i}$ that when merged follow formulas (5.33). Then, when splitting a rule, after a decision about which classes (dimensions of the parameters) belong to separate clusters, we perform merging of these parameters according to this separation. Here, n is the total number of samples within cluster i, n^j is the total number of samples within cluster i belonging to the class j and $\boldsymbol{x}_k^j$ is a k-th sample within cluster i belonging to class j.

$$t_i = n = \sum_{j=1}^{c} n^j = \sum_{j=1}^{c} t_{C,i}^j \tag{5.71}$$

$$\boldsymbol{\eta}_i = \sum_{k=1}^{n} \boldsymbol{x}_k = \sum_{j=1}^{c} \sum_{k=1}^{t_{C,i}^j} \boldsymbol{x}_k^j = \sum_{j=1}^{c} \boldsymbol{\eta}_{C,i}^j \tag{5.72}$$

$$\gamma_i = \sum_{k=1}^{n} \boldsymbol{x}_k^2 = \sum_{j=1}^{c} \sum_{k=1}^{t_{C,i}^j} \left(\boldsymbol{x}_k^j\right)^2 = \sum_{j=1}^{c} \gamma_{C,i}^j \tag{5.73}$$

Then, when splitting cluster i, we define two sets of classes, A and B, each of them belonging to a new (split) cluster. Such a division then leads us to formulas (5.74), where the first cluster is denoted by index 1 and the second cluster by index 2. When performing the splitting, we also have to remember that parameters $t_{C,i}$, $\boldsymbol{\eta}_{C,i}$ and $\gamma_{C,i}$ need to be split, simply by dividing them into two groups on the basis of sets A and B.

$$\begin{array}{lll} a)\, t_1 = \sum_{j \in A} t_{C,i}^j & c)\, \boldsymbol{\eta}_1 = \sum_{j \in A} \boldsymbol{\eta}_{C,i}^j & e)\, \gamma_1 = \sum_{j \in A} \gamma_{C,i}^j \\ b)\, t_2 = \sum_{j \in B} t_{C,i}^j & d)\, \boldsymbol{\eta}_2 = \sum_{j \in B} \boldsymbol{\eta}_{C,i}^j & f)\, \gamma_2 = \sum_{j \in B} \gamma_{C,i}^j \end{array} \tag{5.74}$$

5.5.2.4 Splitting consequent parameters

Consequent parameters can be split by two sets of classes A and B. Since we expect that the division is correct, the gain matrix C is initialized and only matrix Π is split. Such a splitting is based on dividing this matrix by its columns, where each column represents a set of parameters for one class (5.77). Similarly, as in the previous section, we denote the two new clusters by indexes $_1$ and $_2$.

$$\Pi_i = \left\{ \boldsymbol{\pi}_i^j \right\} = \begin{pmatrix} \boldsymbol{\pi}_i^1 & \boldsymbol{\pi}_i^2 & \boldsymbol{\pi}_i^3 & \dots & \boldsymbol{\pi}_i^c \end{pmatrix} \tag{5.75}$$

$$j = 1 \dots c \tag{5.76}$$

$$a)\, \boldsymbol{\pi}_1^j = \begin{cases} \boldsymbol{\pi}_i^j & j \in A \\ \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} & j \in B \end{cases} \qquad b)\, \boldsymbol{\pi}_2^j = \begin{cases} \boldsymbol{\pi}_i^j & j \in B \\ \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} & j \in A \end{cases} \tag{5.77}$$

Such an approach also follows the basic idea of CCL; and thus, we use this approach for division of the CCL parameters as well.

5.5.2.5 Splitting rule parameters

As in section 5.5.1.5, we also need to treat splitting of parameters for all measurements to allow the self-organization of our model. Since the vector ω is learned for all the samples, we inherit it in both new clusters in the belief that with more samples, enriching both of these clusters, they will move towards their true values. As for the general inference vector, we follow the procedures from previous subsection within section on Splitting. Thus, when separating a cluster on the basis of sets A ad B, knowing that $\boldsymbol{giv}_i$and tg_i store information about each class j, we assign them values as in (5.78).

$$a)\, giv_1^j = \begin{cases} giv_i^j & j \in A \\ 0 & j \in B \end{cases} \quad c)\, tg_1^j = \begin{cases} tg_i^j & j \in A \\ 0 & j \in B \end{cases}$$
$$b)\, giv_2^j = \begin{cases} giv_i^j & j \in B \\ 0 & j \in A \end{cases} \quad d)\, tg_2^j = \begin{cases} tg_i^j & j \in B \\ 0 & j \in A \end{cases} \tag{5.78}$$

5.5.3 Self-Organized mechanism

In this section we describe in detail the learning of parameters necessary for self-organized clustering as well as the logic behind the automation of choices about rule organization. The solution consists of learning the vigilance parameter, learning rate, and processes of merging, splitting and discarding of rules. The assimilated vigilance parameter is used for resonance or reset when generating rules. It needs to fit the requirements of the model since the organization of rules might change. The learning rate represents the rate at which the vectors $\boldsymbol{\omega}$ for resonance and reset are learned for a new sample. Lastly, the self-organization processes are required for sustaining the correct division of the space in order to correctly satisfy parameter ρ (vigilance). The complete solution is then summarized within section 5.5.3.6. The complete architecture of the system can be seen in Figure 5.2.

5.5.3.1 Learning the learning rate

In this section we introduce learning the learning rate for adapting a representative vector $\boldsymbol{\omega}$ for each rule, such that it is used for calculation of a distance between these rules. We can choose either to adapt this parameter to new situations, risking forgetting the old ones, or to learn new situations, risking too slow adaptation to them. In our approach we choose the learning strategy

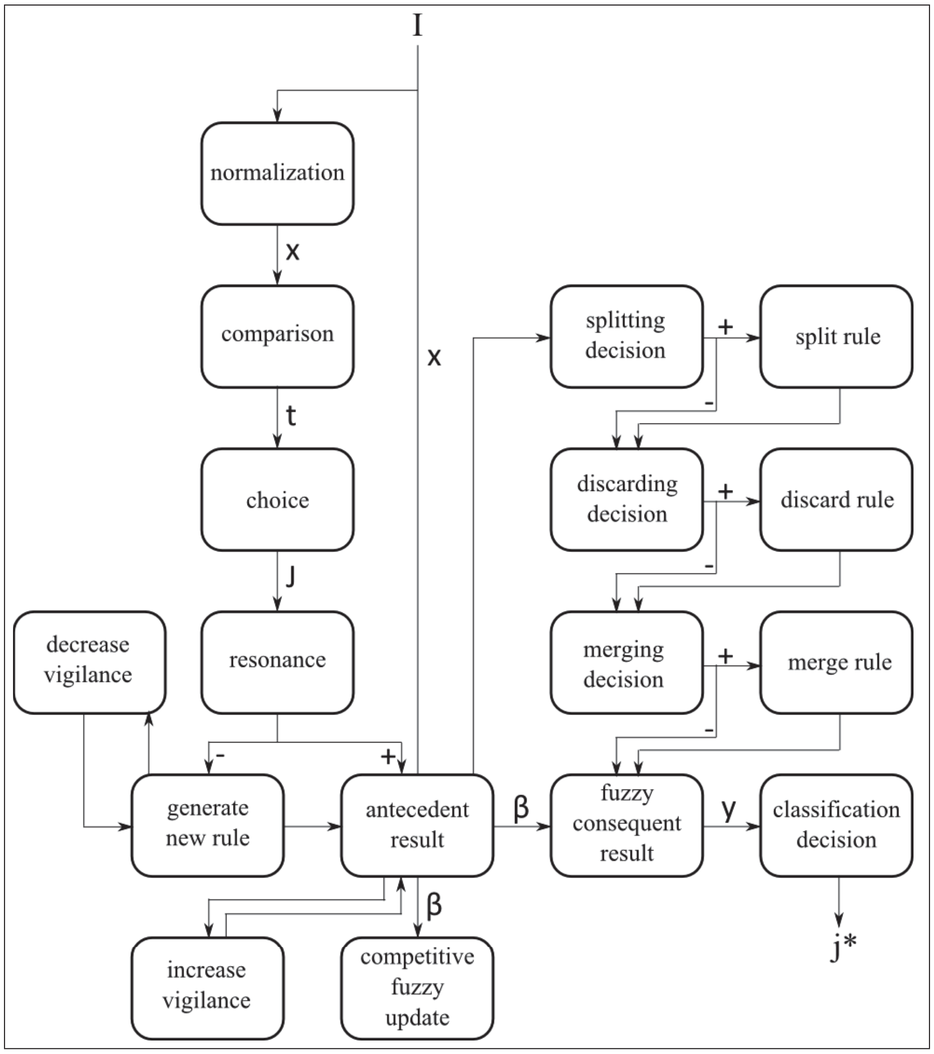

Figure 5.2 SO-ARTIST

that, in our opinion, is more suitable and stable. The representative vector ω can be seen as a centroid for a cluster; with each sample, it moves towards its ideal value for representing all the samples stored in that cluster.

In our solution, we learn the learning rate on the basis of confusion (5.81), where its default value is set to 0.5 (an average of its values). The learning rate is higher and learning faster when the model is confused, and learning is slower when the system performs better. Ideally, if the result of the true class Y^{j^*} equals 1 and a maximum result for any other than the true class j^* $\max_{j=1...C, j\neq j^*} Y^j$ equals zero, confusion is zero and we do not need to learn, since the model is ideal. Conversely, if confusion is at its maximum, equaling 1, the model needs to perform a jump. However, in real applications, Y^j is not necessarily within the range [0,1]. This is caused by the optimization of consequent parameters (and even these do not necessarily have the same range) and the involvement of firing degree. For this reason, we normalize the inference vector in order to achieve range [0, 1], where the function $N(\cdot)$ is the normalization.

$$Y^j = \sum_{i=1}^{r} \beta_i \boldsymbol{x}^T \Pi_i^j \tag{5.79}$$

$$\tilde{Y} = N(Y) \tag{5.80}$$

$$\lambda = \frac{1 - (\tilde{Y}^{j^*} - \max_{j=1\ldots C,\, j\neq j^*} \tilde{Y}^j)}{2} \tag{5.81}$$

5.5.3.2 Learning the vigilance parameter

Assuming the vigilance parameter, in a way similar to that in section 5.5.3.1, we can choose to learn it so that it will converge towards the correct value or be adapted to the correct value. As for the learning rate, our choice is to learn this parameter, so that the system will converge to the proper solution rather than make jumps. Our decision is based on the fact that the learning of the model is incremental and the database can change back and forth. Such changes in the database will in the case of adapting ρ locally lead towards the instability of the system. Instead, we want it to be still partially driven towards previous solutions, still keeping in mind the possibility that they are the correct ones.

For the adjustment of parameter ρ we use confusion as in the previous subsection. Thus, if a system is confused, its learning rate will be faster and the vigilance parameter will be adjusted more towards a new situation. Otherwise, it is kept in its comfort zone. The new possible value for adjustment can be described as a bounding requirement of the system, the necessary conditions for making the right decision.

If the system was in error and the winning rule i was mistaken but a resonance occurs, something has gone wrong. Hence, the vigilance parameter needs to increase towards the maximum requirement, where the reset will occur (5.82).

$$\rho = \lambda \left(\left(\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|} \right)^T \boldsymbol{\omega}_i \right) + (1-\lambda)\,\rho \tag{5.82}$$

On the other hand, if there is a reset, but the winning rule was not mistaken, the system should have made a resonance. Thus, the vigilance parameter needs to be decreased towards the minimum requirement for resonance (5.83).

$$\rho = \lambda \left(\left(\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|} \right)^T \boldsymbol{\omega}_i - 0.1 \right) + (1-\lambda)\,\rho \tag{5.83}$$

The complete process is described in Algorithm 1.

Algorithm 1: Adaptation of vigilance parameter

```
Initialization: $j^{'}$, $j^*$ //winning class and ground truth
                $y_i^{j^{'}}$, $y_i^{j^*}$ //inference of rule i for winning class $j^{'}$ and ground truth $j^*$

function increaseVigilance($j^{'}$, $j^*$)
    if $(resonance)\,\&\,\left(j^i \neq j^*\right)\,\&\,\left(y_i^{j^{'}} \neq y_i^{j^*}\right)$
        adjust $\rho$ according to (5.82);
    end
end

function decreaseVigilance($j^{'}$, $j^*$)
    if $(reject)\,\&\,\left(y_i^{j^{'}} \neq y_i^{j^*}\right)$
        adjust $\rho$ according to (5.83);
    end
end
```

5.5.3.3 Merging process

Given the similarity, inferences and distance matrices (5.25)(5.27), we want to decide which rules to merge. We search for rule pairs $[i, j]$ that satisfy the following requirements:

- distance between rules i and j (5.26)(5.27) is greater than the vigilance parameter ρ;
- similarity between rules i and j (5.25) is greater than the mean similarity between all the rules;
- the opinion of rules i and j about the current sample x is the same (possibilities of classes for this sample are in the same order).

However, during one merging sequence, we can find cases when rule i should be merged with rule j, rule j should be merged with rule k and rule i is nor similar or close enough to rule k to be merged. Hence, after each merging within one merging sequence, we update the inferences to calculate the similarities and also update the *merged* set to recognize, which rules need to be checked again for similarity. Such a procedure is necessary to avoid the merging of dissimilar rules. See Algorithm 2.

Algorithm 2: Merging a set of similar and close enough rules

```
Initialization: $S$, $D$; //similarity and rule distance matrices
                $merged = \{\}$; //set of merged rules for one sequence of merging

function mergeRules(S, D)
    $id = \{[i, j]\}$ //all rule pairs of matrices S and D satisfying vigilance and similarity
    for $id$
        if $((i \in merged) \,||\, (j \in merged)) \,\&\, (i\, is\, similar\, to\, j)$
            merge rules $i$ and $j$ into rule $i$;
            $merged$ += $[i, j]$;
            discard rule $j$;
            update inferences;
        end
    end
end
```

5.5.3.4 Splitting process

Unlike the situation of merging, when splitting one rule, we do not need to consider any collisions. The split takes place when a rule is erroneous, dissimilar and its classes are too distant from each other so that they do not satisfy the vigilance parameter. Thus, we search for a rule *i* that satisfies the following requirements:

- dissimilarity measure (5.70) of rule *i* is less than the mean dissimilarity measure;
- rule *i* achieves an average error (5.66) that is greater than the average errors of the other rules;
- there are inner classes whose inner rule distance (5.68) is less than the vigilance parameter ρ (notice that for this distance the closer two entities are, the greater is the distance measured).

See Algorithm 3.

Algorithm 3: Splitting a rule with distant classes

```
Initialization: $E, cD, dS$; //erroneous vector, class-distance and dissimilarity matrices

if $(e_i = max(e_i))\,\&\,(e_i > mean(e_i))\,\&\,(D < \rho)\,\&\,(i\,is\,dissimilar)$
    id += i;
end

function splitRule(E, cD, dS)
    $id = \{i\}$ //all rules of vector E and matrices cD and dS satisfying vigilance and
      achieving maxima error
    for id
        add new rule to the system;
        split rule i;
        update inferences;
    end
end
```

5.5.3.5 Discarding process

Since we do not receive the database all at once, in incremental learning we face the possibility of creating useless clusters. In our method we detect them by using the erroneous, age and dissimilarity measurements. If a rule is discarded, as in the previous subsections, we need to update the inferences to allow for further organization and updating of the model. See Algorithm 4.

Algorithm 4: Discarding an unimportant rule

```
Initialization: $E, A, dS$; //erroneous, age and dissimilarity matrices

if $(e_i = max(e_i)) \& (e_i > mean(e_i)) \& (i\, is\, old)$
    $id$ += $i$;
end

function discardRule(E, A, dS)
    $id = \{i\}$ //all rules of vectors E, A and dS satisfying the condition for discarding
    for id
        remove rule $i$ from the list of rules;
        update inferences;
    end
end
```

5.5.3.6 Algorithm

As in our previous work, we disconnect the addition of new classes from the necessity of creating new rules for each such occasion. This prevents us from creating too many clusters close to each other and confusing the system. As can be seen in Algorithm 5, we incorporate the self-organization of rules only if a match is performed. However, the adaptation of the vigilance parameter is present also when creating a new rule, because of the nature of its adaptation.

Algorithm 5: SO-ARTIST: A Self-Organized online fuzzy model

```
Initialize: $D$, $cD$; //distance and class distance matrices
            $S$, $dS$; //similarity and dissimilarity matrices
            $E$, $A$; //erroneous and age vectors
            $c$; //number of already known classes

function addClass()
    for each rule $i = 1...r$
        extend consequent parameters $\Pi_i$ by new parameters $\pi_{init} = (0...0)^T$
        extend $giv_i$ by $giv_i^{c+1} = 0$
        extend $tg_i$ by$tg_i^{c+1} = 0$
        set $t_{C,i}^{c+1} = 0$, $\eta_{C,i}^{c+1} = (0...0)^T$ and $\gamma_{C,i}^{c+1} = 0$
    end
end

function SO-ARTIST(x)
    if newClass()
        addClass();
        activate(); (5.10)
    else
        activate(); (5.10)
        $j'$ = winningClass(); (5.4)
        $j^*$ = trueClass();
    end
    if match() > $\rho$
        updateAntecedent(); (5.11)
        updateLearningRate(); (5.81)
        updateART(); (5.7) //learn the $\omega$ vectors
        increaseVigilance($j'$, $j^*$); //Algorithm 1
        splitRule ($E$, $cD$, $dS$); //Algorithm 3
        discardRule ($E$, $A$, $dS$); //Algorithm 4
        mergeRules ($S$, $D$); //Algorithm 2
    else
        addRule();
        decreaseVigilance($j'$, $j^*$); //Algorithm 1
    end
    updateSimilarityVectors(); (5.62) //learn similarities
    updateCCL(); (17) //learn consequent part
end
```

5.6 Results

We evaluate our method on the datasets[1] specified in Table 5.1. The evaluation benchmark consists of both multiclass and two-class problems. In section 5.6.1 we evaluate the self-organized structure of our new model in a comparison to the grid-based search for the best parameters. Then we show the evolution of the vigilance parameter and learning rate by the time new samples enrich the system. In section 5.6.2 we compare our method to the ETS evolving fuzzy model Angelov *et al.* (2004) and online/offline baseline models. It should be noted that all of the online models in our experiments performed initialization of all classes before the learning process. Unlike these, SO-ARTIST (and also ARTIST) and ETS perform this initialization on the fly, and the number or classes is unknown at the beginning of the learning process. At the end, in section 5.6.3 we evaluate the capabilities of CCL on the Handwritten Gestures I dataset. It will be seen that our method is capable of retrieving the best parameters possible and achieving a comparable recognition rate with solutions where these parameters are fixed.

5.6.1 Self-Organization evaluation

In this section, we compare the recognition rate for all datasets specified in Table 5.1. We can see that in many cases the recognition rate of SO-ARTIST exceeds the recognition rate of the baseline model ARTIST that has to fix its parameters. We assume that this phenomenon is caused by the self-organization of the system as well as the adjustment to the local needs of the state of the world. Each result is calculated as an average result from 10 randomly sorted orders of samples from each dataset. The classes are added randomly and on the fly, often causing learning from one example at the introduction of the class. This framework has been

[1] Digits, DNA, Mushrooms, SatImage, Segments and USPS can be found at UCI Machine Learning Repository;
Handwritten Gestures I can be found at http://www.synchromedia.ca/web/ets/gesturedataset;
Handwritten Gestures II can be found at http://www.irisa.fr/;
Gaussian mixtures are mixtures of two overlapping classes generated using Gaussian distributions;
The MNIST dataset in this paper is a random subset of 10000 samples from the original MNIST in LeCun *et al.* (1998) (with a random number of samples per class).

Table 5.1 Datasets

Dataset	# of classes	# of dimensions	# of samples
Digits	10	16	10992
DNA	3	180	2000
Gaussian Mixture I	2	2	800
Gaussian Mixture II	2	2	800
Gaussian Mixture III	2	50	1550
Handwritten Gestures I	17	20	13600
Handwritten Gestures II	21	50	1923
MNIST	10	784	10000
Mushrooms	2	22	8124
SatImage	6	36	4435
Segments	7	19	2310
USPS	10	256	7291

chosen in order to follow the online spirit of the learning process and the fact that the data is unknown in advance.

5.6.1.1 Self-Organized framework evaluation

In this section we compare the performance of the self-organized framework of SO-ARTIST to the one with fixed free parameters in ARTIST by using cross-validation. These results are summarized in Table 5.2, which shows the overall recognition rate on each dataset using no free parameters (SO-ARTIST) and using the best parameters (ARTIST). We will compare the automatic and cross-validated choice of parameters later in section 5.6.1.2.

The recognition rate displayed in Figures 5.3-5.7 is derived as an average error on 500 samples from range [500-*i*, *i*], where *i* is the sample to which the error is assigned. Then, the best parameter is the one that has the model performing the best overall (red curve). We again need to notice that the classes are not initialized at the beginning, and that their number or any other information is unknown to the system at that stage. Then, the classes are added in a random order and at a random time, and their introduction is accompanied by only one sample. Also, the introduction of the classes to the systems is not the same for models SO-ARTIST and ARTIST, but random. In the Figures, the pink curve displays the results of SO-ARTIST and the blue curves display results of ARTIST using different vigilance parameters (ARTIST-n uses vigilance 0.n: n = 1, $\rho = 0.1$).

From the results displayed in Figures 5.3-5.7 we can see that the automatic evolution of the parameters is able to achieve comparable or even superior results (in the form of recognition rate) compared to the model with parameters derived by grid-based cross-validation. For many datasets, the wrong choice of parameters leads to unpleasant results (Figures 5.3, 5.6 and 5.12). In all of these cases our model was able to achieve superior results to the baseline model using the best parameters. Another reason for this behavior is the need of several 'best parameters' for the evolution of the system, which is not possible for the baseline model.

Table 5.2 Comparison of Self-Organized framework to grid-based search for best parameters

	SO-ARTIST	ARTIST
Digits	0.92	0.91
DNA	0.89	0.90
Gaussian mixture I	0.82	0.83
Gaussian mixture II	0.84	0.84
Gaussian mixture III	0.92	0.94
Handwritten Gestures I	0.99	0.98
Handwritten Gestures II	0.91	0.92
Mushrooms	0.95	0.93
SatImage	0.79	0.79
Segments	0.88	0.66
USPS	0.92	0.93

5.6.1.2 Evolution of Vigilance parameter and Learning rate

In this section we inspect the evolution of the vigilance parameter ρ and learning rate λ on datasets from Table 5.1. We can see that the evolution converges towards the good solutions from the grid-based search for fitting the parameters on a concrete dataset. For the SO-ARTIST model, the default value of the vigilance parameter was set to 0.9 and the learning rate to 0.5 for all datasets. For the ARTIST model we searched only for the best vigilance parameter; and

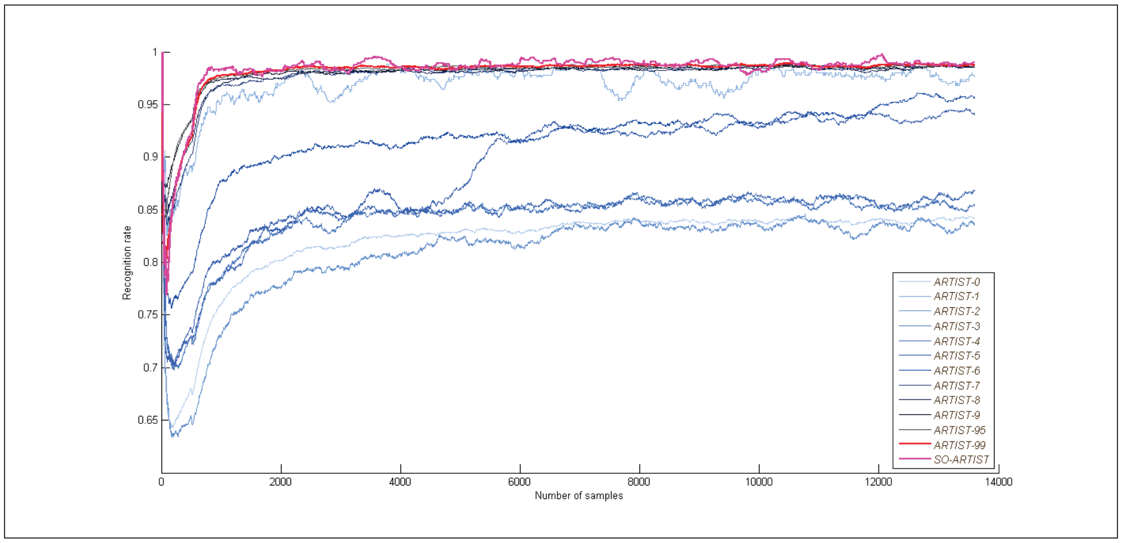

Figure 5.3 Recognition rate comparison of ARTIST and SO-ARTIST on Handwritten Gestures I dataset

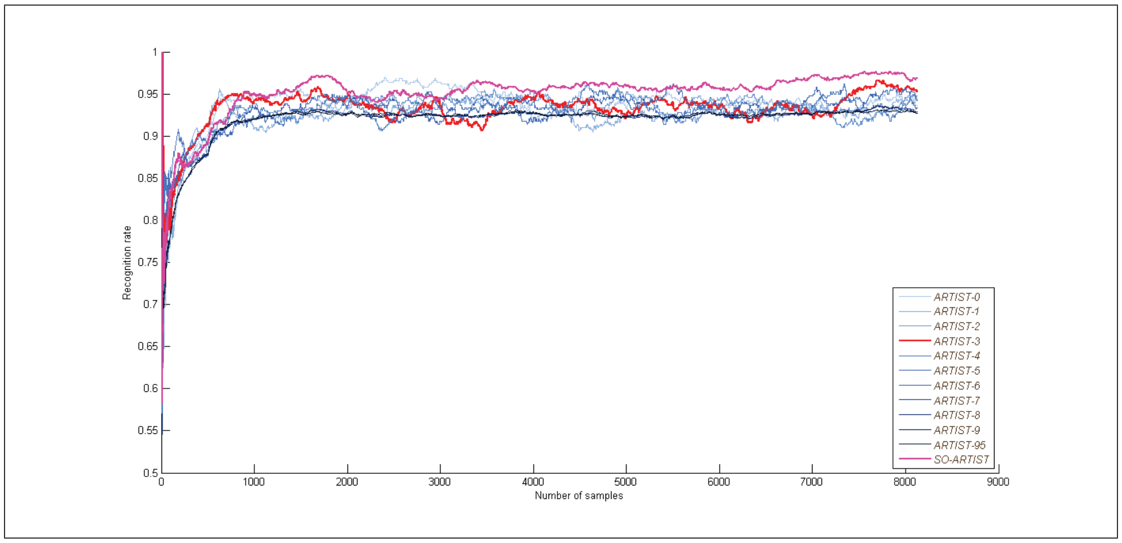

Figure 5.4 Recognition rate comparison of ARTIST and SO-ARTIST on Mushrooms dataset

leaving the exploration of the learning rate using grid-based strategy for future work, we set it by default for 0.1. All the results can be seen in Table 5.3, where the values denote Vigilance parameter / Learning rate.

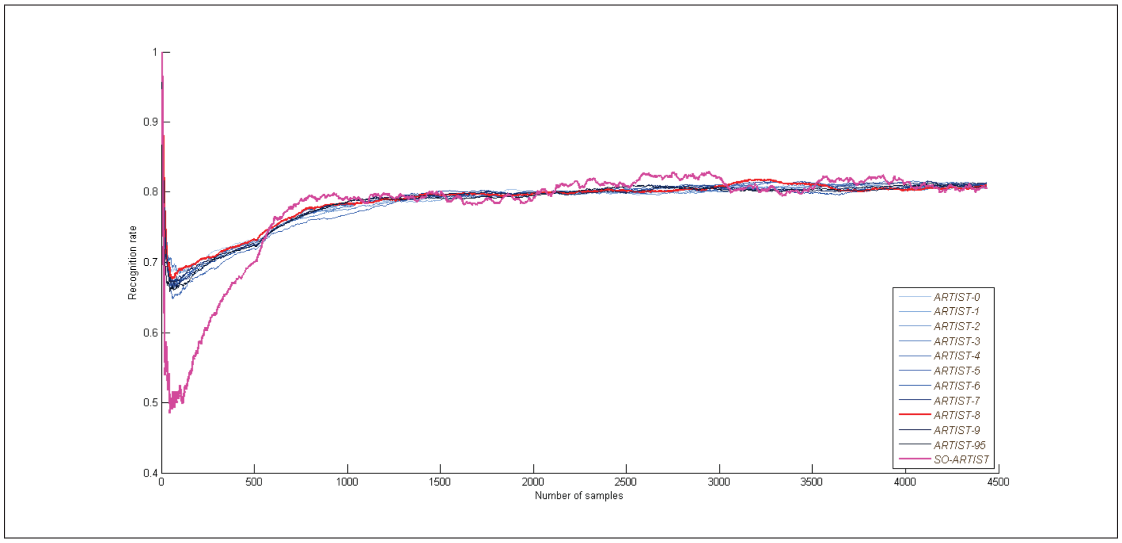

Figure 5.5 Recognition rate comparison of ARTIST and SO-ARTIST on SatImage dataset

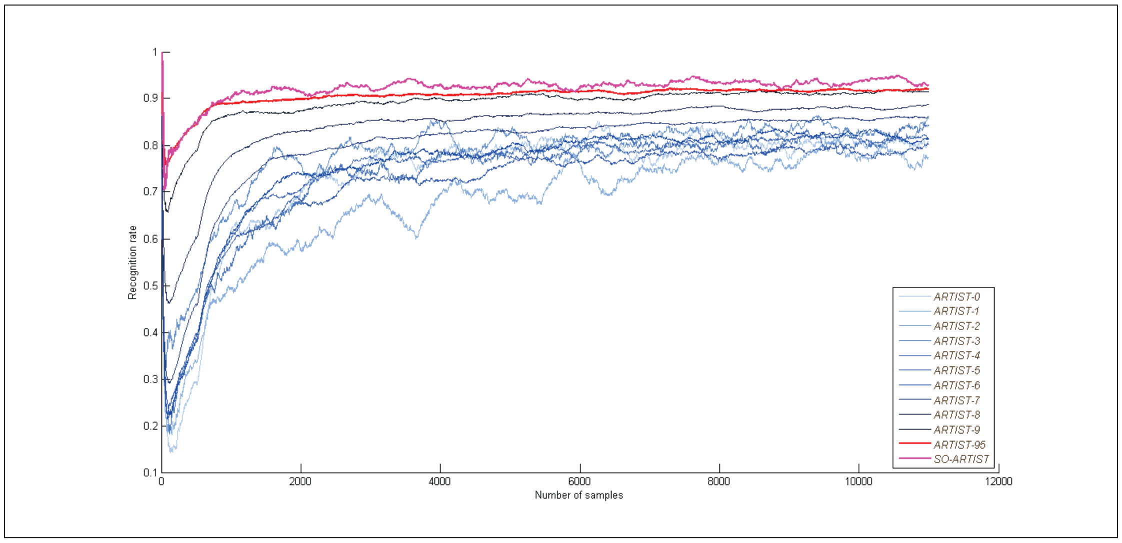

Figure 5.6 Recognition rate comparison of ARTIST and SO-ARTIST on Digits dataset

As can be seen, in some datasets (DNA, Gaussian Mixtures, etc.) the results for SO-ARTIST are slightly worse than the results for ARTIST. It can also be seen that the learning rate for the self-organized model is slightly different. Thus, its adaption to new samples vs. relying on long-term information (from the nature of the ART network learning) is different: in the

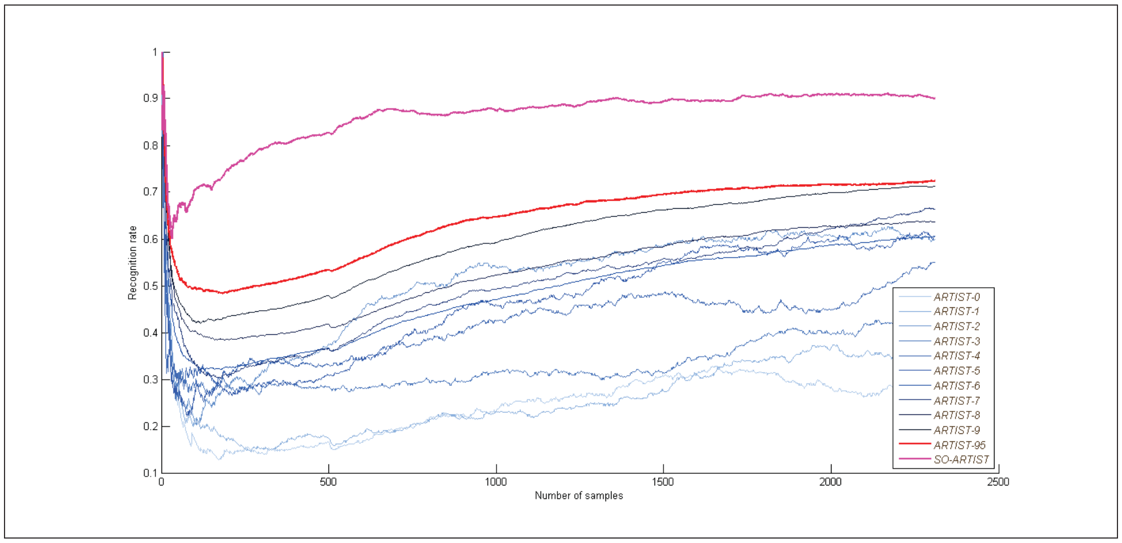

Figure 5.7 Recognition rate comparison of ARTIST and SO-ARTIST on Segments dataset

case of the higher learning rate for SO-ARTIST, it adapts more to the new information; and in the case of lower learning rate, it relies more on the long-term (past) learned information. We assume that this could be the explanation for the difference. We need to notice that considering the default value to be right in the middle, the system tries to adjust it towards the better one. However, the dataset size is small; and it is possible that more samples would lead to a more appropriate value.

Moreover, the significant influence for the recognition rate is done by the vigilance parameter. In most cases, self-adaptation of this parameter reaches a value close to the value from the grid-based search. In the results for the Mushrooms dataset we can see a significant difference. It should be noticed that for this dataset all the parameter settings led to results close to the ones with fixed parameters. However, our new model achieves even better results. Thus, we assume, that changing the value of this parameter both adds some influence and contributes to the robustness of the model.

Figures 5.3-5.12 visualize the evolving values of the vigilance parameter (blue) and the learning rate (pink) by time and its average values per 500 samples and the whole dataset. We can notice that in some cases (Mushrooms, SatImage, Segments) these values change very often and seem to be rather unstable. However, this is the result of their adaptation to the current nature of the system and the current (incoming) sample.

Table 5.3 Evolution of Vigilance parameter and Learning rate

	cross-validation (ρ / λ)	mean value (ρ / λ)	mode value (ρ / λ)
Digits	0.95 / 0.1	0.95 / 0.4	0.98 / 0.42
DNA	0.4 / 0.1	0.4 / 0.3	0.4 / 0.2
Gaussian mixture I	0.8 / 0.1	0.89 / 0.24	0.9 / 0.2
Gaussian mixture II	0.95 / 0.1	0.9 / 0.22	0.94 / 0.2
Gaussian mixture III	0.4 / 0.1	0.15 / 0.12	0.15 / 0.1
Handwritten Gestures I	0.99 / 0.1	0.96 / 0.2	0.97 / 0.2
Handwritten Gestures II	0.9 / 0.1	0.93 / 0.19	0.93 / 0.19
Mushrooms	0.3 / 0.1	0.91 / 0.1	0.9 / 0.1
SatImage	0.8 / 0.1	0.8 / 0.4	0.9 / 0.4
Segments	0.95 / 0.1	0.96 / 0.4	0.96 / 0.42
USPS	0.8 / 0.1	0.8 / 0.35	0.8 / 0.3

5.6.2 Accuracy evaluation

In this section we evaluate our model SO-ARTIST in comparison to a group of baseline models as well as the evolving fuzzy model ETS. We divide these models into two groups for comparison online (Table 5.4) and offline (Table 5.5). Except for ETS, all the other online models needed initialization of classes or at least setting of their number. However, in the case of some classifiers (e.g. AdaBoost) we had to specify a partial classifier for each dataset separately to achieve good results. For our model SO-ARTIST and fuzzy ETS, the classes are learned on the fly, and thus the system starts with no structure built in. In all the other online models, all the classes are initialized at the beginning of the learning. However, we initialized them with

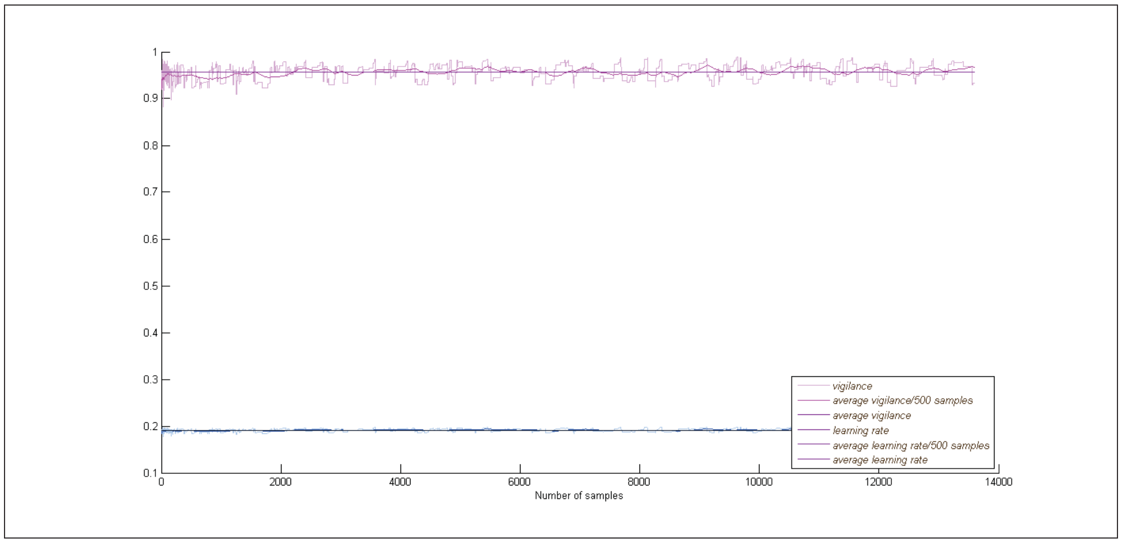

Figure 5.8 Evolution of ρ and λ parameters for Handwritten Gestures I dataset

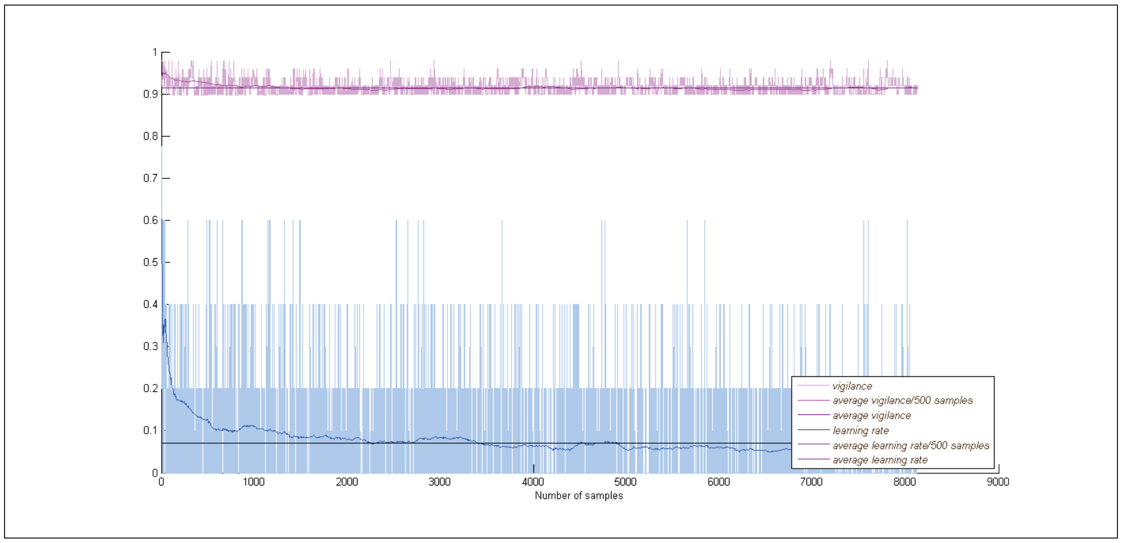

Figure 5.9 Evolution of ρ and λ parameters for Mushrooms dataset

only one sample. Our model is also capable of competing with offline models that already have the whole dataset in advance, and it can adapt to the known situation, unlike the online models (despite the parameter settings).

From the results we can say that our model achieves high stable accuracy for all the used datasets, unlike most of the other models. It can be observed that some models perform in

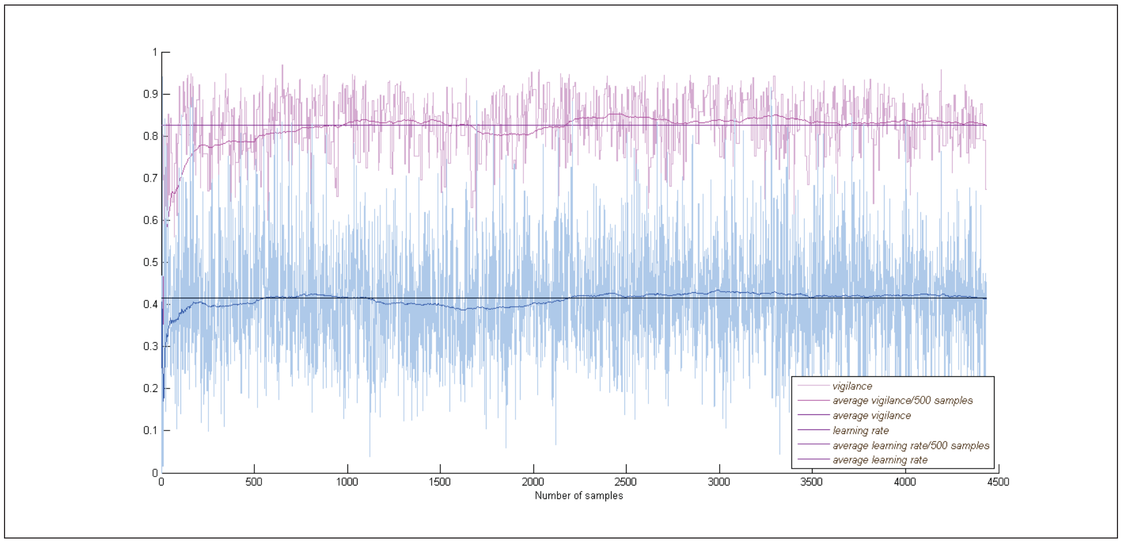

Figure 5.10 Evolution of ρ and λ parameters for SatImage dataset

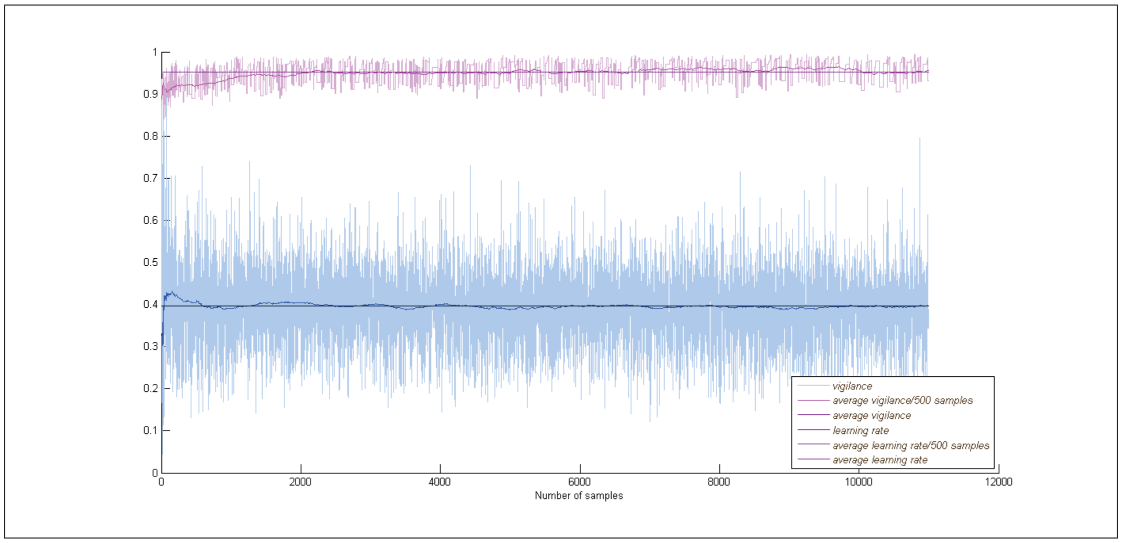

Figure 5.11 Evolution of ρ and λ parameters for Digits dataset

a superior way for some datasets and in an inferior way for others. However, following the no-free-lunch theorem, there is no classifier that applies to all data, and we can see this in our results as well. In the case of our model, even though its performance in some cases is not the best, it still holds a high place among the evaluated models. This we find to be very important in online learning when the data is unknown and we simply cannot adapt to it before the learning process. And it is for this situation that we have built our model.

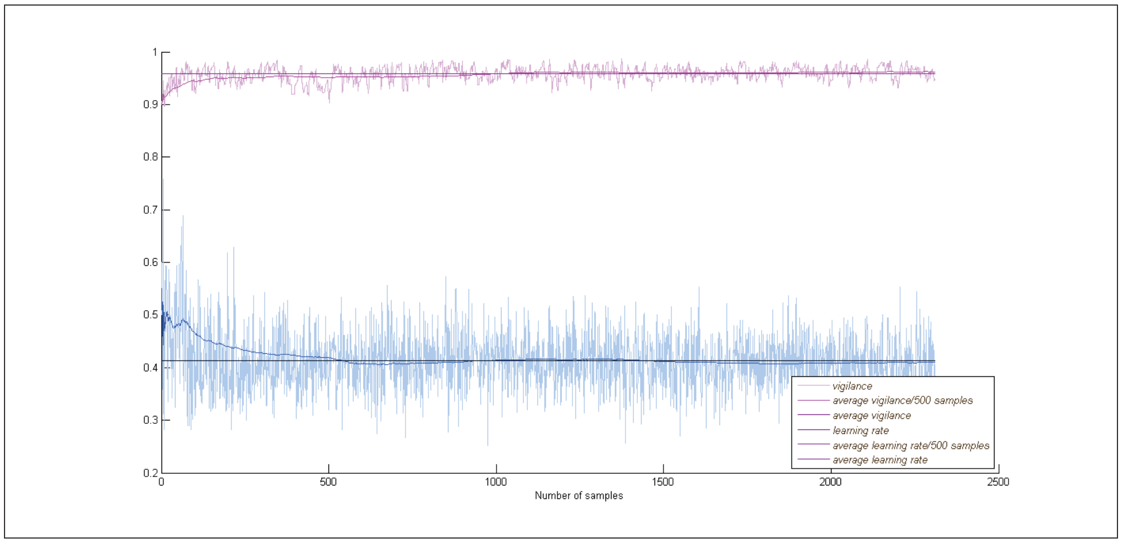

Figure 5.12 Evolution of ρ and λ parameters for Segment dataset

Table 5.4 Recognition rate for Online Learning models

	SO-ARTIST	ETS	IMNB	I-Perceptron	ISGD	OPAC
Digits	0.92	0.86	0.1	0.79	0.79	0.81
DNA	0.89	0.88	0.9	0.86	0.85	0.88
Gaussian mixture I	0.82	0.83	0.49	0.77	0.77	0.74
Gaussian mixture II	0.84	0.85	0.49	0.79	0.79	0.76
Gaussian mixture III	0.92	0.93	0.51	0.91	0.9	0.92
Handwritten Gestures I	0.99	0.98	0.97	0.82	0.82	0.87
Handwritten Gestures II	0.91	0.91	0.85	0.6	0.6	0.65
MNIST	0.81	0.77	0.11	0.79	0.8	0.82
Mushrooms	0.95	0.95	0.52	0.74	0.74	0.69
SatImage	0.79	0.75	0.64	0.7	0.72	0.68
Segments	0.88	0.84	0.73	0.49	0.49	0.51
USPS	0.92	0.88	0.84	0.85	0.85	0.88

Table 5.5 Recognition rate for SO-ARTIST and Offline Learning models

	SO-ARTIST	Naive Bayes	CART	Linear SVM	AdaBoost
Digits	0.92	0.85	0.96	0.87	0.41
DNA	0.89	0.92	0.89	0.93	0.91
Gaussian mixture I	0.82	0.83	0.75	0.83	0.8
Gaussian mixture II	0.84	0.85	0.78	0.85	0.83
Gaussian mixture III	0.92	0.95	0.67	0.94	0.87
Handwritten Gestures I	0.99	0.99	0.98	0.99	0.64
Handwritten Gestures II	0.91	0.92	0.88	0.95	0.82
MNIST	0.81	0.52	0.78	0.85	0.64
Mushrooms	0.95	0.9	0.99	0.87	0.99
SatImage	0.79	0.73	0.81	0.8	0.7
Segments	0.88	0.8	0.95	0.82	0.52
USPS	0.92	0.8	0.88	0.94	0.83

5.6.3 Learning without forgetting

In this section we show the influence of the self-organized nature of our method on the forgetting factor of the system. We evaluate this factor in the Handwritten Gestures dataset for one forgotten class and two forgotten classes. The recognition rate is derived as an average of recognition on the last 500 samples.

Figure 5.13 shows the evaluation of the forgetting of one class. The graph is divided into 4 phases: A: all the classes are learned and the average recognition rate is displayed; B: a random class is unused while the rest continues to be learned and the average recognition rate is displayed; C: the recognition rate of forgotten class without learning is checked and displayed; and D: the recognition rate on the rest of the dataset with learning is displayed. Then, in Figure 5.14 we evaluate the forgetting of two classes. The graph is divided into 6 phases: A: all the classes are learned and the average recognition rate is displayed; B: two random classes are unused while the rest continues to be learned and the average recognition rate is displayed; C: the recognition rate of one of the two forgotten classes without learning is checked and displayed; D: all the classes except the second forgotten one (that was not used in phase C) are learned and the average recognition rate is displayed; E: the recognition rate of the last forgotten class without learning is checked and displayed; and F: the recognition rate on the rest of the dataset with learning is displayed.

We can see that by using CCL old information is preserved and forgetting significantly reduced. This is caused by the nature of CCL, where the learning of consequences is focused only for necessary situations and the parameters in clusters are not overwritten by block learning/missing of samples from some class. We can also notice that SO-ARTIST seems less resistant than ARTIST to the block-forgetting of some class, although the difference is only slight.

In Figure 5.14 we see that the recognition rate of the forgotten classes seem significantly higher than the recognition rate (RR) in the previous steps. However, the difference between stage B and stage C is less than 0.5%, and we can assume they are almost the same. The reason for RR being higher in phase B then in phase A is due to the online learning nature, as the learning process still continues during the phase B, making fewer misclassifications than in phase A.

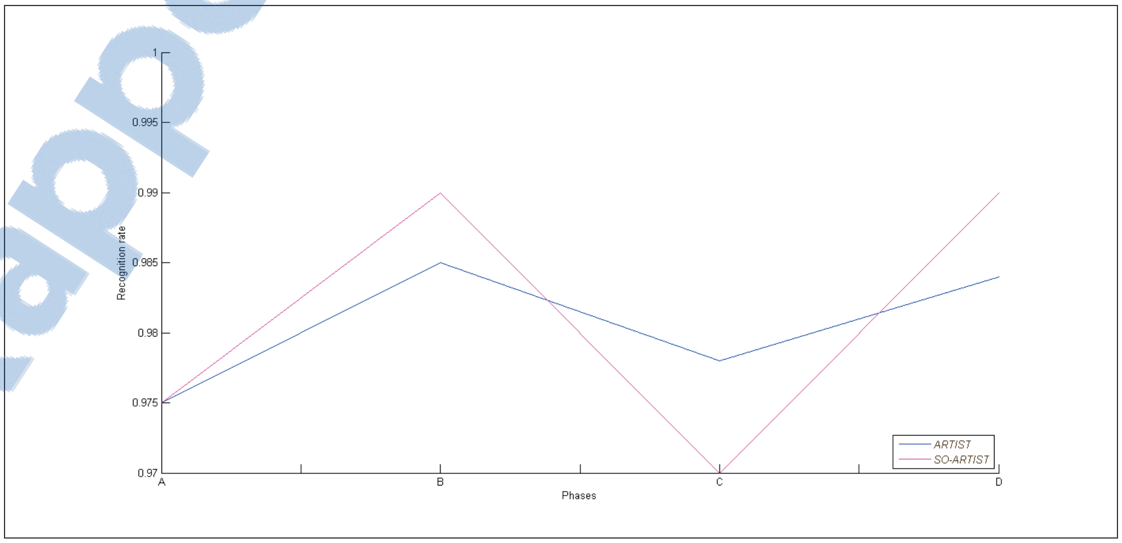

Figure 5.13 Recognition rate when forgetting one class

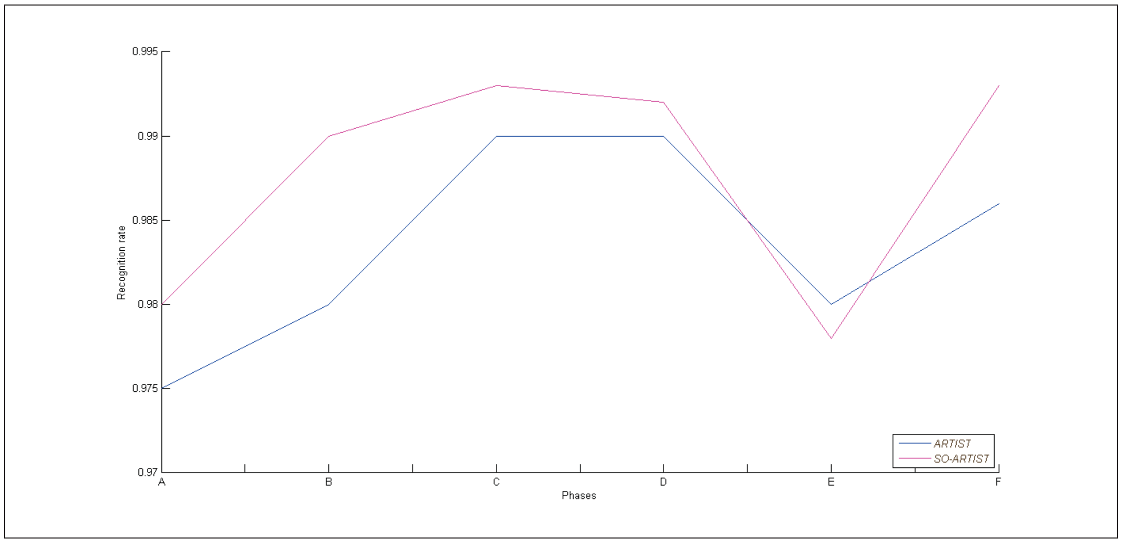

Figure 5.14 Recognition rate when forgetting two classes

5.7 Conclusion and Discussion

The proposed method is an adaptation of TS fuzzy models for online learning, resulting in a novel, self-organized, completely automated system. SO-ARTIST does not require knowledge of the database, such as the number of classes, time of their arrival, or any pre-learning process. It is fully online, and thus in each instance the model is learned and no batch learning is used. We have shown the building steps leading to SO-ARTIST, which consists mainly of merging and splitting solutions, as well as the evolution of parameters ρ and λ.

The results reveal that our method is capable of finding the best parameters in an automatic way. Using this strategy requires no knowledge about the size of the dataset or its properties. The learning rate and vigilance parameter are derived on the basis of the actual state of the world, slowly learning towards the best solution. We can notice slight differences between the resulting parameters and the fixed ones found by the grid-based strategy. This difference is caused by the adjustment to these actual situations. However, the values stabilize after a certain period of learning. It is also noticeable that the whole model stabilizes after a certain period of learning. This is because our datasets are randomly shuffled, and thus most of the classes and differences between them are introduced at the beginning of the learning process in a random order and density.

We have evaluated our model in comparison to online and offline baseline models. We can see that unlike most of them, SO-ARTIST achieves stable results in all datasets and in almost all cases the best results. This fact is very important for our goal, where the dataset is unknown and one cannot fix the model in advance. All these methods (except ETS) needed initialization of the classes at the beginning of the learning.

The ability of self-organization of rules adds to out model's accuracy making it fit to the unknown dataset. In our work, we expected to achieve comparable results to the grid-based search for the best parameters, which we have accomplished. In most of the cases, our automated method outperformed the baseline model ARTIST using fixed parameters. This is due to the adaptability of the model to concrete data as single situations rather than the whole set. At the same time, we tested its accuracy against the forgetting factor, which was empirically proven to be secured. However, the incremental nature of the learning still causes a slower learning at the beginning of the process. This problem is one of the future focuses of our work.

In our future work we want to simplify the self-organized mechanism while keeping the accuracy of the whole system high. At the same time we want to make our model more efficient in terms of accuracy and computational cost. For this purpose we want to explore more works tackling merging and splitting strategies for fuzzy models. As we have stated in the previous sections, some of the works cited in this paper (Lughofer *et al.* (2011); Lughofer (2012); Lemos *et al.* (2013)) compare the antecedent part for the purpose of merging/splitting more significantly, as we do. Therefore we want to explore the possibilities of hybrid merge-split operations in order to find the most appropriate fuzzy rules to merge, split or discard. These studies' measurement of the quality of rules after their merging or splitting is very appealing and an interesting point of view that can further enhance our work. Furthermore, recent research Lughofer *et al.* (2013a) suggests that the usage of all-pairs architecture outperforms the multi-model architecture that we use in this work. Thus, we want to explore this modeling and possibly incorporate it into our solutions. In this work we set default values for our parameters, but without any cross-validation. We believe that there exists a way for their automatic setting so that they will start to evolve before the beginning of the learning itself.

To conclude, our model has achieved automation and high accuracy using online self-organized clustering on Takagi-Sugeno fuzzy model. The contributions of this paper, self-organization of the rules, learning of all parameters for the generation of rules, competitive consequent learning and incremental distance measurement, are shown to work well for the purpose of online learning. In many cases we have outperformed our former model, ARTIST, which uses fixed parameters. This shows that for online learning some level of re-organization is important.

Acknowledgement

This work was supported by the Social Sciences and Humanities Research Council of Canada (SSHRC) as well as the Natural Sciences and Engineering Research Council of Canada (NSERC).

CHAPTER 6

FORGETTING OF UNUSED CLASSES IN MISSING DATA ENVIRONMENT USING AUTOMATICALLY GENERATED DATA: APPLICATION TO ON-LINE HANDWRITTEN GESTURE COMMAND RECOGNITION

Marta Režnáková[1], LukasTencer[1], Réjean Plamondon[2], Mohamed Cheriet[1]

[1] Synchromedia Lab, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] SCRIBENS Laboratory, École Polytechnique de Montréal,
2900 Edouard Montpetit Blvd, Montréal, Québec, Canada H3T 1J4

Paper submitted for publication, October 2016.

6.1 Abstract

In this paper we exploit the use of synthetic data for on-line handwritten gesture commands recognition with an emphasis on the problem of forgetting unused classes. For on-line learning, one of the most crucial moments of the processing is the initialization. In some applications the data is available and these can be fed to the learning model. However, in applications such as user-friendly handwritten gesture recognition, this scenario is not possible. Since from the user perspective it is better to let the user define his own symbols, the learning model is lacking in the amount of data at the initialization. Some strategies have been proposed to acquire synthetic handwritten gesture commands and use these for on-line learning. In this paper we exploit this technique further and focus on the forgetting of unused classes by applying a random buffer and Elastic Memory Learning (EML) to avoid this from happening. In the experiments we search for the proper amount of synthetic data produced for each sample as well as exploit the most appropriate time to stop the generation of synthetic data for learning purposes. We also investigate the influence of synthetic data on forgetting when using the proposed EML. We base the generation of synthetic gesture commands on Kinematic Theory.

6.2 Introduction

Nowadays the market is overflowing with smart devices for personal or commercial uses, including smartphones, tablets, smart boards, etc. When handling these devices, a very natural method is to use a finger or pen. With these, users can perform simple gestures that are translated into commands. An example of such commands can be seen on Figure 6.1. It is important to ensure simplicity for the user to learn these commands in order to easily handle the smart device. Many applications tend to pre-define these gesture commands and force the user to learn them, sometimes providing him with some help to learn how to use them. However, in our work we avoid the pre-definition of gestures, letting the user choose the way to perform the gesture commands on his device. For this reason, in our research we focus on learning from scratch; at the beginning of the learning process no information regarding the database, such as the number of classes or the shape of gesture commands, is defined in advance. Further, it is allowed to add a new class any time during the learning process.

For these reasons we focus on on-line life-long learning with an emphasis on learning from scratch. In on-line learning, each time instance, i.e. each time a new sample is added to the system, both learning and recognition processes are performed, unlike traditional off-line learning. In off-line learning, the whole learning database needs to be known and the model that is learned from these data is then used for the recognition. On the contrary, in on-line learning, the same data that are expected to be recognized are used for the learning of the model. The advantage of such a learning strategy is the possibility of adapting to the dynamics of the real data and not relying only on the data that were known in the past. In off-line learning, if the situation changes, e.g. new classes need to be incorporated to the system, or more data is available, the whole learning process needs to be repeated. This can lead into interruption of the usage of the system. Opposedly, in on-line learning case, the model adapts to the new situation by adjusting its parameters now handling all the known situations. It processes all the known data as they come and does not need to postpone the training until there is enough of it for re-training. By leveraging the knowledge from all the known data, it is better adjusted to the current state of the world opposed to the off-line model that has not been re-trained yet.

Moreover, in many models solving on-line learning constraint, if a new class is introduced to the system, the model is incrementally expanded to incorporate this new class into the model on the line. A challenging constraint of on-line learning as compared to off-line, imposed by the context of gesture commands, is its lower performance due to the missing data that have not yet been revealed and that in the off-line scenario are known from the beginning. For example, if we search for the best division within a data set, we usually achieve better results knowing all the data rather than only a few. Moreover, in the sequential learning scenario and due to the absence of information about future data, the "whole picture" of a situation can be missed. As in our example, if we have only few data, it is much harder to find structures of these data, as these can develop and reveal themselves with more data or with better data in the future.

Furthermore, learning from scratch brings up the learning from very small samples with the expectation of high performance in recognition. If we let a user enter only a few samples or perhaps only one sample for each new class, we also have to expect the user to use his device afterwards. However, learning from one sample, if further samples vary within each class, is very challenging. In our experiments we can notice a significant drop in the recognition rate during the initial learning for each class. One of the solutions is to acquire more data. However, we do not expect the user to give us these data, which would mean to force him to re-enter his gesture commands several times in a line. Thus, we focus on the generation of synthetic data, where these data resemble the original real data, but still vary.

As mentioned earlier in this section, it is not preferred to postpone the usage of the system by learning to recognize the synthetic data. However, the generation of synthetic data and learning them do require some processing time. Not only do we need to learn more samples after only one sample has been input into the system, but we also need to generate these samples. Thus, it is important to find a way to incorporate the synthetic data generation and learning processes into the whole on-line learning process without postponing the recognition of the system or learning from the real samples.

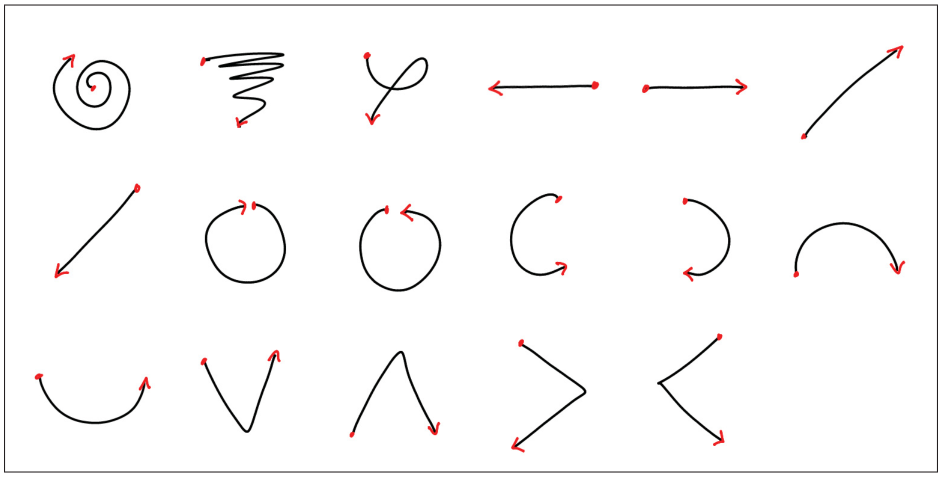

Figure 6.1 Simple handwritten gesture commands. The complete information for the system contains coordinates, time of the acquisition and direction of the gesture command.

In on-line learning problems when learning from scratch, there are few to no examples per class introduced to the model at the beginning of the learning process. This leads to low bias, but high variance, and thus over-fitting. However, more examples are expected to be introduced to the model, which leads to a small recognition rate. The generation of synthetic data can help to improve the initial learning by increasing the amount of data and thus helping the model to generalize well. Yet, the challenge of on-line learning that can be affected by this is the forgetting of unused classes. Usually, in the real world, the classes are not uniformly distributed within the learning process. There are times when some classes of gesture commands are not used for a longer period of time. Since in on-line learning we miss the "whole picture" of the data, it is most likely that these unused classes will be partially or completely forgotten, as all the other classes are continuously learned (see Figure 6.2). Moreover, using synthetic data can amplify this solving of the problem of missing data and help in forgetting the past and unused data. These unused data will be overwhelmed not by a few samples from other classes, but by a large number of samples.

Thus, the challenges of this work are:

a. How to solve the learning from scratch and missing data? How to generate synthetic data?

b. What amount of synthetic data is necessary for solving problem 1?

c. How to avoid postponing the operation of the system?

d. How to prevent forgetting of unused classes?

Thus, the contributions of this paper, as a response to our problems are:

a. Using automatically generated data to support the initial learning,

b. Exploiting the necessities of synthetic data in on-line learning and finding the preferred settings for on-line learning,

c. Proposing a framework for on-line real-time learning that avoid postponing usage of the system,

d. Applying an EML original technique in order to avoid forgetting the unused classes and exploiting the influence of the generation of synthetic data.

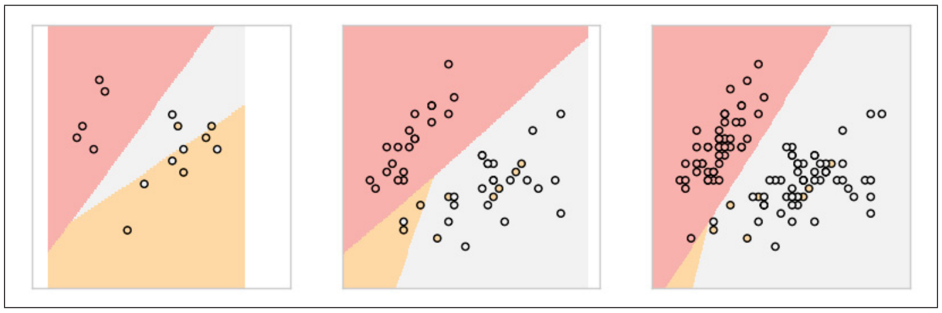

Figure 6.2 Forgetting of unused classes. At first, all classes are learned at the same rate, but gradually one class is left out of the learning process.

In this paper we discuss the on-line learning models, more specifically neuro-fuzzy models and the models for generation of synthetic data in section 6.3. In section 6.4 we describe the Sigma-lognormal model employed for generating synthetic data. Then, in section 6.5 we describe ARTIST, the model selected for this work, followed by Elastic Memory Learning in section 6.6. We describe our framework for on-line real-time learning with the simulation of real-world usage of the system in section 6.7. Lastly we show the results in section 6.8 and conclude this work in section 6.9.

6.3 Related works

In this section we briefly discuss works on on-line and incremental learning with special emphasis on neuro-fuzzy models. At the end we discuss approaches concerning the generation of synthetic data and the use of such data in works already published.

To solve on-line learning problems, where learning and recognition occur at the same time and at every time instance, many researchers have chosen to transform an off-line method into an on-line one. From the most common approaches we make a brief survey of the following groups. In Support Vector Machines (SVMs), the model is composed of support vectors (SVs) that are learned in an off-line mode. In Diehl *et al.* (2003); Carozza *et al.* (2000) the authors propose to incrementally generate these in order to adapt SVMs to an incremental learning environment. Bordes (2005); Tax (2003); Wang *et al.* (2007) the authors choose to update SVs in a sequential manner with every new sample. All these methods try to separate the the classes of the input space, where for our application a multi-class classification approach is needed. The main idea behind the on-line learning approaches is the sequential processing, though needing an initialization. Thus, these methods are not capable of learning from scratch, which is essential for our application. For methods with an incrementally learned subspace, such as Incremental Principal Component Analysis (PCA), there are several applications, such as classification problems for hand gesture recognition in Yao *et al.* (2010), visual tracking in Ross *et al.* (2007), feature extraction in Kasabov *et al.* (2005) and face recognition in Li (2004). These methods use subspace learning rather for the representation than the classification itself. They do require some level on initialization, i.e. small sample of data, but can detect a new class, which is interesting for our research. The Boosting and Bagging based method relies on copies of one weak classifier that is trained on subsets of the data. There are several works adapting such methods for on-line learning, where the weak classifiers as well as their weights are updated accordingly for each sample. On-line Bagging and Boosting algorithms were introduced in Grabner *et al.* (2006); Oza (2005) along with the Adaptive Boosting method (AdaBoost) for two-class problems. This was adapted for multi-class problems in Ditzler *et al.* (2010); Saffari *et al.* (2010). A boosting based on-line model was introduced in Leistner *et al.* (2009) by choosing randomized decision trees as a weak classifier, turning this approach into an On-line Random Forest (ORF). Incremental decision trees with some changes that have been used in ORFs were introduced in Utgoff (1989). All the boosted methods require an initialization of classes, though in our work we need to be able to detect new classes on the fly. Whereas some boosted methods can achieve real-time performance, other methods, such as Random Forests, are more expensive, especially when using high number of trees. In Forster *et al.* (2010); Yu *et al.* (2009) the authors propose to adapt k Nearest Neighbors for the incremental learning with incremental adaptation of the important samples from which the nearest neighbors are selected. It is an interesting idea to use only a subset of data to calculate the nearest neighbors, making the model faster in the prediction phase. However, an initial subset of data is needed in order to begin the on-line learning process. All these methods are capable of learning in a sequential mode; however, they require initialization in the terms of the definition of classes. Thus, they are not applicable to learning from scratch and do not allow adding classes on the fly without learning the whole model again from the beginning.

In this work we have chosen an evolving neuro-fuzzy model based on a fuzzy logic and thus composed of a number of fuzzy rules. There are two main streams for fuzzy models, one was introduced by Mamdani in Mamdani *et al.* (1975) and the other by Takagi-Sugeno in Takagi *et al.* (1985). To adapt these models for on-line learning, the handling of the rules needs to be solved in an on-line manner as well. The work on this topic was done in Babuska (1998), and adapting fuzzy models for non-linear problems was done in Babuška *et al.* (2003); Tan *et al.* (2000). In Ahn *et al.* (2010) the authors use dynamic clustering to handle the rules of fuzzy model, and genetic algorithms for this purpose were used in Cordón, Oscar; Herrera, Francisco; Hoffmann, F.; Magdalena (2001); Cordon *et al.* (2004); Yuan *et al.* (1996). However, these models are not capable of on-line learning. The adaptation of fuzzy models for sequential data streams is surveyed in Lughofer (2011). In Angelov *et al.* (2004); Angelov, Plamen; Filev, Dimitar P.; Kasabov (2010) the authors introduce an incremental Mountain Clustering that can be used for handling of the rules. Another work also based on incremental Mountain Clustering is proposed in Almaksour *et al.* (2010). In Carse *et al.* (1996); Gomez *et al.* (2002); Kasabov (2001) the authors use genetic algorithms for the evolution of the rules, in Lughofer (2008b,a) the authors propose to use incremental Vector Quantization; in Režnáková *et al.* (2012) the authors propose an incremental clustering for the fuzzy rules based on the local results of these rules; in Režnáková *et al.* (2013) the authors propose to use ART-2A Carpenter *et al.* (1991b) for the incremental generation of rules with the self-organized adaptation in Režnáková *et al.* (2015a); in Tencer *et al.* (2015a) the authors propose transductive learning to improve the fuzzy model inference. Most of these models still require some pre-definition, especially in terms of classes; however, they are capable of on-line learning and some of them are capable of learning from scratch. In this work we have chosen a model hereafter referred to as ARTIST Režnáková *et al.* (2013), and we apply our framework and EML to it. Nevertheless, all our propositions within this paper are applicable to any other of these models.

To address the generation of synthetic handwritten data, several models have been proposed. In Schmidt *et al.* (1988) the authors propose to use behavioral models; in Neilson *et al.* (2005); Tanaka *et al.* (2006) the authors use the minimization principles to generate synthetic handwritten data; in Gangadhar *et al.* (2007) the authors use neural networks for this purpose; in Tencer *et al.* (2015b) the authors propose to sample new data in order to enhance the performance of the fuzzy modeling; and in Plamondon *et al.* (2006); Djioua *et al.* (2009) the authors propose to use kinematic models to best describe the human hand movement and produce reliable synthetic handwritten data. The work in this paper is based on the Kinematic Theory describing rapid human movements by the Sigma-lognormal model that was proposed and further extended in Plamondon *et al.* (2014); Djioua *et al.* (2009); Plamondon *et al.* (2003) and applied to handwritten gesture command recognition in Almaksour *et al.* (2011). We will briefly describe this approach later in the paper.

In Almaksour *et al.* (2011); Plamondon *et al.* (2014) the authors propose to use the Sigma-lognormal model to tackle the generation of synthetic samples for recognition of handwritten gesture commands; however, they mainly use these only to boost the learning process for the improvement of the performance. However their work lack the focus on the side effects of using synthetic data as well as the application to the real-world problems.

Usually, on-line learning based models do not tackle the addition of classes on the fly, nor the learning from scratch. Both these aspects result into the low generalization of on-line models when having only a small number of data. There are works proposing the use of artificial data, but only few that are specifically tailored for handwritten gestures, as it is in the case of Sigma-lognormal model. To our knowledge there has not been a work tackling the way of incorporating the synthetic data into the on-line learning process, nor the problem of forgetting of unused classes that is a natural consequence of adding artificial data into the learning stream in blocks, right after they are generated.

Thus, in this work we focus on the these aspects of the generation of synthetic samples and their use for on-line learning, i.e. the non-forgetting of unused classes of gesture commands and the influence of automatically generated samples on this aspect of on-line learning. We also explore the parameters of the process of the generation of synthetic samples, i.e. the stop criterion of the generation itself and the number of the synthetic samples to be generated. This work is a significant extension of our previous work published in Režnáková *et al.* (2015b) in which we introduced the randomized buffer to tackle the real-world use of an evolving system with generation of synthetic samples.

6.4 Sigma-lognormal model

The Sigma-lognormal model is based on the Kinematic Theory of rapid human movements. To be able to mimic the human movements and reliably generate synthetic data, it involves describing the impulse responses of a neuromuscular network Plamondon *et al.* (2014). Each original hand gesture command can be reconstructed as a number of curved vector strokes, with each of these strokes composed of 6 parameters 6.1, with t_0 referring to the time occurrence of each neuromuscular command, D being the magnitude of this command, μ the logtime delay and σ the logresponse time with which the neuromuscular system reacts; θ_s and θ_e refer to the starting and the ending directions of the curvature.

$$\vartheta = P\left[t_0, D, \mu, \sigma, \theta_s, \theta_e\right] \tag{6.1}$$

In the application for on-line learning, each time an original sample is introduced, Sigma-lognormal model reconstructs it as precisely as possible, the system then duplicates it by adding some noise Martin-Albo *et al.* (2014); Režnáková *et al.* (2015b) and translated back into coordinates to generate synthetic samples. The noise is added to the vector ϑ in form of $\vartheta = \vartheta + [0, 0, n_\mu, n_\sigma, 0, 0]$, where $n_\mu \in [-0.15\mu, 0.15\mu]$, $n_\sigma \in [-0.15\sigma, 0.15\sigma]$. Since we want to generate samples as if they were created by the user, we have chosen these parameters describing the neuromuscular timing. When a new gesture is obtained using updated vector, a set of 20 geometrical features is applied, i.e. length of the gesture, curvature, etc. In Figure 6.3 we can see the original data as compared to the data reconstructed by the Sigma-lognormal model without any noise as well as an example of a duplicated sample, a new specimen where some noise has been applied to it. To generate additional 10 samples takes on average 12.96 s. After this process, feature extraction is applied on these newly created synthetic data. In this work, geometrical features have been used as defined in Režnáková *et al.* (2012), where the authors used some geometrical features form Willems *et al.* (2008); Rubine (1991); Peura *et al.* (1997). These synthetic samples are supplied further to the buffer where they are stored until they can be used for the learning purposes (Figure 6.5). More information on the Kinematic Theory and the Sigma-lognormal model can be found in Plamondon *et al.* (2014).

6.5 ARTIST

The ARTIST model applies the ART-2A incremental clustering for the handling of fuzzy rules Režnáková *et al.* (2013); Carpenter *et al.* (1991b). Unlike many other evolving fuzzy models,

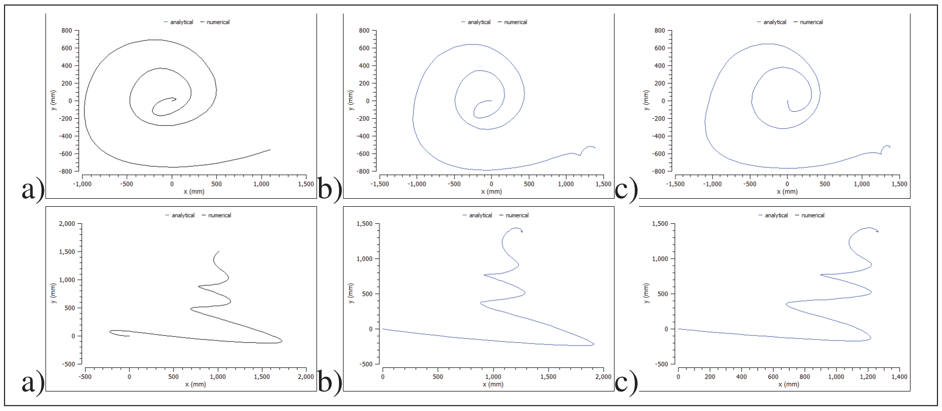

Figure 6.3 Comparison of a) original data to b) data reconstructed by the Sigma-lognormal model without any noise applied to it and c) synthetic data generated by the Sigma-lognormal model from duplicated data after adding some noise. The figure was produced using Scriptstudio O'Reilly *et al.* (2009), a software package that extracts lognormal parameters from a velocity profile and reconstructs it minimizing the reconstruction errors.

it does not take into account the addition of a new class to the system for the purpose of generating rules, and thus it does not force creating a new cluster at each occasion. Since ART-2A is unsupervised, classes can be added on the fly and the learning can start from scratch as no pre-definition of classes is needed. The model itself is based on the Takagi-Sugeno fuzzy modeling, where each rule is in the form of the $IF\,THEN$ condition 6.2. Such a rule is further divided into antecedent and consequent parts, where IS is responsible for the antecedent and $THEN$ is responsible for the consequent. The antecedent part denotes how similar a sample x is to the samples of cluster A according to rule R_i and the consequent part then implies what this rule thinks about this sample. Each rule then gives an opinion for such a sample and for each class, denoting how much it thinks this sample belongs to each class.

$$R_i : \; if \;\; \boldsymbol{x} \; IS \; A \; THEN \; y = \boldsymbol{x\pi} + b \tag{6.2}$$

The consequence $y = \boldsymbol{x\pi} + b$ is thus derived as $y_i = \beta_i \boldsymbol{x}^T \Pi_i$, where β_i is the similarity of sample x to the rule R_i, y_i is the inference of R_i for all classes and $\Pi_i = \left\{ \boldsymbol{\pi}_i^j \right\}_{j=1\ldots c}$ is a set of the consequent parameters of R_i for each class j. After the model obtains all the local results from each rule, a weighted average of y_i is derived 6.3, where the result for each class is represented as 6.4, with r being the number of all rules. Then, the winning class $j^{'}$ is derived as the maximum of all the results Y^j 6.5. This inference is further compared to the ground truth j^* to decide whether an error has occurred.

$$Y = \frac{1}{\sum_{i=1}^{r} \beta_i} \sum_{i=1}^{r} \beta_i \boldsymbol{x}^T \Pi_i \cong \sum_{i=1}^{r} \beta_i \boldsymbol{x}^T \Pi_i \tag{6.3}$$

$$Y^j = \sum_{i=1}^{r} \beta_i \boldsymbol{x}^T \boldsymbol{\pi}_i^j \tag{6.4}$$

$$j^{'} = arg \max_j Y^j \tag{6.5}$$

The whole architecture of the model is shown in Figure 6.4.

6.6 EML: Elastic Memory Learning

In ARTIST the local Recursive Least Squares (RLS) is used for the learning of consequent parameters Π_i. Each time a new sample is introduced to the model, all parameters for all classes within all rules are learned accordingly. Thus, if one class is not used for a longer period of time, samples from all the other classes can cause the parameters for such a class to be overwritten (see Figure 6.2). This can later result in lower recognition of such now forgotten classes by the rules and by the model itself. In Režnáková *et al.* (2013, 2015a) we proposed using Elastic Memory Learning to prevent the forgetting of unused classes.

Generation of synthetic data can lead into the emphasis of forgetting of unused classes (see Figure 6.2). To prevent this, in this work we have chosen to use EML for the RLS algorithm used in neuro-fuzzy models for learning the consequent parameters $\Pi_i = \left\{ \boldsymbol{\pi}_i^j \right\}_{j=1...c}; i = 1...r$

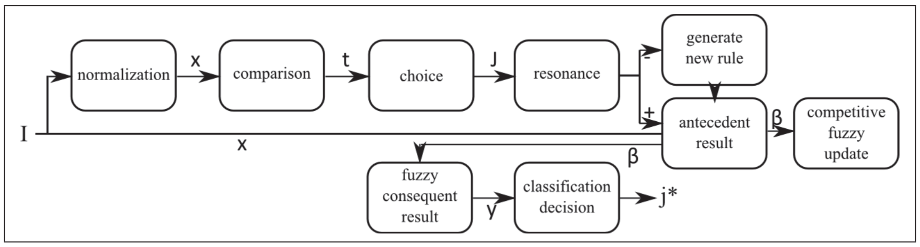

Figure 6.4 ARTIST. This model is composed of fuzzy rules 6.2 that are managed by an ART-2A network, but the inner functionality of the rules is independent of it. First input I is processed by the ART-2A network, where the network is responsible for managing the rules of the model. It is normalized to $\boldsymbol{x}$ so that a cosine distance can be obtained and compared to the existing rules. The choice of the best rule is made and the resonance layer decides whether to update the chosen rule or create a new one. From this point a fuzzy network takes the control and calculates the antecedent part, which is used as a weight β for consequent predictions. After obtaining the consequent results, a prediction is made and used for competitive update 6.10.

for r rules and c classes. By applying competitive updates to the consequent parameters, not all the classes are updated for each sample, but only the important ones. This helps to undersample the classes that are presented to the system in a higher frequency as opposed to other classes.

The aim of RLS is to minimize a squared difference between an actual result $\left\{\beta_i \boldsymbol{x} \boldsymbol{\pi}_i^j\right\}_{j=1\ldots c}$ and an expected result Y_i 6.6.

$$E = \sum_{k=1}^{t} ||\beta_{i,k} \boldsymbol{x}_k \Pi_{i,k} - Y_{i,k}||^2 \tag{6.6}$$

Minimizing this error by sending its derivative to zero gives us formulas for updating a covariance matrix C_i 6.7 and set of parameters $\Pi_{i,t}$ 6.8 for a rule i.

$$\Pi_t = \sum_{i=1}^{t} \beta_i \boldsymbol{x}_i Y_i \left(\sum_{i=1}^{t} \beta_i \boldsymbol{x}_i \beta_i \boldsymbol{x}_i^T \right)^{-1} = M_t C_t^{-1}$$

$$C_t = \sum_{i=1}^{t} \beta_i \boldsymbol{x}_i \beta_i \boldsymbol{x}_i^T = C_{t-1} + \beta_t \boldsymbol{x}_t \beta_t \boldsymbol{x}_t^T$$

$$C_{i,t}^{-1} = C_{i,t-1}^{-1} - \frac{C_{i,t-1}^{-1} \beta_{i,t} \boldsymbol{x}_t \beta_{i,t} \boldsymbol{x}_t^T C_{i,t-1}^{-1}}{1 + \beta_{i,t} \boldsymbol{x}_t^T C_{i,t-1}^{-1} \beta_{i,t} \boldsymbol{x}_t} \tag{6.7}$$

$$\Pi_t = M_t C_t^{-1} = (M_{t-1} + \beta_t \boldsymbol{x}_t Y_t) C_t^{-1} = \Pi_{t-1} \left(C_t^{-1} \left(C_t - \beta_t \boldsymbol{x}_t \beta_t \boldsymbol{x}_t^T \right) \right) + C_t^{-1} \beta_t \boldsymbol{x}_t Y_t$$

$$\Pi_{i,t} = \Pi_{i,t-1} + C_{i,t-1}^{-1} \beta_{i,t} \boldsymbol{x}_t \left(Y_{i,t} - \Pi_{i,t-1} \beta_{i,t} \boldsymbol{x}_t \right) \tag{6.8}$$

Parameters $\Pi_{i,t}$ are composed of a number of local parameters $\boldsymbol{\pi}_{i,t}^j$ for each class j, rule i at a time t, with an updating formula as shown in 6.9. We define a correction parameter ε_i^j that is set to either 0 or 1, on the basis of the inference Y_i^j, given rule i and class j that are set in the updating formula for RLS 6.9 as an ideal result of the inference for sample $\boldsymbol{x}$ at time t using parameters $\boldsymbol{\pi}_{i,t}^j$ for rule i and class j. Thus, for the true class $\varepsilon_i^j = 0$ and for all the other classes $\varepsilon_i^j = 1$.

$$\boldsymbol{\pi}_{i,t}^j = \boldsymbol{\pi}_{i,t-1}^j + C_{i,t-1}^{-1} \beta_{i,t} \boldsymbol{x}_t \left(\varepsilon_i^j - \boldsymbol{\pi}_{i,t-1}^j \beta_{i,t} \boldsymbol{x}_t \right) \tag{6.9}$$

In EML, the parameter ε_i^j does not update the parameters for each sample, i.e. if a sample is not of the class j, it does not necessarily update parameters for such a class to 0. As we stated earlier, such an update can lead to forgetting the class j in cases when this class has not been used for a longer period of time, and its parameters are constantly adjusted to output 0. Thus, we set parameter ε_i^j to update parameters $\boldsymbol{\pi}_{i,t}^j$ only if the class j is the current true class or if the local result of the rule i for class j is greater than the result of the true class j^* 6.10.

$$\varepsilon_i^j = \begin{cases} 1 & j = j^* \\ 0 & y_i^j > y_i^{j*} \\ \boldsymbol{\pi}_{i,t-1}^j \beta_{i,t} \boldsymbol{x}_t & otherwise \end{cases} \tag{6.10}$$

Using synthetic data can indeed help to improve the initial recognition. However, it can also lead to forgetting unused classes and magnifying its effect by the amount of synthetic samples produced for each sample. Thus, in this work we want to investigate the effect of generating synthetic samples on recognition, forgetting and EML and to find a balance in how many samples to generate to keep the initial recognition high, but at the same time to keep the forgetting low.

In the results we separate the EML from ARTIST to show the sole influence of the EML on the forgetting of unused classes.

6.7 Framework for on-line real-time learning using synthetic data

Since in on-line learning data are being introduced to the model sequentially, we need to adapt the generation of synthetic data and their distribution to the model accordingly. Each time a new sample is introduced to the model, a generation of these data starts. However, the processing time of the generation of synthetic data is not negligible. We cannot afford to postpone the usage of the system and let the user wait for the synthetic samples to be produced and finally learned by the model. Thus, we propose a randomized buffer that allows the flow of the real data, which are created by the user himself, without any interruption from the synthetic data generation system. To randomize the buffer in our work is preferred over block learning since block learning can cause the very same problems as setting the correction parameter ε_i^j in our learning to 0 for a class that has not been used for a long period of time (section 6.6). In other words it can lead to forgetting of not only unused class but of all the classes except the current one.

The processing of the sequential data is as follows (k is number of features, r is number of rules, c is number of classes and n is number of samples):

a. Receive a real sample x, a sequence of 2D coordinates

b. Process the sample x

    a. Apply feature extraction, i.e. a set of 20 geometrical features ($O(k)$)

    b. Recognize, i.e. classify x using ARTIST model ($O(krc)$)

    c. Send it to the model to be learned, i.e. process by ART-2A network and learn fuzzy antecedent and consequent parameters ($O(k^2rc)$)

c. If the system is not busy with real data, generate synthetic data using Sigma-lognormal model

    a. Translate the reconstructed sample into vectors ϑ of lognormal strokes 6.2

    b. Add a noise to these strokes, i.e. $\vartheta = \vartheta + [0, 0, n_\mu, n_\sigma, 0, 0]$

    c. Translate duplicated sample (in form of ϑ) back to the coordinate system

    d. Apply feature extraction, as in step 2a

    e. Push new duplicated sample to the buffer

d. In the buffer

    a. Randomly shuffle all the data ($O(n)$)

    b. While the system is not busy with the data, pop data to the model to be learned, i.e. steps 2b-c ($O(1)$)

e. Else, wait for the system to be ready

This process is further depicted in Figure 6.5.

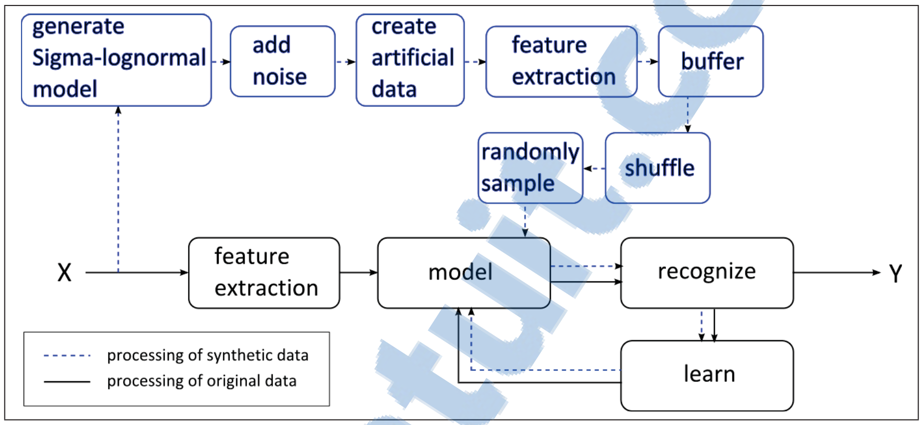

Figure 6.5 Framework for on-line learning from scratch using synthetic data. X refers to an input sample that is fed to the model and recognized. The information from the recognition process is then used for the learning and new data are added to the model (the model is evolving). The solid line refers to the processing of real samples, where on the basis of recognition the system gives an output Y of a predicted label. The dashed line refers to the additional processes related to the generation of synthetic data.

In the next section we can see that the combination of EML with Randomized buffer using synthetic samples leads to increased performance, while preventing the forgetting issue of unused classes. Moreover, we explore the settings of the online framework.

6.8 Experiments

In this section we show our experiments done on the Handwritten Gesture database[1]. In our previous work we have proposed to use the EML in order to avoid the forgetting of unused classes. This issue especially arises in the environment of the generation of synthetic data,where bulks of data of the same class are generated in a short time period. At the same time, to avoid the unnecessary generation of synthetic data, we need to find the best setting of the amount of synthetic data to be generated for each sample as well as the length of the

[1] Handwritten Gestures dataset can be found at http://www.synchromedia.ca/web/ets/gesturedataset

generation itself. For this purpose we evaluate on a number of Scenarios, each dedicated to a different setting as well as introducing a new class to model with more samples per already learned class and the forgetting of unused classes, i.e. not being automatically produced at that time. In the results we will see how using EML significantly improves the overall results and empowers the generation of synthetic data for the initial learning and learning on the fly.

6.8.1 Experimental setting

In this work we use a dataset of handwritten gesture commands[1] that contains 17 classes of ~1950 gesture commands per class. These data were acquired from 20 distinct writers captured by Fujitsu-Siemens Stylistic ST5022 in InKML format. The data are stored in the form of 2D coordinates; and for further processing of the data, we use the feature extraction techniques specified in Režnáková *et al.* (2012). For each of our evaluations (i.e. every distinct run), the order of the samples is different as well as the selected samples for each class (since we focus on the initial learning where the number of samples per class is small). For each sample we generate a different number of synthetic samples and compare the recognition and error rates on these in order to find the best suited amount of automatically generated samples. In the following we describe the 4 scenarios used in this work:

Scenario 1: Learning with classes added on the fly

In this scenario we let the system randomly add the classes on the fly, i.e. at random occurrences. For each sample that is introduced to the system, we generate a different number of duplicated samples (varying from 1 to 50) and add these samples to the randomized buffer. We compare our results to the learning with no synthetic data. Further, we use EML to prevent the forgetting that is caused by blocks of data of the same class.

Scenario 2: Searching for the appropriate stop criterion

In on-line learning, the most crucial part of the process regarding the amount of data is the initialization (i.e. beginning) or the introduction of a new class. Thus, we want to investigate the stop criterion for the generation of synthetic samples, i.e. after what amount of original samples it is reasonable to stop generating synthetic samples.

Scenario 3: Learning with focus on one class introduced later

In this setting we intentionally skip one random class at the beginning of the learning process and introduce it later. We compare the results with no synthetic data added to the randomized buffer with added synthetic data. As in the previous scenarios, we compare these results with the model using EML to prevent forgetting.

Scenario 4: Exploiting the forgetting of an unused class

To investigate the forgetting of an unused class, we use the following benchmark:

a. Phase 1: learn all samples,

b. Phase 2: exclude one random class from the learning process and learn all the others,

c. Phase 3: recognize the unused class without any learning,

d. Phase 4: learn all classes.

We derive the recognition rate for all these phases. We focus mostly on phase 3, where we investigate the recognition rate of the unused class to see how much it has been forgotten.

6.8.2 Results

We divide our results according to the 4 scenarios. In Figures 6.6, 6.7, 6.8, 6.10 and 6.11, the learning curve is derived as 6.11, where the $|errors|_1^t$ is the number of errors from the beginning of the learning until time t for which the recognition rate is being derived. At the beginning of the learning, no error is detected because most of the classes are just being introduced. Therefore the $recognition\,rate_t$ for very small t is high. When the curve drops, this means that there have been more errors encountered; and the steeper the curve declines, the higher the number of errors that have occurred. On the other hand, when the curve grows, it can be understood as meaning that there have been fewer errors. The steeper the curve grows, the fewer the number of errors occurring at times close to (but smaller than) t.

$$recognition\,rate_t = 1 - \frac{|errors|_1^t}{t} \tag{6.11}$$

In Figures 6.12 and 6.13 we derive the $recognition\,rate_{phase}$ as defined in 6.12, where t_B and t_E are the times of the beginning and the end of each phase, i.e. each recognition rate is derived as the cumulative recognition rate for the whole phase.

$$recognition\,rate_{phase} = 1 - \frac{|errors|_{t_B}^{t_E}}{t_E - t_B} \tag{6.12}$$

Scenario 1

In Table 6.1 we show the complete comparison of error rates using only original data (no synthetic data), synthetic data with learning in blocks and synthetic data with the randomized buffer for a different number of automatically generated samples. For each of these options we additionally compare the results without and with EML. Thus, each table entry denotes the combined result for the method used for the handling of synthetic data (rows) and the number of automatically generated samples (columns: 1-50). Both of these combined entries (separated by '/') correspond to the error rates using the ARTIST model. We want to show the difference EML brings to the whole concept of ARTIST; thus, the first of the combined entries denotes the result without EML (i.e. using the original RLS), and the second entry denotes the results for ARTIST with EML. Then, the best result among all results for each column (i.e. the number of automatically generated samples) is chosen and is shown in the table in bold. For example, the entry for the second row, third column (0.362/**0.312**) denotes the combined result for the learning using 10 synthetic samples generated for each sample, using block learning, where the first entry is the result for ARTIST not using EML (0.362), and the second entry is the result for ARTIST using EML (0.312). We can note that for 10 synthetic samples, the best result is achieved by ARTIST using EML when the synthetic samples are added to the system in blocks. We can also note that for the first row, all the results are identical. This is because these results are derived using only original samples, and thus the number of synthetic samples has no influence on them.

Then in Figure 6.6 we can see the complete results for recognition rates using EML and the randomized buffer for a different number of automatically generated samples. From the results we can see that using synthetic data can lead to forgetting, as there are blocks of data of the same class. Especially at the beginning of the learning, all the other classes are vulnerable to the forgetting problem becasue they are poorly represented. Using EML helps to solve this problem; and in the combination with synthetic data, it outperforms using only original data as well as synthetic data without EML.

As we can note from these and further results, using only EML tends to have problems in the initial learning. This can be solved by using automatically generated data that increase the number of samples that EML is choosing from. Further, EML prevents the forgetting that automatically generated data blocks can cause. Both of these methods, in combination, lead to superior results compared to the use of only original data, only synthetic data or only EML with original data. We can also note, that in such settings and for the data used in this work, the best choice for the number of automatically generated data is ~25. We choose this number for the rest of our experiments.

In Figure 6.7 we compare the use of the randomized buffer using a selected number of automatically generated data, with and without EML to the recognition rate when using only original data for learning purposes.

Table 6.1 Exploring the error rates on the first 100 original samples compared with learning with original data vs. learning with synthetic data in blocks vs. learning with synthetic data using the randomized buffer, with emphasis on forgetting when not using EML (i.e. RLS) vs. using EML.

RLS / EML	1	5	10	15	20	25
original data	0.32/0.328	0.32/0.328	0.32/0.328	**0.32**/0.328	0.32/0.328	0.32/0.328
block learning	0.29/0.346	**0.3**/0.328	0.362/**0.312**	0.328/0.346	0.366/**0.282**	0.374/0.26
randomized buffer	**0.284**/0.378	**0.3**/0.356	0.354/0.326	0.344/0.33	0.374/0.29	0.354/**0.254**

RLS / EML	30	35	40	45	50
original data	0.32/0.328	0.32/0.328	0.32/0.328	0.32/0.328	0.32/0.328
block learning	0.33/**0.262**	0.364/0.31	0.362/0.35	0.374/**0.292**	0.37/0.272
randomized buffer	0.312/0.292	0.322/**0.306**	0.328/**0.318**	0.352/0.294	0.384/**0.26**

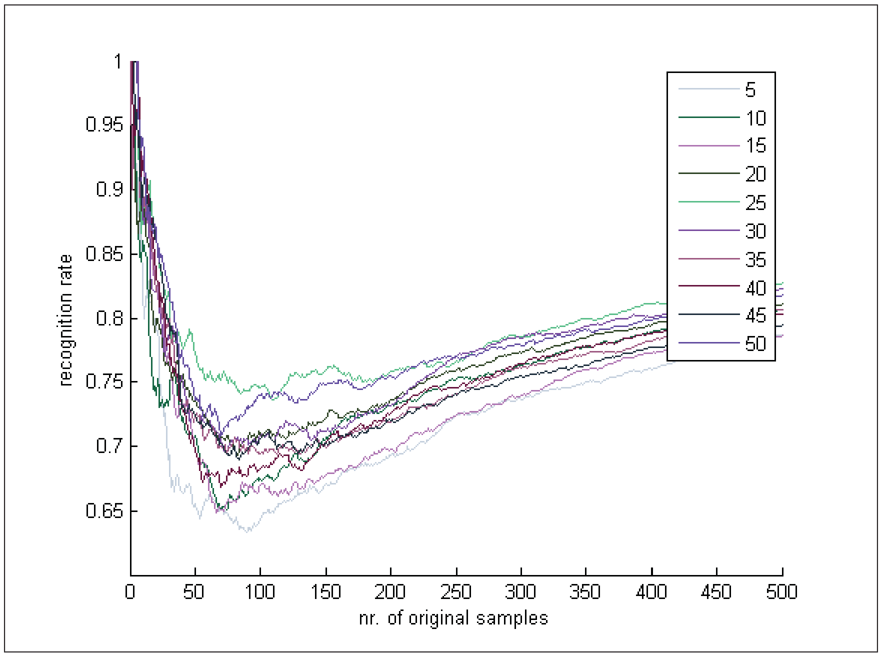

Figure 6.6 Recognition rates for randomized buffer with EML using a different number of synthetic samples.

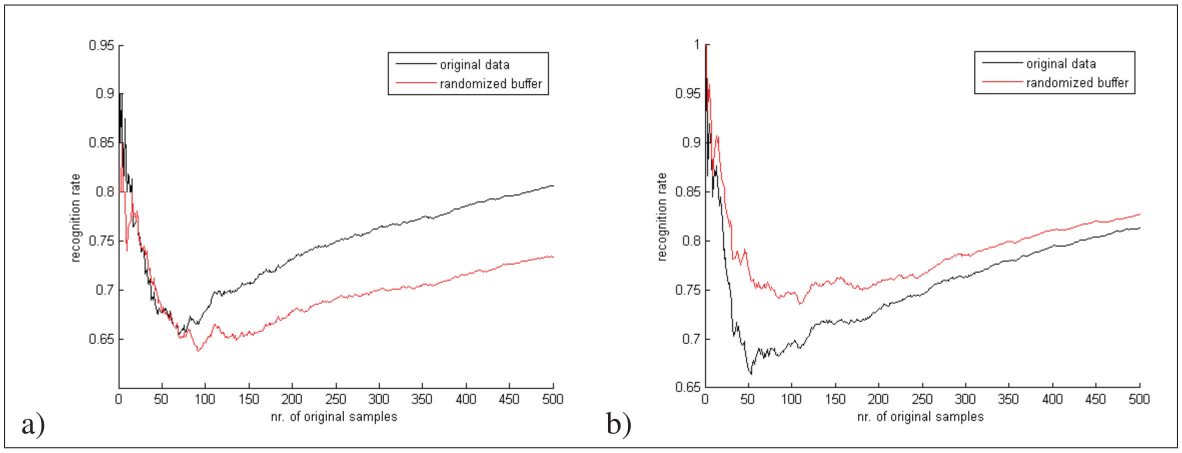

Figure 6.7 Recognition rate when using only original data compared to the one using 25 automatically generated samples for each original sample with the use of the randomized buffer. The comparison is done for a) learning without EML and b) learning with EML.

As well in the previous figure, we can note that using the combination of EML and automatically generated data helps to improve the recognition rate as well as prevent forgetting. We can also note that after a certain number of original samples, using the automatically produced data do not add any improvement. We investigate this amount further in scenario 2.

In Figure 6.8 we compare the recognition rates for using only original data to the rate when using automatically generated data learned in blocks (with the possible postponing of the system) and using automatically generated data with the randomized buffer (with respect to real-world use). All these models use our EML. From the Figure we can note that, again, using the combination of synthetic data and EML outperforms other settings. We can see for all cases that after a certain number of samples, the performance does not vary as much as for the initial learning, i.e. for learning at the introduction of classes when the lack of data is most significant.

On Figure 6.9 we can see the comparison of our model using EML and randomized buffer to a number of baseline models that have been adapted to perform on-line learning. As we can see, ARTIST highly outperforms all these models despite the fact, that they do need to initialize all the classes ahead.

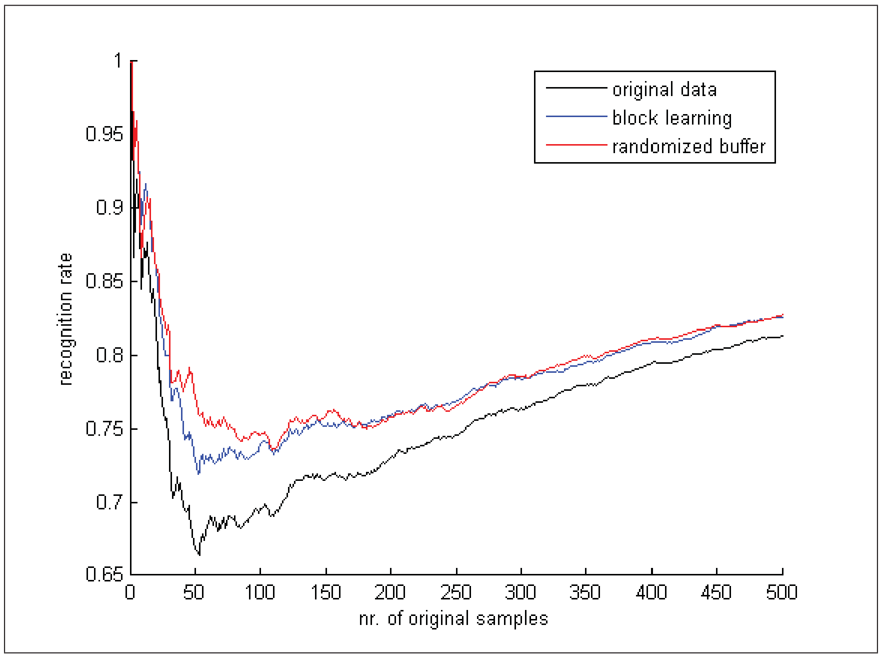

Figure 6.8 Comparison of the recognition rate using original data, synthetic data with block learning and synthetic data with the randomized buffer, using 25 synthetic samples generated for each real sample.

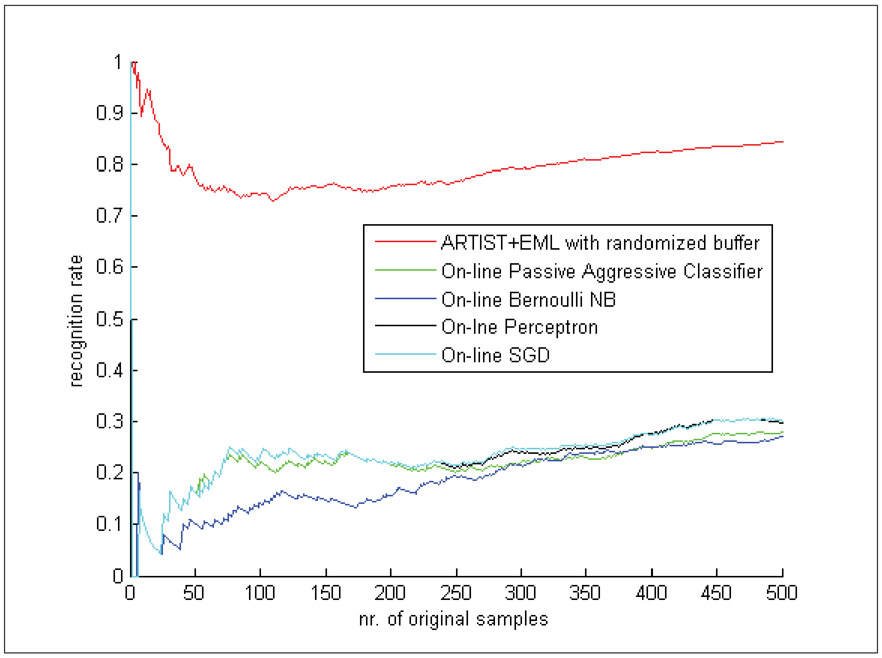

Figure 6.9 Comparison of the recognition rate using our model (ARTIST+EML with randomized buffer) to baselines (from sklearn library).

Scenario 2

In this scenario we investigate the amount of original samples for which synthetic samples are generated. Figure 6.10a) shows the results without our EML. For this setting, the amount of 250-300 original samples for 17 classes (~15-18 samples for each class) is sufficient to achieve a recognition rate (accumulated by the latter samples only) of ~82% as compared to 80% and lower for the smaller stop criterion. Figure 6.10b) shows the results using our EML. We can notice that by using this strategy, top results of ~87% are achieved by the highest amounts of synthetic samples. Thus, it seems that when using EML we need to prolong the generation or synthetic samples, which is only natural as there are fewer samples being used for the learning, i.e. only those that win the competition for the learning. However, the differences among these results are only slight (less than 2%) and we can agree with previous results that the number ~300 is appropriate for the amount of original samples for which synthetic samples are to be generated. This number not only results into top results, but is also not the highest one.

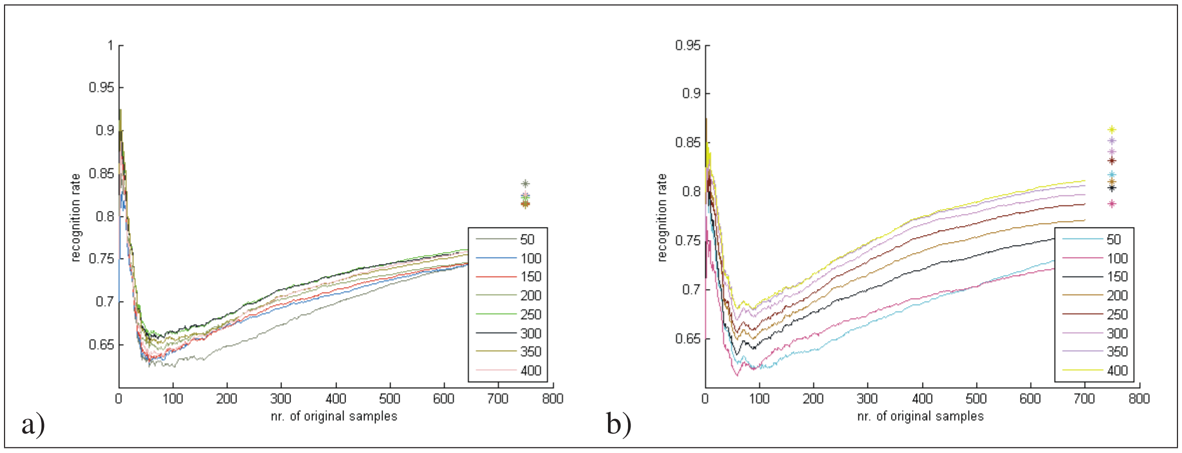

Figure 6.10 Comparison of different times to stop the generation of synthetic samples, i.e. different number of original samples for which synthetic samples are generated: a) without EML and b) with EML. At the very right an error on last 100 samples is visualized by stars to show the converged recognition rate of the system.

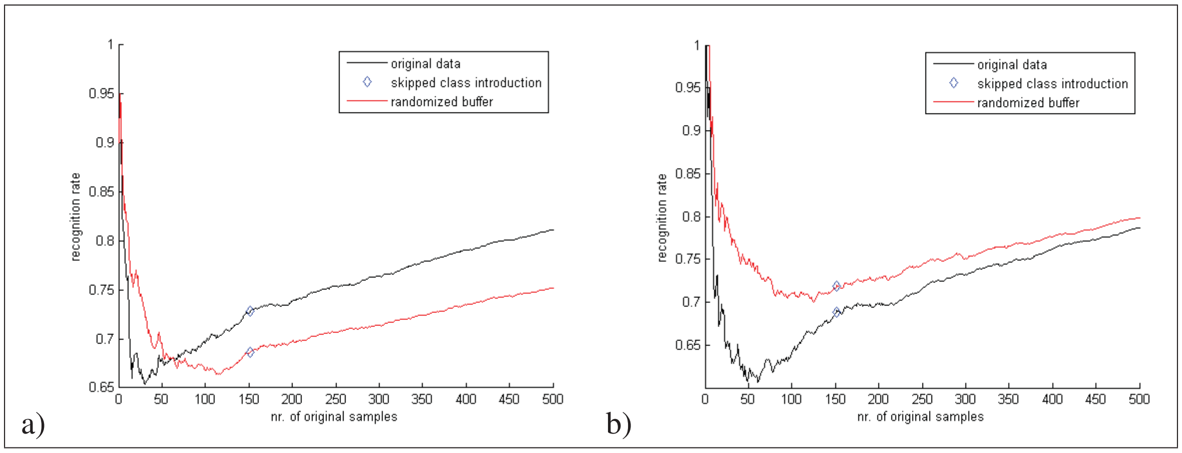

Figure 6.11 Comparison of recognition rates with one skipped class and after its introduction to the system. We compare the rates using only original data to those using the randomized buffer with synthetic samples when learning a) without EML and b) with EML. The introduction of the skipped class occurs at the 151st original sample.

Scenario 3

As with the previous scenarios, we allow the system to add classes on the fly except for one randomly selected class. After this class is added back to the data stream, we can compare the recognition rates for using only original data and using synthetic data, as shown in Figure 6.11. Unlike the previous scenarios, where the focus was on initial learning, here the focus is on learning on the fly.

In the Figure 6.11 a slight narrowing of the learning curve can be noted after the introduction of a new class, especially in the case without synthetic data. This is caused by a higher increase in erroneous results, i.e. more occurrences of miss-classifications. Thus, using synthetic samples indeed helps to improve the recognition rates in such scenarios.

Scenario 4

In this scenario we focus on phase 3, which is denoting the forgetting of the unused class that was unused in phase 2 and later used again in phase 4. From Figure 6.12a) we can note that using synthetic samples decreases the recognition rate for the unused class and magnifies the forgetting rate. However, by using EML, as shown in Figure 6.12b), we are able to significantly improve this. We can see that using synthetic data does not affect the forgetting significantly anymore, while at the same time improves initial learning.

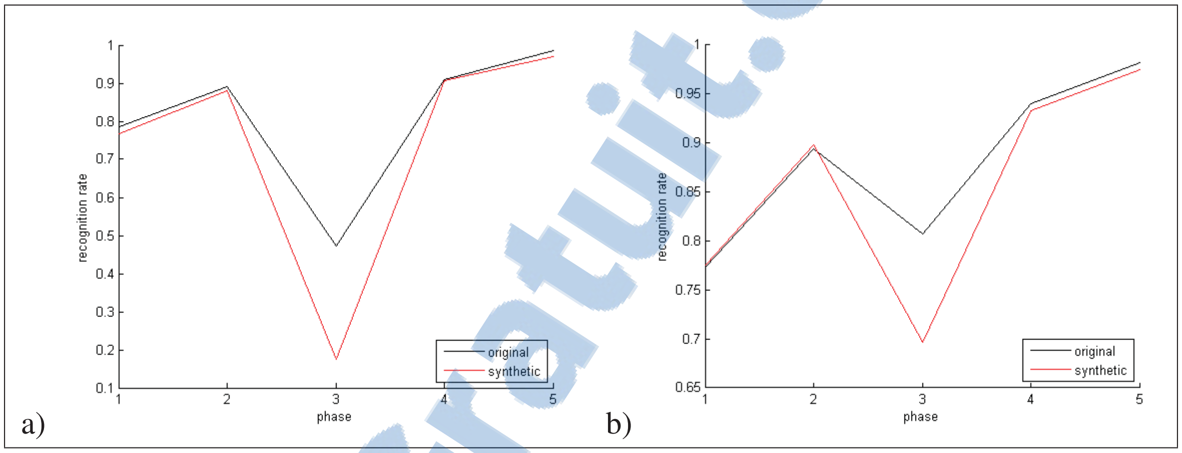

Figure 6.12 Recognition rate for the forgetting scenario a) without EML, b) with EML. Phase 1 learns and recognizes samples from all classes, Phase 2 learns and recognizes samples from all classes except one, phase 3 does not learn but recognizes the unused class from phase 2, and phase 4+ learn and recognize samples from all classes.

In Figure 6.13 we can note that ARTIST is less prone to the forgetting of unused classes (phase 3) as well as in comparison to other evolving models, ETS Angelov *et al.* (2004) and ETS+ Almaksour *et al.* (2010). This underlines the previous results and shows that even with increased number of samples per class, the system can be immune to forgetting of unused classes. This is very important for the sake of initial learning and the need of synthetic samples in situations where such samples are not easy to come by.

It should be stated that all our experiments were performed using a specific tuning for noise that was added to the synthetic data. We believe that this noise plays a significant role in the whole process and needs to be further explored.

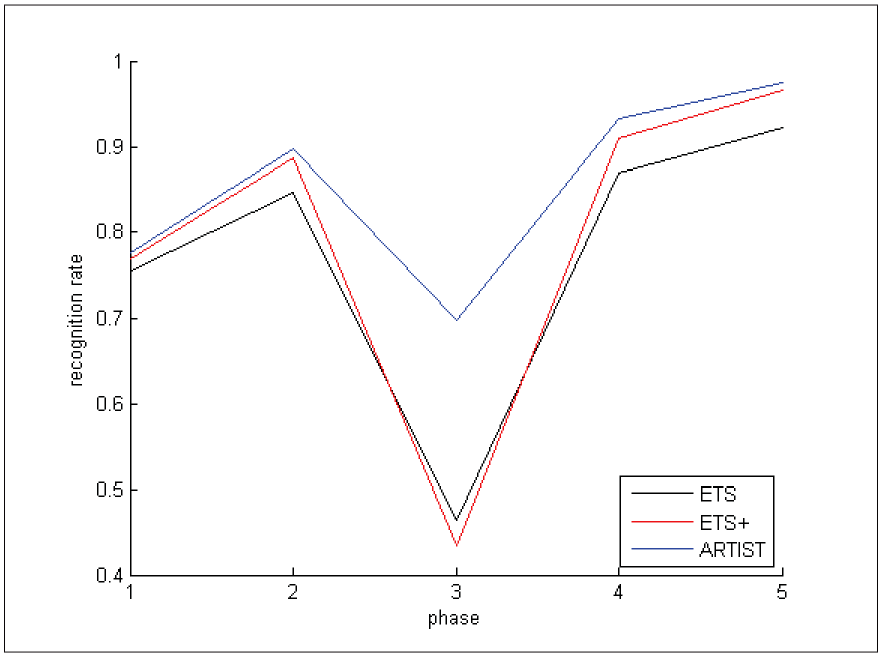

Figure 6.13 Comparison of the recognition rates for the forgetting scenario using different models. Phase 1 learns and recognizes samples from all classes, phase 2 learns and recognizes samples from all classes except one, phase 3 does not learn but recognizes the unused class from phase 2, and phase 4+ learn and recognize samples from all classes.

6.9 Conclusion and Discussion

In this paper we focus on solving the problem of initial learning and learning on the fly. One option as to how to increase performance when the number of samples is very low and often close to 1, is to generate synthetic data. However, using blocks of data can lead to forgetting of the unused classes, especially in the on-line learning environment where, for learning purposes, not all data are available at once. Here we show that by using Elastic Memory Learning we can avoid the forgetting of classes that are not being multiplied by synthetic data. We also show that the generation of synthetic data is not required for the whole learning process and is needed only for the initial learning period and for the introduction of a new class on the fly.

As we state in the section 6.2, we faced several problems concerning the on-line learning of handwritten gesture commands. We successfully addressed these problems by

a. applying the Sigma-lognormal model for the reconstruction of real handwritten specimens and then applying noise to the lognormal strokes thus obtained to generate duplicated specimens;

b. showing in the results the advantages of using synthetic data for learning from scratch (initial learning) and learning on the fly (randomly adding new classes);

c. proposing a randomized buffer to fit the behavior of real-time on-line learning and showing its positive influence on the performance of the system;

d. applying our proposed Elastic Memory Learning to solve the forgetting problem in on-line learning. In the results we show the significant improvement that EML brings to the whole system by making use of synthetic data, a very powerful tool for the improvement of the performance of on-line learning based systems when there are only a few real examples for the learning process.

Our experiments show promising results for the solution of the most important and crucial problems in on-line learning in the application for recognition of handwritten gesture commands. However we still see a room for improvements, especially in terms of feature extraction. We believe that proper handling if 2D data along with synthetic data will result in better performance.

Thus in our future work we will explore various feature extraction techniques for digital handwritten strokes as well as other applications, such as the generation of synthetic data for other handwriting applications, such as letter or digit recognition. At the same time we will explore the different types of noise applied to the vector strokes of the Sigma-lognormal model. We believe that this noise can influence EML and increase not only initial learning but also create greater immunity to forgetting.

Acknowledgments

The authors thank NSERC RGPIN 138344 & 2014-04649 and SSHRC 412-2010-1007 of Canada for their financial support.

CHAPTER 7

GENERAL DISCUSSION

The main objective of this thesis was to find a model capable of online incremental learning with the emphasis on avoiding the use of hyper-parameters. The main motivation of such restrictions was user-friendly online handwritten gesture recognition. Thus besides the online incremental restriction we had to focus on real-time processing, learning from scratch and no relying on the knowledge of the future data. We wanted the user to be able to define his/hers own gestures at any point of the usage of the system. However, besides this application we have shown the strengths of our solution also on other applications in a form of various datasets. Further in this section we discuss our contributions following the scheme of our objectives and articles.

7.1 Incremental Similarity

The main objective of the thesis was to find a system based on online learning with the emphasis on learning from scratch, learning on the fly and user-friendliness. The Incremental Similarity (IS) brings a trade-off between the computational cost and accuracy. The main reason for searching for such trade-off is the real-time setting of many online and incremental learning systems. Here, the computational cost is one of the crucial factors for the decision about the learning and recognition model. In our work we have focused on real-time online learning with starting from scratch. Our method offers a simple updating formula that does not rely on covariance matrix and at the same time is able to deliver high accuracy. Rather than the distance from the mean and corresponding covariances, it describes the distance from all points in a specific structure, i.e. a cluster. Using this novel technique we were able to avoid the covariance matrix, but still take into account the covariances in the data, while using simple updates. However, using simpler description of the data slightly lowered the accuracy in some cases compared to using full covariance matrix. Nevertheless, compared to both, using only variance vs covariance matrix, the accuracy and the processing time has been brought to an iinteresting trade-off.

7.2 Elastic Memory Learning

To tackle the problem of forgetting of unused classes we have developed an Elastic Memory Learning. We have applied our solution on Recursive Least Squares, as this is the optimization technique we have chosen n all of our approaches. During our experiments we have notices that in cases that some class is not present in the data stream for a longer period of time, it becomes under-sampled in the local model, and thus it is not easy to retrieve. It happens, that in such occurrences, a decision boundary is shifted to recognize the negative class well, but not the positive class. This results in a small recall of the positive class. Elastic Memory Learning is based on a competitive choice of the learning, i.e. the parameters are learned only when necessary. In other words, we under-sample the negative class and keep only the important samples that bring the confusion to the system.

Such competitiveness in updating of the consequent parameters leads to better results even if some of the classes is not used for a longer period of time, preventing the model from forgetting it. This ability is very important especially when the distribution of classes of incoming examples is not necessarily uniform. However, we have also noticed, that such under-sampling can contribute to a lowered performance, especially at the beginning of the learning. Since the learning from scratch is a difficult task by itself and there is a need for more data, by under-sampling we feed even less data to the model. We also need to state, that the lowered performance is very slight.

7.3 Self-Organized incremental model

In our work we have developed a few models based on Takag-Sugeno neuro-fuzzy model. All of these are capable on online incremental learning. However, our aim was to find a model that will not need to rely on the knowledge of the data that should not be known ahead. Thus, we have adapted on of our models, ARTIST to a self-adapting version. Here, all the structures, i.e. the rules, are organized automatically in order to follow the adaptive nature of all the parameters of ARTIST.

Both of these models are besides neuro-fuzzy modeling based on ART-2A neural network. This network has two hyper-parameters, vigilance and learning rate. These are used to decide about the creation of a new cluster, n our case fuzzy rule, for the former, and the 'speed' of adapting of the parameters used to derive the similarity to these clusters, for the latter. We have developed a formulas to adapt these parameters based on the responses of the model to the data and its predictions.

Since these parameters responsible for the creation of new rules, especially the vigilance parameter, it was necessary to find a way for the rules to re-organize. Thus we have developed methods to allow the rules to merge, split and discard, based on the changing parameters and the needs of the model to adapt.

All these solutions have contributed to the development of the self-organized model that does not rely on the information of the unknown data and can adapt itself based on the dynamic changes in the environment. However, all this has also contributes to the complexity of the model and thus also the processing time.

7.4 The problem of missing data

As we have stated in the objectives of this thesis, in online learning the data are introduced in a sequence. Thus, at the beginning of the learning the model lacks the amount of data which results into a higher variance and over-fitting. This is caused by that the model is learned to recognize specific examples rather than their expected and thus more general value. To be able to increase the generalization power of the model at the beginning of the learning process as well as after introducing new class, we generate new synthetic data so that the parameters derived for each class are more representative and include some variance within the data.

In our work we have developed a framework to incorporate a generation of synthetic data to the online process such that it does not create blocks of data of the same kind. At the same time we have estimated the necessary amount of the synthetic samples to be generated and the time span of their generation.

Using the synthetic data we were able to avoid the problem of generalization at the beginning of the process, even more when in combination with our Elastic Memory Learning, that has helped to handle the amount of the synthetic data to be used. However this solution is partially restricted to the handwritten gesture recognition, since the synthetic data generation is used solely for the handwriting.

CONCLUSION AND RECOMMENDATIONS

In this work we have presented original contributions to the state of the art in the field of online learning. We were able to tackle all our posed problems within online learning as well as address all the objectives that has raised from these problems. Our work has led to a number of publications in international journals and conferences that have shown the usefulness of our work.

To find the similarity metric that deals with the accuracy vs processing time problem is very necessary for online learning, especially for real-time processing. Many approaches choose either high performance in the sense of accuracy, precision, recall, etc. or the low computational cost. Usually, the higher performance results into higher complexity and oppositely lower complexity results into lower performance. We were able to propose and develop Incremental Similarity measure that we have applied to Neuro-Fuzzy models and shown that while achieving lower complexity it retains the high performance. We have compared this metric on a number of models and various data sets showing its applicability to various aspects of the machine learning domain. While we see the success of our work and that we have successfully accomplished our objective, we still see a room for improvement. We believe that in the future, the science will move forward with the real-time processing either by introducing new methods to handle this problem or by the computational capacity of machines.

The forgetting of unused classes is an important issue that has not been vastly tackled in the state of the art. However it is a very natural occurrence and thus we have posed the objective to tackle this problem. As we could see in previous chapters, we were able to successfully develop a solution for Recursive Least Squares. We have applied our solution for various models and evaluated it on various machine learning benchmarks, showing its immunity towards the forgetting of unused classes. In many cases, we were able to almost completely eliminate the forgetting aspect of the learning process. On the other hand, we can see a slight drop in the performance when we continue the learning process using all the classes. We explain this by the fact, that in our solution we do not learn from all the examples, but competitively choose the proper ones, whereas in the original RLS all the data is used. Nevertheless, this drop is not significant, and we can state that our solution achieves superior results.

In our work we were able to develop a self-organized model based on neuro-fuzzy modeling. This model is capable of adding new rules to the ensemble and adding new classes to the complete system. Moreover, as a part of self-organized mechanism it is capable of determining the necessity for merging, splitting and discarding of the rules. This is a consequence of the self-learning parameters that handle the rules and their generation. Since these are changing in time, it is necessary to allow the model to change its organization, such that it is at its best with the knowledge of all the data until that specific time. In the results we have shown that our proposed self-organized model outperforms the model using free parameters that have been fixed by cross-validation. Moreover, we have compared this model to a number of online and offline state of the art models. We can see that our model either outperforms or sustains comparably high performance on all listed data sets. We need to note, that the vast majority of the listed models need some initialization and do not start from scratch, while at the same time require cross-validation. Thus, the results of our model prove that by self-organization the machine is capable to adapt dynamically to the current state of the environment, which as a result leads to superior results. This is due to the fact that when using fixed parameters and structure, this is not the best setup for all the dynamical changes in the learning process, but only in general. Thus, in online learning we find it preferable to use dynamically structured models in order to cope with dynamically changing environment.

To be able to cope with the high variance at the beginning that is caused by the lack of data, we have proposed the generation of synthetic data. We have developed a framework for real-time online processing with randomized buffer that leads to a significantly lower variance during the initial stages of the data stream processing. However we have shown, that using synthetic data can magnify the forgetting factor. Thus, we have applied our solution for this problem and shown, that the combination of synthetic data with the competitive nature of EML is the proper solution for the high variance and the low performance. In the results we have shown that we can get close to the elimination of the lowered performance while avoiding the unwanted forgetting of the other classes. Moreover, we have explored the length of the generation process as well as the amount of synthetic examples to be generated, which has led to improved results for this application.

We have applied our solutions to various tasks and shown their usefulness. The problems that we have tackled are often ignored in online learning field, where the highest emphasis is given to the sequential nature of the data. While this is an important aspect of online learning, by ignoring all the other problems that we have raised, these models cannot properly cope with the real-time setup, where the data are processed in their natural order and cannot be shuffled.

In conclusion, we have successfully approached the objectives of this thesis and incremented the state of the art techniques for online learning. Online learning is very interesting field within machine learning and its importance rises with the rise of the use of the internet, where the data are created in a big amounts and the need of dynamic models rises as well. There is a number of various applications that can benefit from online learning and the increase in the variety of online models will help to tackle various problems specific for these applications. In this work some problems of online learning have been listed, however with the increase in the applications, new problems will need to be tackled. Online learning similarly to its own nature, is a dynamic field that need to adapt to all the new situations in the state of the real world. We believe that our research will help scientists in the future to develop solutions that will help the society to go beyond all that we have ever thought.

Future work

Our work has mainly focused on the self-organized aspect of online learning applied to the neuro-fuzzy models. We have also proposed the framework for synthetic generation of data that has been applied on our online incremental model ARTIST. We believe that such framework can be an interesting addition to the self-organized models. However, the kinematic model is directly connected to human movements, and thus not applicable for other domains. To be able to generate synthetic data is of a great interest especially in the case of learning from scratch or cold-start learning.

One of the flaws that we see in our solution is the approximation needed for the update of some parameters. This is caused by the choice of the optimization, RLS, and the application to the self-organized model. We believe that a better choice of the optimization technique can lead to a superior results. This choice can be also inspired by a change in the consequent learning, that may lead to a simplified parameter learning.

To contribute to the computational cost of the system, an interesting proposal is to add a feature selection mechanism to the model. One way is to choose the best features from the complete set of features, where these best features are the ones that contribute to the right decision the most.

The feature extraction is an open problem, where the pool of possible features from handwritten data can always be increased. The higher choice of features can also complement our previous proposal, the feature selection.

In our work we have used a feature vector representation of the input data. Especially in the application of handwritten gestures, there are other interesting approaches of representing the data, such as markov models or neural networks. Here, one gesture may be represented by one model, where these models are added incrementally into an ensemble model. The models themselves need to be able to update their parameters in an incremental manner as well, so that they can change with the increasing number of data.

Another interesting task is to apply our solution along with proper data representation to a general gesture recognition, where on the input we have gestures produced by a movement.

There are still and always will be many more challenges in the area of machine learning and of online learning specifically, and we hope that our research has contributed to resolving them.

Summary of contributions

In this work we have shown our solutions for the problems posed in the Introduction and followed the objectives resulting from them. As such, we were able to achieve number of contributions to the state of the art. We list these contributions in the following:

- Incremental similarity for tackling the accuracy vs processing time problem,
- Elastic Memory Learning to tackle the forgetting of unused classes that in online learning setup are not distributed uniformly in the data stream and thus can become sparse,
- Self-Organized model to enable learning from scratch, learning on the fly and online learning in general, where the structures of the model are organized in an automated way along with the learning of all the parameters necessary for high performance,
- Framework for generating the synthetic data for online learning to tackle the problem of high variance at the beginning of the learning process and every time a new class is added to the system.

Articles in peer reviewed journals

[1] M. Režnáková, L. Tencer, R. Plamondon, M. Cheriet, Forgetting of unused classes in missing data environment using artificially generated data. Application to on-line handwritten stroke recognition (under review), Pattern Recognition.

[2] M. Režnáková, L. Tencer, M. Cheriet, Elastic Memory Learning for Fuzzy Inference Models (under review), Applied Soft Computing.

[3] M. Režnáková, L. Tencer, M. Cheriet, Incremental Similarity for real-time on-line incremental learning systems, Pattern Recognition Letters 74 (2016), pp. 61-37.

[4] M. Režnáková, L. Tencer, M. Cheriet, SO-ARTIST: Self-Organized ART- 2A inspired clustering for online Takagi Sugeno fuzzy models, Applied Soft Computing 31 (2015), pp. 132-152.

Articles in peer reviewed conference proceedings

[1] M. Režnáková, L. Tencer, R. Plamondon, M. Cheriet, The generation of synthetic handwritten data for improving on-line learning, in: 17th Biennial Conference of the International Graphonomics Society, 2015.

[2] M. Režnáková, L. Tencer, M. Cheriet, ARTIST: ART-2A driven Generation of Fuzzy Rules for Online Handwritten Gesture Recognition, in: Document Analysis and Recognition, 2013. ICDAR '13. 12th International Conference on, 2013.

[3] M. Režnáková, L. Tencer, M. Cheriet, Online handwritten gesture recognition based on Takagi-Sugeno fuzzy models, in: Information Science, Signal Processing and their Applications (ISSPA), 2012 11th International Conference on, 2012, pp. 1247-1252.

Conference Organization

IEEE 11th International Conference on Information Sciences, Signal Processing and their Applications (ISSPA '12), volunteer.

Paper Reviewing

i ASC 2015-2016

ii IJCNN 2016

iii JMLC 2015

Research at Pandora Media

Currently I am a part of Pandora Media company that produces a radio application daily accessed by over 80 millions users. My work is focused on machine learning towards the discovery of new artists and genres. My main focus is the use of deep learning techniques.

BIBLIOGRAPHY

Ahn, K. K., Pham, H. & Anh, H. (2010). Inverse double narx fuzzy modeling for system identification. *Ieee/asme transactions on mechatronics*, 15(1), 136–148. doi: 10.1109/TMECH.2009.2020737.

Almaksour, A., Anquetil, E., Quiniou, S. & Cheriet, M. (2010). Evolving fuzzy classifiers: Application to incremental learning of handwritten gesture recognition systems. *2010 20th international conference on pattern recognition*, 4056–4059.

Almaksour, A., Anquetil, E., Plamondon, R. & O'Reilly, C. (2011). Synthetic handwritten gesture generation using sigma-lognormal model for evolving handwriting classifiers. *15th biennial conference of the international graphonomics society*.

Anagnostopoulos, G. C. et al. (2000). Hypersphere art and artmap for unsupervised and supervised, incremental learning. *Neural networks, 2000. ijcnn 2000, proceedings of the ieee-inns-enns international joint conference on*, 6, 59–64.

Angelov, P., Lughofer, E. & Zhou, X. (2008). Evolving fuzzy classifiers using different model architectures. *Fuzzy sets and systems*, 159(23), 3160–3182. doi: http://dx.doi.org/10.1016/j.fss.2008.06.019.

Angelov, P. P. et al. (2004). An approach to online identification of takagi-sugeno fuzzy models. *Ieee transactions on systems, man, and cybernetics. part b, cybernetics : a publication of the ieee systems, man, and cybernetics society*, 34(1), 484–98.

Angelov, P. (2008). Evolving fuzzy-rule-based classifiers from data streams. *Ieee transactions on fuzzy systems*, 16(6), 1462–1475. doi: 10.1109/TFUZZ.2008.925904.

Angelov, Plamen; Filev, Dimitar P.; Kasabov, N. (2010). *Evolving Intelligent Systems: Methodology and Applications*. New York: John Wiley & Sons.

Babuska, R. (1998). *Fuzzy Modeling for Control* (ed. 1st). Norwell, MA, USA: Kluwer Academic Publishers.

Babuška, R. et al. (2003). Neuro-fuzzy methods for nonlinear system identification. *Annual reviews in control*, 27(1), 73–85.

Beringer, J. et al. (2006). Online clustering of parallel data streams. *Data & knowledge engineering*, 58(2), 180–204.

Bifet, A., Holmes, G., Kirkby, R. & Pfahringer, B. (2010). MOA: Massive Online Analysis. *Journal of machine learning research*, 11, 1601–1604.

Bordes, A. (2005). The Huller : A Simple and Efficient Online SVM. *Machine learning: Ecml 2005 16th european conference on machine learning, porto, portugal, october 3-7, 2005. proceedings*, pp. 505–512.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010* (pp. 177–186). Springer.

Breiman, L. E. O. (2001). Random Forests. *Machine learning*, 45(1), 5–32.

Caridakis, G., Karpouzis, K., Drosopoulos, A. & Kollias, S. (2010). SOMM: Self organizing Markov map for gesture recognition. *Pattern recognition letters*, 31(1), 52–59. doi: 10.1016/j.patrec.2009.09.009.

Carozza, M. et al. (2000). Towards an incremental SVM for regression. *Neural networks, 2000. ijcnn 2000, proceedings of the ieee-inns-enns international joint conference on*, pp. 405–410.

Carpenter, G., Grossberg, S. & Rosen, D. (1991a). Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural networks*, 4(6), 759–771.

Carpenter, G. et al. (2010). *Adaptive Resonance Theory*.

Carpenter, G. A., Stephen, G. & Rosen, D. B. (1991b). ART 2-A: An adaptive resonance algorithm for rapid category learning and recognition. *Neural networks*, 4(4), 493–504.

Carpenter, G. A. et al. (2017). Adaptive resonance theory. *Springer us*.

Carse, B., Fogarty, T. C. & Munro, A. (1996). Evolving fuzzy rule based controllers using genetic algorithms. *Fuzzy sets and systems*, 80(3), 273–293. doi: 10.1016/0165-0114(95)00196-4.

Cernuda, C., Lughofer, E., Suppan, L., Röder, T., Schmuck, R., Hintenaus, P., Märzinger, W. & Kasberger, J. (2012). Evolving chemometric models for predicting dynamic process parameters in viscose production. *Analytica chimica acta*, 725(0), 22–38. doi: http://dx.doi.org/10.1016/j.aca.2012.03.012.

Chapelle, O., Haffner, P. & Vapnik, V. N. (1999). Support vector machines for histogram-based image classification. *Ieee transactions on neural networks*, 10(5), 1055–1064.

Cheng, J., Liu, H., Wang, F., Li, H. & Zhu, C. (2016). Silhouette analysis for human action recognition based on supervised temporal t-sne and incremental learning. *Ieee transactions on image processing*, 24, 3203 - 3217.

Cordon, O., Gomide, F., Herrera, F., Hoffmann, F. & Magdalena, L. (2004). Ten years of genetic fuzzy systems: current framework and new trends. *Fuzzy sets and systems*, 141(1), 5–31. doi: 10.1109/NAFIPS.2001.943725.

Cordón, Oscar; Herrera, Francisco; Hoffmann, F.; Magdalena, L. (2001). *Genetic Fuzzy Systems*. World Scientific.

Deng, D. et al. (2003). On-line pattern analysis by evolving self-organizing maps. *Neurocomputing*, 51(0), 87–103.

Diehl, C. P. et al. (2003). SVM incremental learning, adaptation and optimization. *Neural networks, 2003. proceedings of the international joint conference on*, 4, 2685–2690 vol.4.

Ditzler, G., Muhlbaier, M. D. & Polikar, R. (2010). Incremental Learning of New Classes in Unbalanced Datasets: Learn + + .UDNC. In *Multiple Classifier Systems* (pp. 33–42).

Djioua, M. et al. (2009). Studying the variability of handwriting patterns using the Kinematic Theory. *Human movement science*, 28(5), 588–601.

Forster, K., Monteleone, S., Calatroni, A., Roggen, D. & Troster, G. (2010). Incremental kNN Classifier Exploiting Correct-Error Teacher for Activity Recognition. *2010 ninth international conference on machine learning and applications*, pp. 445–450. doi: 10.1109/ICMLA.2010.72.

Frank, T., Kraiss, K. F. & Kuhlen, T. (1998). Comparative analysis of fuzzy ART and ART-2A network clustering performance. *Ieee transactions on neural networks / a publication of the ieee neural networks council*, 9(3), 544–59. doi: 10.1109/72.668896.

Freund, Y. et al. (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. *European conference on computational learning theory*, pp. 23–37.

Gail A. Carpenter, Stephen Grossberga, J. H. R. (1991). ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural networks*, 4(5), 565–588.

Gama, J. a. (2010). *Knowledge Discovery from Data Streams*. Chapman and Hall/CRC. doi: 10.1201/EBK1439826119.

Gangadhar, G., Joseph, D. & Chakravarthy, V. (2007). An oscillatory neuromotor model of handwriting generation. *International journal of document analysis and recognition (ijdar)*, 10(2), 69–84. doi: 10.1007/s10032-007-0046-0.

Glotin, H., Warnier, P., Dandurand, F., Dufau, S., Lété, B., Touzet, C., Ziegler, J. C. & Grainger, J. (2009). An Adaptive Resonance Theory account of the implicit learning of orthographic word forms. *Journal of physiology, paris*, 104(1-2), 19–26. doi: 10.1016/j.jphysparis.2009.11.003.

Gomez, J. et al. (2002). Evolving Fuzzy Classifiers for Intrusion Detection. *Proceedings of the 2002 ieee workshop on information assurance.*, (June 2001).

Grabner, H. et al. (2006). On-line Boosting and Vision. *Computer vision and pattern recognition, 2006 ieee computer society conference on*, 1, 260–267.

Gua, B., Shenge, V. S., Wangf, Z., Hog, D., Osmanh, S. & Li, S. (2015). Incremental learning for v-support vector regression. *Neural networks*, 67, 140-150.

H., W. D. et al. (1997). No free lunch theorems for optimization. *Ieee transactions on evolutionary computation 1.1*, 67-82.

Hall, P. M., Marshall, D. & Martin, R. R. (1998). Incremental eigenanalysis for classification. *British machine vision conference i*, (May), 286–295.

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6), 417.

Hsu, C.-C. et al. (2008). Incremental clustering of mixed data based on distance hierarchy. *Expert systems with applications*, 35(3), 1177–1185.

Kasabov, N. (2001). Evolving fuzzy neural networks for supervised/unsupervised online knowledge-based learning. *Systems, man, and cybernetics, part b: Cybernetics, ieee transactions on*, 31(6), 902–918. doi: 10.1109/3477.969494.

Kasabov, N., Zhang, D. & Fang, P. S. (2005). Incremental Learning in Autonomous Systems : Evolving Connectionist Systems for On-line Image and Speech Recognition. *Proceedings of ieee workshop on advanced robotics and its social impacts*, pp. 120–125.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the ieee*, 78(9), 1464–1480.

LeCun, Y. et al. (1998). The MNIST database of handwritten digits.

Leistner, C. et al. (2009). On-line Random Forests. *Computer vision workshops (iccv workshops), 2009 ieee 12th international conference on*, pp. 1393 – 1400.

Lemos, A., Caminhas, W. & Gomide, F. (2013). Adaptive fault detection and diagnosis using an evolving fuzzy classifier. *Information sciences*, 220(0), 64–85. doi: http://dx.doi.org/10.1016/j.ins.2011.08.030.

Li, Y. (2004). On incremental and robust subspace learning. *Pattern recognition*, 37(7), 1509–1518. doi: 10.1016/j.patcog.2003.11.010.

Lin, X., Soergel, D. & Marchionini, G. (1991). A self-organizing semantic map for information retrieval. *Sigir '91 proceedings of the 14th annual international acm sigir conference on research and development in information retrieval*, pp. 262–269.

Liu, C.-L. (2007). Normalization-cooperated gradient feature extraction for handwritten character recognition. *Ieee transactions on pattern analysis and machine intelligence*, 29(8), 1465–1469.

Ljung, L. (1999). *System Identification: Theory for the User*. Upper Saddle River, New Jersey: Prenntice Hall PTR, Prentic Hall Inc.

Lughofer, E. (2011). *Evolving Fuzzy Systems: Methodologies, Advanced Concepts and Applications*. Springer, Berlin Heidelberg.

Lughofer, E. et al. (2013a). Reliable All-Pairs Evolving Fuzzy Classifiers. *Fuzzy systems, ieee transactions on*, 21(4), 625–641.

Lughofer, E. (2008a). FLEXFIS: a robust incremental learning approach for evolving Takagi-Sugeno fuzzy models. *Fuzzy systems, ieee transactions on*, 16(6), 1393–1410.

Lughofer, E. (2008b). Extensions of vector quantization for incremental clustering. *Pattern recognition*, 41(3), 995–1011. doi: 10.1016/j.patcog.2007.07.019.

Lughofer, E. (2012). A Dynamic Split-and-Merge Approach for Evolving Cluster Models. *Evolving systems*, 3(3), 135–151. doi: 10.1007/s12530-012-9046-5.

Lughofer, E. (2013). On-line assurance of interpretability criteria in evolving fuzzy systems – achievements, new concepts and open issues. *Information sciences*, 251, 22-46.

Lughofer, E., Bouchot, J.-L. & Shaker, A. (2011). On-line Elimination of Local Redundancies in Evolving Fuzzy Systems. *Evolving systems*, 2(3), 165–187. doi: 10.1007/s12530-011-9032-3.

Lughofer, E., Cernuda, C., Kindermann, S. & Pratama, M. (2015). Generalized smart evolving fuzzy systems. *Evolving systems*, 6, 269–292.

Lughofer, E. et al. (2013b). Reliable all-pairs evolving fuzzy classifiers. *Ieee transactions on fuzzy systems*, 21, 625 - 641.

Lühr, S. et al. (2009). Incremental clustering of dynamic data streams using connectivity based representative points. *Data & knowledge engineering*, 68(1), 1–27.

MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of the fifth berkeley symposium on mathematical statistics and probability*, 1(14), 281–297.

Mamdani, E. et al. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. *International journal of man-machine studies*, 7(1), 1–13.

Martin-Albo, D., Plamondon, R. & Vidal, E. (2014). Training of on-line handwriting text recognizers with synthetic text generated using the kinematic theory of rapid human movements. *Frontiers in handwriting recognition (icfhr), 2014 14th international conference on*, pp. 543–548.

Moamar-Mouchaweh, Sayed; Lughofer, E. (2012). *Learning in Non-Stationary Environments: Methods and Applications*. New York: Springer.

Neilson, P. D. et al. (2005). An overview of adaptive model theory: solving the problems of redundancy, resources, and nonlinear interactions in human movement control. *Journal of neural engineering*, 2(3), S279.

O'Reilly, C. et al. (2009). Development of a Sigma–Lognormal representation for on-line signatures. *Pattern recognition*, 42(12), 3324–3337.

Oza, N. C. (2005). Online bagging and boosting. *Systems, man and cybernetics, 2005 ieee international conference on*, 3, 2340–2345. doi: 10.1109/ICSMC.2005.1571498.

Peura, M. et al. (1997). Efficiency of simple shape descriptors. *Proceedings of the third international workshop on visual form*, 443, 451.

Plamondon, R., Feng, C. & Woch, A. (2003). A kinematic theory of rapid human movement. Part IV: a formal mathematical proof and new insights. *Biological cybernetics*, 89(2), 126–138. doi: 10.1007/s00422-003-0407-9.

Plamondon, R., O'Reilly, C., Galbally, J., Almaksour, A. & Anquetil, E. (2014). Recent developments in the study of rapid human movements with the kinematic theory: Applications to handwriting and signature synthesis. *Pattern recognition letters*, 35, 225–235. doi: http://dx.doi.org/10.1016/j.patrec.2012.06.004.

Plamondon, R. et al. (2006). A multi-level representation paradigm for handwriting stroke generation. *Human movement science*, 25(4–5), 586–607.

Reddy, K. K. et al. (2009). Incremental action recognition using feature-tree. *Computer vision, 2009 ieee 12th international conference on*, (Iccv), 1010–1017.

Režnáková, M., Tencer, L. & Cheriet, M. (2012). Online handwritten gesture recognition based on Takagi-Sugeno fuzzy models. *Information science, signal processing and their applications (isspa), 2012 11th international conference on*, pp. 1247–1252.

Režnáková, M., Tencer, L. & Cheriet, M. (2013). ARTIST: ART-2A driven Generation of Fuzzy Rules for Online Handwritten Gesture Recognition. *Document analysis and recognition, 2013. icdar '13. 12th international conference on*.

Režnáková, M., Tencer, L. & Cheriet, M. (2015a). SO-ARTIST: Self-Organized ART-2A inspired clustering for online Takagi–Sugeno fuzzy models. *Applied soft computing*, 31, 132–152. doi: http://dx.doi.org/10.1016/j.asoc.2015.02.022.

Režnáková, M., Tencer, L., Plamondon, R. & Cheriet, M. (2015b). The generation of synthetic handwritten data for improving on-line learning. *17th biennial conference of the international graphonomics society*.

Režnáková, M., Tencer, L. & Cheriet, M. (2016a). Incremental Similarity for real-time on-line incremental learning systems. *Pattern recognition letters*, 74, 61–67.

Režnáková, M., Tencer, L. & Cheriet, M. (2016b). Elastic Memory Learning for Fuzzy Inference Models (under review). *Applied soft computing*.

Režnáková, M., Tencer, L., Plamondon, R. & Cheriet, M. (2016c). Forgetting of unused classes in missing data environment using artificially generated data. Application to on-line handwritten stroke recognition (under review). *Pattern recognition*.

Rong, H.-J., Sundararajan, N., Huang, G.-B. & Zhao, G.-S. (2010). Extended sequential adaptive fuzzy inference system for classification problems. *Evolving systems*, 2(2), 71–82. doi: 10.1007/s12530-010-9023-9.

Ross, D. a., Lim, J., Lin, R.-S. & Yang, M.-H. (2007). Incremental Learning for Robust Visual Tracking. *International journal of computer vision*, 77(1-3), 125–141. doi: 10.1007/s11263-007-0075-7.

Rubine, D. H. (1991). The Automatic Recognition of Gestures.

Saffari, A., Godec, M., Pock, T., Leistner, C. & Bischof, H. (2010). Online multi-class LPBoost. *Computer vision and pattern recognition (cvpr), 2010 ieee conference on*, pp. 3570–3577. doi: 10.1109/CVPR.2010.5539937.

Schmidt, R. A. et al. (1988). *Motor control and learning*. Human kinetics.

Snyman, J. (2005). *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*. Springer Science & Business Media.

Song, F., Liu, H., Zhang, D. & Yang, J. (2008). A highly scalable incremental facial feature extraction method. *Neurocomputing*, 71(10-12), 1883–1888. doi: 10.1016/j.neucom.2007.09.022.

Takagi, T. et al. (1985). Fuzzy identification of systems and its applications to modeling and control. *Systems, man and cybernetics, ieee transactions on*, SMC-15(1), 116–132.

Tan, K., Jing, W. & David, L. (2000). T-S Fuzzy Model with Linear Rule Consequence and PDC Controller : A Universal Framework for Nonlinear Control Systems. *Fuzzy systems, 2000. fuzz ieee 2000. the ninth ieee international conference on*, pp. 549 – 554.

Tanaka, H., Krakauer, J. W. & Qian, N. (2006). An optimization principle for determining movement duration. *Journal of neurophysiology*, 95(6), 3875–3886.

Tax, D. (2003). Online SVM learning: from classification to data description and back. *Neural networks for signal processing, 2003. nnsp'03. 2003 ieee 13th workshop on*, pp. 499 – 508.

Tencer, L., Reznakova, M. & Cheriet, M. (2015a). TITS-FM: Transductive incremental Takagi-Sugeno fuzzy models. *Applied soft computing*, 26, 531–544. doi: 10.1016/j.asoc.2014.09.024.

Tencer, L., Režnáková, M. & Cheriet, M. (2015b). Universum Learning for Semi-Supervised Signature Recognition from Spatio-Temporal Data. *17th biennial conference of the international graphonomics society*.

Utgoff, P. E. (1989). Incremental Induction of Decision Trees. *Machine learning*, 4(2), 161–186.

Vesanto, J. et al. (2000). Clustering of the Self-Organizing Map. *Neural networks, ieee transactions on*, 11(3), 586–600.

Wang, H., Pi, D. & Sun, Y. (2007). Online SVM regression algorithm-based adaptive inverse control. *Neurocomputing*, 70(4-6), 952–959. doi: 10.1016/j.neucom.2006.10.021.

Wang, W., Men, C. & Lu, W. (2008). Online prediction model based on support vector machine. *Neurocomputing*, 71(4-6), 550–558. doi: 10.1016/j.neucom.2007.07.020.

Wen, X., Shao, L., Xue, Y. & Fang, W. (2015). A rapid learning algorithm for vehicle classification. *Information sciences*, 295, 395-406.

Willems, D. J. M. et al. (2008). Definitions for Features used in Online Pen Gesture Recognition Space-based Features. (5).

Yao, M., Qu, X., Gu, Q., Ruan, T. & Lou, Z. (2010). Online PCA with Adaptive Subspace Method for Real-Time Hand Gesture Learning and Recognition. *Wseas transactions on computers*, 9(6), 583–592.

Yu, C., Zhang, R., Huang, Y. & Xiong, H. (2009). High-dimensional kNN joins with incremental updates. *Geoinformatica*, 14(1), 55–82. doi: 10.1007/s10707-009-0076-5.

Yuan, Y. et al. (1996). A genetic algorithm for generating fuzzy classification rules. *Fuzzy sets and systems*, 84(1), 1–19.

Zhang, D. et al. (2004). Semantic, hierarchical, online clustering of web search results. *Asia-pacific web conference*, pp. 69–78.