One Class Classification using Kernel Ridge Regression for Non-stationary Environment

Undergraduate Thesis

Submitted in partial fulfillment of the requirements of
CS 493 B.Tech Project

By

Vedaanta Agarwalla
ID No. 140001038

Under the supervision of:

Dr. Aruna Tiwari

INDIAN INSTITUTE OF TECHNOLOGY INDORE
November 2017

Declaration of Authorship

I, Vedaanta Agarwalla , declare that this Undergraduate Thesis titled, 'One Class Classification using Kernel Ridge Regression for Non-stationary Environment' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for BTP Project at IIT Indore.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed: _____

Date: _____

Certificate

This is to certify that the thesis entitled, "*One Class Classification using Kernel Ridge Regression for Non-stationary Environment*" and submitted by Vedaanta Agarwalla ID No. 140001038 in partial fulfillment of the requirements of CS 493 B.Tech Project embodies the work done by him under my supervision.

Supervisor
Dr. Aruna Tiwari
Associate Professor,
Indian Institute of Technology Indore
Date:

"Sometimes life hits you in the head with a brick. Don't lose faith."

Steve Jobs

INDIAN INSTITUTE OF TECHNOLOGY INDORE

Abstract

Bachelor of Technology

One Class Classification using Kernel Ridge Regression for Non-stationary Environment

Kernel Ridge Regression (KRR) based One-class Classification (OCC) is explored in the past for stationary environment but not for non-stationary environment. This paper presents a KRR based one-class classifier, which can adapt the non-stationarity present in the data stream. In this paper, Type-2 Fuzzy logic along with meta-cognition is employed with KRR based one-class classifier for handling data in non-stationary environment. The meta-cognition enables the one-class classifier for the decision of which inputs to train and how to train. And Type-2 fuzzy logic generates a Type-2 fuzzy kernel which helps building the model which makes its decision boundary adaptable to new incoming data. Additionally, employs a forgetting mechanism under the Meta-cognition framework, which boosts the ability of the classifier to eliminate the impact of irrelevant data as well as assures the execution of the proposed method within the limited memory consumption. Moreover, the proposed method is tested on different types of non-stationary artificial and real datasets to verify its behaviour under various drifting conditions of normal and outlier samples, and compared the performance with the state-of-the-art kernel based online one-class classifiers.

Acknowledgements

I would like to thank my B.Tech Project supervisor **Dr. Aruna Tiwari** for her constant support in structuring the project and her valuable feedback throughout the course of this project.She gave me an opportunity to discover and work in such an interesting domain.Her guidance proved really valuable in all the difficulties I faced in the course of this project.

I am really grateful to **Mr. Chandan Gautam** who also provided valuable guidance and helped with the problems while working with the research. He provided the initial pathway for stating the project in right manner and provided useful directions to proceed along whenever necessary.

I am also thankful to my family members, friends and colleagues who were a constant source of motivation. I am really grateful to Dept. of Computer Science & Engineering, IIT Indore for providing with the necessary hardware utilities to complete the project. I offer sincere thanks to everyone who else who knowingly or unknowingly helped me complete this project.

Contents

List of Figures

List of Tables

Abbreviations

OCC	**O**ne **C**lass **C**lassifier
KRR	**K**ernel **R**idge **R**egression
OCSVM	**O**ne **C**lass **S**upport **V**ector **M**achine
FOU	**F**ootprint **O**f **U**ncertainty
CPU	**C**entral **P**rocessing **U**nit
IOT	**I**nternet **O**f **T**hings
IOT	**S**upport **V**ector **M**achine
SVDD	**S**upport **V**ector **D**ata **D**escription
PCA	**P**rinciple **C**omponent **A**nalysis
KKT	**K**arush **K**uhn **T**ucker
OCKRR	**O**ne **C**lass **K**ernel **R**idge **R**egression
AAKELM	**A**uto **A**ssociative **K**ernelized **E**xtreme **L**earning **M**achine
UCI	**U**niversity **C**alifornia **Irvine**
CDT	**C**lass **D**iagonal **T**ranslation
CHT	**C**lass **H**orizontal **T**ranslation
CVT	**C**lass **V**ertical **T**ranslation
UG	**U**nimodal **G**aussian

This work is dedicated to my parents and friends.

Chapter 1

Introduction

1.1 Background

With sensors pervading our everyday lives, we are seeing an exponential increase in the availability of streaming, time-series data. Largely driven by the rise of the Internet of Things (IoT) and connected real-time data sources, we now have an enormous number of applications with sensors that produce important data that changes over time. Analyzing these streams effectively can provide valuable insights for any use case and application.

The detection of anomalies in real-time streaming data has practical and significant applications across many industries. Use cases such as preventative maintenance, fraud prevention, fault detection, and monitoring can be found throughout numerous industries such as finance, IT, security, medical, energy, e-commerce, agriculture, and social media. Detecting anomalies can

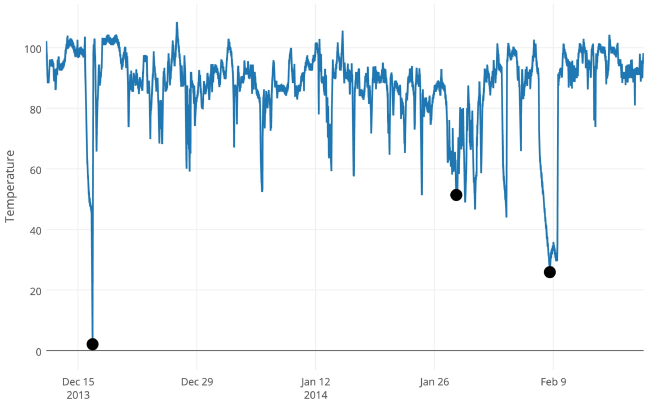


FIGURE 1.1: The figure shows real-world temperature sensor data from an internal component of a large industrial machine. Anomalies are labeled with circles. The first anomaly was a planned shutdown. The third anomaly was a catastrophic system failure. The second anomaly, a subtle but observable change in the behavior, indicated the actual onset of the problem that led to the eventual system failure.

give actionable information in critical scenarios, but reliable solutions do not yet exist. To this end, we propose a novel and robust solution to tackle the challenges presented by real-time anomaly detection.

We define an anomaly as a point in time where the behavior of the system is unusual and significantly different from previous, normal behavior. An anomaly may signify a negative change in the system, like a fluctuation in the turbine rotation frequency of a jet engine, possibly indicating an imminent failure. An anomaly can also be positive, like an abnormally high number of web clicks on a new product page, implying stronger than normal demand. Either way, anomalies in data identify abnormal behavior with potentially useful information. Anomalies can be spatial, where an individual data instance can be considered anomalous with respect to the rest of data, independent of where it occurs in the data stream, like the first and third anomalous spikes in 1.1. An anomaly can also be temporal, or contextual, if the temporal sequence of data is relevant; i.e., a data instance is anomalous only in a specific temporal context, but not otherwise. Temporal anomalies, such as the middle anomaly of 1.1, are often subtle and hard to detect in real data streams. Detecting temporal anomalies in practical applications is valuable as they can serve as an early warning for problems with the underlying system.

1.1.1 Streaming Data

Streaming applications impose unique constraints and challenges for machine learning models. These applications involve analyzing a continuous sequence of data occurring in real-time. In contrast to batch processing, the full dataset is not available. The system observes each data record in sequential order as they arrive and any processing or learning must be done in an online fashion. Let the vector xt represent the state of a real-time system at time t. The model receive a continuous stream of inputs:

..., xt-2,xt-1,xt,xt+1,xt+2, ...

Consider for example, the task of monitoring a datacenter. Components of xt might include CPU usage for various servers, bandwidth measurements, latency of servicing requests, etc. At each point in time t we would like to determine whether the behavior of the system is unusual. The determination must be made in real-time, before time t+1. That is, before seeing the next input (xt+1), the algorithm must consider the current and previous states to decide whether the system behavior is anomalous, as well as perform any model updates and retraining. Unlike batch processing, data is not split into train/test sets, and algorithms cannot look ahead. Practical applications impose additional constraints on the problem. Typically, the sensor streams are large in number and at high velocity, leaving little opportunity for human, let alone expert, intervention; manual parameter tweaking and data labeling are not viable. Thus, operating in an unsupervised, automated fashion is often a necessity.

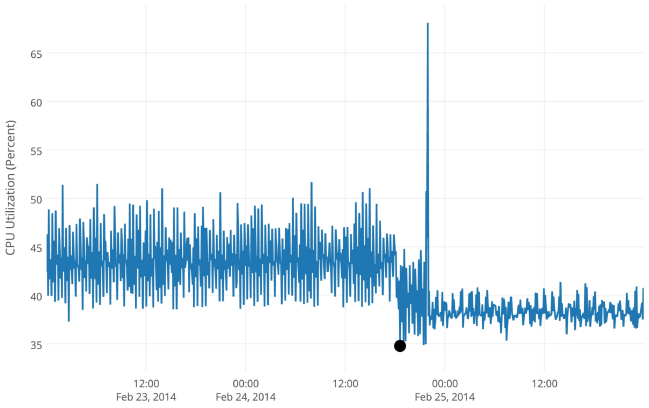


FIGURE 1.2: CPU utilization (percent) for an Amazon EC2 instance (data from the Numenta Anomaly Benchmark). A modification to the software running on the machine caused the CPU usage to change. The initial anomaly represents a changepoint, and the new system behavior that follows is an example of concept drift. Continuous learning is essential for performing anomaly detection on streaming data like this.

In many scenarios the statistics of the system can change over time, a problem known as concept drift. Consider again the example of a production datacenter. Software upgrades and configuration changes can occur at any time and may alter the behavior of the system 1.2. In such cases models must adapt to a new definition of "normal" in an unsupervised, automated fashion.

In predictive analytics and machine learning, the concept drift means that the statistical properties of the target variable, which the model is trying to predict, change over time in unforeseen ways. This causes problems because the predictions become less accurate as time passes.

The term concept refers to the quantity to be predicted. More generally, it can also refer to other phenomena of interest besides the target concept, such as an input, but, in the context of concept drift, the term commonly refers to the target variable.

In streaming applications early detection of anomalies is valuable in almost any use case. Consider a system that continuously monitors the health of a cardiac patient's heart. An anomaly in the data stream could be a precursor to a heart attack. Detecting such an anomaly minutes in advance is far better than detecting it a few seconds ahead, or detecting it after the fact. Detection of anomalies often gives critical information, and we want this information early enough that it's actionable, possibly preventing system failure. There is a tradeoff between early detections and false positives, as an algorithm that makes frequent inaccurate detections is likely to be ignored.

1.1.2 Non-stationarity data

On the theoretical side this happens because the distribution from which your data in extracted changes. In practice, this kind of difference could be due to spatial or temporal changes on new data. Suppose you build a classifier to differentiate between male/female faces just using human images. But, somehow, you start to get dogs/cats images to classify. This could lead to poor performance.

The same could happen with spatial processes, for example suppose we build a classifier that, given latitude/longitude of a house, we can predict the family income. But we only have information about some parts of the city (downtown and close surroundings). Now we get the lat/long of a new house in the city, far from the downtown area. Due to a totally unexpected change on a neighborhood characteristic like a slum being very close to a fancy neighborhood on the city we could get wrong results. (Type 'morumbi paraisopolis' at google images to see it) or due to drastic changes on a terrain of some kind on satellite images.

Another situation could be due to behavioural changes over time. Imagine you use a Naive Bayes to build a spam blocker. But somehow, the spammers understand the rules your filter apply and then they adapt their messages in order to surpass your blocker. Your model will start to perform poorly and you will need to 'retrain' your classifier.

Customers also change their behavior over time due to lots of things like technological and competitors changes (Netflix/Blockbuster), laws (One child policy), social behavior (Gay marriage is being more accepted in some places, which could lead to people who would declare themselves single in the past now declare married).

1.2 Objectives

Given the above requirements, we define the ideal characteristics of a real-world anomaly detection algorithm as follows:

1. To design an enhanced online one class classifier.
   (a) The algorithm must learn continuously without a requirement to store the entire stream.
   (b) The algorithm must run in an unsupervised.
   (c) Algorithms must adapt to dynamic environments and concept drift.
   (d) Algorithm should handle fuzziness in the data.

2. To test the proposed algorithm on standard non stationary datasets and achieve higher accuracy than current algorithms.

Taken together, the above requirements suggest that anomaly detection for streaming applications is a fundamentally different problem than static batch anomaly detection. As discussed further below, the majority of existing anomaly detection algorithms (even those designed for time-series data) are not applicable to streaming applications.

Chapter 2

Literature Survey

This chapter provides an overview of current algorithms and problem domain. The chapter starts with problem description i.e. One Class Classification and current algorithms for OCC. Then it proceeds to explain non stationary data and prospective solution to modify OCC algorithms to handle non-stationary datasets.

2.1 One Class Classification

Conventional multi-class classification algorithms aim to classify an unknown object into one of several pre-defined categories. A problem arises when the unknown object does not belong to any of those categories. In one-class classification [25], one of the classes (referred to as the positive class or target class) is well characterized by instances in the training data. For the other class (nontarget), it has either no instances at all, very few of them, or they do not form a statistically-representative sample of the negative concept. To motivate the importance of one-class classification, let us consider some scenarios. One-class classification can be relevant in detecting machine faults, for instance. A classifier should detect when the machine is showing abnormal/faulty behaviour. Measurements on the normal operation of the machine (positive class training data) are easy to obtain. On the other hand, most faults will not have occurred so one will have little or no training data for the negative class. As another example, a traditional binary classifier for text or web pages requires arduous pre-processing to collect negative training examples. For example, in order to construct a homepage classifier [3], collecting sample of homepages (positive training examples) is relatively easy, however collecting samples of nonhomepages (negative training examples) is very challenging because it may not represent the negative concept uniformly and may involve human bias.

2.1.1 OCC vs. Multi-class Classification

In a conventional multi-class classification problem, data from two (or more) classes are available and the decision boundary is supported by the presence of example samples from each class. Moya et al. [16] originate the term One-Class Classification in their research work. Different researchers have used other terms to present similar concepts such as Outlier Detection [18], Novelty Detection [3] or Concept Learning [11]. These terms originate as a result of different applications to which OCC has been applied. The drawbacks that are encountered in the conventional classification problems, such as the estimation of error rates, measuring the complexity of a solution, the curse of dimensionality, the generalization of the method, and so on, also appear in OCC, and sometimes become even more prominent. As stated earlier, in OCC tasks, the negative data is either absent or limited in its distribution, so only one side of the classification boundary can be determined definitively by using the data. This makes problem of one-class classification harder than the problem of conventional multi-class / binary classification. The task in OCC is to define a classification boundary around the positive (or target) class, such that it accepts as many objects as possible from the positive class, while it minimizes the chance of accepting non-positive (or outlier) objects. Since only one side of the boundary can be determined, in OCC, it is hard to decide, on the basis of just one class how tightly the boundary should fit in each of the directions around the data. It is also harder to decide which attributes should be used to find the best separation of the positive and non-positive class objects. In particular, when the boundary of the data is long and non-convex, the required number of training objects might be very high. Hence it is to be expected that one-class classification algorithms will require a larger number training instances relative to conventional multi-class classification algorithms.

Based on reviewing past research that has been carried out in the field of OCC by using different algorithms, methodologies and application domains, we propose a taxonomy with three broad categories for the study of OCC problems. The taxonomy can be summarized as (see Fig. 2.1):

(a) Availability of Training Data: Learning with positive data only (or with a limited amount of negative samples) or learning with positive and unlabeled data (b) Methodology Used: Algorithms based on One Class Support Vector Machines (OSVMs) or methodologies based on algorithms other than OSVMs (c) Application Domain Applied: OCC applied in the field of text/document classification or in other application domains

The proposed categories are not mutually exclusive, so there may be some overlapping among the research carried out in each of these categories. However, they cover almost all of the major research conducted using the concept of OCC in various contexts and application domains. The key contributions in most OCC research fall into one of the above-mentioned categories.

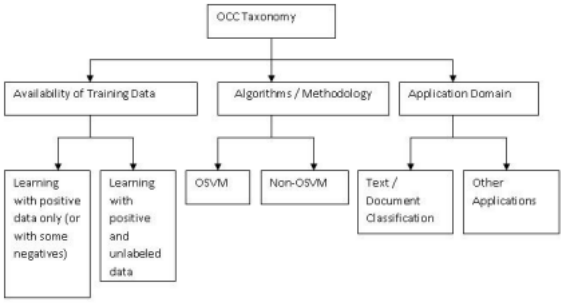


FIGURE 2.1: Taxonomy for the Study of OCC Techniques.

2.1.2 OCSVM

The one-class classification problem is often solved by estimating the target density [16], or by fitting a model to the data support vector classifier. Tax and Duin [26] seek to solve the problem of OCC by distinguishing the positive class from all other possible patterns in the pattern space. Instead of using a hyper-plane to distinguish between two classes, a hyper-sphere is found around the positive class data that encompasses almost all points in the data set with the minimum radius. This method is called the Support Vector Data Description (SVDD). Thus training this model has the possibility of rejecting some fraction of the positively-labeled training objects, when this sufficiently decreases the volume of the hyper-sphere. Furthermore, the hyper-sphere model of the SVDD can be made more flexible by introducing kernel functions. Tax considers a Polynomial and a Gaussian kernel and found that the Gaussian kernel works better for most data sets. A drawback of this technique is that they often require a large data set; in particular, in high dimensional feature spaces, it becomes very inefficient. Also, problems may arise when large differences in density exist. Objects in low-density areas will be rejected although they are legitimate objects. Scholkopf et al. [20] present an alternative approach to the above mentioned work of Tax and Duin on OCC using a separating hyper-plane. The difference between theirs and Tax and Duins approach is that instead of trying to find a hyper-sphere with minimal radius to fit the data, they try to separate the surface region containing data from the region containing no data. This is achieved by constructing a hyper-plane which is maximally distant from origin, with all data points lying on the opposite side from the origin and such that the margin is positive. Their paper proposes an algorithm that computes a binary function that returns +1 in small regions (subspaces) that contain data and -1 elsewhere. The data is mapped into the feature space corresponding to the kernel and is separated from the origin with maximum margin. They evaluate the efficacy of their method on the US Postal Services data set of handwritten digits and show that the algorithm is able to extract patterns which are very hard to assign to their respective classes and a number of outliers were identified. Manevitz and Yousef [15] propose a different version of the one class SVM which is based on identifying outlier data as representative of the second class. The idea of this methodology is to work first in the feature

space, and assume that not only is the origin the second class, but also that all data points close enough to the origin are to be considered as noise or outliers. The vectors lying on standard sub-spaces of small dimension (i.e. axes, faces, etc.) are treated as outliers. They evaluate their results on Reuters Data set1 and the results are worse than the OSVM algorithm presented by Scholkopf et al. [20]. Classifiers are commonly ensembled to provide a combined decision by averaging the estimated posterior probabilities. When Bayes theorem is used for the combination of different classifiers, under the assumption of independence, a product combination rule can be used to create classifier ensemble. The outputs of the individual classifiers are multiplied and then normalized (also called the logarithmic opinion pool). In OCC, as the information on the non-positive data is not available, in most cases, the outliers are assumed to be uniformly distributed and the posterior probability can be estimated. Tax mentions that in some OCC methods, distance is estimated instead of probability for one class classifier ensembling. Tax observes that the use of ensembles in OCC improves performance, especially when the product rule is used to combine the probability estimates. Yu [28] proposes an OCC algorithm with SVMs using positive and unlabeled data, and without labeled negative data, and discusses some of the limitations of other OCC algorithms [24][20][15]. Yu comments that in the absence of negative examples, OSVM requires a much larger amount of positive training data to induce an accurate class boundary

2.1.3 Non-OCSVM

Ridder et al. [23] conduct an experimental comparison of various OCC algorithms, including: (a) Global Gaussian approximation; (b) Parzen density estimation; (c) 1-Nearest Neighbor method; and (d) Gaussian approximation (combines aspects of (a) and (b)). Manevitz and Yousef [15] trained a simple neural network to filter documents when only positive information is available. To incorporate the restriction of availability of positive examples only, they used a three-level feed forward network with a “bottleneck”. DeComite et al. [4] modify the C4.5 decision tree algorithm [17] to get an algorithm that takes as input a set of labeled examples, a set of positive examples, and a set of unlabeled data, and then use these three sets to construct the decision tree. Letouzey et al. [5] design an algorithm which is based on positive statistical queries (estimates for probabilities over the set of positive instances)and instance statistical queries (estimates for probabilities over the instance space). They design a decision tree induction algorithm, called POSC4.5, using only positive and unlabeled data. They present experimental results on UCI data sets that are comparable to the C4.5 algorithm. Wang et al. [27] investigate several one-class classification methods in the context of Human-Robot interaction for face and non-face classification. Some of the noteworthy methods used in their study are: (a) SVDD; (b) Gaussian data description; (c) KMEANS-DD; (d) Principal Component Analysis-DD. In

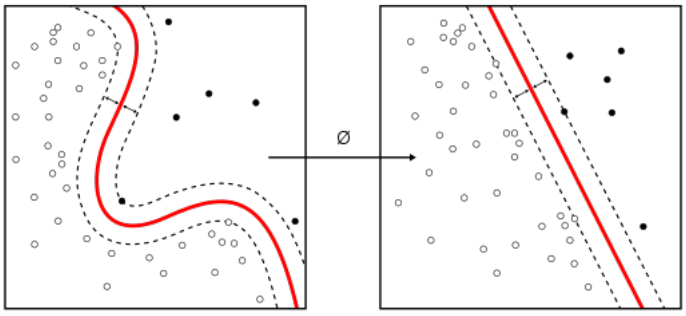


FIGURE 2.2: Kernel Feature Mapping.

their experimentation, they observe that SVDD attains better performance than the other OCC methods they studied.

2.2 Kernel Method

In machine learning, kernel methods are a class of algorithms for pattern analysis, whose best known member is the support vector machine (SVM). The general task of pattern analysis is to find and study general types of relations (for example clusters, rankings, principal components, correlations, classifications) in datasets. For many algorithms that solve these tasks, the data in raw representation have to be explicitly transformed into feature vector representations via a user-specified feature map: in contrast, kernel methods require only a user-specified kernel, i.e., a similarity function over pairs of data points in raw representation.

Kernel methods owe their name to the use of kernel functions, which enable them to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space. This operation is often computationally cheaper than the explicit computation of the coordinates. This approach is called the "kernel trick"[1]. Kernel functions have been introduced for sequence data, graphs, text, images, as well as vectors.

Algorithms capable of operating with kernels include the kernel perceptron, support vector machines (SVM), Gaussian processes, principal components analysis (PCA), canonical correlation analysis, ridge regression, spectral clustering, linear adaptive filters and many others. Any linear model can be turned into a non-linear model by applying the kernel trick to the model: replacing its features (predictors) by a kernel function[citation needed].

Most kernel algorithms are based on convex optimization or eigenproblems and are statistically well-founded. Typically, their statistical properties are analyzed using statistical learning theory

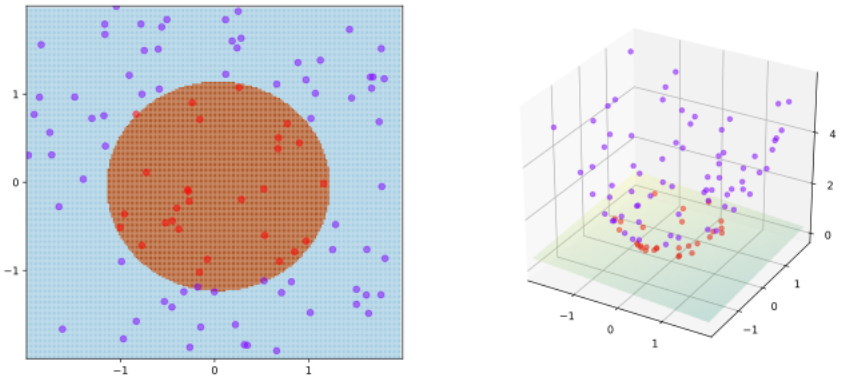


FIGURE 2.3: The training points are mapped to a 3-dimensional space where a separating hyperplane can be easily found.

2.3 Online Learning

In predictive analytics and machine learning, the concept drift means that the statistical properties of the target variable, which the model is trying to predict, change over time in unforeseen ways. This causes problems because the predictions become less accurate as time passes.

To prevent deterioration in prediction accuracy because of concept drift, both active and passive solutions can be adopted. Active solutions rely on triggering mechanisms, e.g., change-detection tests (Basseville and Nikiforov 1993; Alippi and Roveri, 2007) to explicitly detect concept drift as a change in the statistics of the data-generating process. In stationary conditions, any fresh information made available can be integrated to improve the model. Differently, when concept drift is detected, the current model is no more up-to-date and must be substituted with a new one to maintain the prediction accuracy (Gama et al., 2004; Alippi et al., 2011). On the contrary, in passive solutions the model is continuously updated, e.g., by retraining the model on the most recently observed samples (Widmer and Kubat, 1996), or enforcing an ensemble of classifiers (Elwell and Polikar 2011).

Contextual information, when available, can be used to better explain the causes of the concept drift: for instance, in the sales prediction application, concept drift might be compensated by adding information about the season to the model. By providing information about the time of the year, the rate of deterioration of your model is likely to decrease, concept drift is unlikely to be eliminated altogether. This is because actual shopping behavior does not follow any static, finite model. New factors may arise at any time that influence shopping behavior, the influence of the known factors or their interactions may change.

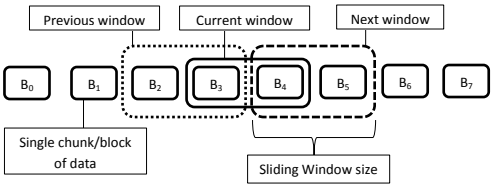


FIGURE 2.4: Data stream with new samples which leads to deletion of old samples.

Concept drift cannot be avoided for complex phenomenon that are not governed by fixed laws of nature. All processes that arise from human activity, such as socioeconomic processes, and biological processes are likely to experience concept drift. Therefore periodic retraining, also known as refreshing, of any model is necessary.

We use Meta Cognitivity to adapt our offline classsifier to Online stream of data possessing Concept Drift.

2.4 Metacognition

Metacognition is "cognition about cognition", "thinking about thinking", "knowing about knowing", becoming "aware of one's awareness" and higher-order thinking skills. The term comes from the root word meta, meaning "beyond". Metacognition can take many forms; it includes knowledge about when and how to use particular strategies for learning or for problem-solving. There are generally two components of metacognition: (1) knowledge about cognition and (2) regulation of cognition.

Meta cognition is a technique that enables the self-regulation of the learning process [10]. Instead of training on all samples as in a classical case we use meta cognitive approach that decides *what-to-learn, how-to-learn, what-to-learn* from the data stream which in machine learning domain is implemented by *sample delete strategy, neuron growth strategy, parameter update strategy, sample reserve strategy.* Meta cognition over neural networks as in [19], neuro-fuzzy systems as in [23], have been researched but lack a forget mechanism. Giduthuri et al. propose a projection based learning algorithm including a forget mechanism in [2].

2.5 Fuzzy Logic

Fuzzy logic has become an important tool for number of different applications ranging from the control of engineering system to artificial intelligence. Practical applications of fuzzy logic pose a unique set of problems. The design of systems, which apply fuzzy logic to make use of human knowledge and experience, is a daunting task without facing engineering problems of real world

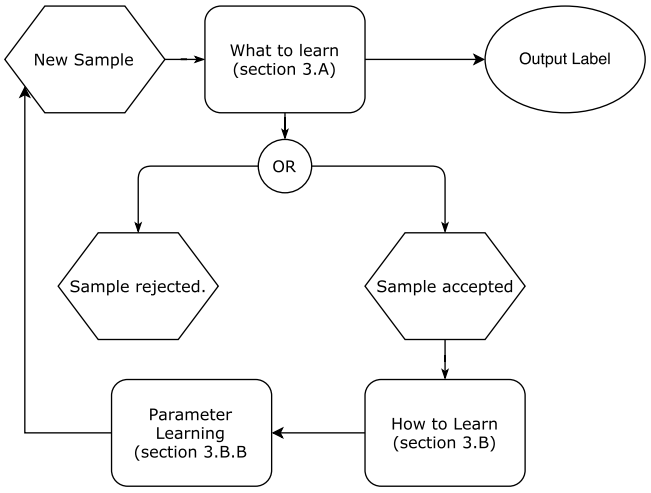


FIGURE 2.5: Metacognition.

systems. Fuzzy logic is a set of mathematical principles for knowledge representation based on degrees of membership . Fuzzy logic is a form of many-valued logic; it deals with reasoning that is approximate rather than fixed and exact. Compared to traditional binary sets (where variables may take on true or false values), fuzzy logic variables may have a truth value that ranges in degree between 0 and 1. Fuzzy logic has been extended to handle the concept of partial truth, where the truth value may range between completely true and completely false . When linguistic variables are used, these degrees may be managed by specific functions. Fuzzy logics provide the basis for logical systems dealing with vagueness, e.g. for formalizing common natural language predicates such as “tall” or “fast”. Design choices in this framework are made as to which real numbers to take as truth values, and which properties connectives should have. In fact logics based on real numbers occur in a number of areas in logic. Fuzzy logic is based on the theory of fuzzy sets, which a generalization of the classical is set theory. Saying that the theory of fuzzy sets is a generalization of the classical set theory means that the latter is a special case of fuzzy sets theory. To make a metaphor in set theory speaking, the classical set theory is a subset of the theory of fuzzy sets. A fuzzy set is a set without a crisp, not clearly defined boundary. It can contain elements with a partial degree of membership with multi-valued logic. Fuzzification comprises the process of transforming discrete values into grades of membership (continuous) for linguistic terms of fuzzy sets. The membership function is used to associate a grade to each linguistic term. Defuzzify evaluate several membership sets established by the system designer for a fuzzy logic based control system, such as ”speed too fast,” ”speed too slow” and ”speed about right” at a specific input value. Degree of membership is a specific value that defines how each point in the input space is mapped to the specific environment being studied lying between 0 and 1. Linguistic Variable means relating to language, (plain language words and statements). While variables in mathematics usually take numerical values, in fuzzy logic, the non-numeric linguistic variables are often used to facilitate the expression of rules and facts . A

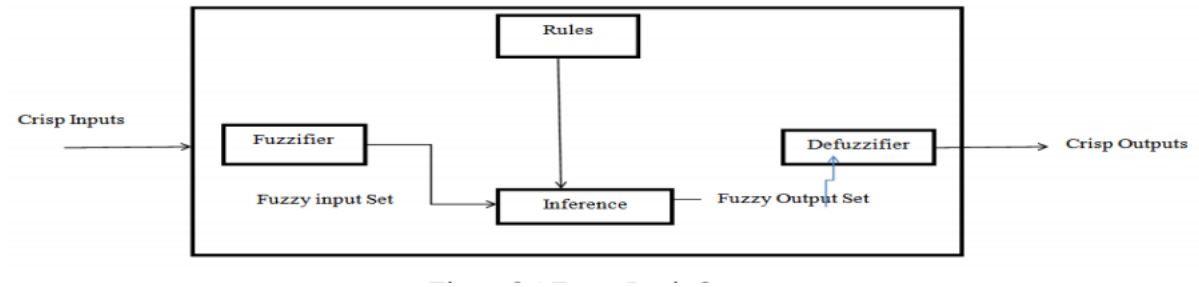


FIGURE 2.6: Fuzzy Logic System

Fuzzy Logic System consists of four main parts: fuzzier, rules, inference engine, and defuzzifier. These components and the general architecture of a Fuzzy Logic System is shown in 2.6.

The process of fuzzy logic involves obtaining a crisp set of input data are gathered and converted to a fuzzy set using fuzzy linguistic variables, fuzzy linguistic terms and membership functions . This step is known as fuzzication. Afterwards, an inference is made based on a set of rules and lastly, the resulting fuzzy output is mapped to a crisp output using the membership functions, in the defuzzication step.

2.6 Type 2 Fuzzy

Type-2 fuzzy sets and systems generalize Type-1 fuzzy sets and systems so that more uncertainty can be handled. From the very beginning of fuzzy sets, criticism was made about the fact that the membership function of a type-1 fuzzy set has no uncertainty associated with it, something that seems to contradict the word fuzzy, since that word has the connotation of lots of uncertainty. So, what does one do when there is uncertainty about the value of the membership function? The answer to this question was provided in 1975 by the inventor of fuzzy sets, Prof. Lotfi A. Zadeh [29], when he proposed more sophisticated kinds of fuzzy sets, the first of which he called a type-2 fuzzy set. A type-2 fuzzy set lets us incorporate uncertainty about the membership function into fuzzy set theory, and is a way to address the above criticism of type-1 fuzzy sets head-on. And, if there is no uncertainty, then a type-2 fuzzy set reduces to a type-1 fuzzy set, which is analogous to probability reducing to determinism when unpredictability vanishes,.

In order to symbolically distinguish between a type-1 fuzzy set and a type-2 fuzzy set, a tilde symbol is put over the symbol for the fuzzy set; so, A denotes a type-1 fuzzy set, whereas $\tilde{A}$ denotes the comparable type-2 fuzzy set. When the latter is done, the resulting type-2 fuzzy set is called a general type-2 fuzzy set (to distinguish it from the special interval type-2 fuzzy set).

Prof. Zadeh didn't stop with type-2 fuzzy sets, because in that 1976 paper [29] he also generalized all of this to type-n fuzzy sets. The present article focuses only on type-2 fuzzy sets because

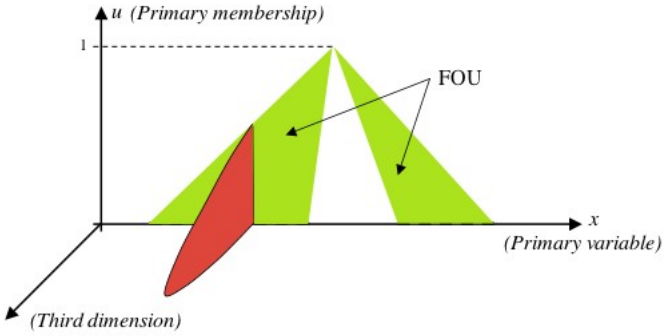


FIGURE 2.7: The membership function of a general type-2 fuzzy set is three-dimensional. A cross-section of one slice of the third dimension is shown. This cross-section, as well as all others, sits on the FOU. Only the boundary of the cross-section is used to describe the membership function of a general type-2 fuzzy set. It is shown filled-in for artistic purposes.

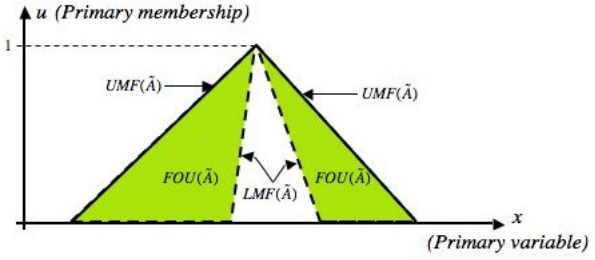


FIGURE 2.8: FOU for an interval type-2 fuzzy set. Many other shapes are possible for the FOU.

they are the next step in the logical progression from type-1 to type-n fuzzy sets, where n = 1, 2, ... Although some researchers are beginning to explore higher than type-2 fuzzy sets, as of early 2009, this work is in its infancy.

The membership function of a general type-2 fuzzy set, $\tilde{A}$, is three-dimensional (Fig. 2.7), where the third dimension is the value of the membership function at each point on its two-dimensional domain that is called its footprint of uncertainty (FOU).

For an interval type-2 fuzzy set that third-dimension value is the same (e.g., 1) everywhere, which means that no new information is contained in the third dimension of an interval type-2 fuzzy set. So, for such a set, the third dimension is ignored, and only the FOU is used to describe it. It is for this reason that an interval type-2 fuzzy set is sometimes called a first-order uncertainty fuzzy set model, whereas a general type-2 fuzzy set (with its useful third-dimension) is sometimes referred to as a second-order uncertainty fuzzy set model.

The FOU represents the blurring of a type-1 membership function, and is completely described by its two bounding functions (Fig. 2.8), a lower membership function (LMF) and an upper membership function (UMF), both of which are type-1 fuzzy sets! Consequently, it is possible to use type-1 fuzzy set mathematics to characterize and work with interval type-2 fuzzy sets. This

means that engineers and scientists who already know type-1 fuzzy sets will not have to invest a lot of time learning about general type-2 fuzzy set mathematics in order to understand and use interval type-2 fuzzy sets.

Work on type-2 fuzzy sets languished during the 1980s and early-to-mid 1990's, although a small number of articles were published about them. People were still trying to figure out what to do with type-1 fuzzy sets, so even though Zadeh proposed type-2 fuzzy sets in 1976, the time was not right for researchers to drop what they were doing with type-1 fuzzy sets to focus on type-2 fuzzy sets. This changed in the latter part of the 1990s as a result of Prof. Jerry Mendel and his student's works on type-2 fuzzy sets and systems. Since then, more and more researchers around the world are writing articles about type-2 fuzzy sets and systems.

Chapter 3

Analysis and Design

This section describes the construction of KRR for One class classification, a technique to modify the kernel according to the new samples, i.e., Meta-cognition 3.2 followed by the construction of the type-2 fuzzy kernel in 3.3 and finally is concluded with pseudo code in 3.4.
Though the rest of the algorithm is trained and tested with streaming data, in order to initialize the classifier with a set of normal training samples, we take an initial chunk of training data where all the samples are of the normal class, i.e., with class label 1. All parameters describing this initial chuck of data with a subscript of 0. So, the initial chunk of training samples is denoted by $\{\boldsymbol{X_0}, \boldsymbol{R_0}\}$ of size t_0. Section 3.1 describes this initialization phase. Next, in Section 3.2 we describe the procedure to delete samples and insert new samples in the kernel matrix and it's inverse efficiently in order to alter the information stored in the model. This is necessary as the data is non-stationary. This is followed by Section 3.3 where we discuss a procedure to enable fuzzy handling in the kernel computed in Section 3.1 and Section 3.2. It is concluded by Section 3.4, where we discuss how $McTOC$ computes the class label of the new incoming samples with the help of a pseudo-code.

3.1 Kernel Based methods for one class classification

In Boundary framework based ***OCKRR*** i.e. ***OCKRR(B)***, model is trained by only target data X and endeavored to approximate all data to any real number. Fig. 3.1 shows OCC with single output node architecture. In Fig. 3.1, given a stream of training data $\boldsymbol{X}$, $\{(\boldsymbol{x}_1, c_1), (\boldsymbol{x}_2, c_2), ..., (\boldsymbol{x}_t, c_t), ...\}$, where $\boldsymbol{x}_t = [x_t^1, x_t^2, ..., x_t^n] \in \Re^n$ is n-dimensional input of the t^{th} sample and c_t is the class label of the target class, which is same for all the training data. Input layer takes data for t^{th} input sample is coded as $(\boldsymbol{x}_t, R_t)$ because model has to approximate all data to any real number R. Target output vector $\boldsymbol{R}$ is represented as $[R_1, R_2....R_t, ...]$, however, value of R_t will be same for all samples. Here, value of R_t is considered as 1 for all the experiments.

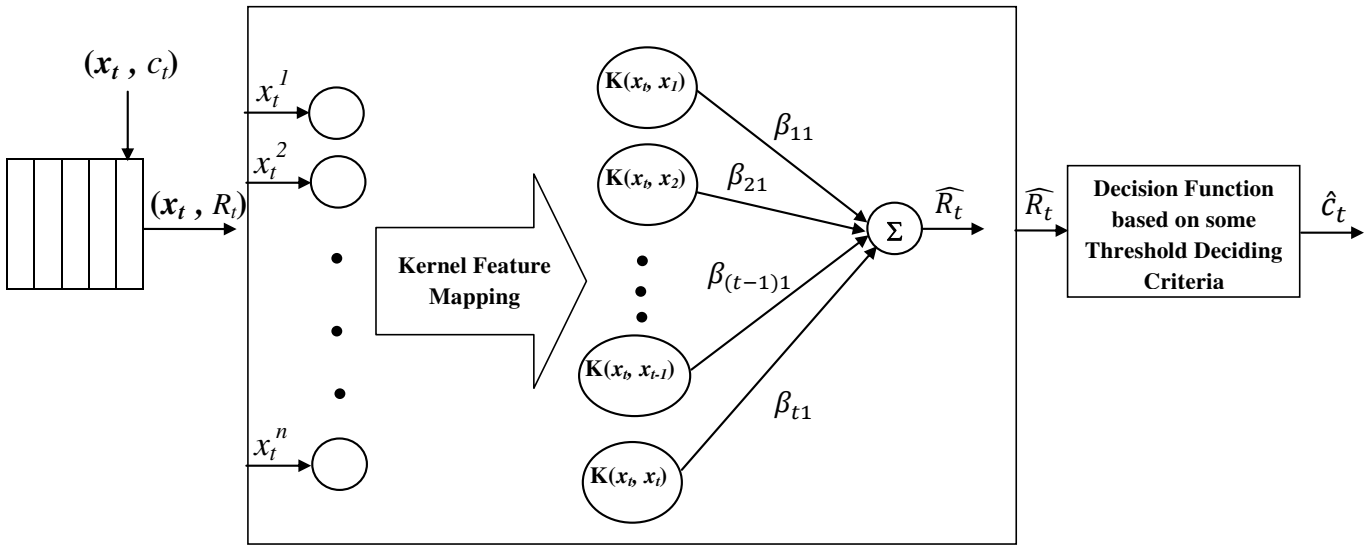


FIGURE 3.1: Schematic Diagram of Sequential One Class Kernel Ridge Regression: Boundary Based

Further, kernel feature mapping has been employed between input and hidden layer. Different symbol has been used for kernel matrix to avoid confusion between kernel $\boldsymbol{\Phi}$ and random feature mapping $\boldsymbol{H}$. Here, kernel matrix is represented as $\boldsymbol{\Phi} = \boldsymbol{H^T H} = \boldsymbol{K(X, X)} = \boldsymbol{\phi(X)}$ and $\boldsymbol{H = h(X)}$. During training, hidden layer output or kernel matrix $\boldsymbol{\Phi}$ will be a square symmetric matrix of size $[t \times t]$. Output weight $\boldsymbol{\beta}$ for any t samples in sequence of data is represented as $[\beta_{11}, \beta_{21}....\beta_{(t-1)1}, \beta_{tt}]$. $\widehat{\boldsymbol{R}} = [\widehat{R}_1, \widehat{R}_2...\widehat{R}_t, ...]$ is the predicted output vector and $\widehat{R}_t$ is the predicted output for t^{th} sample. $\widehat{\boldsymbol{c}} = [\widehat{c_1}, \widehat{c_2}...\widehat{c_t}, ...]$ is the predicted class vector, where, $\widehat{c_t}$ is the predicted class for t^{th} sample.

3.1.1 Kernel Ridge Regression based OCC (KOC)

OCC using KRR was proposed by Leng et al. [13], where the target output t_i is same for all the training samples. Thus, the model tries to map all positive data to a real number $r \in \Re$. This process corresponds to the following optimization problem:

$$\begin{aligned} \text{Minimize} : L &= \frac{1}{2}\|\boldsymbol{\beta}\|^2 + \frac{C}{2}\sum_{i=1}^{N}\|e_i\|_2^2 \\ \text{Subject to} : \ & \boldsymbol{\beta^T \phi_i} = r - e_i, \ i = 1, 2, ..., N \end{aligned} \tag{3.1}$$

After solving the minimization problem in (3.1) we obtain following output weight vector:

$$\boldsymbol{\beta} = \left(\boldsymbol{\Phi\Phi^T} + \frac{\boldsymbol{I}}{C}\right)^{-1}\boldsymbol{\Phi r} \tag{3.2}$$

Note: $\boldsymbol{r} \in \Re^N$ in bold face is a vector consists of all elements equal to r (r without bold face is a scalar). This representation is same throughout the paper.

After calculating output weight vector $\boldsymbol{\beta}$, the output of the network for a given input vector $\boldsymbol{x}_p$ is given by:

$$\boldsymbol{o_p} = \boldsymbol{f(x_p)} = \boldsymbol{\beta^T \phi_p} \tag{3.3}$$

Whether $\boldsymbol{x}_p$ belongs to the positive (i.e. normal) class or it is an outlier is decided based on the following rule:

$$\begin{aligned} &\text{If } abs(\boldsymbol{o_p} - r) \leq \varepsilon, \quad \boldsymbol{x}_p \text{ belongs to normal class} \\ &\text{Otherwise,} \qquad\qquad\qquad\qquad \boldsymbol{x}_p \text{ is an outlier} \end{aligned} \tag{3.4}$$

In the above, ε is a threshold value which is calculated based on the error obtained on the training data by rejection of few deviant training samples after completion of training.

3.1.2 Initialization Phase

For the initial chunk $\boldsymbol{\{X_0, R_0\}}$ of size N_0, the objective is to minimize the output weight $\boldsymbol{\beta_0}$ as well as error E_0 between expected $\boldsymbol{(R_0)}$ and predicted value $\boldsymbol{(\phi(X_0)\beta_0)}$. For regularization, λ is used as a regularization parameter with minimization problem. First we will write basic KRR formulation for random feature mapping with hidden layer matrix H and later, we will derive this formulation for kernel mapping using Representer Theorem[1]. Hence, minimization problem with hidden layer matrix for $\boldsymbol{X_0}$ i.e. $\boldsymbol{H_0 = h(X_0)}$ with weight vector $\boldsymbol{w_0}$ can be written as follows:

$$\begin{aligned} \text{Minimize} : L &= \frac{1}{2} \|\boldsymbol{w_0}\|^2 + \lambda \frac{1}{2} \|\boldsymbol{E_0}\|^2 \\ \text{Subject to} : \quad & \boldsymbol{H_0 w_0 = R_0 - E_0} \end{aligned} \tag{3.5}$$

Representer Theorem [1] is exploited in (3.6), which describes the weight vector $\boldsymbol{w_o}$ as a linear combination of the training data representation in KRR space ($\boldsymbol{H_0}$) and a reconstruction vector ($\boldsymbol{\beta_0}$) as follows:

$$\boldsymbol{w_0} = \boldsymbol{H_0 \beta_0}. \tag{3.6}$$

Further, minimization problem in (3.5) can be reformulated as follows using Representer theorem:

$$\begin{aligned} \text{Minimize} : L = \frac{1}{2}\boldsymbol{\beta_0^T H_0^T H_0 \beta_0} + \lambda\frac{1}{2}\left\|\boldsymbol{E_0}\right\|^2 \\ \text{Subject to} : \ \boldsymbol{\beta_0^T H_0^T H_0 = R_0 - E_0} \end{aligned} \tag{3.7}$$

Now, substituting kernel matrix $\boldsymbol{\Phi_0 = \phi(X_0) = (H_0)^T H_0 = K(X_0, X_0)}$ in the above equation and obtained:

$$\begin{aligned} \text{Minimize} : L = \frac{1}{2}\boldsymbol{\beta_0^T \Phi_0 \beta_0} + \lambda\frac{1}{2}\left\|\boldsymbol{E_0}\right\|^2 \\ \text{Subject to} : \ \boldsymbol{\beta_0^T \Phi_0 = R_0 - E_0} \end{aligned} \tag{3.8}$$

According to Karush-Kuhn-Tucker (KKT) theorem [8], (3.8) can be written as dual optimization problem:

$$\begin{aligned} \text{Minimize} : L = \frac{1}{2}\boldsymbol{\beta_0^T \Phi_0 \beta_0} + \lambda\frac{1}{2}\left\|\boldsymbol{E_0}\right\|^2 \\ -\alpha(\boldsymbol{\beta_0^T \Phi_0 - R_0 + E_0}) \end{aligned} \tag{3.9}$$

where α is Langrange multiplier, which is employed to combine the constraint with minimization problem. Further take partial derivatives of (3.9) with respect to all variables $\boldsymbol{\beta_0}, \boldsymbol{E_0}$ and α:

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{\beta_0}} = 0 \Rightarrow \boldsymbol{\beta_0} = \alpha \\ \frac{\partial L}{\partial \boldsymbol{E_0}} = 0 \Rightarrow \lambda \boldsymbol{E_0} = \alpha \\ \frac{\partial L}{\partial \alpha} = 0 \Rightarrow \boldsymbol{\beta_0^T \Phi_0 - R_0 + E_0} = 0 \end{aligned} \tag{3.10}$$

Following weight is obtained from (3.10):

$$\boldsymbol{\beta_0} = (\boldsymbol{\Phi_0} + \frac{1}{\lambda}\boldsymbol{I})^{-1}\boldsymbol{R_0} \tag{3.11}$$

Here, $\boldsymbol{I}$ is an identity matrix.

Finally, we have weight matrix for initial samples which needs to update for each upcoming sample is as follows:

$$\begin{aligned} \boldsymbol{\beta_0} = \boldsymbol{P_0 R_0} \\ \boldsymbol{P_0} = \boldsymbol{\Phi_0^{-1}} \\ \boldsymbol{\Phi_0} = \boldsymbol{\phi(X_0)} \end{aligned} \tag{3.12}$$

Here, $\boldsymbol{\Phi_0}$ is a kernel matrix for initial N_0 of size $N_0 \times N_0$. Kernel matrix $\boldsymbol{\Phi_0}$ is defined based on Mercer's condition. Hence, any kernel method which satisfies Mercer's condition can be adopted as the kernel for the proposed classifier. For initial N_0 sample, kernel matrix will be defined as:

$$\boldsymbol{\Phi} = \boldsymbol{\Phi_0} = \left(\begin{bmatrix} \Omega_{11} & \Omega_{12} & \dots & \Omega_{1N_0} \\ \Omega_{21} & \Omega_{22} & \dots & \Omega_{2N_0} \\ \vdots & \vdots & \ddots & \vdots \\ \Omega_{N_0 1} & \Omega_{N_0 2} & \dots & \Omega_{N_0 N_0} \end{bmatrix} + \frac{1}{\lambda} \boldsymbol{I} \right) \tag{3.13}$$

$$\boldsymbol{P} = \boldsymbol{P_0} = \boldsymbol{\Phi_0^{-1}} = \left(\begin{bmatrix} \Omega_{11} & \Omega_{12} & \dots & \Omega_{1N_0} \\ \Omega_{21} & \Omega_{22} & \dots & \Omega_{2N_0} \\ \vdots & \vdots & \ddots & \vdots \\ \Omega_{N_0 1} & \Omega_{N_0 2} & \dots & \Omega_{N_0 N_0} \end{bmatrix} + \frac{1}{\lambda} \boldsymbol{I} \right)^{-1} \tag{3.14}$$

3.2 Metacognition

We use the three basic meta cognitive principles of what-to-learn, how-to-learn, when-to-learn. The new samples are passed through What-to-learn part if the sample is accepted by what-to-learn it is passed to how-to-learn part.

3.2.1 What To Learn

When a new sample arrives output given by current model is calculated and projected on 1. If the new training sample has potential drift the output will be at a higher distance from 1 and if the new training sample doesn't have any concept drift it would lie in our current model. Thus, not providing any additional information to our model. In such cases evolving our model with these samples won't have any effect and it will be a waste of time. So, we reject such samples. We define a threshold distance from 1 before which we will delete the new sample.

$$Projection = \textbf{Output} - 1 \tag{3.15}$$

$$Projection < \alpha \tag{3.16}$$

The result of What-to-learn depends on this value of alpha. If it is very close to 1 then there won't be many sample deleted and we would be training on already learned sample. If it is very far away from 1 we will be loosing the samples that have drift. Empirically, we found this value to be 0.3. What-to-learn prevents evolving of model with samples already learned, and thus, helping us in reducing over-training and the computational work required.

3.2.2 How To Learn

The sample that is accepted by What to learn is passed to How to learn. This section can be divided in two parts:

1. ***Sample Delete Strategy***:What past sample needs to be pruned.
2. ***Model Update Strategy***:How to update the current model to learn the new knowledge that the sample passed from previous section represents.

3.2.2.1 Decremental Learning

Selecting The Sample Each sample has its value age value.Each sample is assigned a confidence value. When the new sample arrives the sample with least importance is unlearned and new sample is added to the model using block inverse.

Confidence is defined as below:

$$\overline{X} = \frac{\sum_{i=1}^{N}(X_i)}{N} \tag{3.17}$$

$$C_i = (\overline{X} - X_i)(\overline{X} - Xi)` + \theta * age_i \tag{3.18}$$

The value of constant theta is determined empirically. The sample with least confidence value is deleted as described below.

Deleting The Sample During online learning, data increases continuously and it creates two issue i.e. how the algorithm will learn continuously (i) if the distribution of training samples change and (ii) if the memory of system exhausted as memory can't be infinite. Both issues are addressed by forgetting mechanism with a sliding window as shown in Fig. 3.3. This mechanism unlearns the old or irrelevant samples by unlearning the trained model. Further, relearning on new samples can be done by online learning as discussed in previous sections.

Forgetting Mechanism for KRR based One-Class Classifier:
Suppose, we have currently $\boldsymbol{P_{curr}} = \boldsymbol{\Phi_{curr}^{-1}} \in \boldsymbol{\Re^{s \times s}}$ is the inverse of current kernel matrix $\boldsymbol{\Phi_{curr}} \in \boldsymbol{\Re^{s \times s}}$. Now, we want to remove the impact of learning of old f samples. In this mechanism, two things viz., kernel matrix and the inverse of that kernel matrix, needs to be updated before moving to learning of new samples. Modified kernel matrix $\boldsymbol{\phi_{new}}$ can be simply generated by removing rows and columns of the corresponding samples from the current kernel

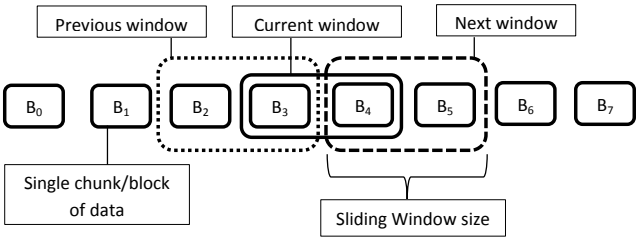


FIGURE 3.2: Illustration of Sliding Window for a Given Data Stream

matrix $\boldsymbol{\Phi_{curr}}$ and modified inverse matrix $\boldsymbol{P_{new}}$ are calculated after removing (forgetting) few samples from $\boldsymbol{\Phi_{curr}}$ as follows:

$$\boldsymbol{P_{curr}} = \boldsymbol{\Phi_{curr}^{-1}} = \begin{bmatrix} \boldsymbol{F_{11}} & \boldsymbol{F_{12}} \\ \boldsymbol{F_{12}^T} & \boldsymbol{R_{22}} \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{Fi_{11}} & \boldsymbol{Fi_{12}} \\ \boldsymbol{Fi_{12}^T} & \boldsymbol{Ri_{22}} \end{bmatrix} \tag{3.19}$$

$$\begin{bmatrix} \boldsymbol{Fi_{11}} & \boldsymbol{Fi_{12}} \\ \boldsymbol{Fi_{12}^T} & \boldsymbol{Ri_{22}} \end{bmatrix} \begin{bmatrix} \boldsymbol{F_{11}} & \boldsymbol{F_{12}} \\ \boldsymbol{F_{12}^T} & \boldsymbol{R_{22}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \tag{3.20}$$

Suppose, if we need to delete $\boldsymbol{F_{11}, F_{12}}$ and $\boldsymbol{F_{12}^T}$ from $\boldsymbol{\Phi_{curr}}$ and calculate the inverse of the remaining block $\boldsymbol{R_{22}}$ by reusing the currently available inverse $\boldsymbol{P_{curr}}$. $\boldsymbol{R_{22}^{-1}}$ is calculated from (3.20) by multiplying the (3.21) on both sides as follows:

$$\begin{bmatrix} \boldsymbol{I} & 0 \\ -\boldsymbol{Fi_{11}^{-1}}\boldsymbol{Fi_{12}^T} & \boldsymbol{I} \end{bmatrix} \tag{3.21}$$

$$\begin{aligned} \begin{bmatrix} \boldsymbol{Fi_{11}} & \boldsymbol{Fi_{12}} \\ \boldsymbol{0} & \boldsymbol{Ri_{22}} - \boldsymbol{Fi_{12}}\boldsymbol{Fi_{11}^{-1}}\boldsymbol{Fi_{12}^T} \end{bmatrix} \begin{bmatrix} \boldsymbol{F_{11}} & \boldsymbol{F_{12}} \\ \boldsymbol{F_{12}^T} & \boldsymbol{R_{22}} \end{bmatrix} \\ = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ -\boldsymbol{Fi_{11}^{-1}}\boldsymbol{Fi_{12}^T} & \boldsymbol{I} \end{bmatrix} \end{aligned} \tag{3.22}$$

From (3.22), $\boldsymbol{R_{22}^{-1}}$ can be obtained after deletion of $\boldsymbol{F_{11}, F_{12}}$ and $\boldsymbol{F_{12}^T}$ from $\boldsymbol{\Phi_{curr}}$ as follows:

$$\begin{aligned} \left(\boldsymbol{Ri_{22}} - \boldsymbol{Fi_{12}}\boldsymbol{Fi_{11}^{-1}}\boldsymbol{Fi_{12}^T}\right)\boldsymbol{R_{22}} &= \boldsymbol{I} \\ \boldsymbol{P_{new}} = \boldsymbol{R_{22}^{-1}} &= \boldsymbol{Ri_{22}} - \boldsymbol{Fi_{12}}\boldsymbol{Fi_{11}^{-1}}\boldsymbol{Fi_{12}^T} \end{aligned} \tag{3.23}$$

3.2.2.2 Incremental Learning

Initially, $\boldsymbol{\Phi}$ and $\boldsymbol{P}$ will be equal to $\boldsymbol{\Phi_0}$ and $\boldsymbol{P_0}$ respectively. Here, $\boldsymbol{\Phi}$ represents kernel matrix and $\boldsymbol{P}$ represents inverse of this kernel matrix for all the arrived samples till now for training.

Here, $\boldsymbol{\Phi}$ and $\boldsymbol{P}$ will be updated continuously for any upcoming new samples X^v as per (3.24), where, $\boldsymbol{X^v}$= $\{(\boldsymbol{x}_1^v, c_1), (\boldsymbol{x}_2^v, c_2), ..., (\boldsymbol{x}_s^v, c_s)\}$ and $\boldsymbol{X^v} \subset \boldsymbol{X}$. $\boldsymbol{X^v}$ is just next chunk of the data. Current value of $\boldsymbol{\Phi}$ is represented as $\boldsymbol{\Phi_u}$, which is generated by using old samples $\boldsymbol{X^u} \subset \boldsymbol{X}$. Here, u and v simply denote old and new values respectively. Now, calculate $\boldsymbol{\Phi}$ after arrival of new sample as follows:

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\Phi_u} & \boldsymbol{\Phi_{u,v}} \\ \boldsymbol{(\Phi_{u,v})^T} & \boldsymbol{\Phi_v} \end{bmatrix} \tag{3.24}$$

Here, $\boldsymbol{\Phi}$ is combination of four block matrices. $\boldsymbol{\Phi_u}$ is the old value of $\boldsymbol{\Phi}$. Block matrix $\boldsymbol{\Phi_{u,v}}$ and $\boldsymbol{\Phi_v}$ in (3.24) are calculated as per (3.25), which is discussed below.

Let the number of samples processed till now be b and number of samples in the current chunk be s. b is initially equal to N_0. Update b and s each time when calculation starts for new samples. The block matrices $\boldsymbol{\Phi_v}$ and $\boldsymbol{\Phi_{u,v}}$ can be defined as follows:

$$\boldsymbol{\Phi_v} = \left(\begin{bmatrix} \boldsymbol{K}(x_1^v, x_1^v) & \dots & \boldsymbol{K}(x_1^v, x_s^v) \\ \vdots & \ddots & \vdots \\ \boldsymbol{K}(x_s^v, x_1^v) & \dots & \boldsymbol{K}(x_s^v, \boldsymbol{x_s^v}) \end{bmatrix} + \frac{1}{\lambda}\boldsymbol{I} \right) \tag{3.25}$$

$$\boldsymbol{\Phi_{u,v}} = \begin{bmatrix} \boldsymbol{K}(x_1^u, x_1^v) & \dots & \boldsymbol{K}(x_1^u, \boldsymbol{x_s^v}) \\ \vdots & \ddots & \vdots \\ \boldsymbol{K}(x_b^u, x_1^v) & \dots & \boldsymbol{K}(x_b^u, x_s^v) \end{bmatrix} \tag{3.26}$$

Now, value of $\boldsymbol{P}$ will be inverse of $\boldsymbol{\Phi}$ in (3.24) as follows:

$$\boldsymbol{P} = \boldsymbol{\Phi^{-1}} = \begin{bmatrix} \boldsymbol{\Phi_u} & \boldsymbol{\Phi_{u,v}} \\ \boldsymbol{(\Phi_{u,v})^T} & \boldsymbol{\Phi_v} \end{bmatrix}^{-1} \tag{3.27}$$

Further, compute the inverse in (3.27) using block matrix inverse [22].

$$\boldsymbol{S} = \boldsymbol{D^{-1}} = \begin{bmatrix} \boldsymbol{D_{11}} & \boldsymbol{D_{12}} \\ \boldsymbol{D_{21}} & \boldsymbol{D_{22}} \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{S_{11}} & \boldsymbol{S_{12}} \\ \boldsymbol{S_{21}} & \boldsymbol{S_{22}} \end{bmatrix} \tag{3.28}$$

$\boldsymbol{S}$ in (3.28) can be written as follows to obtain inverse: $\boldsymbol{H}$

$$\begin{bmatrix} \boldsymbol{D_{11}} & \boldsymbol{D_{12}} \\ \boldsymbol{D_{21}} & \boldsymbol{D_{22}} \end{bmatrix} \begin{bmatrix} \boldsymbol{S_{11}} & \boldsymbol{S_{12}} \\ \boldsymbol{S_{21}} & \boldsymbol{S_{22}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \tag{3.29}$$

where, $\boldsymbol{S_{21} = S_{12}^T}$ and $\boldsymbol{D_{21} = D_{12}^T}$.
After solving (3.29), following solution is obtained:

$$\begin{aligned}
\boldsymbol{S_{11}} &= \boldsymbol{(D_{11} - D_{12}D_{22}^{-1}D_{21})^{-1}} \\
\boldsymbol{S_{12}} &= \boldsymbol{-D_{11}^{-1}D_{12}(D_{22} - D_{21}D_{11}^{-1}D_{12})^{-1}} \\
\boldsymbol{S_{21}} &= \boldsymbol{-D_{22}^{-1}D_{21}(D_{11} - D_{12}D_{22}^{-1}D_{21})^{-1}} \\
\boldsymbol{S_{22}} &= \boldsymbol{(D_{22} - D_{21}D_{11}^{-1}D_{12})^{-1}}
\end{aligned}$$

Hence, (3.27) will be rewritten as follows:

$$\boldsymbol{P = \Phi^{-1}} = \begin{bmatrix} \boldsymbol{\Phi_u} & \boldsymbol{\Phi_{u,v}} \\ \boldsymbol{(\Phi_{u,v})^T} & \boldsymbol{\Phi_v} \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{P_{11}} & \boldsymbol{P_{12}} \\ \boldsymbol{P_{21}} & \boldsymbol{P_{22}} \end{bmatrix} \tag{3.30}$$

$\boldsymbol{P_{11}}$, $\boldsymbol{P_{12}}$, $\boldsymbol{P_{21}}$ and $\boldsymbol{P_{22}}$ in (3.30) can be written as follows:

$$\begin{aligned}
\boldsymbol{P_{11}} &= \boldsymbol{(\Phi_u - \Phi_{u,v}\Phi_v^{-1}\Phi_{u,v}^T)^{-1}} \\
\boldsymbol{P_{12}} &= \boldsymbol{-\Phi_u^{-1}\Phi_{u,v}P_{22}} \\
\boldsymbol{P_{21}} &= \boldsymbol{-\Phi_v^{-1}\Phi_{u,v}^T P_{11}} \\
\boldsymbol{P_{22}} &= \boldsymbol{(\Phi_v - \Phi_{u,v}^T\Phi_u^{-1}\Phi_{u,v})^{-1}}
\end{aligned} \tag{3.31}$$

$\boldsymbol{P_{11}}$ can be expanded by employing Woodbury formula[9] as:

$$\begin{aligned}
\boldsymbol{P_{11}} &= \boldsymbol{(\Phi_u - \Phi_{u,v}\Phi_v^{-1}\Phi_{u,v}^T)^{-1}} \\
&= \boldsymbol{\Phi_u^{-1} - \Phi_u^{-1}\Phi_{u,v}(\Phi_{u,v}^T\Phi_u^{-1}\Phi_{u,v}} \\
&\boldsymbol{+ \Phi_v^{-1})^{-1}\Phi_{u,v}^T\Phi_u^{-1}}
\end{aligned} \tag{3.32}$$

Woodbury formula[9] is employed instead of computing direct inverse because now we need to calculate two inverse i.e. Φ_u^{-1} and Φ_v^{-1} independently, where, Φ_u^{-1} is already computed in previous iteration. How it impacts time and storage complexity is described in the Section **??**. In a similar fashion, (3.31) can be explored for $\boldsymbol{P_{12}}$, $\boldsymbol{P_{21}}$ and $\boldsymbol{P_{22}}$.

3.3 Type-2 Fuzzy Kernel

When the data stream suffers from various uncertain conditions the performance of 'model' will be hindered. Hence, we use an Interval Type-2 Fuzzy Kernel for 'model' to tackle uncertainty. This will enable efficient classification even with data uncertainty.

3.3.1 Interval Design

Following is a Type-2 Fuzzy set $\tilde{A}$ that has a membership function such that the secondary membership is equal to one for all primary memberships:

$$\tilde{A} = \{(x,u), \mu_{\tilde{A}}(x,u) | \forall x \in A, \forall u \in J_x \subseteq [0,1], \mu_{\tilde{A}}(x,u) = 1\} \tag{3.33}$$

Such a Type 2 Fuzzy set is called an Interval Type 2 Fuzzy set(IT2). J_x, the primary membership function of a sample x, can be represented as:

$$J_x = [\overline{\mu}(x), \underline{\mu}(x)] \tag{3.34}$$

A similar concept is used to represent the kernel uncertainty, with J_{K^m} being the primary membership of the kernel K^m. Following is a Interval Type-2 Fuzzy Kernel set $\tilde{\urcorner}$:

$$\tilde{\urcorner} = \{(x,u), \mu_{\tilde{A}}(x,u) | \forall x \in \urcorner, \forall u \in J_x \subseteq [0,1], \mu_{\tilde{\urcorner}}(x,u) = 1\} \tag{3.35}$$

Different kernel models with different kernel parameters will be used to compute the upper and lower primary memberships. Hence, primary membership of K^m,J_{K^m}, will be:

$$J_{K^m} = [\overline{\mu}(K^m), \underline{\mu}(K^m)] \tag{3.36}$$

Now let $\{(K^m, P) | P \in \Delta\}$ be a set where P is the kernel parameter and K^m be a model from the set of kernels that satisfies the Mercer conditions. So a kernel model K^m will be tested with a J kernel parameters to make a kernel set:

$$\{(K^m, P_j^m)\}, j = 1, \ldots, J \tag{3.37}$$

Obtain a vector with values calculated using kernel computation between samples that are converted either column-by-column/row-by-row and denote it by W_j^m. Calculate the mean as follows that will denote the mean of output value kernel computation:

$$\overline{W^m} = \sum_{j=1}^{J} W_j^m \tag{3.38}$$

Now the mean variance can be calculated as follows:

$$V^m = \frac{\sum_{j=1}^{J}(W_j^m - \overline{W^m})^2}{J} \tag{3.39}$$

Let C_j^m describe the output of confidence of the output values of the kernels which as be obtained as follows:

$$C_j^m = 1 - \frac{(W_j^m - \overline{W^m})^2}{V^m} \tag{3.40}$$

So we can obtain a confidence value for any kernel computation. The modified kernel set that includes the confidence value of the kernel can be written as follows:

$$K^m, P_j^m, C_j^m, m = 1, \ldots, M, j = 1, \ldots, J \tag{3.41}$$

To calculate the fuzzy membership interval of the kernel model K^m we use the confidence values of the different parameters as follows:

$$\overline{u}(K^m) = max(C_j^m), j = 1, \ldots, J \tag{3.42}$$

$$\underline{u}(K^m) = min(C_j^m), j = 1, \ldots, J \tag{3.43}$$

3.3.2 Type Reduction

A centroid type reducer, as in [12], is used for the Type Reduction step of the Interval Type 2 Fuzzy kernels.

The upper Kernel, $\overline{K}$, is given by:

$$\overline{K} = \frac{\sum_{m=1}^{M} \overline{\mu}(K^m).K^m}{\sum_{m=1}^{M} \overline{\mu}(K^m)} \tag{3.44}$$

The lower Kernel, $\underline{K}$, is given by:

$$\underline{K} = \frac{\sum_{m=1}^{M} \underline{\mu}(K^m).K^m}{\sum_{m=1}^{M} \underline{\mu}(K^m)} \tag{3.45}$$

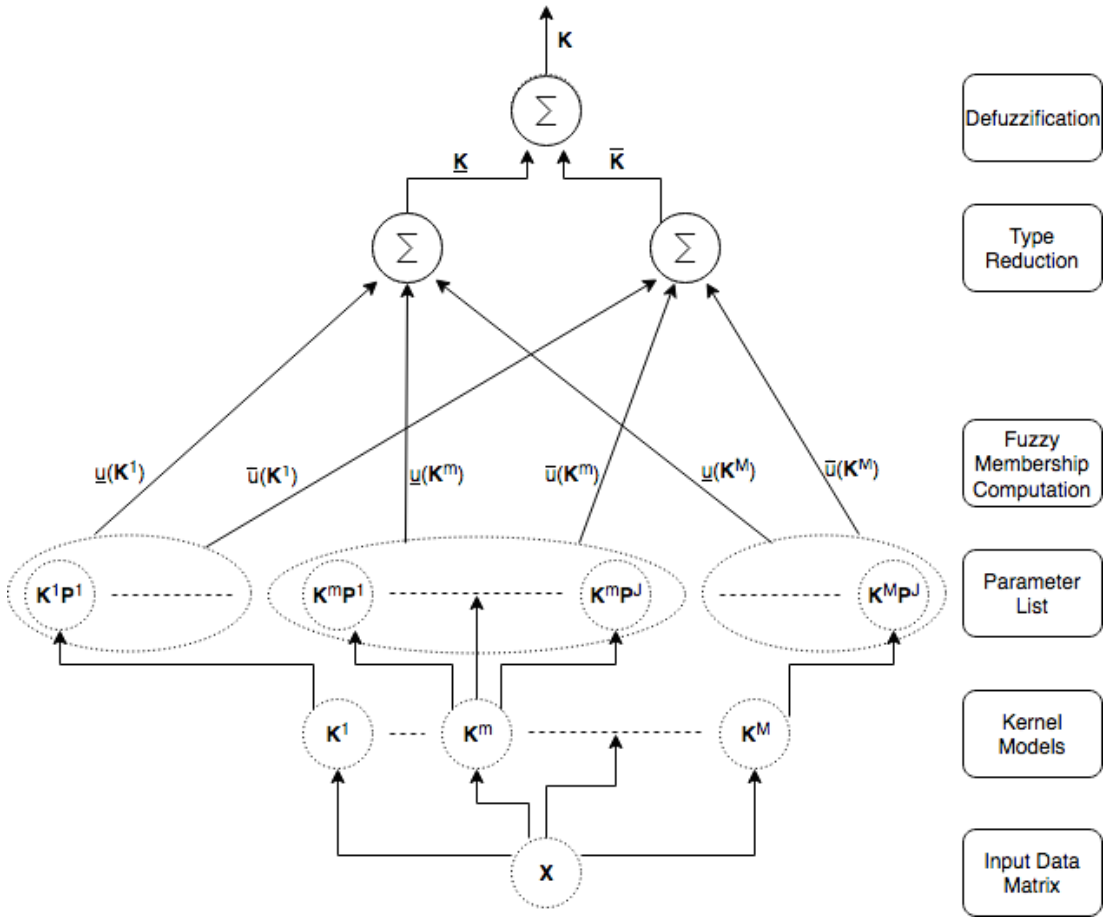


FIGURE 3.3: Type-2 Fuzzy Kernel Flowchart

3.3.3 Defuzzification

To defuzzify the primary membership interval $[\overline{K}, \underline{K}]$ obtained from Type Reduction, we use the mean of $\overline{K}$ and $\underline{K}$ as the output. Hence, the final defuzzified kernel can be written as:

$$K = \frac{\overline{K} + \underline{K}}{2} \tag{3.46}$$

3.4 Pseudo-Code

Algorithm 1 ***ORK-OCELM(B)***:Boundary Based Approach

Input: Training set $\boldsymbol{X}= (\boldsymbol{x}_1, c_1), (\boldsymbol{x}_2, c_2), ..., (\boldsymbol{x}_{N_0}, c_{N_0}), ...,$
$(\boldsymbol{x}_t, c_t), ...$

Output: Whether Target or Outlier corresponding to each sample

Initialization Phase

1: Pass initial set of samples $\boldsymbol{X_0}, \boldsymbol{R_0}$ to the classifier as: $\{(\boldsymbol{x}_1, R_1), (\boldsymbol{x}_2, R_2), ..., (\boldsymbol{x}_{N_0}, R_{N_0})\}$
// For first chunk of N_0 samples, following steps are required

2: Employ kernel feature mapping: $\boldsymbol{\Phi_0 = \phi(X_0)}$.

3: Output Weight $\boldsymbol{\beta_0}$ for $\boldsymbol{(X_0, R_0)}$:

$$\boldsymbol{\beta_0} \leftarrow \boldsymbol{P_0 R_0}$$
$$\boldsymbol{P_0} \leftarrow \boldsymbol{\Phi_0^{-1}}$$
$$b \leftarrow N_0$$

End of Initialization Phase

// For second chunk onwards, following steps are required

$$\boldsymbol{\Phi} \leftarrow \boldsymbol{\Phi_0}$$
$$\boldsymbol{P} \leftarrow \boldsymbol{P_0}$$

4: **for** $i = 1$ to last chunk of data in $\boldsymbol{X}$ **do**

5: Size of chunk at the current stage $= s$

6: Update the final kernel matrix $\boldsymbol{\Phi}$ and its inverse $\boldsymbol{P}$ in two steps:

Step 1: Forgetting Phase:

7: Remove the impact of s old samples from the current inverse $\boldsymbol{P}$ using (3.23):

$$\boldsymbol{P_{new}} = \boldsymbol{R_{22}^{-1}} = \boldsymbol{Ri_{22}} - \boldsymbol{Fi_{12}} \boldsymbol{Fi_{11}^{-1}} \boldsymbol{Fi_{12}^{T}}$$

8: Update the kernel matrix $\boldsymbol{\Phi}$ by removing those rows and columns which were generated due to those old s samples.

Step 2: Retraining Phase

9: Update the kernel matrix $\boldsymbol{\Phi}$ as per (3.24):

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\Phi_u} & \boldsymbol{\Phi_{u,v}} \\ \boldsymbol{(\Phi_{u,v})^T} & \boldsymbol{\Phi_v} \end{bmatrix} \tag{3.24}$$

10: Calculate $\boldsymbol{\Phi_v}$ for i^{th} chunk by using (3.25)

11: Calculate $\boldsymbol{\Phi_{u,v}}$ for i^{th} chunk by using (3.26)

12: Compute the inverse of updated kernel matrix $\boldsymbol{\Phi}$, $\boldsymbol{P = \Phi^{-1}}$ using block inverse as discussed in (3.28)-(3.32)

13: $b = b + s$

14: Update the Output Weight as per the value of $\boldsymbol{R}$ and updated $\boldsymbol{P}$:

$$\boldsymbol{\beta = PR}$$

15: Compute the predicted value by using output function $f(X_k)$,

16: Calculate distances(d) between predicted value of training sample and R:

$$\boldsymbol{d(x_t)} = |\boldsymbol{f(x_t)} - R_t| = \left|\hat{R}_t - R_t\right|$$

17: Sort the distances in decreasing order

18: Compute θ: $\theta = d(\lfloor \eta * N \rfloor)$

19: Use (3.4) to decide whether a new sample z belongs to target or not

$$Sign(\theta - d(\boldsymbol{z})) = \begin{cases} 1, & \boldsymbol{z} \text{ is classified as target} \\ -1, & \boldsymbol{z} \text{ is classified as outlier} \end{cases} \tag{3.4}$$

20: **end for**

Chapter 4

Setup and Implementation

This chapter discuss about the testing of the algorithm on artificial dataset and real dataset. The first section talks about the specification of datasets whose results and comparative analysis are portrayed in next chapter

4.1 Datasets

Datasets used for testing can be classified in two types-

1. ***Artificial datasets*** - Data was created artificially to test drift with different direction and speed. Details of each dataset is provided in Table 4.1.

2. ***Real datasets*** - Data was taken from real world with abrupt concept drift.

4.1.1 Environment

- **Software Specifications**:
  - Language used - Matlab
  - IDE - Matlab 2016b
- **Hardware Specifications**:

  - Processor - Intel i5 processor

  - RAM - 4 GB

TABLE 4.1: Datasets Description

Dataset Name	No. of Classes	No. of Attributes	Drift	No. of Samples
1CDT[21]	2	2	400	16,000
2CDT[21]	2	2	400	16,000
1CHT[21]	2	2	400	16,000
2CHT[21]	2	2	400	16,000
4CR[21]	4	2	400	144,400
4CRE-V1[21]	4	2	1,000	125,000
4CRE-V2[21]	4	2	1,000	183,000
5CVT[21]	5	2	1,000	40,000
1CSurr[21]	2	2	600	55,283
4CE1CF[21]	5	2	750	173,250
FG-2C-2D[6]	2	2	2,000	200,000
UG-2C-2D[7]	2	2	1,000	100,000
UG-2C-3D[7]	2	3	2,000	200,000
UG-2C-5D[7]	2	5	2,000	200,000
MG-2C-2D[7]	2	2	2,000	200,000
GEARS-2C-2D[7]	2	2	2,000	200,000

TABLE 4.2: Real Datasets Description

Dataset Name	Drift	No. of Samples
Electricity[14]	Unknown	45,312
Keystroke[14]	Unknown	1,600
Abalone[14]	Unknown	4,177
Sea[14]	Unknown	16,000
Poker[14]	Unknown	8,29,201

4.2 Implementation

In this section we consider the code implementation of the proposed algorithm on MATLAB 2016b. It is described in Section 3.4. Here, Section 4.2.1 covers the implementation of the algorithm to generate the fuzzy kernel from a provided set of kernel models and corresponding parameters. It is followed by Section 4.2.2, the code for the forgetting mechanism, whose formulation was covered in Section 3.2.

4.2.1 Type-2 Fuzzy Kernel

Implementation of Type-2 fuzzy kernel matrix. Input to the function is the input data and kernel parameters.

```
function kernel = fuzzy_kernel(Xtrain, kernel_list, Xtest)

    disp(nargin);
```

```
if nargin<3
        sz = size(Xtrain*Xtrain');
    else
        sz = size(Xtrain*Xtest');
end

K_upper = zeros(sz);
K_lower = zeros(sz);
u_upper_sum = 0;
u_lower_sum = 0;

M = 0;
for kernel_type_cell = keys(kernel_list)
    M = M + 1;
    kernel_type = char(kernel_type_cell);
    pars_list = kernel_list(kernel_type);

    J = 0;
    K = {};
    for kernel_pars = pars_list
        J = J + 1;
        if nargin<3
            K{J} = kernel_matrix(Xtrain, kernel_type, kernel_pars);
        else
            K{J} = kernel_matrix(Xtrain, kernel_type, kernel_pars, Xtest);
        end
    end

    unique = 0;
    for j = 2:J
       if K{1} ~= K{j}
           unique = 1;
       end
    end

    if unique == 0
        K_upper = K{1};
        K_lower = K{1};
        u_upper_sum = 1;
        u_lower_sum = 1;
    else

        C = zeros(1,J);
        for i = 1:sz(1)
            W_mean = zeros(1, sz(2));
            for j = 1:J
                W_mean = W_mean + K{j}(i,:);
            end
            W_mean = W_mean/J;

            MV = 0;
            for j = 1:J
               MV = MV + (K{j}(i,:) - W_mean)*(K{j}(i,:) - W_mean)';
            end
            MV = MV/J;
```

```
                if(MV==0)

                end

                for j = 1:J
                    x = abs(MV - (K{j}(i,:) - W_mean)*(K{j}(i,:) - W_mean)')/MV;
                    if (~(0<x&&x<=1+10^-9))
                       keyboard
                    end
                    C(j) = C(j) + 1 - x;
                end
            end

            for j = 1:J
               C(j) = C(j)/sz(1);
            end

            [u_upper, max_idx] = max(C);
            [u_lower, min_idx] = min(C);

            K_upper = K_upper + u_upper * K{max_idx};
            K_lower = K_lower + u_lower * K{min_idx};
            u_upper_sum = u_upper_sum + u_upper;
            u_lower_sum = u_lower_sum + u_lower;

        end
    end

    if(u_upper_sum~=0)
        K_upper = K_upper/u_upper_sum;
    end
    if(u_lower_sum~=0)
        K_lower = K_lower/u_lower_sum;
    end

    kernel = (K_upper + K_lower)/2.0;

    if isnan(kernel)
       assert(0);
    end

end
```

4.2.2 Forgetting Mechanism

Following code unlearns the old sample and learns the new sample.

```
function [W] = forget_mechanism(W, forgetting_factor)

%%% get the data from a trained_model as it is in the form of prmapping
```

```
%============================ forgetting mechanism ============================
if ~isempty(W)
    training_a = W.training_a;
    fracrej = W.fracrej;
    HTrain = W.HTrain;
    R = W.R;
else
    error('Pass the trained model as structure trained_model is empty');
end

ind_del = forgetting_factor; %%% Index of those samples which needs to be deleted/removed. Deleted as p
ind_remain = setdiff(1:size(R,1),ind_del); %%% Index of those samples which remains after deleting few
Rnew = R(ind_remain,ind_remain) - R(ind_remain,ind_del) * inv(R(ind_del,ind_del)) * (R(ind_remain,ind_d

%%% Just clear before updating the necessary variable in the structure of trained model
W.training_a = [];
W.threshold = [];
W.HTrain = [];
W.beta = [];
W.R = [];

training_b=training_a; HTrain_b = HTrain;
%%%% Update the variable
%%%% Calculate updated Beta i.e. OutputWeight by Rnew %%%
training_a = training_a(ind_remain,:);
HTrain = HTrain(ind_remain,ind_remain);

%%%%% Just checking whether forgetting is working properly or not
%             a1inv=inv(HTrain+speye(size(HTrain,1))/1);
%             %chk1=round(Rnew,3); chk2=round(a1inv,3);
%             chk1=Rnew; chk2=a1inv;
%          if ~isequal(chk1,chk2)
%         keyboard
%          end
%%%% End of checking

[m,~] = size(training_a);
T = ones(m,1);
beta = Rnew * T;
Y=HTrain * beta;
out = abs(Y-T);
[sout,~] = sort(out);

W.training_a = training_a;
W.HTrain = HTrain;
W.threshold = sout(ceil(m*(1-fracrej)),1);
W.beta = beta;
W.R = Rnew;
end
```

The results, compared with present algorithms, discussed in next chapter.

Chapter 5

Experimental Results

Here we discuss results obtained and compare performance with other online adaptive one class classification algorithms. We evaluate the proposed model and provide detailed discussion of findings to help the ongoing research of online learning and one class classification.

We tested the proposed model against other one class classifier for online learning with concept drift. Other algorithms don't consider fuzziness in the data. Algorithms evaluated includes OKPCA, incSVDD, AAKELM. These algorithms were implemented accoeding to their respective papers. Exhaustive testing was used to tune the parameters of these algorithms.

Datasets taken for testing purpose fall in two categories-artificial datasets and real datasets. Artificial datasets are public benchmark datasets provided to evaluate learning algorithms in non stationary environment. These datasets have incremental and gradual drifts. While these datasets were generated for multi class classification, we choose one class as our standard samples and consider the rest of the classes as outlier.

Table 5.1 summarizes the result for artificial datasets. Overall we note that proposed model provides better results across all algorithms tested.

Drift only in anomalies: In the datasets 1CHT, 1CDT, 4CE1CF only the anomalies are changing characteristics, drift is only observed in anomaly class the normal class remains static, thus in this case static algorithms also give good results as can be seen in Table 5.1.

Drift in both the normal class and anomalies: For Gears-2C-2D, 4CRE-V1, 4CRE-V2 and 5CVT datasets, drift occurs in both the normal and outlier classes which causes the static algorithms to fail. Online algorithms, like OKPCA and proposed algorithm gives better accuracy as they detect the drift and update the model accordingly. Accuracy of static algorithms plunges on the occurence of the drift while online algorithm catches fast as can be seen in Fig. 5.1. Proposed algorithm outperforms other online algorithms in these datasets.

TABLE 5.1: Accuracy of all algorithms on Artificial datasets

Datasets	McTOC	McTOC	AAKELM (S)	incSVDD	OKPCA	Block_Size	Sliding Window Size
1CDT	94.92	97.63	**97.79**	95.85	96.22	50	150
2CDT	90.42	**90.49**	54.24	88.17	87.34	50	150
1CHT	94.69	**96.58**	93.11	94.89	94.97	50	150
2CHT	**82.86**	79.92	55.61	78.06	77.50	50	150
4CR	97.18	**98.92**	69.17	97.26	98.34	50	150
4CRE-V1	96.01	**97.06**	71.44	96.10	95.40	50	150
4CRE-V2	**93.38**	92.26	63.76	92.25	88.68	50	150
5CVT	**89.48**	89.14	68.68	88.55	86.38	50	150
1CSurr	96.75	**98.13**	66.06	95.70	96.27	50	150
4CE1CF	97.28	**98.04**	97.03	96.32	96.75	50	150
UG-2C-2D	92.08	**92.49**	51.13	89.54	89.36	50	150
MG-2C-2D	**88.2**	87.88	47.56	84.51	83.79	50	150
FG-2C-2D	86.5	**88.32**	66.49	84.72	84.09	50	150
UG-2C-3D	86.1	87.24	52.99	**87.45**	84.11	50	150
UG-2C-5D	83	**84.22**	56.74	83.10	77.79	50	150
GEARS-2C-2D	93.81	**96.11**	83.22	92.99	87.37	50	150

TABLE 5.2: Accuracy of all algorithms on Real datasets

Datasets	McToc	McToc	AAKELM (S)	incSVDD	OKPCA	Block_Size	Sliding Window Size
ELEC	58.15	61.16	55.14	**61.25**	58.67	200	2500
Keystroke	97.03	**97.37**	18.83	97.31	85.86	50	150
Abalone	73.42	**77.37**	57.66	75.32	66.77	50	150
sea	**76.86**	76.1254	44.34	67.75	57.74	50	150
Poker	**77.29**	77.8	49.9	73.29	66.95	50	150

For 2CDT and 2CHT datasets, the drift is non periodic and increases continuously. Static algorithm perform worst in this case giving the accuracy of just more than 50 %. The proposed algorithm outperforms other online algorithm by a large margin here as seen in Table 5.1.

Table 5.2 provides the results for real datasets. Real datasets provide abrupt concept drift and fuzziness in the data. Data source range from Electricty, change in keystroke of a user, poker, sea and abalone dataset

Electricty data is collected from real world markets where the price varies with respect to the demand and the supply. Task here is to find the change of price relative to the moving average. Proposed OCC achieves better results than other algorithms.

Poker has largest number of samples of the datasets presented here. It originally had 10 classes, one non poker hand class and others poker hand. Al poker hand classes were treated as single class for one class classification. Task was to identify the poker hand where the model was trained on non poker hand. Proposed model outperformed all other algorithm by a large margin.

Keystroke is another real world dataset, which represents the rhythm of user typing on keyboard. Here user authentication is done based on the typing rythm of user instead of the traditional way of user id and password. With type typing rhythm of user changes which result in drift. One user data is taken for learning and further authentication of user is done through the trained

TABLE 5.3: Number of samples rejected in Online learning

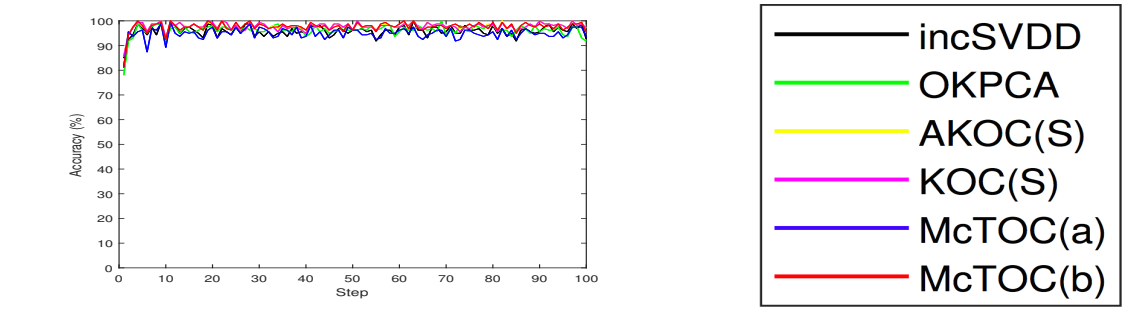
Datasets	Count Rejected
1CDT	454
2CDT	450
1CHT	455
2CHT	448
4CR	2876
4CRE-V1	2488
4CRE-V2	3649
5CVT	472
1CSurr	1127
4CE1CF	3450
FG-2C-2D	3989
UG-2C-2D	2876
UG-2C-3D	14122
UG-2C-5D	5795
MG-2C-2D	5736
GEARS-2C-2D	14084
Real World	
ELEC	1198
Keystroke	20
Abalone	41
sea	5341
Poker	32997

model. Result of the proposed algorithm is way better than OKPCA and is similar to that of incSVDD.

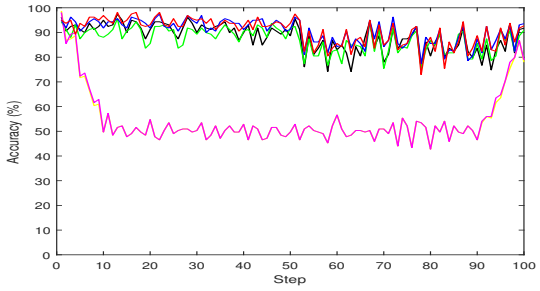
The Abalone dataset is taken from the UCI repository [14]. Original dataset requires to predict the age of Abalone from physical measurements. It had 29 classes. Classes 9 to 29 were merged into 1 to give normal class and class 1 to 8 were taken as anomaly. The proposed algorithm performs better than both OKPCA and incSVDD.

As the data comes in a stream and in many cases it is an infinite stream, learning of new sample is not needed when new sample doesn't have any concept drift. This is where the evaluation of What-to-learn comes, we need to reject some of the samples. Rejecting these sample leads to a significant speed up of the algorithm which is the requirement of online learning. Table 5.3 gives us the count of number of samples rejected in each dataset.

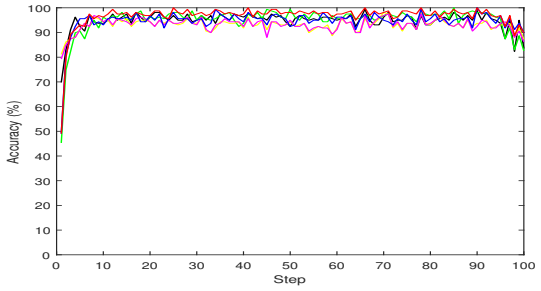
In most of the datasets, our algorithm beats other algorithm by a good margin and produce similar results for other datasets. Real data sets possesses higher level of fuzziness in the data, Type 2 Fuzzy produces good results in these datasets as can be seen in Table [**real**] proposed algorithm beats other algorithms by far greater margin than in artificial datasets.



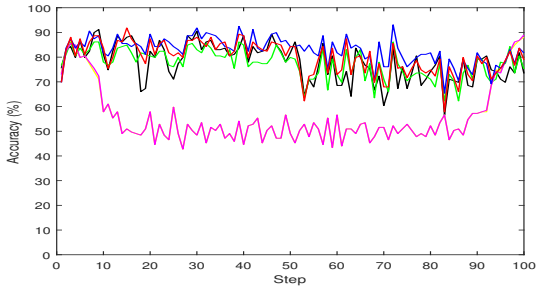
(A) 1CDT



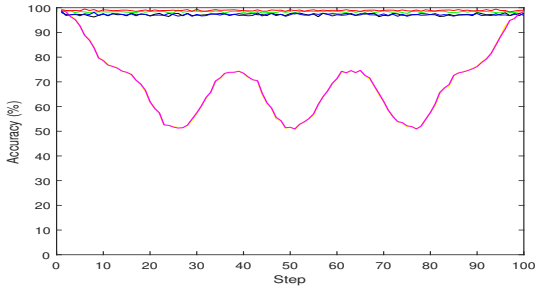
(B) 2CDT



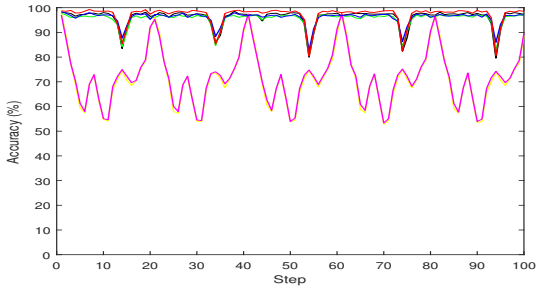
(C) 1CHT



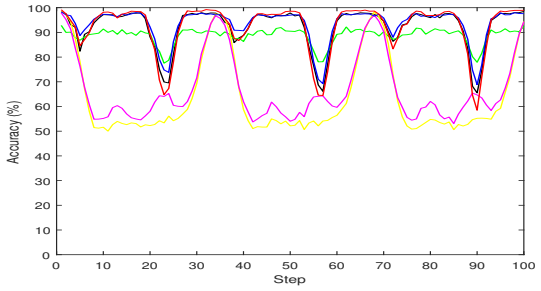
(D) 2CHT



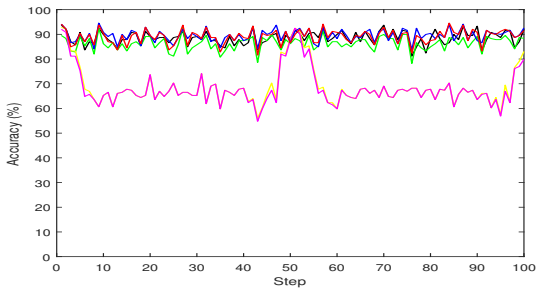
(E) 4CR



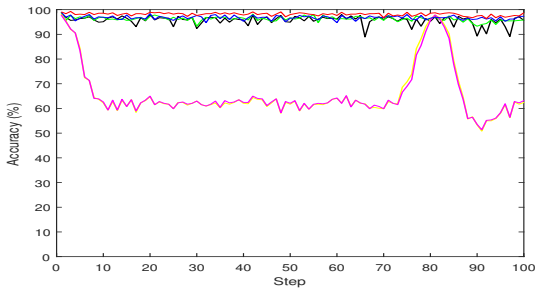
(F) 4CRE-V1



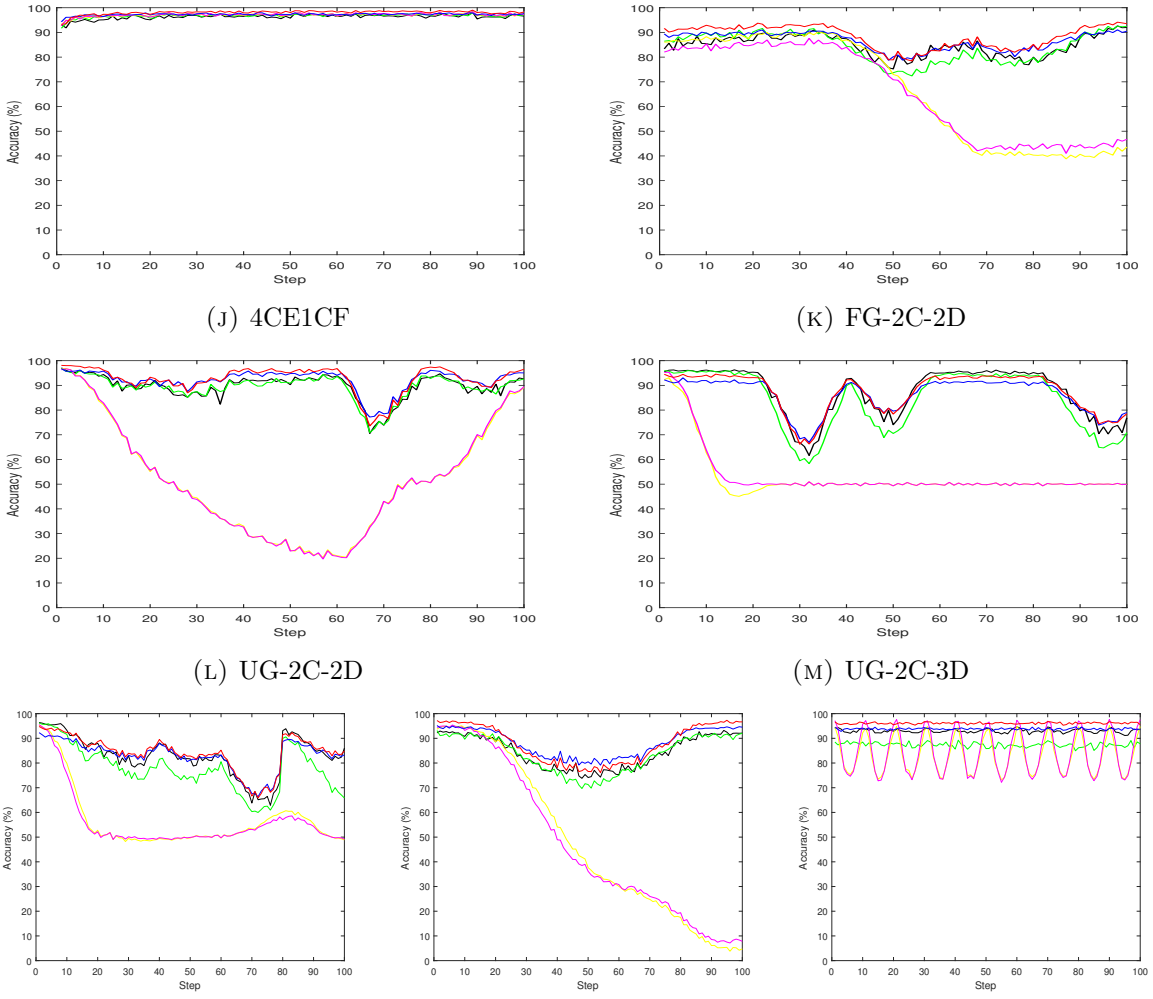
(G) 4CRE-V2



(H) 5CVT



(I) 1CSurr



(J) 4CE1CF (K) FG-2C-2D

(L) UG-2C-2D (M) UG-2C-3D

(N) UG-2C-5D (O) MG-2C-2D (P) GEARS-2C-2D

FIGURE 5.1: Performance over 16 datasets in 100 steps

Offline classifier can be considered a special case of online classifier, so the algorithm proposed works for offline classifer too.

Chapter 6

Conclusion and Future Work

With rising number of devices, transactions etc, the amount of data is immense and is changing characteristic over time. To handle fraud, failure in large systems, anomaly detection has become indispensable.

Kernel Ridge Regression (KRR) based One-class Classification (OCC) has been enhanced for non-stationary environment. This thesis proposes a KRR based one-class classifier, which can adapt the non-stationarity present in the data stream. Type-2 Fuzzy logic along with meta-cognition is employed with KRR based one-class classifier for handling data in non-stationary environment. The meta-cognition enables the one-class classifier for the decision of which inputs to train and how to train. And Type-2 fuzzy logic generates a Type-2 fuzzy kernel which helps building the model which makes its decision boundary adaptable to new incoming data. Moreover, a forgetting mechanism is employed under the Meta-cognition framework, which boosts the ability of the classifier to eliminate the impact of irrelevant data as well as assures the execution of the proposed method within the limited memory consumption. The proposed method is tested on different types of non-stationary artificial and real datasets to verify its behaviour under various drifting conditions of normal and outlier samples, and compared the performance with the state-of-the-art kernel based online one-class classifiers.

We achieved far better results than present algorithms. We, for the first time, handled fuzziness in the data stream. Meta-cognition handles the non stationarity of the data very well. We proposed rejection of new samples based on knowledge of current system which further enhanced the result and time.

For future work, Number of samples rejected by What-to-learn can be improved significantly. Currently, we empirically set the parameters in What-to-learn. An automatic way based on the number of samples rejected and average difference in projection can be used to automate and vary the parameters while learning.

Bibliography

[1] A. Argyriou, C. A. Micchelli, and M. Pontil. "When is there a representer theorem? Vector versus matrix regularizers". In: *Journal of Machine Learning Research* 10.Nov (2009), pp. 2507–2529.

[2] G. S. Babu and S. Suresh. "Sequential Projection-Based Metacognitive Learning in a Radial Basis Function Network for Classification Problems". In: *IEEE Transactions on Neural Networks and Learning Systems* 24.2 (2013), pp. 194–206. ISSN: 2162-237X.

[3] C.M. Bishop. "Novelty Detection and Neural Network Validation". In: 141 (Sept. 1994), pp. 217 –222.

[4] Francesco De Comité et al. "Positive and Unlabeled Examples Help Learning". In: *Algorithmic Learning Theory: 10th International Conference, ALT'99 Tokyo, Japan, December 6–8, 1999 Proceedings*. Ed. by Osamu Watanabe and Takashi Yokomori. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 219–230. ISBN: 978-3-540-46769-4. DOI: 10.1007/3-540-46769-6_18. URL: https://doi.org/10.1007/3-540-46769-6_18.

[5] François Denis, Rémi Gilleron, and Fabien Letouzey. "Learning from positive and unlabeled examples". In: *Theoretical Computer Science* 348.1 (2005). Algorithmic Learning Theory (ALT 2000), pp. 70 –83. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2005.09.007. URL: http://www.sciencedirect.com/science/article/pii/S0304397505005256.

[6] G. Ditzler and R. Polikar. "Incremental Learning of Concept Drift from Streaming Imbalanced Data". In: *IEEE Transactions on Knowledge and Data Engineering* 25.10 (2013), pp. 2283–2301. ISSN: 1041-4347.

[7] K. B. Dyer, R. Capo, and R. Polikar. "COMPOSE: A Semisupervised Learning Framework for Initially Labeled Nonstationary Streaming Data". In: *IEEE Transactions on Neural Networks and Learning Systems* 25.1 (2014), pp. 12–26. ISSN: 2162-237X.

[8] P. E. Gill, W. Murray, and M. H Wright. "Practical optimization". In: (1981).

[9] Nicholas J Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002.

[10] Randy Isaacson and Frank Fujita. "Metacognitive knowledge monitoring and self-regulated learning: Academic success and reflections on learning". In: 6 (Sept. 2006), pp. 39–55.

[11] Nathalie Japkowicz. "Concept Learning in the Absence of Counterexamples: An Autoassociation-based Approach to Classification". AAI9947599. PhD thesis. New Brunswick, NJ, USA, 1999. ISBN: 0-599-49831-5.

[12] N. N. Karnik, J. M. Mendel, and Qilian Liang. "Type-2 fuzzy logic systems". In: *IEEE Transactions on Fuzzy Systems* 7.6 (1999), pp. 643–658. ISSN: 1063-6706.

[13] Q. Leng et al. "One-Class Classification with Extreme Learning Machine". In: *Mathematical Problems in Engineering* (2014), pp. 1–11.

[14] M. Lichman. *UCI Machine Learning Repository*. 2013. URL: `http://archive.ics.uci.edu/ml`.

[15] Larry M. Manevitz and Malik Yousef. "One-class Svms for Document Classification". In: *J. Mach. Learn. Res.* 2 (Mar. 2002), pp. 139–154. ISSN: 1532-4435. URL: `http://dl.acm.org/citation.cfm?id=944790.944808`.

[16] M.M. Moya, M.W. Koch, and L.D. Hostetler. "One-class classifier networks for target recognition applications". In: (Jan. 1993).

[17] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN: 1-55860-238-0.

[18] Gunter Ritter and María Teresa Gallegos. "Outliers in statistical pattern recognition and an application to automatic chromosome classification". In: *Pattern Recognition Letters* 18.6 (1997), pp. 525 –539. ISSN: 0167-8655. DOI: `https://doi.org/10.1016/S0167-8655(97)00049-4`. URL: `http://www.sciencedirect.com/science/article/pii/S0167865597000494`.

[19] G. Sateesh Babu and S. Suresh. "Meta-cognitive Neural Network for Classification Problems in a Sequential Learning Framework". In: *Neurocomput.* 81 (Apr. 2012), pp. 86–96. ISSN: 0925-2312. URL: `http://dx.doi.org/10.1016/j.neucom.2011.12.001`.

[20] Bernhard Schölkopf et al. "Support Vector Method for Novelty Detection". In: *Proceedings of the 12th International Conference on Neural Information Processing Systems*. NIPS'99. Denver, CO: MIT Press, 1999, pp. 582–588. URL: `http://dl.acm.org/citation.cfm?id=3009657.3009740`.

[21] V. M. A. Souza et al. "Data Stream Classification Guided by Clustering on Nonstationary Environments and Extreme Verification Latency". In: *Proceedings of SIAM International Conference on Data Mining (SDM)*. 2015, pp. 873–881.

[22] Volker Strassen. "Gaussian elimination is not optimal". In: *Numerische mathematik* 13.4 (1969), pp. 354–356.

[23] K. Subramanian, S. Suresh, and N. Sundararajan. "A Metacognitive Neuro-Fuzzy Inference System (McFIS) for Sequential Classification Problems". In: *IEEE Transactions on Fuzzy Systems* 21.6 (2013), pp. 1080–1095. ISSN: 1063-6706.

[24] D. M. J. Tax. "One-class classification; Concept-learning in the absence of counter-examples". In: *ASCI dissertation series* 65 (2001).

[25] David M. J. Tax and Robert P. W. Duin. "Uniform Object Generation for Optimizing One-class Classifiers". In: *J. Mach. Learn. Res.* 2 (Mar. 2002), pp. 155–173. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=944790.944809.

[26] David M.J. Tax and Robert P.W. Duin. "Support Vector Data Description". In: *Machine Learning* 54.1 (2004), pp. 45–66. ISSN: 1573-0565. DOI: 10.1023/B:MACH.0000008084.60811.49. URL: https://doi.org/10.1023/B:MACH.0000008084.60811.49.

[27] X. Wang and M. Han. "Online sequential extreme learning machine with kernels for nonstationary time series prediction". In: *Neurocomputing* 145 (2014), pp. 90–97.

[28] Hwanjo Yu. "Single-Class Classification with Mapping Convergence". In: *Machine Learning* 61.1 (2005), pp. 49–69. ISSN: 1573-0565. DOI: 10.1007/s10994-005-1122-7. URL: https://doi.org/10.1007/s10994-005-1122-7.

[29] L.A. Zadeh. "The concept of a linguistic variable and its application to approximate reasoning—I". In: *Information Sciences* 8.3 (1975), pp. 199 –249. ISSN: 0020-0255. DOI: https://doi.org/10.1016/0020-0255(75)90036-5. URL: http://www.sciencedirect.com/science/article/pii/0020025575900365.