CLUSTERING TO IMPROVE ONE-CLASS CLASSIFIER PERFORMANCE IN DATA STREAMS

RICHARD HUGH MOULTON

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the
Master of Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

The classification task requires learning a decision boundary between classes by making use of training examples from each. A potential challenge for this task is the class imbalance problem, which occurs when there are many training instances available for a single class, the majority class, and few training instances for the other, the minority class [58]. In this case, it is no longer clear how to separate the majority class from something for which we have little to no knowledge. More worrying, often the minority class is the class of interest, e.g. for detecting abnormal conditions from streaming sensor data.

The one-class classification (OCC) paradigm addresses this scenario by casting the task as learning a decision boundary around the majority class with no need for minority class instances [110]. OCC has been thoroughly investigated, e.g. [20, 60, 90, 110], and many one-class classifiers have been proposed. One approach for improving one-class classifier performance on static data sets is *learning in the context of concepts*: the majority class is broken down into its constituent sub-concepts and a classifier is induced over each [100].

Modern machine learning research, however, is concerned with data streams: where potentially infinite amounts of data arrive quickly and need to be processed as they arrive. In these cases it is not possible to store all of the instances in memory, nor is it practical to wait until "the end of the data stream" before learning. An example is network intrusion detection: detecting an attack on the computer network should occur as soon as practicable. Many one-class classifiers for data streams have been described in the literature, e.g. [33, 108], and it is worth investigating whether the approach of *learning in the context of concepts* can be successfully applied to the OCC task for data streams as well.

This thesis identifies that the idea of breaking the majority class into sub-concepts to simplify the OCC problem has been demonstrated for static data sets, [100], but has not been applied in data streams. The primary contribution to the literature made by this thesis is the identification of how the majority class's sub-concept structure can be used to improve the classification performance of streaming one-class classifiers while mitigating the challenges posed by the data stream environment. Three frameworks are developed, each using this knowledge to a different degree. These are applied with a selection of streaming one-class classifiers to both synthetic and benchmark data streams with performance compared to that of the one-class classifier learning independently. These results are analyzed and it is shown that scenarios exist where knowledge

of sub-concepts can be used to improve one-class classifier performance.

Acknowledgments

I would like to begin by acknowledging that the University of Ottawa is located on land which is the traditional unceded territory of the Algonquin Anishnaabeg People.

The research conducted for this thesis was supported by the Natural Sciences and Engineering Research Council of Canada, the Province of Ontario and the University of Ottawa.

I have learned throughout the entire process of developing ideas, conducting experiments, and, of course, writing this thesis. This would not have been possible without the help of a number of individuals in the academic community.

First and foremost, I would like to thank my supervisors, Dr. Nathalie Japkowicz and Dr. Herna Viktor, for their guidance over the last two years. I would also like to thank Dr. João Gama, who hosted me at the University of Porto and provided guidance for a portion of this work. I also received help and support from my peers at both the University of Ottawa and the University of Porto. Thank you to Ali Pesaranghader, Thiago Andrade and Wesllen Sousa.

As well, this thesis would not have been possible without the support for friends and family as well. To my Mum and Dad, Elizabeth, Scott, Connie, Steve and, of course, Haley: I am extremely grateful for everything.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AUC area under the curve

BIRCH Balanced Iterative Reducing and Clustering using Hierarchies

CBTT correlated Bayesian t-test

CDBD Confidence Distribution Batch Detection

CIP Cluster Inclusion Probability

CMM Cluster Mapping Measure

DCT Discrete Cosine Transform

DSCA data stream clustering algorithm

ECDD EWMA for Concept Drift Detection

EWMA exponentially weighted moving average

FIFO first in-first out

FN False negative

FP False positive

GFMMNN generalized fuzzy min-max neural network

g-mean geometric mean

GNG Growing Neural Gas

HDDDM Hellinger distance drift detection method

HS-Tree Half-Space Tree

IDS intrusion detection system

IFCS incremental fuzzy classification system

i.i.d. independent and identically distributed

LVQ Learning Vector Quantization

MELE mixture of explicitly localised experts

MILE mixture of implicitly localised experts

MINAS MultIclass learNing Algorithm for DSs

MOA Massive Online Analysis

MVND multivariate normal distribution

NHST null hypothesis significance testing

NN-d nearest neighbour data description method

OCC one-class classification

OCSVM One-Class Support Vector Machine

OcVFDT One-class Very Fast Decision Tree

OLINDDA OnLIne Novelty and Drift Detection Algorithm

PCA Principal Components Analysis

pdf probability distribution function

RBF radial basis function

ROC Receiver Operating Characteristic

ROPE region of practical equivalence

SA Streaming Autoencoder

SMLC semi-supervised multi-layered clustering model

SMOTE synthetic minority over-sampling technique

SOM self-organizing map

SPASC semi-supervised pool and accuracy-based stream classification

SPC statistical process control

SVDD Support Vector Data Description

SVM Support Vector Machine

TN True negative

TP True positive

UCI University of California Irvine

UPNB Uncertain Positive Naïve Bayes

VFDT Very Fast Decision Tree

VI variation of information

WOCSVM Weighted One-Class Support Vector Machine

Chapter 1

Introduction

How can a machine learning algorithm learn with few or no examples of different classes? Examples of some classes may be hard to collect because they represent rare events, e.g. stellar phenomena, or because they are expensive to obtain e.g. requiring the use of expensive equipment or human expertise. The task of learning a good classification model without these examples is a difficult one, especially because the classes of interest are likely those lacking training instances.

The class imbalance problem [58] and its extreme form, OCC [110], are well known in the machine learning literature for static data sets. Specific aspects of these problems include small disjuncts and rare sub-concepts in the majority class and learning to discriminate the minority class with few or no examples to learn from. These characteristics have motivated a change in the classification task from discrimination to recognition.

Machine learning now goes beyond static data sets, however, and the data stream environment has been an important area in machine learning for the past decade [40, 63, 67, 101]. Data streams are not just an area of theoretical interest: modern, real-world applications commonly involve streaming data as well. This change in the nature of the underlying data from static to dynamic provokes a host of complications and it is not straight forward to transform algorithms for static data sets into algorithms for the data stream environment. In the context of the OCC problem, data streams exacerbate the problems of small disjuncts and rare sub-concepts and potentially add a temporal aspect to the majority and minority classes, requiring the learner to keep its model up to date.

These individual challenges have, of course, been addressed in the machine learning literature. One technique that has been applied to OCC, for example, is to improve recognition of the majority class by dividing it into its constituent sub-concepts [100]. Whether this idea can be applied to OCC in the data stream environment has not been investigated, however, which inspires the work done in this thesis. Although the principle has been shown to work for static data sets, it is unclear how to adapt it in order to account for the new challenges

found in the data stream environment.

The research question addressed in this thesis is "how can knowledge of the majority class's sub-concept structure be used to improve one-class classifier performance in data streams?" We will discuss a motivating scenario, list the contributions made in this thesis, and then describe the organization of the remaining sections.

1.1 Motivation

The motivating scenario for this thesis is streaming sensor data. This scenario matches our previous description of the OCC problem: it can be impractical to collect data for rarely occurring events; the sensor itself may be expensive to operate, limiting collection; and the classes of interest are usually the *abnormal* classes against a backdrop of *normal* background activity. This streaming sensor could be an airborne sensor performing environmental surveys or a scientific sensor used during the course of an experiment.

Airborne Sensor In the first case, imagine an airborne sensor monitoring ground cover as part of an environment survey. Among its many tasks might be a responsibility to detect invasive species or signs of natural disaster, such as flooding. An airborne sensor is expensive to operate, which will limit the amount of training data available. As well, although our sensor will be able to see many examples of *normal* terrain, we are most interested in all of the *abnormal* terrain, even if it has never been seen before.

Our machine learning technique must also handle streaming sensor data from many different terrain conditions, e.g. mountainous terrain, boreal forest, lakes and prairies, possibly all during the same mission. We would like to use this sensor and its associated machine learning techniques in every area of the country, to cut down on the cost of these surveys.

It seems reasonable in this case to divide the overall concept of "terrain" into its constituent sub-concepts, one for each terrain condition. Our machine learning technique can then learn to recognize what *normal* looks like for each kind of terrain. If a sensor reading is not recognized as representing *normal* terrain, then a signal is generated which alerts the user to *abnormal* terrain.

Scientific Sensor In the second case, imagine a telescope situated on the ground that is continuously observing the sky for stellar phenomena. Predictably, many of the observations will be of standard conditions which are uninteresting from a scientific perspective, but we would like to recognize rare and interesting scientific phenomena, even (especially) if they have not been seen before.

These observations will take place in a variety of continually changing contexts such as different atmospheric conditions or the relative position of the sun (i.e. day or night). We would like to be able to observe in as wide a range of conditions as possible in order to increase the sensor's operational time. We

would also like to do so without requiring human intervention, which would needlessly increase the cost and latency of the system. Phenomena of interest should also be signalled in real-time so that more focused observations can be made.

Here again, it seems reasonable to divide the different observational contexts into sub-concepts and to learn to recognize what *normal* observations look like for each. Our system would then generate a signal if an observation is not recognized as *normal*, which cues the system to pay attention to an *abnormal* phenomena that is potentially of interest.

1.2 Contributions

This thesis makes five contributions to the machine learning literature, in the areas of data stream clustering and OCC in data streams.

Answering this thesis's research question results in the main contribution of this thesis: three frameworks that use knowledge of the majority class's sub-concept structure in a OCC problem in the context of data streams, along with theoretical and experimental support for their use. Although sub-concept structure has been previously exploited in the case of static data sets, the volume, velocity and volatility of the data stream environment all present additional challenges for machine learning techniques. The temporal aspect of data streams also forces stream learning algorithms to maintain their performance throughout. The frameworks presented adapt to the dynamic nature of data streams by: separating the initialization and online phases; passively forgetting old instances; and synthetically generating instances for underrepresented sub-concepts.

The second contribution is one of the theoretical supports given for these frameworks. Although the whole training set is always available for static data sets, streaming algorithms must contend with constantly changing training set. One risk associated with this is that a given training set may not contain enough instances from each concept. To address this, a proof is given regarding the size of window necessary to assure, to a given degree of confidence, that there will be a minimum number of instances from a rare sub-concept. This proof is also illustrated with empirical data to show its application to real data streams. Although in this thesis the contribution is used to select a parameter for the three frameworks, it is equally applicable for any other stream learning algorithm.

The third contribution is a novel Cluster Distance Function based on Cluster Inclusion Probability (CIP). As the literature does not contain an appropriate Cluster Distance Function, one is proposed in order to measure the distance between any two clusters, regardless of their shape and of which data stream clustering algorithm (DSCA) they were produced by. This function is also proved to be a metric.

The fourth contribution is an in-depth study of DSCAs and their behaviour in challenging data stream environments. It is observed that there is a gap in the literature concerning the use of DSCAs in frameworks for data streams,

specifically regarding how they can be expected to perform in the presence of concept drift. The existing literature concerning the task of data stream clustering is reviewed and an experiment is conducted in order to make an informed decision regarding which DSCA to use in this thesis.

Finally, the fifth contribution is a novel method to produce synthetic real-valued data streams with precise quantitative concept drift. In order to conduct the study of DSCAs, the ability to control the experiment's dependent variables is key. The literature does not describe a way of generating real-valued data streams with specific quantitative concept drift, however, so a new method is proposed based mathematically on mixture models of multivariate distributions and the Hellinger distance between probability distributions.

1.3 Organization

Having established that the OCC problem is made more challenging by the data stream environment, we will now proceed with an in depth review of the literature in the relevant areas. In Chapter 2 we review the class imbalance problem and OCC for static data sets as well as the idea of improving classification by using a "divide and conquer" approach. We also review: the data stream environment; data stream clustering; concept drift in data streams; OCC in data streams; and uses of "divide and conquer" in data streams.

Following the review, we present the contributions made by this thesis in Chapter 3. As identified above, these include three frameworks making use of sub-concept structure, a proof regarding appropriate window size, a novel Cluster Distance Function, a novel synthetic data stream generator, and in-depth study of DSCAs in data streams where concept drift is present.

The experimental designs for testing DSCAs as well as the three frameworks are described in Chapter 4 and the results of both experiments are presented and discussed in Chapter 5.

We conclude in Chapter 6 by summarizing the results and contributions of this thesis and by identifying paths for future research.

Chapter 2

Literature Review

2.1 Class Imbalance

Class imbalance occurs when one class in a data set is represented by many examples while the other class(es) is (are) represented by only a few [58, 114]. As described by Kubat and Matwin, this adding of instances to the majority class can actually leaded to poorer performance from a learning algorithm [70]. One of the approaches for dealing with this challenge is OCC: the task of learning to accurately recognize the majority class using only training instances from the majority class.

The task of OCC is closely related to the tasks of anomaly detection, outlier detection, novelty detection and data description and the terminology used depends on which aspect of the problem is of interest [20, 36, 60, 110]. For example, Chandola et al. describe anomalies as points of interest for an analyst, in contrast with noise, which is unwanted and must be removed, and novelties, which are new patterns to incorporate into the majority class [20]. Similarly, Gama distinguishes novelties as concepts from outliers as sparse, independent examples; both differ from anomalies which are instances that domain knowledge allows us to label as abnormal [36, Ch. 9]. We will focus on the OCC task and use its associated terminology for the remainder of this thesis.

2.1.1 Definitions

In their study of the nature of the class imbalance problem, Japkowicz and Stephen determine that it is characterized by three dimensions: the size of the training set; the complexity of the classes; and the degree of the imbalance [58]. These were formalized as mathematical definitions, Definitions 1, 2 and 3 respectively. In line with intuition, they found that high degrees of class imbalance result in higher degree of difficulties for the classifiers.

Definition 1 (Training Set Size)**.** A training set is described as being at size level s when the total number of instances present is given by:

$$|TrainingSet| = n_{min} + n_{maj} \propto 2^s$$

Larger values of s mean larger training sets.

Definition 2 (Class Complexity)**.** A training set is described as having class complexity c if the whole data set is composed of 2^c regular intervals, each of which is a subspace of the feature space containing an equivalent number of instances from only one class.[1] Larger values of c mean a more complex data set.

Definition 3 (Degree of Imbalance)**.** A training set is described as having a degree of imbalance i if the minority class is reduced to the following portion of its expected number of instances:

$$n_{min} \propto n_{maj} \times 2^{i-MAX}$$

Where MAX represents some maximum level of imbalance possible in the data set. Lower values of i mean a larger degree of class imbalance because a smaller proportion of the expected instances are present for the minority class.

2.1.2 Challenges

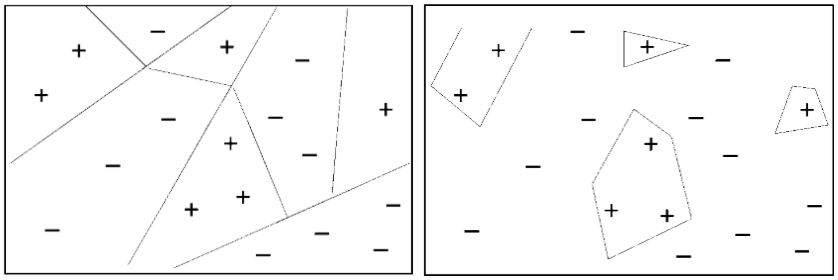
It is clear from Definition 3 that the most extreme degree of imbalance for a data set is $i = 0$. As long as it is permitted by the value MAX, this leads to $n_{min} = 0$, where there are no training instances available for the minority class. This is the OCC problem, where the classification task changes from one of *discrimination* (as in binary or multi-class classification) to one of *recognition* (see Figure 2.1). In the *recognition* paradigm, a model is trained to recognize the instances in its training set; if a test instance is recognized by the model then it is labelled as belonging to the majority class [54, 100]. Khan and Madden articulated the challenge to this approach: only one side of the boundary being determinable [60].

The challenge presented by the lack of training instances from the minority class is by no means the only difficulty faced by one-class classifiers, however. Additional wrinkles crop up in real-world problems with class imbalance and conspire to make the task more difficult. As discussed above, Japkowicz and Stephen identified two other aspects of the class imbalance problem: class complexity and training set size [58]. Both of these are present in OCC as well.

Complex Domains The aspect of complex domains is common to many problems in machine learning; this motivates techniques such as feature engineering in order to transform a data set and make it more suitable to learn from [32]. Class complexity is no less applicable to the OCC problem: just because our training examples are from only one class does not mean that this

[1] Japkowicz and Stephen used a backbone model to define their data sets, but the concept can be extended to any feature space.

Figure 2.1: Two approaches to binary classification (from Japkowicz [54])



(a) Discrimination learns boundaries between classes

(b) Recognition learns boundaries around the majority class

class is easily described or that all its constituent instances are generated by the same process. One method for addressing complex domains is to break the problem domain up into sub-domains, we review the literature in this area in a later section (Section 2.3).

As well as the characteristics, multiple authors have also identified class overlap as a related issue for machine learning algorithms [10, 92, 106]. Prati et al. focused on the effect of class overlap by comparing the degradation in performance by a C4.5 decision tree classifier when a) the level of imbalance between classes was increased and b) the level of overlap between classes was increased. They found that while increased imbalance did decrease classifier performance for well separated clusters, increased overlap had a more drastic effect [92]. Similarly, Batista et al. showed that this relationship held for each of five balancing methods applied to a variety of imbalanced and overlapping data sets [10].

Stefanowski also observed the problem posed by minority class instances overlapping with the majority class, either as outliers or as rare cases [106]. He considered four different class balancing approaches: random oversampling; cluster-based oversampling [59]; nearest cleaning rule [73]; and SPIDER [107]. His experiments paired these approaches with two baseline classifiers, C4.5 decision tree and decision rules, and applied them to data sets with varying degrees of overlap. Results showed that while the random oversampling and cluster-based oversampling generally performed best with lower degrees of overlap, they were outperformed by nearest cleaning rule and SPIDER when there were high degrees of overlap [106].

Training Set Size The size of the training set is of primary concern in machine learning. As highlighted by Domingos: "More data beats a cleverer algorithm" [32]. In OCC, the problem of too few training examples is most clearly seen in the problem of small disjuncts: portions of the majority class that are represented only by rare cases. In this case, there are areas of the feature space that should belong to the majority class but are represented by only a few

training instances. This can cause the classifier to learn too tight a boundary in this area or it disregard it altogether. Jo and Japkowicz concluded that small disjuncts can, in fact, be an underlying source of the difficulties in the class imbalance problem and recommended dealing with the problems simultaneously [59].

Weiss demonstrated that the problem of *with-in class imbalance*,[2] exemplified by small disjuncts, occurs because the ability to learn a good boundary around a disjunct, and therefore the error rate that will result from this learned boundary, is highly dependent on the size of the disjunct. The larger the disjunct, the better the learned boundary and the lower the error rate for that disjunct [114].

Weiss also highlighted that smaller disjuncts are vulnerable to absolute rarity, where there may be insufficient instances in the disjunct to be learned, and noise, which may form its own disjuncts that should not be learned [114]. One method to rectify absolute rarity is oversampling. Oversampling by naïve replication readily leads to overfitting, because this encourages classifiers to learn boundaries around very specific areas of the feature space instead of around the likely area occupied by the minority class [21]. Hoens and Chawla noted that Chawla et al.'s synthetic minority over-sampling technique (SMOTE) avoids this issue by creating synthetic instances which are convex combinations of existing training instances [51].

SMOTE, which is based on a combination of undersampling the majority class and oversampling the minority class [21], generates synthetic minority class instances instead of replicating them. This process is given in Algorithm 2.1. Chawla et al.'s experiments with benchmark data sets showed that both the $C4.5$ and $RIPPER$ classifiers performed better when trained on data sets that had had the minority class "SMOTE-d" and the majority class undersampled [21]. Both Chen and He [22] and Hoens and Chawla identify SMOTE as appropriate for addressing class imbalance as well.

Regarding undersampling, Hoens and Chawla note that one drawback to this approach is the potential of throwing away useful information. They identify multiple methods proposed in the literature to combat this by ensuring that only redundant or noisy instances are removed [51].

2.2 One-Class Classifiers

The literature contains a number of classifiers that are able to learn with training examples from only one class. These are called one-class classifiers and they employ a variety of methods, which Tax groups into three approaches: defining the majority class by its density in the feature space; learning to reconstruct majority class instances; and learning a boundary around the majority class [110, Ch. 3]. This paradigm was also used by Mazhelis [83] in a survey of OCC techniques for application in the context of "mobile-masquerader detection." Khan

[2] as opposed to *between class imbalance*, which is the general class imbalance problem where one class had many instances available and the other has few or no instances available.

Algorithm 2.1 The synthetic minority over-sampling technique algorithm (adapted from Chawla et al. [21])

INPUT: T, minority class samples; N, percentage of synthetic samples to produce; and k, number of nearest neighbours to consider.
if $N < 100$ **then**
  Randomly select $N\%$ of the minority class samples in T
end if
$N = (\text{int})N/100$ (N should be an integer multiple of 100.)
Initialize *syntheticSamples*, an array to store the synthetic samples
for all $min_i \in T$ **do**
  Compute the k nearest neighbours of min_i
  for j from 1 to N **do**
    Choose a random number r between 1 and k
    Select nn_r, the r^{th} nearest neighbour of min_i
    Initialize synthetic instance *syn*
    for all Attributes of min_i: min_{ia} **do**
      diff = $min_{ia} - nn_{ra}$
      gap = random number between 0 and 1
      $syn_a = min_{ia} + (\text{diff} \times \text{gap})$
    end for
    Add *syn* to *synSamples*
  end for
end for
Return T plus *syntheticSamples*

and Madden instead propose a taxonomy based partially on whether the one-class classifier is an One-Class Support Vector Machine (OCSVM), stating that the amount of development that had occurred for this family of one-class classifiers justified setting them apart from all others [60]. This is less descriptive of the classifiers' internal workings and Tax's three approaches are used in the remainder of this thesis.

2.2.1 Density-based

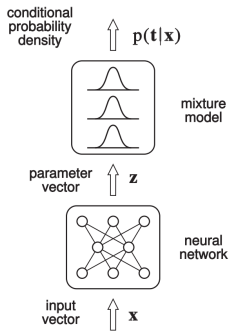
One approach to one-class classification is to define the majority class according to its density in specific areas of the feature space. In practice this can be done using any probability density function, but the normal distribution is commonly used. This can be used as a single probability distribution or in a mixture model [110, pg.64-66].

Examples of these approaches are given by Bishop [13] and Gopinath [44], both of whom highlight that mixture models of normal distributions can be used to model arbitrary probability density functions. Bishop presented the Mixture Density Network (Figure 2.2), which uses a neural network to transform the input, x, into a parameter vector. The mixture model is then evaluated at the

parameter vector to produce the output $Pr(t|x)$: the probability of the target data given the initial input. With $\phi_i(t|x)$ as the conditional probability of t for the ith model and $\alpha_i(x)$ as the mixing coefficients of the mixture model, this is given in (2.1) [13].

$$Pr(t|x) = \sum_{i=1}^{m} \alpha_i(x)\phi_i(t|x) \tag{2.1}$$

Figure 2.2: A Mixture Density Network (from Bishop [13])



Goparinath modelled training data using normal distributions with the principle of Maximum Likelihood. Although this was a multi-class application, each class's parameters were obtained separately and the mixture model of normal distributions was used to model all classes. The technique was applied to a speech recognition task and the author observed that modelling all classes as a single mixture model improved discrimination [44].

Representing the majority class's density as a probability density function also allows for an adaptation of the classic Naïve Bayes classifier, as demonstrated by He et al. [49]. In their work, the authors proposed the Uncertain Positive Naïve Bayes (UPNB): a Naïve Bayes classifier capable of learning from only positive and unlabelled instances with both categorical and numeric attributes. Considering numeric attributes, the problem becomes solving Equation 2.2. The only user provided parameter required is p, the proportion of instances belong to the majority class, as the others can be approximated using Ren et al.'s formula-based method, which derives a closed-formula for the Gaussian kernel [94]. This requirement can be avoided as well, however, as He et al. demonstrate that p can be adequately estimated using a validation data set [49].

$$Pr(X_i) = (Pr(X_i|1) \times p) + (Pr(X_i|0) \times (1 - p)) \tag{2.2}$$

The majority class's density can also be used as a basis for training a supervised learner. Hempstalk et al. started the idea of learning to distinguish between the training data, T, and artificial data generated from a known distribution, A; from this idea and Bayes' Theorem, they derived Equation 2.3.

Bagged decision trees were then trained as class probability estimators, providing $\hat{Pr}(T|X)$ for an instance X [50]. The authors showed that this approach outperformed benchmark one-class classifiers on a selection of data sets from the University of California Irvine (UCI) Machine Learning repository. One specific advantage that Hempstalk et al. identified for their approach was that the detection threshold could be adjusted at prediction time [50].

$$Pr(X|T) = \frac{(1 - Pr(T))Pr(T|X)}{Pr(T)(1 - Pr(T|X))} Pr(X|A) \tag{2.3}$$

2.2.2 Reconstruction-based

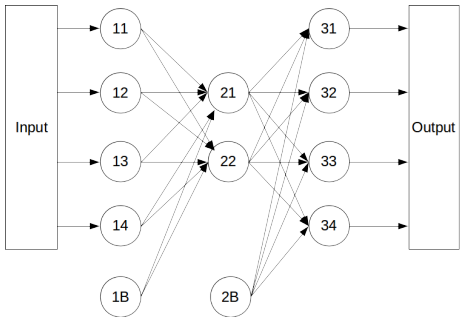
Determining whether a test instance comes from a high-density or a low-density area of the majority class is a form of recognition, but it is not the only one. Another approach to do so is to take a compressed representation of the majority class and to test whether it recognizes the test instance. Tax identifies k-means clustering, Learning Vector Quantization (LVQ) and self-organizing maps (SOMs) among the simplest reconstruction methods [110, pg. 73]. Each represents the training data with prototype objects chosen to minimize some error function. This error function is then used during testing to determine whether a test instance belongs to the majority or minority class. Although compact representations can be achieved using these techniques, they are defined using a distance function (generally the Euclidean Distance) and are therefore sensitive to feature scaling [110, pg. 75]. If the data exists in a subspace (or in a certain number of sub-spaces) then Principal Components Analysis (PCA) can be used as the training set's compressed representation as well [110, pg. 76].

The autoencoder is a type of neural network that can be trained on majority class instances only, which makes it an obvious technique to apply the problem of class imbalance. An autoencoder transforms the input and then tries to reconstruct it and have previously been used in novelty detection [56]. Mathematically, Dong and Japkowicz describe an autoencoder as a two-part process: an encoder, the map $\phi : \mathcal{X} \to \mathcal{F}$; and a decoder, the map $\psi : \mathcal{F} \to \mathcal{X}$. The maps ϕ and ψ are used to minimize the reconstruction error, comparing the test instance with the reconstructed instance, which is then used to determine how anomalous the input is, (2.4).

$$\arg\min_{\phi,\psi} \|x - \psi(\phi(x))\|^2 \tag{2.4}$$

The autoencoder network is fully connected, meaning that all neurons in a given layer receives input from all neurons in the preceding layer [46, pg. 399]. The input and output layers each have one neuron per dimension in the attribute space. The hidden layer should be smaller than these layers to ensure that the identity function is not learned [33]. The input and hidden layers both also have bias unit neurons, which are always activated. A sample autoencoder network is shown in Figure 2.3.

Figure 2.3: A sample autoencoder network



Each neuron incorporates an activation function that transforms the sum of the activations it has received to the range $[0, 1]$ [46, pg.402]. Examples of activation functions include: the logistic function; the hyperbolic tangent function; and the binary step function.

2.2.3 Boundary-based

In contrast with density-based and reconstruction-based methods, boundary-based OCC methods are designed for cases where there is only a small amount of training data available [110, pg. 67-78].

The simplest such method is Ypma and Duin's k-centers algorithm, which covers the training set, X, with S_k, a set of k balls of radius r centred on training instances (Equations 2.8 and 2.7). The radius for a given set of centres, $r(S_k)$ is calculated as the maximum distance between a point $x_i \in X$ and its nearest ball centre $y_j \in S_k$ (Equation 2.6. The set of centres that minimizes $r(S_k)$ is taken as the algorithm's output, J [117]. Both k and the number of iterations of the algorithm need to be specified by the user [110, pg. 69].

$$x_i \in R_p \text{ when } (p = \arg\min_j d(x_i, y_j)) \wedge (d(x_i, y_j) \leq r) \tag{2.5}$$

$$r(S_k) = \max_i d(x_i, y_j) \text{ such that } x_i \in R_j, y_j \in S_k \tag{2.6}$$

$$|S_k| = k \tag{2.7}$$

$$J = \arg\min_{S_k} r(S_k) \tag{2.8}$$

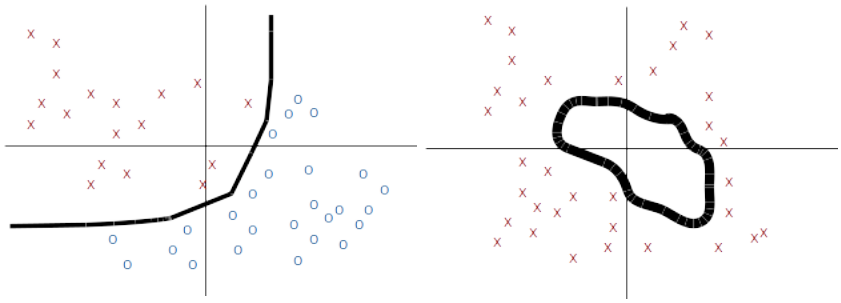
Khan and Madden identify two techniques that are based on the concept of a support vector[3]: the OCSVM and Support Vector Data Description (SVDD).

The adaptation of the Support Vector Machine (SVM), inherently a binary classifier, to the OCC problem is known as the OCSVM. Instead of trying to learn a maximally separating hyper-plane between two classes, an OCSVM

[3]A support vector is based on the idea that not every training vector (instance) is required to define a boundary, only those (the supports) which are essential to the boundary's definition.

maps the data into an appropriate feature space and learns a hyper-plane that separates the majority class instances from the origin [97]. Figure 2.4 illustrates the similarity in these approach.

Figure 2.4: SVMs and OCSVMs both learn hyper-planes



(a) Binary classification (b) One-class classification

The computation of the separating hyper-plane is quadratic [97] and Khan and Madden identify many contributions to the literature aimed at optimizing this aspect of the algorithm [60]. These techniques include transforming the quadratic programming problem so that linear programming can be used[4], finding parameters using particle swarm optimization, and making the OCSVM sensitive to different classification error costs [60].

SVDD is an approach described by Tax and Duin that seeks to improve on the concept of the OCSVM [112]. The SVDD of a training set results in a hyper-sphere (or hyper surface) that contains the training instances and is defined by certain support vectors. Example SVDDs produced using the Gaussian kernel and with varying kernel widths, s, and regularization terms, C, are shown in Figure 2.5.

Finally, the nearest neighbour data description method (NN-d) is derived from local density estimation [110, pg. 69-71]. Instead of needing to estimate densities, however, Tax showed how the decision rule could be modified to produce a boundary using the indicator function, I, the instance's nearest neighbour and the nearest neighbour's nearest neighbour, (2.9).

$$f(x) = I\left(\frac{\|x - NN^{tr}(x)\|}{\|NN^{tr}(x) - NN^{tr}(NN^{tr}(x))\|} \leq \tau\right) \tag{2.9}$$

Where x is the test instance, τ is a threshold, and $NN^{tr}(\cdot)$ if a function that returns the argument's nearest neighbour. An example NN-d is shown in Figure 2.6; it is produced from the same banana-shaped data set as in Figure 2.5.

[4]This is done by inverting the OCSVM formulation and finding a hyper-plane that tightly contains the majority class instances around the origin.

Figure 2.5: Support Vector Data Description for a banana shaped data set. s = kernel width, C = regularization term. From Tax and Duin [112].

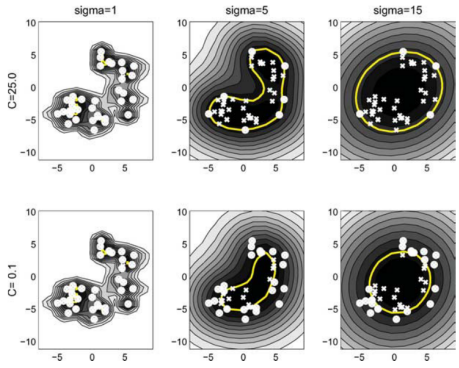
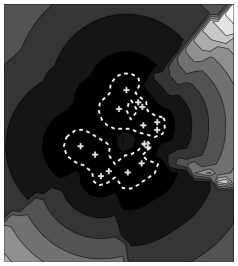


Figure 2.6: Nearest Neighbour Data Description for a banana shaped data set. From Tax [110].



2.2.3.1 Comparison of One-Class Classifier Approaches

Each of Tax's three approaches for one-class classifiers is capable of performing the OCC task, but they are designed with specific scenarios in mind and they provide different outputs to the user. These are summarized in Table 2.1.

Density-based one-class classifiers model the majority class for the entire feature space which, if the model is accurate, provides very complete knowledge about the problem. The question of the model's accuracy depends on the training data available as well as the probability density function being used and both must be appropriate for density-based classifiers to be used.

The use of probability distribution functions (pdfs) (generally mixture models of normal distributions) is a powerful way of representing this model and comes with both advantages and disadvantages. The main advantage is that techniques from the mathematical field of probability are readily applicable. This can be useful when developing new machine learning techniques and it also provides a framework within which theoretical justifications can be constructed for existing techniques. This also means that density-based one-class classifiers inherently produce probabilistic scores for test instances, which is useful when

Table 2.1: Pros and cons for three approaches for one-class classifiers

Approach	Pros and Cons
Density	**Pros** Models majority class for the entire feature space Able to make use of mathematical techniques from probability Reconstruction error provides a range of scores for test instances **Cons** Requires fully representative training set Representations of the majority class may be unwieldy
Reconstruction	**Pros** Provides a compact model of the majority class, for the entire feature space Model size independent of training set size and complexity Produces probabilistic scores for test instances **Cons** Some models may not be human interpretable (e.g. autoencoder network) Requires fully representative training set
Boundary	**Pros** Models majority class based on available training set Provides model that is easy to visualize **Cons** Produces binary scores for test instances Model size often dependent on training set size and complexity

determining appropriate threshold values. The disadvantage of this approach is that pdfs necessary to represent the full majority class may be very awkward, even with the power of mixture models. As well, the resulting functions may not be straight-forward to evaluate.

Reconstruction-based one-class classifiers take a very different approach and have different strengths and weaknesses. The major strength of these classifiers is that they result in a compact model of the majority class whose size is independent of the majority class's size and complexity. Similar to the probabilistic score produced by density-based classifiers, the reconstruction error is useful in inherently defining how likely it is that a test instance belongs to the majority class.

One weakness of the reconstruction-based classifiers is that although the resulting model is compact, it can also be difficult to interpret. Instead of a pdf or

a boundary that can be visualized over the feature space, the model provides instructions for reconstructing a test instance from its compressed form. Although effective, this is not an intuitive way for humans to consider the problem. The other main weakness is shared with density-based classifiers: a representative training set is required. Without this, the discovered methods of compression and reconstruction may not generalize to the portions of the majority class that are not represented.

Finally, boundary-based one-class classifiers offer a very intuitive way of modelling the majority class. Of the three approaches, these produce the model that is the easiest to visualize and the easiest for humans to interpret. Because the boundaries are decided based on local factors only, these one-class classifiers are also able to produce quality models from the available training set (of course, the more representative the data set the better the model, but the quality of the model is relatively unaffected by the presence or absence of "far-away" training instances).

The weaknesses of these classifiers are their model size and the scores they produce for test instances. Model sizes are very dependent on the size of the training set, for example the NN-d method's model is the training set. The exception to this is the OCSVM, although this reduction in model size comes at the cost of greatly increased computation (quadratic versus NN-d's constant train time). The other weakness is that they produce a binary score that reflects which side of the boundary the test instance finds itself. Although this can be alleviated by accounting for "distance from the boundary" or slack areas, boundary-based methods do not produce a range of scores as easily as density- or reconstruction-based classifiers.

2.3 The Divide and Conquer Approach

Addressing the classification task through a "divide and conquer" approach has been used to good effect many times over in the literature. The idea is to find more tractable sub-problems whose solutions can be combined to provide a solution to the overall problem. In considering that complex domains are a factor in making the class imbalance and OCC problems more challenging, making use of "divide and conquer" is an obvious way forward.

2.3.1 Sub-Division Applied to Classification

Jacobs et al. [53] showed that if a set of training cases was known to consist of multiple distinct cases, then a separate neural network could be trained on each case and combined with a high-level function to determine which expert a test instance should be directed to. This is something that the authors described as a "modular version of a multilayer supervised network."

In a recent survey of approaches incorporating local experts, Masoudnia and Ebrahimpour distinguished between two approaches to "divide and conquer": a mixture of implicitly localised experts (MILE) and a mixture of explicitly

localised experts (MELE) [81]. In the former case, the expert classifiers are assigned a region of the feature space based on where they perform best while in the latter case the expert classifiers are assigned to defined regions on the basis of domain knowledge. The authors identified that MELE methods generally outperform MILE methods as a direct result of the information contained in the explicit sub-domains. This was caveat-ed, though, with the observation that such explicit sub-domains may not be available and that they may not be suitable for the chosen classifier to learn.

With a mixture (or ensemble) of experts, there are a variety of ways that their knowledge can be combined. Although using a high-level deciding function to select one expert classifier is intuitive, it is not the only option. Tax and Duin investigated "if and how" one-class classifiers should be combined for the benchmark problem of recognizing handwritten digits [111]. Using a range of one-class classifiers they considered methods for combining classifier votes based on the mean and product of weighted and unweighted votes. As a result of experimentation they observed a product combination of one-class classifiers led to a lower outlier acceptance rate compared to a mean combination of votes, which estimated a larger area for the majority class.

Kuncheva [71] investigated the trade-off between selecting one classifier from the ensemble and fusing multiple classifiers together and recommended a framework that combines the two. For classifier selection, a cluster-based approach was chosen where the training set is clustered and a classifier with the highest accuracy for a given cluster is assigned. For a given test instance, the nearest cluster centre is found and that cluster's assigned classifier is selected. If no classifier performed sufficiently well on a cluster, then decision template fusion is used. In this case, the squared Euclidean distance between each classifier's decision profile and pre-computed decision template is combined.

2.3.2 Sub-Division Applied to One-Class Classification

One application of problem sub-division to OCC is improving the resampling that is often performed when addressing class imbalance [51]. For example, Weiss observed that many resampling techniques address imbalance between classes and not within classes; he suggested that resampling be done to balance the size of disjuncts in the training data [114]. Indeed, Nickerson et al. showed that using the subcomponent structure of a class to guide resampling improved the quality of that oversampling by ensuring that each subcomponent is adequately represented. Acknowledged limitations of their work was the requirement that the subcomponent structure be well understood [85]. As well, Jo and Japkowicz showed that a method of cluster-based oversampling could be used successfully to increase the number of training examples from small disjuncts. They concluded that it was worthwhile addressing the problems of class imbalance and of small disjuncts simultaneously [59].

Another use of problem sub-division was identified by Jacobs et al. [53], namely the recognition that a single class may be composed of several constituent sub-classes with their own semantic meaning. Japkowicz identified that

labels provided by domain experts do not always result in homogeneous classes and explored using unsupervised learning (clustering) to break up a large, heterogeneous class into more easily modelled sub-classes [58]. In three of five case studies using benchmark data sets from the UCI Machine Learning repository this technique did improve classification accuracy, suggesting that the underlying intuition may be correct. In the information retrieval domain, this idea is seen in Lipka et al.'s proposed cluster-based one-class ensemble [77]. Lipka et al. specifically identified multimodal target classes as being difficult for a classifier to accurately model and demonstrated that OCSVMs trained on clusters produced by the k-means algorithm outperformed an OCSVM trained on the whole data set for both artificial and benchmark data sets.

The idea of representing the majority class through its constituent sub-concepts was greatly developed by Sharma [100], who identified that a one-class classifier might have a range of knowledge concerning this sub-concept structure. Three meaningful levels of knowledge were identified: complete knowledge, where every instance is known to belong to one sub-concept; fuzzy knowledge, where the sub-concepts are known but it is not trivial to assign instances; and no knowledge, where a sub-concept structure is believed to exist but is not defined [100]. These are shown in Table 2.2.

Table 2.2: Three scenarios for sub-concept detection (adapted from Sharma [100])

Scenario Name	Description
Complete Knowledge	The sub-concept structure is known and it is available throughout.
Fuzzy Knowledge	The sub-concept structure is known as part of expert domain knowledge. This labelling is expensive and available only during training.
No Knowledge	The sub-concept structure exists but is unknown.

Sharma showed experimentally that using this knowledge of the majority class's sub-concept structure did improve one-class classifier performance for both autoencoders and OCSVMs [100]. Limitations to the work are that it used k-means as the only clustering algorithm without specific investigation and that is considered static data sets only.

2.3.3 Summary

As we have seen, sub-division can be applied to many aspects of the classification and OCC tasks. A problem's sub-divisions can be the result of domain knowledge or of unsupervised learning and they can be put to use in assisting multiple parts of the data mining process. These are summarized in Table 2.8.

Both subcomponent-/cluster-based resampling and learning from sub-concepts recognize that sub-dividing the majority class can result in "easier-to-learn" rep-

Table 2.3: Uses for sub-division in one-class classification

Use	Benefit	Source
Resampling	Subcomponent-based oversampling ensures each sub-component's relative weight is maintained	[85]
	Cluster-based oversampling helps address problem of small disjuncts along with class imbalance	[59]
Learning from Sub-Concepts	Breaking heterogeneous classes into homogeneous sub-concepts improves classifier performance	[58]
	Clusters can be useful summarizations upon which to train OCSVMs	[77]
	Sub-concept structure, whether known or inferred, can improve classifier performance	[100]

resentations and can minimize the challenge presented by small disjuncts. The common thread between these uses is the recognition that the class structure of a problem may not be the ideal form to learn. The actual reason for this in a real world data set can vary: the classes may be heterogeneous, they may be complex to represent, and they may contain semantically different sub-concepts. In each of these cases, a traditional machine learning technique has benefited from representing the class structure though its constituent sub-concepts.

2.4 Data Streams

The OCC problem, like many other problems in machine learning, has been well-addressed for the case of static data sets. As data streams are a reality in today's world, e.g. computer network traffic, streaming sensor data, financial transactions, we are interested in OCC in this learning environment. The idea of online learning, i.e. learning in the data stream environment, has been studied for some time in the literature. For example, Schlimmer and Granger described the STAGGER program, which is able to learn incrementally from noisy data, [96] and Maass's demonstration that online learning could occur for what he termed "oblivious" environments [80]. Before continuing, we briefly review the terminology and definitions used to describe these streams.

In their discussion of open research challenges for data stream mining, Krempl et al. described data streams as being characterized by three Vs: *volume*, *velocity*, and *volatility* [67]. While each of these qualities may be present in static data sets, their fundamental importance to the data stream environment make it a challenging learning environment. Specific challenges include: time restrictions require data to be processed in limited time; memory restrictions require data to be incorporated incrementally; and changes in the underlying process require "old information" to be safely forgotten [42, 67, 101, 115].

There are multiple approaches to formalizing data streams as a mathemat-

ical construct. Silva et al. formalize a data stream, $\mathcal{DS}$, as a potentially infinite sequence of data objects, $\{x^1, x^2, x^3, \ldots, x^N\}, N \rightarrow \infty$. Each data object is defined by an n-dimensional vector drawn from an attribute space, $x^i = [x^i_j]^n_{j=1} \in \Omega$. Alternatively, Webb et al. describe data streams as data sets with a temporal aspect and generated by some underlying process. This process can be modelled as a random variable, χ, and the data stream's instances as objects drawn from this random variable. An object, o, is a pair $\langle x, y \rangle$ where x is the object's feature vector and y is the object's class label. Each is drawn from a different random variable, X and Y: $x \in \text{dom}(X)$, $y \in \text{dom}(Y)$ and $o \in \text{dom}(X, Y) = \text{dom}(\chi)$ [113]. Webb et al.'s formalization is used in this thesis, as it lends itself naturally to describing the phenomenon of concept drift.

For the rest of this literature review we will cover the important concepts of concept drift (Section 2.5) and data stream clustering (Section 2.6) before addressing the primary topic of this thesis, OCC in data streams (Section 2.7).

2.5 Concept Drift in Data Streams

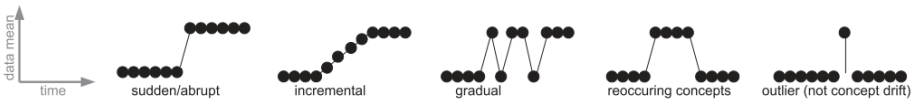
The volatility of a data stream and the requirement for a stream learning algorithm to forget both speak to the idea that a data stream can change over time. This is called concept drift and it is the major challenge when adapting machine learning techniques from static data sets to data streams. This can be thought of as happening because the behaviour underlying the data stream changes, e.g. the kind and quantities of online transactions change through the year, peaking notably for Black Friday and in the lead up to Christmas. The problem introduced by concept drift is that learning done on older instances can actually be detrimental to predictions made in the present, thus the learner needs to both learn and forget in an effective manner.

2.5.1 Qualitative Descriptions of Concept Drift

Concept drift has been conceptualized qualitatively by many authors [40, 120]. Qualitative types of concept drift include: abrupt, where one concept is immediately replaced by another, gradual, with an alternation between concepts, and incremental, with a series of intermediate concepts (Fig. 2.7). Krempl et al. identified four characteristics to use in the categorization of concept drift: its smoothness (sudden vs. gradual); the context it takes place in; whether domain knowledge can predict its trajectory; and whether the drift represents a new normal or just noise [67]. Concept drift could occur, for example, abruptly in traffic data when construction closes an intersection or gradually in medical readings as a patient recovers from illness.

A concept in a data stream is defined as the probability associated with an underlying generative process; Definition 4 [40, 98, 121]. Concept drift occurs between points of time t and u in the data stream when the data stream's concept at time u is not what it was at time t. This is expressed mathematically in Definition 5.

Figure 2.7: Different qualitative types of concept drift (from Gama et al. [40])



Definition 4 (Concept)**.** $Concept = P(X, Y) = P(\chi)$

Definition 5 (Concept Drift)**.** $P_t(X, Y) \neq P_u(X, Y)$

This concept drift could be the result of changes in $P(X)$ or in $P(Y|X)$[5]. In classification tasks it is changes in $P(Y|X)$ that are of interest, given that the goal is to predict y. For clustering task, however, class labels are not available and concept drift affecting only $P(X)$ can be of interest as well. Concept evolution, when a novel pattern emerges, is a special case of concept drift [34, 82]. Although a changing number of classes is usually discussed in the context of supervised learning [82, 28], $P(X)$ is also likely to affected and so it should be considered for unsupervised learning as well. Concept evolution might occur on a computer network where certain activities only occur at certain times, e.g. system updates are downloaded overnight.

It is also true, of course, that concept drift does not only occur in the clean ways depicted in Figure 2.7. Concept drift can occur at different rates for different attributes, recurrent concepts could be observed with one of the constituent concepts drift itself, different qualitative types can occur in the same data stream, etc. This is labelled as "heterogeneous concept drift" by Losing et al.; in their paper proposing the Self-Adjusting Memory model for the k-nearest neighbour algorithm the authors highlight that classifiers capable of handling only one kind of concept drift are likely to perform poorly when confronted with other qualitative types [78].

2.5.2 Quantitative Descriptions of Concept Drift

Although the qualitative descriptions of concept drift seen in Figure 2.7 are intuitive and easy to understand, they lack a degree of rigour that can occasionally be desired. As seen in Losing et al.'s discussion on the topic, it can also be unclear how to use these qualitative types to describe what actually occurs in real world data streams.

To address this shortcoming, Webb et al. proposed a framework for quantitatively describing concept drift, based on Definition 4 [113]. One example of a quantity that can be easily captured under a quantitative framework is the magnitude of a concept drift: this is the distance between the concepts at times t and u as measured by a distribution distance function, D, (Def. 6) [113]. Magnitude is not an aspect captured by Figure 2.7's qualitative types, but it allows us to describe an important aspect of the drift. For example, magnitude

[5]Gama et al. term the former *virtual concept drift* and the latter *real concept drift*.

allows cases with overlapping concepts to be distinguished from cases with divergent concepts - the latter are most likely examples of more "severe" concept drift whether the drift happens abruptly or gradually. As an example, Webb et al. implemented a data stream generator capable of generating categorical data with a precise magnitude of concept drift and used it to conduct experiments with various data stream classifiers [113].

Definition 6 (Drift Magnitude)**.** $Magnitude_{t,u} = D(P_t(\chi), P_u(\chi))$

Another aspect that is illuminated through its quantitative description is the duration of a concept drift: the time during which the drift occurs (Def. 7). Drift duration distinguishes between drifts of different lengths and illustrates that the boundary between abrupt and extended concept drift is simply a threshold value [113].

Definition 7 (Drift Duration)**.** $Duration_{t,u} = u - t$

Webb et al.'s framework also allows more complex aspects of concept drift to be captured, such as path length and drift rate, as well as the relationship between different periods of drift, such as drift frequency and cycle duration [113]. These are not used in this thesis, so we omit them from this review.

2.5.3 Concept Drift Detection

Sethi and Kantardzic identify that many classifiers assume that the instances in both the training and testing sets are independent and identically distributed (i.i.d.) from the same distribution and that this assumption is violated by concept drift [98]. This difference between learning environments for batch and stream learners is also noted by Gama et al. [39]. The violation of this assumption by concept drift is actually tautological, since we defined that concept drift occurs when the probability distribution representing the data stream's concept changes (Definition 5).

Given this, it can be of interest to detect when concept drift has occurred in order to take informed action. Although some algorithms are inherently incremental and can passively adapt to concept drift, many others require a signal to trigger active adaptation [33, 40]. On top of this, Gama et al. identify in their survey on concept drift adaptation that concept drift detection can be valuable in its own right as it can provide information about the data stream itself [40]. The concept drift detection methods which send this signal belong to two families: explicit, which require labelled data, and implicit, which do not [98].

Gama et al. further identify explicit concept drift detectors as based on sequential analysis, SPC, differences between distributions on two time windows, or heuristics [40]. Sethi and Kantardzic largely agree with this, providing three categories: sequential analysis, SPC, and window-based. Sethi and Kantarzic also break implicit methods into three categories: clustering-based, multivariate distribution monitoring, and model dependent monitoring [98]. These categories

Table 2.4: Categories of concept drift detection methods (from Gama et al. [40] and Sethi and Kantarzic [98])

Explicit Methods	Sequential Analysis
	Statistical process control (SPC)
	Differences between distributions (window-based)
	Heuristics
Implicit Methods	Clustering-based
	Multivariate distribution monitoring
	Model dependent monitoring

are summarized in Table 2.4 and exemplary algorithms identified by the surveys from each category are reviewed in the following sections.

2.5.3.1 Explicit Concept Drift Detection based on Sequential Analysis

The Page-Hinkley test [89] is a variant of the cumulative sum technique and an example of an explicit method based on sequential analysis [40]. Gama et al. demonstrated that the Page-Hinkley test could be used to detect concept drift by monitoring a classifier's error rate, Equations 2.10 through 2.12 [39]. In these equations x_t is the error at time t, $\bar{x}_T$ is the mean error throughout the data stream, δ is the allowed magnitude of change, and λ is the user-specified detection threshold.

$$m_T = \sum_{t=1}^{T} (x_t - \bar{x}_T - \delta) \tag{2.10}$$

$$M_T = \min(m_T, t = 1 \dots T) \tag{2.11}$$

$$PH_T = \begin{cases} \text{alarm if } m_T - M_T > \lambda \\ \text{silent otherwise} \end{cases} \tag{2.12}$$

2.5.3.2 Explicit Concept Drift Detection based on Statistical Process Control

Ross et al.'s EWMA for Concept Drift Detection (ECDD) algorithm makes use of exponentially weighted moving average (EWMA) charts and is an example of a statistical process control (control chart)-based method [95]. EWMA makes use of an estimator Z_t.

$$Z_t = \begin{cases} (1 - \lambda) Z_{t-1} + \lambda X_t \text{ for } t > 0 \\ \mu_0 \text{ for } t = 0 \end{cases} \tag{2.13}$$

Independent from the stream's distribution, the mean and standard deviation of Z_t are related to the stream's mean and standard deviation using quantities that can be estimated from the stream itself. The estimator Z_t can then be used to detect concept drift if it is too far away from its initial value, Z_0, as seen in Equation 2.14 [95].

$$ECDD = \begin{cases} \text{alarm if } Z_t > Z_0 + L\sigma_{Z_t} \\ \text{silent otherwise} \end{cases} \tag{2.14}$$

Challenges with the ECDD approach include finding the appropriate values for λ and L. The appropriate value for λ is likely between 0.1 and 0.3 according to Ross et al., but determining the optimal value requires full knowledge of the concept drift. The appropriate value for L, meanwhile, varies throughout the stream and can be computationally expensive to determine [95].

Ross et al. conducted experiments to assess the ECDD approach using synthetic data streams with both abrupt and gradual concept drift as well as real world data streams. The results showed that using ECDD resulted in improved classifier[6] performance as well as two baseline concept drift detectors. Each of the concept drift detectors helped the classifier perform better than the classifier itself [95].

2.5.3.3 Explicit Concept Drift Detection based on Distribution Differences

The clearest example of detecting concept drift by monitoring the differences between distributions is Bach and Maloof's paired learners approach. The paired learners approach compares two models: a stable model, trained on all the instances it has seen, and a reactive model, trained on only the most recent instances. Bach and Maloof highlighted that this method uses model accuracy directly to flag when concept drift is affecting model accuracy, something that results in clear and simple outputs [6].

The specific mechanism used to flag concept drift consider a stable model S that has been trained on t instances and a reactive model R_w that has been trained on the most recent window of w instances. Their performance is compared by counting the number of instances that are correctly classified by R_w but misclassified by S. This suggests that the learning from the $t - w$ instances outside the recent window is hindering the ability of S to make predictions and triggers the replacement of S with a new stable model [6].

Bach and Maloof's experiments showed that their paired learner approach performed comparably or better than a number of other methods in the literature (Dynamic Weighted Majority, Streaming Ensemble Algorithm, and Accuracy Weighted Ensemble) while using less memory and computational time [6].

[6] Linear Discriminant Analysis or K-Nearest Neighbours.

2.5.3.4 Explicit Concept Drift Detection based on Heuristics

Bouchachia's incremental fuzzy classification system (IFCS) algorithm is a context-based concept drift detector that uses prototypes to represent the classes present in the data stream [15, 40]. The prototypes are hyperbox fuzzy sets in a generalized fuzzy min-max neural network (GFMMNN) (i.e. nodes in the GFMMNN's hidden layer) and are dynamically weighted according to three methods [15]:

Staleness Each hyperbox is weighted according to a staleness mechanism, (2.15). If w_i^t falls below a threshold, then hyperbox i is removed.

$$w_i^t = \zeta^{(t-a_i)} \tag{2.15}$$

Here t is the current time and a_i is the last time hyperbox i "won" an incoming instance and ζ is the forgetting factor parameter;

Penalization Each hyperbox is also exponentially penalized based on s_i: the number of errors for which it is responsible, (2.16);

$$z_i^t = \zeta^{s_i} \tag{2.16}$$

Drift Detection Method Finally, Gama et al.'s Drift Detection Method [37] is used to detect abrupt and gradual concept drift. Once a concept drift is detected, the IFCS algorithm reduces the weights of all hyperboxes which speeds up their replacement [15].

Bouchachia demonstrated experimentally, with a data set modelling an ambient intelligent environment, that the adaptive features of the IFCS algorithm allowed for greatly improved performance in the face of concept drift when compared to the same base GFMMNN learner without the adaptive features.

2.5.3.5 Implicit Concept Drift Detection based on Clustering

An example of an implicit method is Spinosa et al.'s OnLIne Novelty and Drift Detection Algorithm (OLINDDA) [105]. OLINDDA performs k-means clustering on unlabelled data in short-term memory throughout the data stream and monitors future clusters against a clustering that models *normal*. Candidate clusters are first assessed for density, and then assessed as either a *novelty* or as a *concept drift* on the basis of distance between the candidate cluster and a global centroid [105]. The authors compared OLINDDA experimentally against other implicit concept drift detection methods and concluded that it performed favourably across eight benchmark data sets [105].

2.5.3.6 Implicit Concept Drift Detection based on Multivariate Distributions

Ditzler and Polikar proposed the Hellinger distance drift detection method (HDDDM) as a "feature based drift detection method" that computes the

Hellinger distance between two histograms of data (representing the current and reference data) [31]. This is done by computing an intermediate Hellinger distance for each feature in the feature space and then averaging them together (2.17).

$$\delta_H(t) = \frac{1}{d}\sum_{k=1}^{d}\sqrt{\sum_{i=1}^{b}\left(\sqrt{\frac{P_{i,k}}{\sum_{j=1}^{b}P_{j,k}}} - \sqrt{\frac{Q_{i,k}}{\sum_{j=1}^{b}Q_{j,k}}}\right)^2} \tag{2.17}$$

Where d is the number of features in the data stream and $P_{i,k}$ and $Q_{i,k}$ are the frequency count of the ith bin of the histograms P and Q respectively for the kth feature. Both histograms have $b = \lfloor\sqrt{N}\rfloor$ bins, where N is the number of instances in the current batch [31].

This computation of the Hellinger distance at time t is compared to the previous Hellinger distance to produce the difference in divergence, $\epsilon(t)$. This difference is compared to a dynamically adjusted threshold value, $\beta(t)$; if it exceeds the threshold value then concept drift is signalled (2.18) [31].

$$HDDDM(t) = \begin{cases} \text{alarm if } \epsilon(t) = \delta_H(t) - \delta_H(t-1) > \beta(t) \\ \text{silent otherwise} \end{cases} \tag{2.18}$$

The authors showed that the HDDDM could be used to improve the performance of a Naïve Bayes classifier for data streams exhibiting both incremental[7] and abrupt concept drift.

2.5.3.7 Implicit Concept Drift Detection based on Models

The Confidence Distribution Batch Detection (CDBD) approach was presented by Lindstrom et al. to take advantage of the classifier's own output as a proxy for concept drift [76]. CDBD requires that the classifier with which it is paired produce a confidence score reflecting how much the test instance is recognized. These confidence scores are put into bins and then the Kullback-Liebler divergence is calculated between the current batch of bins and a reference batch [76].

CDBD flags a concept drift using a modified version of the *Western Electric rules*.[8] Experiments on synthetic data streams and real world data streams with induced concept drift showed that CDBD performed as well in triggering classifier rebuilds as an explicit concept drift detector (the Window Resize Algorithm for Batch Data [72]).

[7] Ditzler and Polikar describe the concept drift as *gradual*, however it consisted of the data stream moving from one concept to another through intermediate concepts. This scenario is described as *incremental* concept drift in this thesis.

[8] In the *Western Electric rules*, a rule is met if x out of the last y observations all surpass a given threshold.

2.5.3.8 Summary

As with the classification and clustering tasks, there are a variety of approaches to detecting concept drift in data streams. In their survey, Gama et al. identified one difference between explicit concept drift detectors was that those based on the difference between two distributions may provide better information along with increased computational costs [40]. Between explicit and implicit methods, of course, the primary difference is dependence on the labels: explicit methods have access to much more information, but implicit methods can be applied to situations where timely access to labels cannot be assumed. Experimental results have also shown that there are scenarios where implicit detectors can perform on par with explicit detectors (e.g. Lindstrom et al.'s CDBD [76])

2.6 Data Stream Clustering

As with static data sets, the role of a DSCA is to discover the clusters that are present in a data stream. That is to say, how best to group the data stream's instance together so that instances within a cluster are similar while instances in separate clusters are dissimilar. On top of the challenges inherent in the clustering task, DSCAs must also contend with the data stream environment.

Barbará argued DSCAs have three requirements: compact representation of instances; fast, incremental processing of new instances; and clear and fast identification of outliers [8]. Most DSCAs produce clusterings with an online component to summarize the data stream's instances and an offline component to cluster this summary [101, 42]. Although each DSCA uses a different approach, we note that there are four general methods of clustering data streams that are discussed in the literature: partitioning, density-based, hierarchical and grid-based. These match Han et al.'s methods for clustering static data sets [46, p.450].

Partitioning Partitioning-based methods produce a partition on the feature space so that objects are similar to the other objects in their partition and dissimilar to objects in other partitions. These partitions can be defined by a mean point, as by StreamKM++ [2], a representative point, as by Stream LOCALSEARCH [87], or a map node, as by G-Stream [41]. A common drawback to these methods is that they are only capable of producing clusters in the shape of hyperspheres and often have difficulty representing arbitrarily shaped clusters.

Density-based Density-based methods address the concern regarding arbitrarily shaped clusters by using the paradigm of clusters as dense regions separated by less dense regions - or regions with many points separated spaces with few points. Both DenStream [19] and OpticsStream [109] model objects with microclusters. Depending on their density, these microclusters are labelled as either core, potentially core or outlier. A key to these methods is defining what constitutes a dense region; this is usually done by user-defined parameters.

Hierarchical Hierarchical-based methods use the approach of organizing objects into a hierarchical structure, like a tree. In this way, objects are closer in the tree structure to objects that are similar and further away in the tree from those that are dissimilar. Both ClusTree [62] and BIRCH [118] produce hierarchical representations as their online summarization. The hierarchical nature of the tree structure allows different clusterings to be produced by inspecting the tree at different levels. This comes with the associated computational cost of rebuilding the whole tree when necessary.

Grid-based Grid-based methods do not deal with objects themselves but instead partition the feature space itself into grids. These grids act as a summarization of the data stream and track as data stream objects arrive that are within their partition. D-Stream takes advantage of this to put an upper bound on the cost of calculations, no matter how many objects arrive they are represented with a constant number of grids [23]. DGClust, on the other hand, makes use of this summarization technique to harmonize online summaries for the distributed clustering problem [38]. The coarseness (or fineness) of the feature space grid becomes an important consideration and generally requires user-defined parameters.

Table 2.5: Methods for clustering data streams (and exemplar algorithms)

Partitioning	**Density-based**	**Hierarchical**	**Grid-based**
CluStream [3]	DenStream [19]	ClusTree [62]	D-Stream [23]
StreamKM++ [2]	OpticsStream [109]	BIRCH [118]	DGClust [38]
G-Stream [41]			Fractal Clustering [9]

A more detailed way of characterizing DSCAs are the seven aspects identified by Silva et al. in their survey of data stream clustering [101]. The highest level aspect is the clustering problem that is addressed: algorithms can be distinguished as performing either object-based clustering or attribute-based clustering. We focus on the former as its applications are more common [101]. The second aspect is the number of user-defined parameters. Possible DSCA parameters include window sizes, decay rates and thresholds. Most notable is that some DSCAs require the number of clusters to find - k - which Silva et al. highlight as handicapping an algorithm's ability to deal with a data stream's dynamic behaviour [101]. The other five aspects are discussed based on how they fit into the online component (Section 2.6.1) or the offline component (Section 2.6.2).

2.6.1 Online Component

A DSCA's online component allows it to process new instances quickly and incrementally. The third aspect is the data structure used to represent the unbounded instances in a compact manner. This can be done using a feature

Table 2.6: Aspects of the online component (adapted from Silva et al. [101])

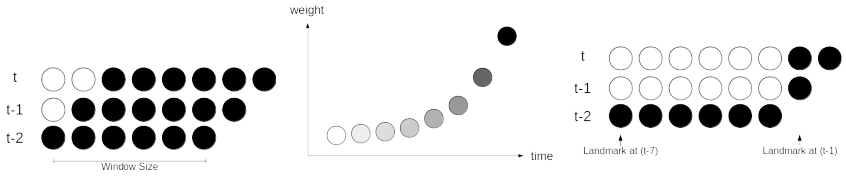
Aspect	Examples
Data Structure	feature vector, representative objects, coreset trees, feature space grids
Window Model	sliding window, damped window, landmark window
Outlier Detection	buffers, user-defined threshold, deleting sporadic grids

vector, introduced by Zhang et al. for their algorithm BIRCH [118]. A microcluster of N instances, $x_1, x_2, ..., x_N$, is summarized as the triple $\langle N, LS, SS \rangle$, where LS is the linear sum of those instances ($\sum_{i=1}^{N} x_i$) and SS is the square sum of those instances ($\sum_{i=1}^{N} x_i^2$) [118]. This is a powerful form of summarization because adding an instance to a feature vector requires only arithmetic and because feature vectors are additive: the feature vector of $(C_1 \cup C_2)$ is equal to the feature vector of C_1 added to the feature vector of C_2 [3, 118]. The data stream can also be summarized using representative objects. Prototype arrays summarize a batch of instances using medioids or centroids from each partition; these prototypes can then be summarized themselves later on [101]. Alternatively these representative objects can be a node in a self-organizing map or a neuron in a growing neural gas [41]. StreamKM++ uniquely uses a data structure called a coreset tree, which organizes $2m$ instances into a tree, and extracts a coreset of m instances. Similar to prototype arrays, two coresets can eventually be combined into another coreset tree and further reduced [2]. Finally, the feature space itself can be divided into grids. Each grid then summarizes the data stream objects that arrive in its portion of the feature space [23].

Reasoning that recent objects are more relevant, the fourth aspect is the window model used to passively forget old concepts. Sliding windows store instances in a first in-first out (FIFO) data structure and remove the oldest instance every time a new one is added. Damped windows weight instances by age, often by using an exponential function, until they are forgotten. Meaningful time-based or stream-based landmarks can be used to break the stream into non-overlapping chunks - landmark windows [101]. As noted by Gama et al., these windows can be of fixed or variable length [40]. The three window structures are illustrated in Figure 2.8 with fixed lengths.

Finally, the fifth aspect of a DSCA is the online component's outlier detection mechanism. The algorithm must decide if a point that doesn't fit the existing clusters represents a new cluster or an outlier to be discarded [8]. Outlier detection mechanisms include buffering outliers until they can be included in the summarization [19, 118], deleting microclusters whose density or relevance is below a threshold [3, 19] and forgetting sporadic grid cells that have stored few instances [23].

Figure 2.8: Window models used by data stream clustering algorithms (adapted from Silva et al. [101])



(a) Sliding window (b) Damped window (c) Landmark window

2.6.2 Offline Component

Table 2.7: Aspects of the offline component (adapted from Silva et al. [101])

Aspect	Examples	
Clustering algorithms	partition-based	density-based
Cluster shape	hyper-sphere	arbitrary

A DSCA's offline component is called when a clustering is required. The summarization is often treated as a static data set and traditional clustering algorithms are used. Here, the DSCA's sixth aspect, its choice of offline clustering algorithm, directly decides the seventh aspect, the shape of the resulting clusters.

The first main approach for the offline clustering algorithm is the k-means family of clustering algorithms. This includes k-means applied to the statistical summary or to a weighted statistical summary, selecting medoids with k-medoids, and using an initial seeding with k-means++. As expected, DSCAs making use of these algorithms result in hypersphere-shaped clusters. The other main approach is to use density-based clustering algorithms, such as DBSCAN, applied to feature vectors, grid cells or frequent states. DSCAs that use density-based clustering have the ability to find arbitrarily-shaped clusters [101].

2.6.3 Data Stream Clustering Algorithms

Having looked at the different taxonomies used to describe DSCAs, we now review those found in the literature. Attention is paid to ensure that both the four methods described in Table 2.5 as well as Silva et al.'s "13 most relevant data stream clustering algorithms to date" are well represented.

2.6.3.1 BIRCH

Zhang et al. described Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) as one of the first DSCAs. The authors' goal was to create a one-pass DSCA that is able to produce high-quality clusters in the presence of noise while being resource-efficient; they achieved this by introducing the Clustering Feature vector and the CF tree [118].

The Clustering Feature vector is a 3-tuple that summarizes all of the available information concerning a cluster (Definition 8). These Clustering Feature vectors are stored in a tree structure with branching factor B and threshold T. Every inner node of the tree has at most B children and every leaf node contains entries whose radius is less than T [118].

Definition 8 (BIRCH Clustering Feature Vector)**.** Given N d-dimensional data points in a cluster, $\{X_1, X_2, \ldots, X_N\}$, the Clustering Feature vector of the cluster is defined as a triple $CF = (N, LS, SS)$. These quantities are sufficient to calculate a variety of the cluster's characteristics, including its centroid and its radius.

$$LS = \sum_{i=1}^{N} X_i \tag{2.19}$$

$$SS = \sum_{i=1}^{N} X_i^2 \tag{2.20}$$

The authors do identify a couple of issues which arise from the manner in which data points are added to the CF tree. First, because nodes are limited in the number of entries they can carry, they may not represent natural clusters. This is addressed by a phase that globally reorders leaf entries across nodes. Second, the same data point inserted twice at two different times in the data stream may end up in different leaf nodes. This is addressed by a further, offline, phase of cluster refinement [118].

2.6.3.2 CluStream

CluStream was introduced by Aggarwal et al. in order to move past what they identified as the conception that DSCAs were simply "a class of one-pass clustering algorithms." Instead, they highlighted that a DSCA needed to be able to handle the dynamic nature of data streams and to produce a clustering that reflected that data stream's state in the present and not too far back in the past [3].

The first aspect they introduced to achieve this was the micro-cluster - a temporal extension of BIRCH's cluster feature vector [3]. They defined a micro-cluster according to Definition 9.

Definition 9 (CluStream Micro-cluster)**.** A micro-cluster for a set of d-dimensional points $X_{i_1}, \ldots, X_{i_n}$ with time stamps $T_{i_1}, \ldots, T_{i_n}$ is defined as the $((2 \times d) + 3)$ tuple $(\bar{CF2}^x, \bar{CF1}^x, CF2^t, CF1^t, n)$.

$C\bar{F}2^x$ is the sum of the squares of the data values for each dimension.

$C\bar{F}1^x$ is the sum of the data values for each dimension.

$CF2^t$ is the sum of the squares of the time stamps.

$CF1^t$ is the sum of the time stamps.

n is the number of data points represented by the microcluster.

These microclusters are very easy to maintain throughout the data stream as the online component's summarization and are stored in snapshots on a pyramidal time frame.[9] A fixed number of microclusters is maintained as the online component's summarization. This summarization is then passed to the offline component, which runs a variant of the k-means algorithm on these microclusters [3].

2.6.3.3 ClusTree

ClusTree was proposed by Kranen et al. as a parameter-free DSCA that self-adapted to whatever characteristic the data stream exhibits [62]. ClusTree stores incoming instances using BIRCH's cluster feature tuples $CF = (n, LS, SS)$ and stores these tuples in an index tree structure (Definition 10). This index tree is balanced to ensure that the time required to reach any of the leaf nodes from the root node is approximately equal.

One of Kranen et al.'s demonstrations regarding the ClusTree algorithm is its ability to keep the tree structure up to date no matter the speed at which instances arrive. This is done by leaving instances partially descended through the tree structure, to be "picked up" the next time another instance passes through that node [62].

Definition 10 (The ClusTree)**.** Given fanout parameters m, M, and leaf node capacity parameters l and L, the ClusTree is a balanced multi-dimensional indexing structure with these properties:

1. The root node has at least one entry, inner nodes each contain between m and M entries, and leaf nodes each contain between l and L entries;
2. An entry in an inner node consists of: a CF of the object it summarize; a CF of the objects in its buffer; and a pointer to its child node;
3. An entry in a leaf node consists of a CF of the object(s) it represents; and
4. The path from the root node to any leaf node is the same length (the tree is balanced).

[9] In the pyramidal time frame, snapshots are stored at different levels of granularity with respect to time. The sampling of newer snapshots is more fine and the sampling of older snapshots is more coarse.

Taking the CF tuples from the leaf nodes as the summary of the data stream, ClusTree produces a clustering using an off-line clustering algorithm such as k-means or DBSCAN [62].

2.6.3.4 D-Stream

Chen and Tu proposed the D-Stream algorithm in order to remove DSCAs dependence on the parameter k [23]. Instead of using k-means or one of its variants to cluster the online component's microclusters, D-Stream uses a density-based clustering algorithm modelled on DBSCAN to cluster grid cells in the feature space.

The use of grid cells is one of D-Stream's uniquely defining features: instead of specifically tracking instances (or their microcluster facsimiles) throughout the data stream, D-Stream partitions the whole feature space into grids and has each cell track the instances that have arrived within its area using a characteristic vector (Definition 11). Although the total memory requirement for this set up is dependent on user parameters, it is completely independent of the data stream itself [23].

Definition 11 (D-Stream Characteristic Vector)**.** The characteristic vector of a grid g is a tuple $(t_g, t_m, D, label, status)$.

t_g is the last time that g was updated.
t_m is the last time that g was removed from being tracked.
D is the grid's density at the last update.
$label$ denotes whether the grid is SPARSE, TRANSITIONAL or DENSE.
$status$ denotes whether the grid is NORMAL or SPORADIC.

Like micro-clusters, D-Stream's characteristic vectors can be easily maintained throughout the data stream. Memory requirements are further alleviated by only tracking those grids that will affect the final clustering; when a clustering is required, clusters are formed out of neighbouring dense grids [23].

2.6.3.5 DenStream

DenStream is Cao et al.'s density-based DSCA with the goals of making no assumptions about the number of clusters, discovering arbitrarily shaped clusters, and handling outliers in the data stream. The authors note that finding arbitrarily-shaped clusters in the data stream environment can be challenging, due to the memory constraints faced by online learners, and propose using a core-micro-cluster synopsis to summarize the clusters [19].

The online component maintains a summarization made up of p-micro-clusters (potential core micro-clusters) and o-micro-clusters (outlier micro-clusters) with the difference between the two being a user specified threshold on micro-cluster weight. Whenever the offline component is required to produce a clustering, a variant of DBSCAN is run on the online component's p-micro-clusters.

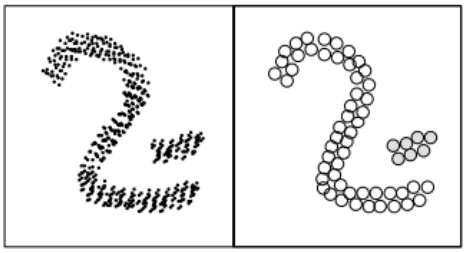
This DBSCAN variant includes two parameters, μ and ϵ, and forms clusters using the concept of direct density reachability between c-micro-clusters (core-micro-clusters, Definition 12).

Definition 12 (DenStream's "directly density-reachable" concept)**.** A p-micro-cluster c_p is directly density-reachable from a p-micro-cluster c_q with respect to ϵ and μ, if:

1. the weight of c_q is above μ (i.e., c_q corresponds a c-micro-cluster); and
2. $dist(c_p, c_q) \leq 2 \times \epsilon$, where $dist(c_p, c_q)$ is the distance between the centres of c_p and c_q.

The relationship of "directly density-reachable" can be chained and the resulting cluster is made up of p-micro-clusters that are all density-connected[10] (Figure 2.9) [19].

Figure 2.9: The points (left) and resulting c-micro-clusters (right) produced by DenStream (from Cao et al. [19])



2.6.3.6 Fractal Clustering

Barbará and Chen noted that DSCAs face many challenges, including data set size, high dimensional data, and the presence of noise. They proposed taking advantage of the fractal properties of data sets using Fractal Clustering: an incremental method that naturally keeps similar points together and dissimilar points apart. Fractals are cited as a promising mathematical formalism to view the clustering problem through since they are the structures that appear when self-similarity is present [9].

Fractal Clustering begins by bootstrapping itself using a traditional offline clustering algorithm to find clusters in a given data set. Once it is given a starting clustering it is able to incrementally add points to this clustering in such a way that the *fractal impact* is minimized. This is done using the fractal dimension, F_d,[11] and shown in Algorithm 2.2 [9].

[10] For any two p-micro-clusters, $c_p, c_q \in C$, there is some p-micro-cluster $c_m \in C$ such that there exists a chain of p-micro-clusters from c_m to c_q AND some chain of p-micro clusters from c_m to c_p such that each successive p-micro-cluster in the chain is directly density-reachable from its predecessor.

[11] specifically Barbará and Chen used a box counting algorithm named FD3, which is based on the ideas of Liebovitch and Toth [75].

Algorithm 2.2 Fractal Clustering - Incremental Phase (from Barbará and Chen [9])

Given a clustering made up of k clusters, $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$.
while $\mathcal{DS}$ has more instances **do**
  Get next instance, $inst$
  for all Clusters C_i in $\mathcal{C}$ **do**
    Let $C_i' = C_i \cup inst$
    Compute the fractal dimension of C_i', $F_d(C_i')$
  end for
  Find $\hat{\imath} = \arg\min_i(|F_d(C_i') - F_d(C_i)|)$
  if $|F_d(C_{\hat{\imath}}') - F_d(C_{\hat{\imath}})| > \tau$ **then**
    Discard $inst$ as noise
  else
    Place $inst$ in $C_{\hat{\imath}}$
  end if
end while

2.6.3.7 G-Stream

Using the idea of what came to be called Kohonen feature maps, [61], Fritzke developed a Growing Neural Gas (GNG) network able to incrementally perform unsupervised learning and dimensionality reduction [35].

Making use of the GNG paradigm, Ghesmoune et al. described G-Stream, which uses the neurons in a GNG to represent any number of arbitrarily-shaped clusters. Unique defining features of G-Stream compared to other DSCAs is that its clustering can begin with as few as two points and all of its functions are performed online [41].

G-Stream represents the data stream with a graph, $\mathcal{C}$, that is made up of nodes, where each node represents a cluster. Incoming points are assigned to the two closest nodes in $\mathcal{C}$. These two nodes are connected in $\mathcal{C}$ and then the nearest node and all of its topological neighbours have their prototypes adjusted towards the new point. Alternatively, if the incoming point is too far from any of the nodes it is added to the reservoir to be dealt with later [41].

The graph is managed throughout the data stream and nodes can be added, removed and merged as required. At any point, the clustering is represented by the collection of nodes, each of which has a prototype and a distance threshold based on the points that have been assigned to them [41].

2.6.3.8 StreamKM++

K-means is one of the classic clustering algorithms and there are many variants on the idea in the literature [46, pg. 451-455]. Ackermann et al.'s algorithm StreamKM++ [2] is a data stream adaptation of one of these variants – Arthur and Vassilsitskii's k-means++ [5].

Ackermann et al. proposed representing the data stream with a coreset suit-

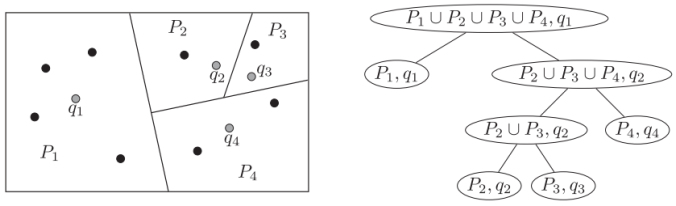
able for the k-means++ algorithm and they accomplished this using a coreset tree. A coreset tree is easily maintainable in an online fashion and is defined in Definition 13 [2].

Definition 13 (StreamKM++'s Coreset Tree)**.** A coreset tree T for a point set P is a binary tree associated with a hierarchical divisive clustering of P.

1. The coreset tree begins with a root node containing the whole point set P;
2. Successively divide nodes into two child nodes such that the subcluster in each child node is far away from the subcluster in the other child node; and
3. Stop when the desired number of leaf nodes are present.

Each node v in the coreset tree contains the point set P_v, a representative point $q_v \in P_v$, and a weight (Figure 2.10). Inner nodes' weights are the sum of the children and leaf nodes' weights are the sum of squared distances from each point in their point set to their representative point. Once the coreset tree has m leaf nodes, a coreset tree of size m can be easily extracted by collecting each leaf node's representative point. This coreset can then be clustered offline using the k-means++ algorithm [2].

Figure 2.10: A coreset tree (from Ackermann et al. [2])



2.7 One-Class Classification in Data Streams

Data streams where the OCC task is likely to be relevant are those where normal behaviour constitutes the vast majority of instances, while the instances of interests - the anomalies or outliers - occur very infrequently. Potential scenarios could include incoming data from an airborne sensor, diagnosis of a rare medical condition, or an intrusion detection system for a computer network.

While Krawczyk and Woźniak recently observed that "the OCC problem for data streams, especially in the presence of concept drift, has not been investigated thoroughly"[12] [65], some one-class classifiers have been adapted for use

[12]echoing the observation of He and Garcia in 2009 [48].

in the data stream environment, while other one-class classifiers have been proposed specifically for the data stream environment. One notable review in the field is Chen and He's chapter concerning "Nonstationary Stream Data Learning with Imbalanced Class Distribution," although the authors restrict themselves to the scenario where minority class instances are available [22]. We review the literature for OCC in data streams here, allowing for the scenario where no minority class instances are available.

2.7.1 Challenges for One-Class Classification in Data Streams

A number of challenges related to class imbalance and one-class classification were highlighted throughout Section 2.1.2. These challenges must also be overcome (or neutralized) in the data stream environment and are sometimes exacerbated by it. These challenges relate to both the changed nature of the data (stream vs. static) and the different kind of learning algorithm required for this environment (online vs. traditional).

2.7.1.1 Challenges Related to the Data

The problem of complex domains in OCC for static data sets is exacerbated by the dynamic nature of the data stream environment. Consider small subconcepts or small disjuncts, which were vulnerable to absolute rarity. Unlike static data sets, in data streams training instances are not equally available throughout. While most windows may contain a representative sample of instances, it is possible that some windows will not. In fact, given enough windows (a reasonable assumption due to the data stream's infinite size) this is guaranteed to occur, even for some sub-concepts that are not "rare." The impact of this problem can be minimized by selecting an adequate window size and we provide an in-depth discussion in the following chapters.

One manner of addressing the problem of small disjuncts in data streams is to use concept specific buffers, where instances are stored according to their sub-concept membership. Although tempting, Chen and He highlighted that this can fall afoul of concept drift: if old instances are being used to learn rare sub-concepts, then it is possible that concept drift has already made them irrelevant [22]. This problem must also be kept in mind if buffers are used as the basis for oversampling techniques. Chen and He recommended the selective discarding of these instances over restricting learning to a single batch only, arguing that the latter would likely result in unnecessary forgetting [22].

Concept drift affects more than just small disjuncts, however. As with all data stream classification, the possibility of a change in the data stream's underlying process means the classifier's model must also be able to change. There are multiple ways that a traditional one-class classifier can be adapted for this requirement and these are addressed in the next section.

2.7.1.2 Challenges Related to the Learner

In discussing learners adapted for data streams, two common adjectives are *incremental* and *online*. Losing et al. define incremental learners as those that, given a data stream $\{x^1, x^2, \ldots, x^N\}$, produce a sequence of models $h_1, h_2, \ldots, h_N$ where model h_i depends solely upon model h_{i-1} and a strictly limited number of recent training objects [78]. They then specify that online learners are incremental learners that are also restricted in terms of model complexity and run-time: an online learner should be able to continue forever while consuming only limited resources [78].

The presence of concept drift in data streams is a major factor in the changes made when transitioning classifiers from static data sets. A variety of strategies for dealing with concept drift have been proposed which are applicable to one-class classifiers as well: the classifier could learn a new model when signalled by a concept drift detection method; the classifier could always learn a new model over the most recent batch of instances; or the classifier could continuously update its model [22, 40, 119]. The first method requires an explicit concept drift detector, as reviewed in Section 2.5.3, which increases the framework complexity. The second likely leads to unnecessary forgetting, as mentioned above [22]. Finally, the third requires that the learner be capable of incrementally updating its model.

Another requirement for online learners is that they be able to provide a prediction at any point in the data stream [119]. Unlike classifiers for static data sets and leaving aside the issue of infinite length, there is no ability to see the whole data stream before making decisions. Most learners do need a certain amount of training before they are capable of producing a reasonable decision: decision trees must be grown; neural networks must have their weights converge; and nearest neighbour approaches need a neighbourhood. This can be achieved by providing a one-class classifier with an initialization period at the beginning of a data stream, where instances are provided for training only and not for testing.

Finally, online learners are required to complete their task using limited time and memory [78]. Although the performance of an algorithm is certainly an important consideration, a more fundamental limitation is that not all of a data stream's instance can be maintained. It is for this reason that Losing et al. base their description of an online learner on an incremental learner, which ensures that the classifier's model can always be adapted and not retrained from scratch [78]. This approach also permits passive adaptation by the classifier to concept drift, which makes it a promising option to consider.

2.7.2 Streaming One-Class Classifiers

As reviewed in Section 2.2, there are many one-class classifiers in the literature. While it is clear that they cannot be applied to data streams straight away, the underlying OCC problem that they address has not changed dramatically. Our survey of streaming one-class classifiers reveals a high degree of similarity with

one-class classifiers for static data sets and so we use Tax's three approaches to categorize them.

2.7.2.1 Density-based

As for static data sets, density-based one-class classifiers for data streams build a model of the majority class's density throughout the entire feature space and use it to predict whether a test instance belongs to the majority class or not. Because of the nature of data streams, this is often done using tree structures.

One Class Very Fast Decision Tree Li et al. identified that little work had been done to date concerning the OCC problem as it applied to data streams and developed the One-class Very Fast Decision Tree (OcVFDT) is a direct adaptation of the well-known Very Fast Decision Tree (VFDT) algorithm, regarding which much work had been done [74].

Considering data streams with discrete attributes only, both positive and unlabelled instances, and no concept drift, Li et al. described an approach for growing OcVFDTs using one-class Information Gain and the Hoeffding bound. Each OcVFDT is grown assuming a different level of class imbalance in the stream and one of these is selected using a set of validation instances. At test time, each leaf node of the chosen OcVFDT is labelled with the label of the majority of instances that traverse to there [74].

Experiments with synthetic and real-world data streams showed that, even with up to 80% of the data stream unlabelled, OcVFDT could nearly match the performance (as measured by accuracy and $F1$) of a VFDT classifier with access to labelled training instances from both the majority and minority classes.

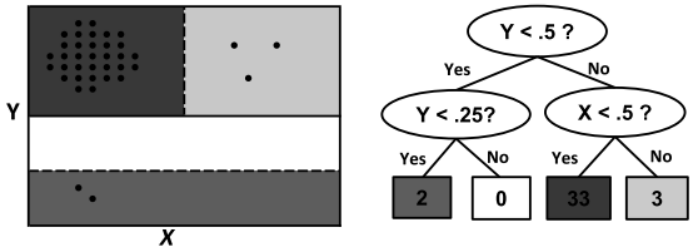
Streaming Half-Space Trees Similar to the random forest, an ensemble of decision trees [46, pg. 382-383], and Hempstalk et al.'s Bagged Decision Trees [50] is Tan et al.'s Streaming Half-Space Trees (HS-Trees) [108]. Streaming HS-Trees uses the relative mass of the different leafs in its trees to detect anomalies; these are updated blindly after every window, meaning that the algorithm adapts passively to concept drift. Like the Bagged Decision Trees, Streaming HS-Trees are a density-based one-class classifier.

Uniquely, these trees are not induced based on instances but instead on random perturbations of the data space itself. Each node in the HS-Tree randomly selects an attribute and grows two children leafs to represent either half of the attribute, as seen in Fig. 2.11 [108].

Every test instance is assigned an anomaly score to reflect how normal or anomalous the HS-Trees believe it to be. This anomaly score is generated by passing the test instance to each tree in the ensemble; each tree passes the instance through its structure until it reaches a terminal node, $Node^*$, and reports that node's mass (2.21). The depth of $Node^*$ in tree T is $Node^*.k$; the number of instances present in $Node^*$ is $Node^*.r$.

These scores are then summed to provide the instance's final anomaly score (2.22) [108]. It is worth noting that this formulation of the anomaly score is

Figure 2.11: A Streaming HS-Tree's partition of the data space (from Tan et al. [108])



more of a density score, and so normal instances will have higher scores while outliers or anomalies will have lower scores.

$$AnomalyScore_T(x) = Node^*.r \times 2^{Node^*.k} \tag{2.21}$$

$$AnomalyScore_{Total}(x) = \sum_{T \in HS-Trees} AnomalyScore_T(x) \tag{2.22}$$

2.7.2.2 Reconstruction-based

Reconstruction-based one-class classifiers again work to compress the training instances of the majority class and then reconstruct or recognize them, with the reconstruction error used to determine how well a test instance fits in with the majority class. Their compact, training set-independent model makes them a natural option to consider for the data stream learning environment.

Streaming Autoencoders Autoencoders are naturally applicable to data streams as they inherently adjust their parameters in an incremental manner [33]. Dong and Japkowicz described Streaming Autoencoders (SAs), which is in adaptation of the standard autoencoder. SAs model the data identically to static data autoencoders, which means they remain reconstruction-based.

An initial window of instances is used over multiple training epochs to reduce the effect of random starting weights; once this initialization is complete then instances are passed to the SA which updates its weights [33]. The number of initial epochs must be chosen so as to avoid overfitting, although the later instances received while offline can also help mitigate this problem.

The autoencoder's incremental weight updating is an advantage in the data stream environment and it allows SAs to passively adapt to concept drift. Threaded ensembles of SAs were tested on benchmark data sets and were found to perform better than the benchmark VFDTs in terms of area under the curve (AUC) and faster than both the VFDTs and Streaming Multilayer Perceptrons [33].

DETECTNOD Another compression and reconstruction-based approach to OCC in data streams is DETECTNOD. Hayat and Hashemi proposed the Discrete Cosine Transform (DCT)-based approach to address the task of novelty and concept drift detection in data streams [47].

DETECTNOD works by compressing the data stream into a few DCT coefficients and using these as a generative model for the existing "normal" behaviour. First, a window of instances is clustered using k-means, then each cluster is further clustered using k-means again. Each sub-cluster is brought to a threshold value number of instances through interpolation or down sampling. These sub-clusters are then transformed into DCT coefficients that can then be used to recognize the "normal" behaviour later on [47].

Throughout the data stream a similar process occurs and the new DCT coefficients are compared to the generative model's coefficients using Equation 2.23, giving the Unknown Score of instance x [47].

$$UnExSco(x) = \sqrt{\sum_{i=1}^{NumDCT} (GenerativeModel_i - CurrentModel_i)^2} \quad (2.23)$$

The appropriate number of DCT coefficients to use is determined experimentally and the threshold on $UnExSco$ is set depending on the underlying data (for example, if a sub-cluster is normally distributed then the threshold $\mu \pm (3 \times \sigma)$ is used). In experiments performed on a range of benchmark data sets from the UCI Machine Learning repository, DETECTNOD performed favourably against a range of other novelty detection methods when evaluated for over- or under-fitting [47]. Sethi and Kantardzic identified that DETECTNOD could also be used to perform cluster-based implicit concept drift detection [98].

2.7.2.3 Boundary-based

Boundary-based one-class classifiers again offer a simple and intuitive way to represent the majority class. In terms of their suitability for the data stream environment, though, these classifiers do vary in their strengths and weaknesses.

Nearest Neighbour Data Description The nearest neighbour family of lazy learners is a family of instance-based supervised learners. These learners are often used with static data sets because they take no time to train and instead only compare instances at classification time [46, pg. 423]. This aspect meets the requirements of an online algorithm in the data stream environment, in fact the lazy learner approach seems immediately applicable to data streams.

Inspired by streaming adaptations of nearest neighbour classifiers (e.g. Shaker and Hüllermeier's IBLStream [99] and Losing et al.'s SAM-kNN [78]), herein we adapt Tax's NN-d method to the data stream. This requires only one change: instead of a static neighbourhood, the classifier keeps track of a neighbourhood of the last w instances. This neighbourhood is initialized as a FIFO list and

filled; future instances are added only if the classifier believes they are from the majority class.

The size of neighbourhood to maintain is the major parameter for this algorithm: a smaller window will allow quicker adaptation to concept drift while a larger window will ensure more robustness to noise. Either way, the neighbourhood is kept up to date as new instances are added to replace the old, which ensures that the algorithm adapts passively to concept drift. The decision function is the same as the NN-d method for static data sets, (2.9), so the streaming implementation of NN-d remains a boundary-based method.

Incremental Weighted One Class Support Vector Machine Although not inherently incremental (the hyper-plane and its associated support vectors are learned all at once), modified versions of the OCSVM have been described in the literature for use with data streams, e.g. Krawczyk and Woźniak's adaptive Weighted One-Class Support Vector Machine (WOCSVM) [64].

At the heart of Krawczyk and Woźniak's method is an incremental method for calculating the WOCSVM's support vectors, where newer or more outlying instances are given higher weights. Each instance's weight is decreased over time, with or without considering their original weight, allowing the classifier to slowly forget old instances that should not impact the current decision boundary [64].

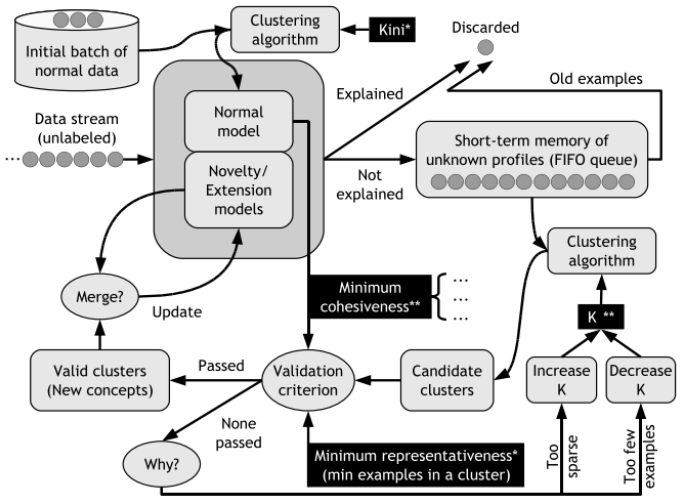
Although the authors were not able to draw generalizable conclusions, noting that the performance seemed to be data set-specific, they did observe that weighting training instances based on their age and gradually decreasing them did produce the best performance [64].

OLINDDA Extended from the concept drift detection method OLINDDA [105], Spinosa et al. proposed this version of OLINDDA which incorporates a number of improvements [104]. Starting with an initial batch of training data, OLINDDA uses k-means clustering (although the authors note that other algorithms could be used) to produce a static model of "normal" behaviour.

During the online portion of the algorithm, test instances are compared against this model. If they are not within the boundaries of the model's hyperspheres, they are added to a FIFO queue for short-term memory [104]. Every time a test instance is added to short-term memory, k candidate clusters are produced and checked for validity. If a cluster is determined to be valid (based on two aspects: cohesiveness and representativeness) it is then labelled as either an extension of the normal model or a novel concept altogether [104].

Spinosa et al. identified that OLINDDA was both capable of learning novel concepts and was robust to outliers. They did note as a shortcoming however, that the k-means algorithm produces hyper-sphere shaped clusters, which may not describe the actual distributions very well [104].

Figure 2.12: An overview of the OLINDDA Novelty Detection approach (from Spinosa et al. [104])



2.7.3 Summary

We identified that a number of the challenges associated with class imbalance and OCC are made worse when considering the data stream environment. There are also new challenges to be dealt with that exist solely because of the data stream's dynamic nature. Challenges due to the data include the rarity of instances from disjuncts in certain training windows, generating sufficient instances from these small disjuncts without relying on outdated data, and preparing to track a changing majority class. Challenges due to the learner's requirements include having an adaptable model, providing predictions throughout the data stream, and doing both of these with limited time and memory.

A range of one-class classifiers are described in the literature. In the case of the data stream environment, however, the literature is less rich than it is for static data sets. Nonetheless, we see that Tax's three approaches to OCC are again embodied by streaming one-class classifiers and that algorithms have been proposed for each approach. The pros and cons of each approach for the OCC problem remain the same as in Table 2.1 and these will not be re-examined. The approaches do, however, offer different pros and cons with respect to the data stream environment.

Density-based approaches need to represent the entire feature space for a dynamic environment. This leads to the use of decision trees, which are quick to build, quick to evaluate and can quickly break a large feature space into many small sub-spaces represented by leaf nodes. One problem with this approach can be that the tree complexity required may scale with the number of instances, which not ideal for a stream learning algorithm. Related is the problem of relearning trees: as noted by Dong and Japkowicz, decision trees can be brittle

in the presence of concept drift and can be costly to relearn [33].

Reconstruction-based approaches represent the entire feature space with a compact model that is independent of training set size – an ideal characteristic for a stream classifier dealing with memory constraints. If this model can be updated incrementally, as with the SA, then it is very well suited for the data stream environment. If not, as with DETECTNOD's DCT representation, then some additional system needs to be put in place in order to trigger the creation of a new generative model.

Boundary-based one-class classifiers exhibit a range of characteristics that are friendly and antagonistic to the data stream environment. Lazy learners, for example, are easily applied to data stream learning because of their constant train time while on the other hand the WOCSVM has to solve a quadratic problem. In terms of model size, though, the NN-d method requires the full training set while the WOCSVM needs only the support vectors. For both considerations, OLINDDA represents something of a middle ground. Its clusters and memory represent a compressed training set, though possibly larger than the set of support vectors. In terms of training time, OLINDDA's requirement to cluster the short-term memory instances leads to longer times than lazy learners, but shorter times than an OCSVM.

The other important aspect of this discussion is that active concept drift detection is not a specific requirement for streaming one-class classifiers learners. Stream learners can be designed to adapt passively to changing data streams, preempting the need for explicit concept drift detection. This is especially clear when inspecting the cluster-based implicit detectors given by Sethi and Kantardzic: many of these can be used as one-class classifiers and/or novelty detection frameworks in their own right (e.g. OLINDDA [105, 104], DETECTNOD [47], MINAS [28]) [98]. This indicates a relationship between passive adaptation by one-class classifiers and implicit concept drift detection, a relationship that will be utilized when designing the frameworks proposed in this thesis.

2.8 Divide and Conquer in Data Streams

Constructing ensembles of stream learners also makes use of the idea that a problem can be usefully divided into more easily solved sub-problems. Both Gomes et al. [43] and Krawczyk et al. [63] identified that ensembles are a well-grounded method to improve performance over a single classifier. One reason for this is that it can be easier to combine several weak classifiers into a strong ensemble than it is to learn an equally strong classifier [43]. Krawczyk et al. highlighted that a specific strength of ensembles is that they incorporate the idea of "divide and conquer," allowing different classifiers to concentrate on performing well in their area of strength[13] [63].

[13]Respecting Wolpert's *no free lunch* theorem [116].

2.8.1 Sub-Division Applied to Data Streams

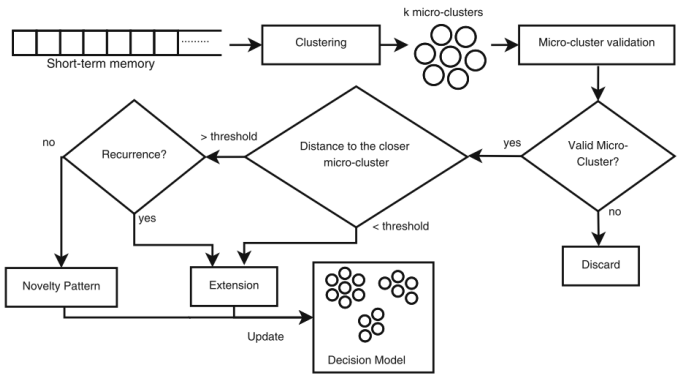
Four recent frameworks or algorithms that make use of the idea of a sub-concept structure have been proposed. Each builds from the assumption that clusters can provide a meaningful representation of the data which then makes it easier to learn a model.

The ClusFirstClass algorithm, proposed by Sobhani et al. [102], uses clustering to improve classifier performance for imbalanced data sets. ClusFirstClass clusters the majority class in a binary class data set and then performs cluster-based undersampling. Using clusters to guide the undersampling (as in Jo and Japkowicz's cluster-based oversampling [59]) ensures that information from small disjuncts is not lost and was shown to out perform a simple ensemble as well as cluster-based oversampling [102].

Methods which use clustering to assist semi-supervised learning in data streams have been proposed [4, 52]. Hosseini et al. propose semi-supervised pool and accuracy-based stream classification (SPASC), which maintains a pool of semi-supervised classifiers that define their underlying concept through clusters. Hosseini et al. showed that this structure was useful in detecting and adapting to recurring concepts in the data stream since different classifiers can be used to represent different concepts [52].

Separately, Al-Jarrah et al. described semi-supervised multi-layered clustering model (SMLC), which produces a series of overlapping clusterings and identifies fully labelled, partially labelled and unlabelled clusters [4]. Fully labelled clusters have a binary classifier trained on them; partially labelled clusters have a tri-training model trained on them; and unlabelled clusters have their centroids labelled using the nearest neighbour approach before training a tri-training model. SMLC was shown experimentally to perform as well as machine learning techniques trained on fully labelled instances for two intrusion detection data sets [4].

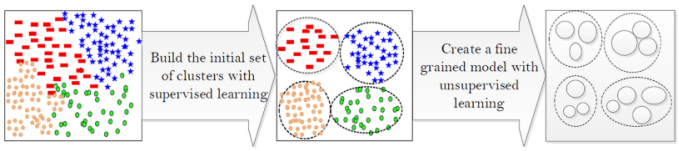
Figure 2.13: MINAS's novelty detection procedure (from Faria et al. [28])



Faria et al. describe MultIclass learNing Algorithm for DSs (MINAS), which clusters the data stream to produce sub-concepts for the novelty detection task

in a multi-class scenario [28]. Microclusters are used to model the patterns of all the classes present in the data stream. This model is then updated throughout the data stream by either extending existing patterns or by incorporating novel patterns that appear in the data stream [28]. This process is illustrated in Figure 2.13.

Figure 2.14: AnyNovel's base-line learning model (from Abdallah et al. [1])



Also applied to multi-class scenario in data streams, Abdallah et al.'s AnyNovel framework is a another clustering based approach for detecting novel concepts in evolving data streams [1]. AnyNovel clusters each of the classes present in the data stream to produce a fine-grained model of the existing stream behaviour (this is the base-line learning model, shown in Figure 2.14). Novel concepts can then be identified if they are within a class but represent a new sub-concept or if they are outside of a class and its Dynamic Relaxed Boundary, to account for the possibility of concept drift. AnyNovel makes use of active learning in the case of the most uncertain data and when a novel or outdated concept is detected [1].

2.8.2 Summary

Table 2.8: Uses for sub-division in data streams

Use	Benefit	Source
Ensembles of stream learners	Combining several weak classifiers often results in better performance than a single strong classifier	[43, 63]
Semi-supervised learning	Clusters can represent sub-concepts in data streams and guide both the labelling of unlabelled instances as well as recognizing recurring concepts	[4, 52]
Novelty detection	Multiple classes can be represented by clusters or microclusters, which allows for novelty detection	[1, 28]

The dynamic nature of a data stream makes it harder to consistently train a capable classifier as can be done for static data sets. This makes the idea of training an ensemble of weaker classifiers very attractive, as long as they can be effectively combined into a single framework.

Ensembles pair well with the idea of dividing a machine learning problem into sub-problems. This has been demonstrated for semi-supervised learning, active learning, and novelty detection for data streams. The way in which sub-concepts (especially those produced by clusters) have been effectively used in these contexts suggests that there is merit in applying the idea to the OCC problem in data streams as well. Given the dynamic nature of data streams, however, this application is not straight forward and this is the subject of the remainder of this thesis.

2.9 Summary

Classification is a well understood machine learning task. One condition that complicates this task, however, is the class imbalance problem. OCC is the extreme example of the class imbalance problem and requires classifiers to learn without any examples from the minority class. It is made more challenging by possibly complex domains and rare cases resulting in small, hard-to-learn disjuncts.

A variety of techniques have been used to improve classifier performance when faced with a class imbalance or OCC scenario. Oversampling, especially sub-component based oversampling, ensures that there are enough training instances for the classifier(s). Dividing the problem and producing a mixture of experts allows each classifier to specialize on a specific portion of the overall problem. These portions can be decided through domain knowledge or through unsupervised learning.

There are also classifiers that have been specifically designed for OCC: one-class classifiers. Instead of discriminating between classes, one-class classifiers learn to recognize the one-class for which training instances exist. These classifiers make use of either a density-based, reconstruction-based, or boundary-based approach to recognize the majority class.

Although OCC is a difficult problem, performing it in the data stream environment exacerbates existing challenges and adds new ones. First, the data stream environment is challenging because of its (potentially infinite) size, its requirement to produce a decision at any point, and its dynamic behaviour, i.e. the possibility of concept drift. Each of these imposes constraints on the classifier and requires it to continually learn the best model it can with a minimum of data.

OCC in data streams has been studied, but not to the same depth as for static data sets. Although streaming one-class classifiers have been proposed, there exists the possibility to adapt some of the more sophisticated techniques as well. Sharma's framework for training one-class classifiers on the majority class's sub-concepts, for example, has only been demonstrated for static data sets. The idea of sub-division in data streams has been used for semi-supervised learning, active learning, and novelty detection for multi-class classification problems, but it has not been applied specifically to the OCC problem.

The next chapter details the contributions of this thesis. Three frameworks

are described that make use of the majority class's sub-concept structure in order to address the OCC problem while simultaneously adapting to the challenging data stream environment. We define a new Cluster Distance Function, provide a theoretical proof regarding the appropriate window size and provide an experimental framework to select a DSCA for use in our frameworks. To support our experimentation, we also describe three synthetic data stream generators.

Chapter 3

Contributions

This thesis makes five contributions to the machine learning literature. The primary contribution is the development of three frameworks that use knowledge of the majority class's sub-concept structure to improve the performance of streaming one-class classifiers. This is supported by two theoretical contributions, a proof regarding appropriate window sizes and the development of a new Cluster Distance Function, as well as experimental evidence to guide the selection of a DSCA. Finally, a novel synthetic data stream generators is described, as well as two modifications to existing data stream generators, in support of this thesis's experiments.

3.1 Frameworks applying Sub-Concept Knowledge to One-Class Classification in Data Streams

The central contribution of this thesis is to demonstrate how one-class classifiers can use knowledge of the majority class's constituent sub-concepts to achieve better performance in data streams. Although Sharma has addressed the usage of sub-concept knowledge in an in-depth manner for static data sets [100], we reviewed the challenges when it comes to applying it to the data stream environment in Section 2.7. In this chapter we describe the frameworks we will use to apply knowledge of the majority class's sub-concept structure in data streams.

3.1.1 Single Classifier

The baseline for all of our comparisons will be a single classifier with no knowledge of the data stream's sub-concept structure. This is distinct from the third scenario in Table 2.2. In that scenario there is no knowledge of the sub-concept structure, the framework attempts to infer it from the training set. By contrast, the single classifier is ignorant of the possibility of the sub-concept structure and does not make use of this idea. Ideally the performance of a single classifier represents the minimum performance that we expect from our frameworks: at no

point should knowledge of the sub-concept structure lead to worse classifier performance.

3.1.2 Complete Knowledge of Sub-Concepts

The first of Sharma's scenarios is where the sub-concept structure is known and where we can identify which sub-concept each instance belongs to [100]. In the data stream environment, this is equivalent to being able to assign instances to sub-concepts for both the initial training instances and the instances seen during the online phase.

An example of this would be streaming data from airborne sensors. Here, the location being sensed will certainly have an impact on the characteristics of the majority class. This location data is available for the initial training data, perhaps even as part of the instances themselves. It will also be readily available during the online phase, such as from the airplane's global positioning system receiver.

In this case, during initialization a one-class classifier is trained on each concept (Algorithm 3.1). During the online phase (Algorithm 3.2), test instances from a data stream, $\mathcal{DS}$, are checked for sub-concept membership. Test instances are then passed to the appropriate one-class classifier, which returns an anomaly score. Online training can occur whenever the test instance's label is received or, potentially, the classifier can be trained on all test instances it deems to belong to the majority class.

3.1.2.1 The OCComplete Framework

Parameters for the OCComplete framework are: $initialPoints$, the number of training instances to use during initialization; $minPoints$, the minimum number of training instances required for each sub-concept; j, the number of sub-concepts determined by domain knowledge or by inspection; τ, the threshold anomaly score to label a test instance as an outlier; and $Classifier$, the base classifier chosen to train on each of the sub-concepts.

For initialization, the goal is to adequately train a one-class classifier on each sub-concept. This is done by putting the first $initialPoints$ instances from the data stream into the buffer corresponding to their sub-concept. Because complete knowledge is available regarding the sub-concept structure, this process is as simple as a query or a look-up. These buffered instances are then used to train the base classifiers and ensure that the framework is capable of providing predictions for future data stream instances.

Depending on the domain, the available training instances may not include sufficient representation of each of the sub-concepts (see Section 3.2 for an in-depth discussion). This is especially problematic for a one-class classifier that needs to converge, e.g. SA. If this is the case, then sub-concept based oversampling can be performed before training the one-class classifiers (Algorithm 3.1, Step 11).

In this thesis we use the SMOTE, which is a well-known method for over-sampling that has been shown to increase the number of minority class instances available for training without leading to over-training (as with replication) or inappropriately expanding the minority class's region [21]. Further discussion is provided in Section 3.1.5.

Algorithm 3.1 OCComplete - Initialization Phase

Initialize instance buffers $\text{Buffer}_{1...j}$
$numInstances \leftarrow 0$
while $\mathcal{DS}$ has more instances $\wedge$ $numInstances < initialPoints$ **do**
  Get next instance, $inst$
  Query $inst$ to determine which sub-concept it belongs to, $c \in [1, j]$
  Add $inst$ to Buffer_c
  $numInstances++$
end while
for all $c \in [1, j]$ **do**
  while $|\text{Buffer}_c| < minPoints$ **do**
    Add new SMOTE instance to Buffer_c
  end while
  Train a $Classifier$ on Buffer_c
end for
return j Classifiers, each trained on a different sub-concept

Once initialization is complete, the framework is ready for use in an online context. Here each test instance is provided to the classifier corresponding to its sub-concept and this classifier returns an anomaly score. In this way, irrelevant or confusing information from other sub-concepts can be ignored while deciding the nature of the test instance. The result is a MELE, where each one-class classifier is an expert on a given sub-concept.

Given the dynamic nature of data streams and the possibility of concept drift, the various one-class classifiers should be kept up to date. This is done by training each one-class classifier on instances from the data stream which belong to the respective sub-concept. In the case where a one-class classifier can only be trained on majority class instances (e.g. Nearest Neighbour Description) this training can occur when a label is received or on the basis of the instance's anomaly score (Algorithm 3.2, Step 10).

3.1.3 Fuzzy Knowledge of Sub-Concepts

Sharma's second scenario is that the sub-concept structure is known, but it is expensive to identify which sub-concept an instance belongs to [100]. For data streams, this is equivalent to being able to assign the initial training instances to their sub-concepts, but not those seen during the online phase.

An example of this would be the diagnosis of a medical condition. Here, the patient and potential disease can affect the characteristics of the majority class.

Algorithm 3.2 OCComplete - Online Phase

while $\mathcal{DS}$ has more instances **do**
  Get next instance, *inst*
  Determine which sub-concept *inst* belongs to, $c \in [1, j]$
  AnomalyScore is equal to the anomaly score returned by $Classifier_c$ for *inst*
if $AnomalyScore > \tau$ **then**
  Label instance as an **OUTLIER**
else
  Label instance as **NORMAL**
end if
if Conditions for training are met **then**
  Train Classifier$_c$ on *inst*
end if
end while

Although a medical professional can analyze the test data and determine which sub-concept a case belongs to, this is an expensive process and can only be done for the initial training instances. In this case, the sub-concept label is readily available for training instances but unavailable during the online phase.

Here a multi-class classifier is trained during initialization to discriminate between sub-concepts and a one-class classifier is trained on each concept (Algorithm 3.3). During the online phase (Algorithm 3.4), test instances from a data stream, $\mathcal{DS}$, are classified by the multi-class classifier as belonging to a given sub-concept. Test instances are then passed to the appropriate one-class classifier, which returns an anomaly score. Online training can occur in the same manner as before for the one-class classifier while the multi-class classifier can be kept up to date using concept drift detectors.

3.1.3.1 Concept Deciding Classifier

An important choice is which multi-class classifier to use for assigning test instances to sub-concepts. No global recommendation can be made (due to, for example, Wolpert's *no free lunch* theorem [116]) and the specific decision should be made according to domain knowledge and experimentation with the available training instances. Options include Naïve Bayes, an ensemble of binary SVMs, k-Nearest Neighbour approaches, Random Forests, etc.

3.1.3.2 The OCFuzzy Framework

Parameters for the OCFuzzy framework include $initialPoints$, $minPoints$, j, τ, and $Classifier$ as before. An additional parameter is $ConceptDecider$, the multi-class classifier to decide which sub-concept a given test instance belongs to.

As before, the framework begins with an initialization period. Here, though, the concept deciding multi-class classifier must also be trained, on top of the base classifier for each sub-concept. Again, if certain sub-concepts are under-represented in the training data then sub-concept based oversampling using SMOTE can be used.

Algorithm 3.3 OCFuzzy - Initialization Phase

```
    Initialize instance buffers Buffer_{1...j}
    numInstances ← 0
    while DS has more instances ∧ numInstances < initialPoints do
      Get next instance, inst
5:    Determine which sub-concept inst belongs to, c ∈ [1, j]
      Add inst to Buffer_c
      numInstances++
    end while
    for all c ∈ [1, j] do
10:   while |Buffer_c| < minPoints do
        Add new SMOTE instance to Buffer_c
      end while
      Train a Classifier on Buffer_c
    end for
15: Train ConceptDecider on Buffer_{1...j}
    return ConceptDecider trained to distinguish between j sub-concepts; j
    Classifiers, each trained on a different sub-concept
```

Since it is not possible to assign instances to their proper sub-concepts during the online phase, the multi-class classifier is instead used as a usable representation of the domain knowledge. During the online context, the multi-class classifier decides which sub-concept a test instance belongs to and this test instance is passed to the appropriate one-class classifier. As before, only the chosen one-class classifier is asked to provide an anomaly score – ensuring only relevant knowledge is applied – and the resulting framework is a MELE.

While the one-class classifiers can be trained throughout the data stream as previously described, the multi-class classifier presents a challenge. Getting a sub-concept label for a test instance is an expensive process, which is what motivated the OCFuzzy framework in the first place. If concept drift is anticipated, then the multi-class classifier could be trained on a small subset of instances that have been labelled by a domain expert as they occur in the data stream or a concept drift detection method could be used to trigger a complete re-initialization of the framework (Algorithm 3.4, Step 13).

3.1.4 No Knowledge of Sub-Concepts

In the third, worst-case, scenario presented by Sharma, the sub-concept structure exists but it is unknown and must be inferred before it can be used [100].

Algorithm 3.4 OCFuzzy - Online Phase

while $\mathcal{DS}$ has more instances **do**
  Get next instance, *inst*
  Have ConceptDecider return the sub-concept to which *inst* belongs, $c \in [1, j]$
  AnomalyScore is equal to the anomaly score returned by $Classifier_c$ for *inst*
if $AnomalyScore > \tau$ **then**
  Label instance as an **OUTLIER**
else
  Label instance as **NORMAL**
end if
if Conditions for one-class training are met **then**
  Train Classifier$_c$ on *inst*
end if
if Conditions for multi-class training are met **then**
  Train ConceptDecider on *inst*
end if
end while

This is easily understood in the data stream context: although the sub-concept structure is believed to exist, there is no specific knowledge that can guide instance assignment.

An example of this is the network intrusion detection task. An intrusion detection system (IDS) monitoring a computer network would benefit from knowing the context during which a specific packet or system trace occurred. This obviously cannot be done in real time and it is unclear if it could usefully be done for initialization.

Here the initial training points are clustered using a DSCA; the clustering is used as a facsimile of the sub-concept structure and a one-class classifier is trained over each cluster (Algorithm 3.5). During the online phase (Algorithm 3.6), test instances from the data stream, $\mathcal{DS}$, are assigned to the sub-concept of the nearest cluster. Test instances are passed to the associated one-class classifier, which returns an anomaly score. Online training occurs in the same manner for the one-class classifiers while the clustering is kept up to date by training the DSCA on each training instance.

3.1.4.1 The OCCluster Framework

Parameters for the OCCluster framework include *initialPoints*, *minPoints*, τ, and $Classifier$ as before. Additional parameters are: *ClusterAlgorithm*, the DSCA with which to cluster the the data stream, $\mathcal{DS}$; w, the number of instances between updates of *ClusterAlgorithm*'s clustering; and T, the cluster distance threshold above which two clusters are considered to be different.

The goal of initialization for the OCCluster framework is to produce a clus-

tering that adequately represents the sub-concept structure as well as training each of the one-class classifiers on their respective sub-concepts. The sub-concept structure is found using the DSCA: the first clustering is produced at the end of the initialization period in order to allow proper clusters to have formed. Any cluster whose weight is less than a given threshold is removed to ensure that the framework only learns classifiers over instances that are likely to continue occurring (Equations 3.1-3.2).

$$\text{weight}(c) = \frac{\text{num. points in } c}{\text{total points clustered}} \tag{3.1}$$

$$\text{Threshold} = \frac{1}{\|\mathcal{C}\|^2} \text{ where } \|\mathcal{C}\| \text{ is the number of clusters in } \mathcal{C}. \tag{3.2}$$

While sub-concept based oversampling using SMOTE can be performed before training the one-class classifiers, it should not be used for the DSCA in order to avoid skewing the resulting clustering.

During the online context, the cluster closest to the test instance is used to determine which one-class classifier will provide an anomaly score. The clustering is updated every w instances; w must be chosen to minimize the framework's sensitivity to fluctuations in the relative likelihoods of different sub-concepts and to maximize the framework's adaptability to concept drift. Because the one-class classifiers are trained on the basis of clusters in the data stream and not any specific knowledge, OCCluster results in a MILE.

Updating the clustering/classifier pairing after a new clustering is acquired is a challenging task. To do so, each of the new clusters, c' in $\mathcal{C}'$, is compared to the old clusters, c in $\mathcal{C}$ using a cluster distance function, CD, which we define in Section 3.3 (Algorithm 3.6, Step 17). If the distance from a new cluster to its closest old cluster is below the threshold T then the old cluster's classifier is assigned to the new cluster; otherwise a new classifier is trained over the new cluster. Similar to the trade-off with the parameter w, T must be chosen to minimize the framework's forgetting of relevant information and to maximize the framework's forgetting of irrelevant information. Ideal values for these parameters are likely data stream-/domain-specific; users should take these conditions into account.

Throughout the online phase (Algorithm 3.6, Step 17), clusters below a certain relative weight are pruned to avoid learning classifiers over clusters that might occur due to a series of similar anomalies or due to noise during a given window. In order to allow this threshold to dynamically adapt to clusterings with different numbers of clusters, we use the same formulation as used in the initialization phase (Equation 3.2).

3.1.5 Sub-Concept Based Oversampling

Hoens and Chawla identify both undersampling and oversampling as standard approaches to address the class imbalance problem [51]. As discussed in the

Algorithm 3.5 OCCluster - Initialization Phase

Initialize instance buffer InstBuffer
$numInstances \leftarrow 0$
while ($\mathcal{DS}$ has more instances) $\wedge$ ($numInstances < initialPoints$) **do**
  Get next instance, *inst*
  Train $ClusterAlgorithm$ on *inst*
  Add *inst* to InstBuffer
  $numInstances++$
end while
Get clustering $\mathcal{C}$ from $ClusterAlgorithm$
for all clusters c in $\mathcal{C}'$ **do**
  if $weight(c) > \frac{1.0}{\|\mathcal{C}\|^2}$) **then**
    Remove c from $\mathcal{C}$
  end if
end for
Set k equal to the number of clusters in $\mathcal{C}$
Initialize k instance buffers $\text{Buffer}_{1...k}$
for all Instances *inst* in InstBuffer **do**
  Determine the cluster, c, to which *inst* is closest
  Add *inst* to Buffer_c
end for
for all $c \in [1, k]$ **do**
  while $|\text{Buffer}_c| < minPoints$ **do**
    Add new SMOTE instance to Buffer_c
  end while
  Train a Classifier on Buffer_c
end for
return A clustering $\mathcal{C}$ with k clusters; k $Classifier$s, each trained on a different sub-concept

literature review (Section 2.3.2), the benefit of oversampling based on the data set's structure has been demonstrated experimentally by multiple authors. It improves the quality of the resulting training set [85] and it allows the problem of small disjuncts to be addressed at the same time as the problem of class imbalance generally [59].

Although synthesizing new instances is more computationally expensive than replication, we use SMOTE's generation process to over-sample any sub-concepts with insufficient training instances. This is to help the base classifier learn the sub-concept's whole area in the feature space rather than overly concentrating on the instances that have been seen.

In considering undersampling, which is also used in SMOTE, we note that a constant challenge is ensuring that there are enough instances from each sub-concept. Therefore, although we do not specifically undersample the majority class, the goal of minimizing unneeded instances is achieved by selecting as

Algorithm 3.6 OCCluster - Online Phase

while $\mathcal{DS}$ has more instances **do**
  Get next instance, *inst*
  Determine which cluster in $\mathcal{C}$ is closest to *inst*, c
  AnomalyScore is equal to the anomaly score returned by Classifier$_c$ for *inst*
  if $AnomalyScore > \tau$ **then**
    Label instance as an **OUTLIER**
  else
    Label instance as **NORMAL**
  end if
  if Conditions for one-class training are met **then**
    Train Classifier$_c$ on *inst*
  end if
  Train *ClusterAlgorithm* on *inst*
  if $numInstances \bmod w = 0$ **then**
    Get clustering $\mathcal{C}'$ from *ClusterAlgorithm*
    for all clusters c' in $\mathcal{C}'$ for which $weight(c') > \frac{1.0}{\|\mathcal{C}\|^2}$ **do**
      if $(\arg\min_{c \in \mathcal{C}} CD(c, c') < T)$ **then**
        Assign Classifier$_c$ to c'.
      else
        Train new Classifier over c'.
      end if
    end for
  end if
  $numInstances++$
end while

small a window size as possible while still guaranteeing the minimum number of instances in each sub-concept (Section 3.2).

3.2 An Observation Regarding Window Size

Each of the three frameworks makes use of sliding windows, a technique used by stream learning algorithms to achieve the twin objectives of a) keeping memory requirements bounded and b) weighting recent instances more heavily than older instances. Although window sizes can be fixed or variable [40], the latter requires some sort of signal to decide what the window size should be at a given point in the data stream. This would detract from our goal of ensuring that the three presented frameworks learn as passively as possible from the data stream, so we would like to use fixed size sliding windows.

This decision naturally raises the question of what fixed window size should be chosen. Žliobaitė and Kuncheva addressed a related problem for dynamic windows in small sample size classification. They, however, described a window

resizing algorithm based on labelled instances to find the optimal window size to train a classifier from after concept drift [122]. Instead, we wish to address a related question: how to ensure, to a desired degree of confidence, that there will be enough instances from each sub-concept within our window.

We note that it is possible to set a minimum window size that meets this guarantee and we formalize this approach in Theorem 1).

Theorem 1 (Required Window Size to Ensure a Minimal Number of Instances From Each Sub-Concept)**.** Consider a data stream, $\mathcal{DS}$, with an underlying concept, χ, in line with Definition 4. The concept χ is composed of n sub-concepts, $\chi_{1\ldots n}$, and an instance drawn from $\mathcal{DS}$, $o \in \chi$, belongs to one of these sub-concept according to their underlying probabilities, p_i for $i \in 1 \ldots n$:

$$p_i = P(o \in \chi_i) \tag{3.3}$$

The minimal window size required to ensure, with degree of confidence c, that at least τ instances from each sub-concept are present is:

$$n \text{ such that } np - (x \times \sqrt{np_i(1-p_i)}) = \tau \tag{3.4}$$

where $p_{min} = \min_{i \in 1\ldots n} p_i$, x is chosen such that $\Phi(x) - \Phi(-x) = c$, and n and p_{min} are such that a normal approximation of the binomial distribution can be used.

3.2.1 Proof of the Window Size Theorem

The guarantee in Theorem 1 can be proven using probability theory. We begin with a lemma regarding when it is appropriate to approximate the binomial distribution with the normal distribution and then we provide a proof for Theorem 1.

Lemma 1 (Approximating the Binomial Distribution with the Normal Distribution)**.** A strong rule of thumb for deciding whether the binomial distribution $B(n, p)$ can be approximated by the normal distribution $\mathcal{N}(np, np(1-p))$ is the requirement:

$$n > 9\frac{1-p}{p} \text{ and } n > 9\frac{p}{1-p} \tag{3.5}$$

If we take p to be less than 0.5 (as it is the least probable sub-concept that matters in our case), then the more demanding limit is:

$$n > 9\frac{1-p}{p} \tag{3.6}$$

Proof of Theorem 1. Consider a data stream, $\mathcal{DS}$, with an underlying concept, χ, which is composed of a series of n sub-concepts, $\chi_{1\ldots n}$. We will show that, for any degree of confidence, a lower-bound can be given on the number of instances from any sub-concept.

Note the probability that a given object drawn from the data stream belongs to sub-concept χ_i is p_i. We wish to ensure that in any window of n instances from $\mathcal{DS}$ and any given level of confidence, $c \in [0, 1]$, the number of instances from each sub-concept, $|\chi_i|$, is at least τ.

The number of instances from a given sub-concept in such a window is a random variable following the binomial distribution: $B(n, p_i)$. Since we have proposed windows of size 1000 or greater, then n is large enough to make use of lemma 1 and we use the approximation $B(n, p_i) \sim \mathcal{N}(np_i, np_i(1 - p_i))$. Using the properties of the normal distribution, we then set the lower bound on the number of instances for each sub-concept.

$$\text{For } c = 0.68,\ |\chi_i| \in [np_{min} \pm \sqrt{np_{min}(1 - p_{min})}]. \tag{3.7}$$

$$\text{For } c = 0.95,\ |\chi_i| \in [np_{min} \pm (2 \times \sqrt{np_{min}(1 - p_{min})})]. \tag{3.8}$$

$$\text{For } c = 0.99,\ |\chi_i| \in [np_{min} \pm (3 \times \sqrt{np_{min}(1 - p_{min})})]. \tag{3.9}$$

In general, using the cumulative density function of the normal distribution, Φ, we determine τ for a specific confidence level, c, as follows:

$$\tau = np - (x \times \sqrt{np_i(1 - p_i)}) \text{ such that } \Phi(x) - \Phi(-x) = c \tag{3.10}$$

□

3.2.2 Experimental Validation of the Window Size Theorem

The utility of Theorem 1 is that it accounts for the fact that not only do larger windows mean more instances, it also means that the number of instances for a sub-concept will be closer to that sub-concept's overall proportion in the data stream. To see this, we investigate a sample data stream with a variety of window sizes.

The data stream in question has $11,112$ instances divided between five different sub-concepts. The distribution of these instances for the entire data stream is summarized in Table 3.1.

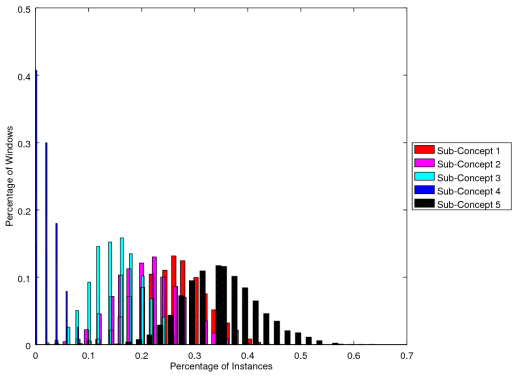
Considering a sliding window of size 50, the number of instances for each sub-concept is an approximately normal binomial distribution (supporting Lemma 1) centred on that sub-concept's overall percentage in the whole data stream.

It can be seen in Figure 3.1 that each sub-concept experiences a range of values for the number of its instances present in a given window. This is most worrisome for the two least likely sub-concepts, three and four. Sub-concept three often makes up less than 10% of the instances in a given window, while sub-concept four is most often absent. Theorem 1 guarantees that by increasing the window size we also reduce the uncertainty of the proportion of instances from each sub-concept in a given window.

Table 3.1: Distribution of instances in a data stream by sub-concept

Sub-Concept	Number of Instances	Percentage of Whole Data Stream
1	2858	25.72%
2	2373	21.36%
3	1711	15.40%
4	231	2.08%
5	3939	35.45%
TOTAL	11112	100%

Figure 3.1: Distributions of percentage of instances from each sub-concept throughout a data stream (window size 50)



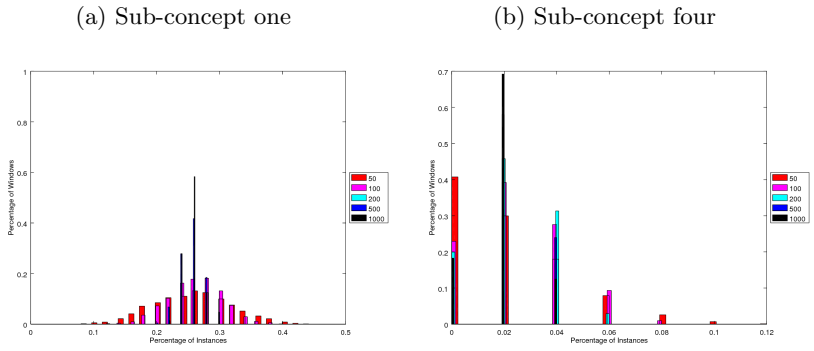
This is seen clearly in Figure 3.2a for sub-concept one, where the number of instances expected in a window becomes more tightly bound around the global value of 25.72%. The effect is even more dramatic for an uncommon sub-concept, such as sub-concept four. Figure 3.2b shows how larger window sizes ensure that significantly fewer windows contain zero instances from this sub-concept.

3.3 Defining a Cluster Distance Function

In the OCCluster framework, it will often be useful to know 'how close are two clusters?' For example, this can help determine whether knowledge learned from an old cluster can be used to bootstrap learning from a new cluster. In order for OCCluster to be as general as possible, it is necessary for the chosen cluster distance function to also be general.

Here we establish our definitions and intuitions regarding a cluster distance function, formally review the literature for existing cluster distance functions,

Figure 3.2: Effect of window size on the distribution of percentage of instances in a given window



develop a distance function between clusters based on Cluster Inclusion Probability, and show that this is a metric.

3.3.1 Definitions

Cluster Description Clustering algorithms such as the k-means family of algorithms produce partitions that attempt to minimize the sum of squared errors between points and the cluster centre to which they are assigned. This results in convex, hyper-sphere shaped clusters that are intuitive and easy to define, needing only a centre point and a radius (Figure 3.3a). Clustering algorithms such as DBSCAN group points together based on the density of their neighbourhoods. Points are put in clusters with all other points that can be reached by following dense areas, while different clusters are separated by sparse areas. This results in arbitrarily-shaped clusters that are harder to define (Figure 3.3b). One approach is to define the cluster as the union of its constituent microclusters or grid cells.

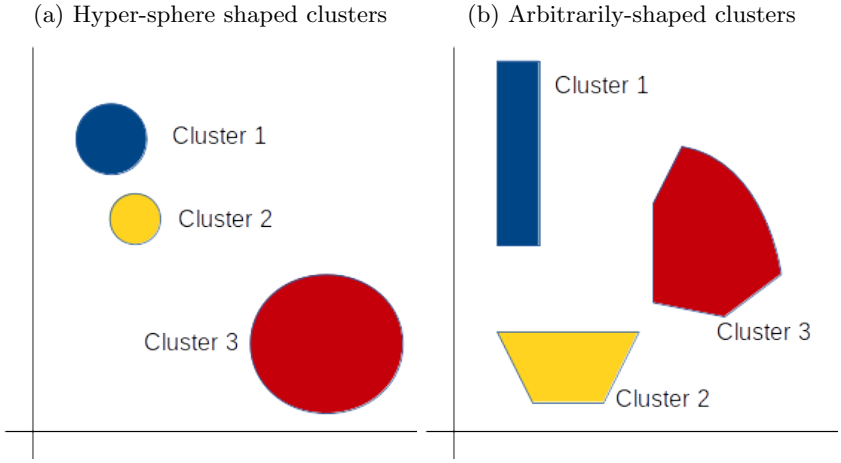
Three types of clusters are identified by Ntoutsi et al. on the basis of how they are defined and how they should be analysed: type A, type B1 and type B2 [86].

Type A "Clusters are discovered within a *dataset-independent* metric space. A cluster is a geometric object, e.g. a sphere like in K- means..." [86]

Type B1 "...A cluster is defined *extensionally* as a set of data records... These algorithms use a metric space to derive a clustering on a dataset, but this space is *data-dependent*, in the sense that the addition of a new record might change the border of a cluster, even if this record does not belong to the cluster." [86]

Type B2 "A cluster is defined *intensionally* as a distribution. For a cluster X

Figure 3.3: Convex and non-convex clusters require different descriptions



of type B2, we denote its cardinality as $card(X)$,its mean as $\mu(X)$ and its standard deviation as $\sigma(X)\ldots$" [86]

Ntoutsi et al. note that each cluster can be given a type B1 definition since all clusters are formed on top of a data set [86]. It is also true, however, that all clusters result in a (hyper-) volume within the feature space. Even if this is not an easily defined geometric object, this volume can be defined using a cluster inclusion probability (CIP) function. Instances within this volume are thought to be specifically linked: e.g. they are instances from the same class or they are generated by the same underlying process.

Cluster Certainty The degree of certainty in cluster membership is another aspect of the clustering problem. Traditionally, cluster algorithms assign instances to a given cluster with certainty. Probabilistic model-based clustering, however, allows instances to be assigned a probability of being assigned to multiple fuzzy clusters [46]. These fuzzy clusters can be modelled using mixture models as the underlying instance-generating process.

Both kinds of clusters are able to return the CIP of an argument instance. In the case of certain clusters, the only clusters used in this thesis, this is simply an indicator function (i.e. the CIP of an instance is 1 if it is in the cluster and 0 if it is not. See equation 3.11).

$$CIP_{\mathcal{C}}(x) = \begin{cases} 1 & \text{if } x \in \mathcal{C} \\ 0 & \text{if } x \notin \mathcal{C} \end{cases} \tag{3.11}$$

Distance Functions A function to tell us how far apart two clusters are will be, mathematically speaking, a distance function. Distance functions are

used to quantify the similarity (or dissimilarity) between two objects and have application in a wide variety of fields. Deza and Deza provide mathematical definitions for both **distance** (Definition 14) and **metric** (Definition 15) in their Encyclopedia of Distances [30, Ch.1].

Definition 14 (Distance)**.** Let X be a set. A function: $d : X \times X \to \mathbb{R}$) is called a **distance** on X if, for all $x, y \in X$, there holds:

1. $d(x, y) \geq 0$ [non-negativity]
2. $d(x, y) = d(y, x)$ [symmetry]
3. $d(x, x) = 0$ [reflexivity]

Definition 15 (Metric)**.** Let X be a set. A function: $d : X \times X \to \mathbb{R}$ is called a **metric** on X if, for all $x, y, z \in X$, there holds:

1. $d(x, y) \geq 0$ [non-negativity]
2. $d(x, y) = 0 \iff x = y$ [identity of indiscernibles]
3. $d(x, y) = d(y, x)$ [symmetry]
4. $d(x, y) \leq d(x, z) + d(z, y)$ [triangle inequality]

3.3.2 Intuitions about Cluster Distance

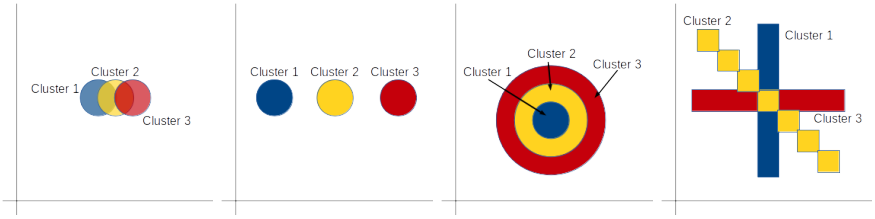
We expect that the more two clusters are overlapped, the closer they will be (Figure 3.4a) and, as with the Euclidean distance, we expect that the distance between non-overlapping clusters will be proportionate to the distance between both their centres and their boundaries (Figure 3.4b). For overlapping clusters (Figure 3.4c) we expect that the distances between two clusters will be inversely proportionate to the extent of the mutual overlapping. Finally, in general - especially for arbitrarily-shaped clusters that do not have neatly defined centres or radii (Figure 3.4d), we expect that the distance between clusters will be proportionate to the amount transformation required to change one cluster to the other.

In all four scenarios illustrated in Figure 3.4, we expect Cluster 1 and 3 to be further apart from each other than either is from Cluster 2. Sample calculations for these scenarios using the Cluster Distance Function defined in this section are given in Appendix B.

3.3.3 Existing Cluster Distance Functions

A variety of algorithms in the literature need to capture "how far apart" clusters or clusterings are in order to make decisions or return assessments. We survey them here and assess whether they are suitable for use with the OCCluster framework.

Figure 3.4: Four clustering scenarios



3.3.3.1 Cluster Separation Measure

Davies and Bouldin presented a cluster separation measure for use in determining whether a clustering contains the appropriate number of clusters [27]. They observed that minimizing the average similarity of each cluster with its most similar cluster would provide a simple way to select k for a clustering. Using S_i as the dispersion of cluster i and M_{ij} as the distance between clusters i and j, the authors defined a general cluster separation measure, (3.12), and showed that it is a metric.

$$R_{ij}(S_i, S_j, M_{ij}) = \frac{S_i + S_j}{M_{ij}} \text{ see: } [1] \tag{3.12}$$

Although it is claimed that this measure is usable with all clustering algorithms, it does make two assumptions: clusters are defined by centroids, and each cluster's constituent points are available for calculations (a type B1 description is available). The first assumption is not met by the arbitrarily-shaped clusters we might encounter and the second assumption is likely to be impractical in a data stream environment. Another difficulty is that the cluster separation measure evaluates at the level of clusterings and it is not clear how individual clusters from two different clusterings could be meaningfully compared.

3.3.3.2 Variation of Information

An information theoretic approach to the question is Meilă's variation of information (VI), which is termed a *criterion for comparing partitions* [84]. For clusterings $\mathcal{C}$ and $\mathcal{C}'$, two different quantities were defined: the entropy of a clustering (3.13), and the mutual information between two clusterings (3.14). For both, $Pr(k) = \frac{n_k}{n}$ is the probability of the outcome of a random variable being in cluster $\mathcal{C}_k$.

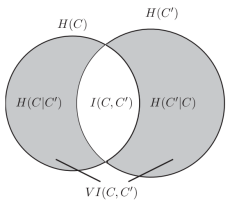
$$H(\mathcal{C}) = -\sum_{k=1}^{K} Pr(k) \log Pr(k) \tag{3.13}$$

$$I(\mathcal{C}, \mathcal{C}') = \sum_{k=1}^{K} \sum_{k'=1}^{K'} Pr(k, k') \log \frac{Pr(k, k')}{Pr(k)Pr(k')} \tag{3.14}$$

Using these concepts, the VI between two clusterings is formulated, (3.15) and Figure 3.5, and shown to be a metric. Meilă argued that this approach was better than criteria based upon either counting pairs or set matching [84]. The main difficulty is that VI, like Davies and Bouldin's cluster separation measure, operates at the level of clusterings and it is not clear how individual clusters representing different points can be meaningfully compared.

$$VI(\mathcal{C}, \mathcal{C}') = H(\mathcal{C}) + H(\mathcal{C}') - 2I(\mathcal{C}, \mathcal{C}') = H(\mathcal{C}|\mathcal{C}') + H(\mathcal{C}'|\mathcal{C}) \tag{3.15}$$

Figure 3.5: Visualization of the variation of information (from Meilă [84])



3.3.3.3 Cluster Overlap

MONIC [103] and MEC [88] are both frameworks for describing how clusters within a clustering evolve over time. Both have defined the concept of cluster movement, which is promising. Both of these concepts of distance, however, require access to the points making up each cluster. As with Davies and Bouldin's cluster separation measure, this imposes an additional memory burden on any framework making use of it, which is not ideal for the data stream environment. Additionally, both make use of MONIC's cluster overlap function, Definition 16, which is non-symmetric [88, 103].

Definition 16 (MONIC's Cluster Overlap Function)**.** Let ζ_i, ζ_j be the clusterings at t_i and t_j for $i < j$. Let $X \in \zeta_i$, $Y \in \zeta_j$ be two clusters. The "overlap of X to Y" is then defined as the normalized sum of the weights of the records in their set intersection:

$$overlap(X, Y) = \frac{\sum_{a \in X \cap Y} age(a, t_j)}{\sum_{x \in X} age(x, t_j)} \tag{3.16}$$

Where $age(x, t_i)$ is a weighting function that accounts for the age of instances x and time t_i.

3.3.3.4 Global Centroid

OLINDDA is an implicit concept drift detection method [105] that was reviewed in Section 2.5.3. As OLINDDA is a cluster-based method, the authors defined a distance function to help determine whether changes in the clustering constitute concept drift or a novelty. Taking $centroid_i$ as the centroid of model i and $centroid_{global}$ as the centroid of the set of all model centroids, this distance function is simply the maximum distance of the centroids of the model to the global centroid (3.17).

$$d_{max} = \arg\max_i dist(centroid_i, centroid_{global}) \tag{3.17}$$

3.3.3.5 Distribution Distances

Finally, there is an established literature for determining the distance between probability distributions, e.g. the Hellinger Distance. Although these are well-defined, and some are provably metrics, they take probability distributions as their arguments and not clusters. Although the two concepts are quite related (especially when considering the type A cluster definition), it would not be proper to treat a cluster as a probability distribution. Nonetheless, it may be profitable to make use of some of these mathematical concepts when formulating a new Cluster Distance Function.

3.3.3.6 Summary

Many authors have touched on problems that are tangential to ours, the question "how far apart are two individual clusters?" As such, many methods have been proposed that seem applicable to our situation. Each method surveyed, however, has some aspect that makes it difficult to apply in its current form.

Some measure the distance between whole clusterings, some measure the distance between probability distributions. Our method should measure the distance between individual clusters. Some of the methods require each cluster's constituent points, ours should require only the cluster's definition. Finally, some methods are not metrics. Ours should be, since the requirements of symmetry, the identity of indiscernibles and the triangle inequality will ensure that the function behaves intuitively and predictably.

We are therefore motivated to propose our own Cluster Distance Function that is usable in the data stream environment, that takes only two clusters as its arguments, and that is provably a metric.

3.3.4 A New Cluster Distance Function

Our cluster distance function should be usable no matter which clustering algorithm is used: no algorithm should produce clusters between which we cannot measure the distance. It should also be possible to measure the distance between different types of clusters. We may, for example, want to compare the

individual clusters produced by different clustering algorithms on the same data set.

For this reason, we will make use of the fact that all cluster descriptions can be expressed as a (hyper-) volume of the feature space that is defined by that cluster's CIP. We can therefore conceive of the distance between two clusters as the amount that these two (hyper-) volumes do not overlap. This converts MONIC's relationship between cluster overlap and cluster distance to a form that can be used with type A clusters.

We will consider only traditional, certain clusters for our cluster distance function and leave the question of measuring distance between fuzzy clusters to future inquiry. Instances will be considered to either belong or not belong to a given cluster, as shown by the indicator function for CIP (3.11).

We will conduct our analysis with numerical dimensions only in the feature space and with the Euclidean distance. Analysis for nominal dimensions will depend on the ordering (if any) of values for that dimension as well as the distance function defined between values. This proposed Cluster Distance Function could be extended to include that nominal dimensions over which a notion of distance and a valid CIP can be defined, however this is left to future work.

3.3.4.1 Mathematical Definition

For a given n-dimensional feature space, $\mathbb{F}$, we define the distance between two clusters, the (hyper-) volumes $\mathcal{C} \in \mathbb{F}$ and $\mathcal{C}' \in \mathbb{F}$, as the amount that these two (hyper-) volumes do not overlap (3.18).

$$CD(\mathcal{C}, \mathcal{C}') = \int_{\mathbb{F}} |IP_{\mathcal{C}}(x) - IP_{\mathcal{C}'}(x)| \tag{3.18}$$

This is inspired by the form of the Hellinger distance, converted from the context of probability distribution to the context of clusters. The Hellinger distance is a metric and it would be useful if we can show that this property holds for the Cluster Distance Function as well.

Proof that the Cluster Distance Function is a metric. To prove that $CD(\cdot, \cdot)$ is a metric, we must show that it satisfies the four conditions of a metric: **non-negativity**, **the identity of indiscernibles**, **symmetry** and **the triangle inequality**.

Non-negativity $|IP_{\mathcal{C}}(x) - IP_{\mathcal{C}'}(x)| \geq 0 \; \forall \; \mathrm{x} \; \in \mathbb{F}$ and the integral of a non-negative function is itself non-negative.

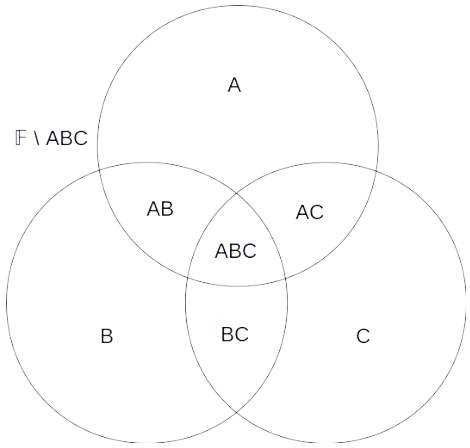
$\therefore CD(\mathcal{C}, \mathcal{C}') \geq 0$ and the condition of **non-negativity** is satisfied.

The identity of indiscernibles $|IP_{\mathcal{C}}(x) - IP_{\mathcal{C}}(x)| = 0 \; \forall \; \mathrm{x} \; \in \mathbb{F}$ and the integral of 0 is itself 0.

$\therefore \mathcal{C} = \mathcal{C}' \implies CD(\mathcal{C}, \mathcal{C}') = 0$
Because $|IP_{\mathcal{C}}(x) - IP_{\mathcal{C}'}(x)| \geq 0 \; \forall \; \mathrm{x} \; \in \mathbb{F}$,
$CD(\mathcal{C}, \mathcal{C}') = 0 \implies |IP_{\mathcal{C}}(x) - IP_{\mathcal{C}'}(x)| = 0 \; \forall \; \mathrm{x} \; \in \mathbb{F} \implies IP_{\mathcal{C}}(x) = IP_{\mathcal{C}'}(x) \; \forall \; \mathrm{x} \; \in \mathbb{F}$. If the CIP for two clusters is the same for all $x \in \mathbb{F}$, that means that they are the same (hyper-) volumes.
$\therefore CD(\mathcal{C}, \mathcal{C}') = 0 \implies \mathcal{C} = \mathcal{C}'$
$\mathcal{C} = \mathcal{C}' \implies CD(\mathcal{C}, \mathcal{C}') = 0$ and $CD(\mathcal{C}, \mathcal{C}') = 0 \implies \mathcal{C} = \mathcal{C}' \therefore CD(\mathcal{C}, \mathcal{C}') = 0 \iff \mathcal{C} = \mathcal{C}'$ and the **identity of indiscernibles** is satisfied.

Symmetry $|IP_{\mathcal{C}}(x) - IP_{\mathcal{C}'}(x)| = |IP_{\mathcal{C}'}(x) - IP_{\mathcal{C}}(x)| \; \forall \; \mathrm{x} \; \in \mathbb{F}$ and so $CD(\mathcal{C}, \mathcal{C}') = CD(\mathcal{C}', \mathcal{C})$.
$\therefore$ the condition of **symmetry** is satisfied.

Figure 3.6: Venn Diagram of clusters $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ in feature space $\mathbb{F}$



The triangle inequality Consider the feature space, $\mathbb{F}$, as the Venn diagram in Figure 3.6. The different areas in the Venn diagram denote the regions of $\mathbb{F}$ that contains instances of the clusters $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$.

$$
\begin{aligned}
CD(\mathcal{A},\mathcal{B}) &= \int_{\mathbb{F}} |IP_{\mathcal{A}}(x) - IP_{\mathcal{B}}(x)| \\
&= \int_{\mathcal{A}} |IP_{\mathcal{A}\setminus\mathcal{B}}(x) - IP_{\mathcal{B}}(x)| + \int_{\mathcal{B}\setminus\mathcal{A}} |IP_{\mathcal{A}}(x) - IP_{\mathcal{B}}(x)| + \\
&\quad \int_{\mathcal{A}\mathcal{B}} |IP_{\mathcal{A}}(x) - IP_{\mathcal{B}}(x)| + \int_{\mathbb{F}\setminus\mathcal{A}\mathcal{B}} |IP_{\mathcal{A}}(x) - IP_{\mathcal{B}}(x)| \\
&= \int_{A\cup AC} 1 + \int_{B\cup BC} 1 + \int_{AB\cup ABC} 0 + \int_{\mathbb{F}\setminus\mathcal{A}\mathcal{B}} 0 \\
&= |A| + |AC| + |B| + |BC|
\end{aligned}
$$

Similarly, $CD(\mathcal{A},\mathcal{C}) = |A| + |AB| + |C| + |BC|$ and $CD(\mathcal{B},\mathcal{C}) = |B| + |AB| + |C| + |AC|$.

$$
\begin{aligned}
CD(\mathcal{A},\mathcal{C}) &\leq CD(\mathcal{A},\mathcal{B}) + CD(\mathcal{B},\mathcal{C}) \\
|A| + |AB| + |C| + |BC| &\leq |A| + |AC| + |B| + |BC| + |B| + \\
&\quad |AB| + |C| + |AC| \\
0 &\leq 2|B| + 2|AC|
\end{aligned}
$$

Since A, B, and C are volumes, they are non-negative. Therefore it is true that $0 \leq 2|B| + 2|AC|$ and the **triangle inequality** is satisfied.

Metric We have shown that $CD(\cdot,\cdot)$ satisfies all four conditions of a metric $\therefore CD(\cdot,\cdot)$ is a metric. □

3.3.5 Summary

We have developed the Cluster Distance Function to calculate the distance between two individual clusters. This Cluster Distance Function is based on the CIP and is usable with all cluster definitions. Our proofs have only considered numeric dimensions and we exclusively use an indicator function for the CIP, limiting the distance function's application to certain clusters.

We note that the formulation presented for the Cluster Distance Function is not able to distinguish between cases where pairs of clusters are completely disjoint. When looking to transfer a learned classifier from one cluster to another, however, the precondition of having some overlap between clusters seems reasonable. Therefore, we accept these limitations when using this Cluster Distance Function for our framework and we leave further development to future work.

3.4 Selecting a Data Stream Clustering Algorithm

With the idea of using clustering to discover the majority class's sub-concept structure, selecting of an appropriate DSCA is of the utmost importance. This is especially true if concept drift is possible within the data stream. The chosen DSCA must produce high quality clusters throughout the data stream, otherwise the summarization provided by its clustering is unlikely to contain any useful information. Indeed, these clusters may hinder the classification process.

Unfortunately, the literature does not contain a comparative study of DSCAs and how they can be expected to behave when concept drift does occur. We therefore conduct our own study as part of this thesis. The experimental design to be used is describe in Section 4.3; the results of the experiment are presented and discussed in Section 5.1.

3.5 The Mixture Model Drift Generator

In order to conduct experiments using real-valued data streams with precise controlled concept drift, we propose a Mixture Model Drift Generator[2] based on Webb et al.'s categorical data generator[3] [113]. Webb et al.'s generator uses a decision tree structure to represent concepts and is limited to data streams with categorical data and with abrupt drift only – that is the magnitude of the concept drift can be varied, but not the duration. Instead, the Mixture Model Drift Generator produces real-valued data streams and allows both the magnitude and duration of the concept drift to be varied. For extended concept drift it also permits the choice of either incremental or gradual concept drift.

The Mixture Model Drift Generator produces its data streams by modelling the periods before and after concept drift - stable concepts - as mixture models with one distribution for each class and a probability vector for choosing between the classes. We use multivariate normal distributions (MVNDs), which are defined by a mean point and a covariance matrix.

3.5.1 Generating the Mixture Models

The generator requires the number of classes present before, n_0, and after, n_1, the concept drift, the stream's dimensionality, a, the drift magnitude, m, the tolerance for the drift magnitude, ϵ, and the drift duration, d, (Alg. 3.7, line 1).

Although any distribution distance functions could be used to measure drift magnitude, we use the Hellinger Distance because it is symmetrical and takes values between 0 and 1, inclusively [113]. Webb et al. also used the Hellinger distance for their categorical data generator, but they used a form appropriate for the categorical nature of their concepts.

[2]available https://doi.org/10.5281/zenodo.1168699
[3]available https://doi.org/10.5281/zenodo.35005

Algorithm 3.7 Mixture Model Drift Generator

1: Input: n_0, n_1, a, m, ϵ
2: Generate M_0: a mixture model of n_0 a-dimensional MVNDs
3: **repeat**
4: Generate M_1: a mixture model of n_1 a-dimensional MVNDs
5: **until** $H(M_0, M_1) = m \pm \epsilon$
6: **return** M_0, M_1

The form of the Hellinger distance between real valued probability density functions, $f(x)$ and $g(x)$, is shown in Equation 3.19. From the last form of Equation 3.19, the Hellinger distance is equal to 0 when the two functions are identical, $f(x) \equiv g(x)$, and equal to 1 when there is no overlap between them, i.e. $(f(x) \neq 0 \implies g(x) = 0) \wedge (g(x) \neq 0 \implies f(x) = 0)$.

$$\begin{aligned} H^2(f(x), g(x)) &= \frac{1}{2} \int \left(\sqrt{f(x)} - \sqrt{g(x)} \right)^2 dx \\ &= 1 - \int \sqrt{f(x)g(x)} dx \end{aligned} \tag{3.19}$$

We are unaware of a method to solve the second mixture model's parameters given the first mixture model (line 1) and m. Instead, mixture models are generated (line 4) and their Hellinger distance from the first mixture model is calculated (line 5) until an appropriate second mixture model is found (line 6).

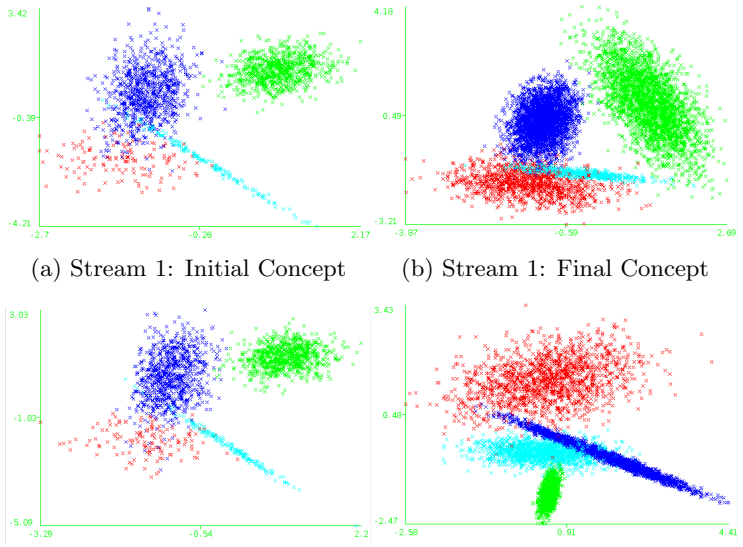
3.5.2 Drawing Instances from the Generator

During a stable concept, sampling the mixture model provides the attributes, x, while the specific MVND that was selected provides the class label, y. Together, these form the data stream object $o \in Dom(\chi)$ as introduced in Sec. 2.5.

Figure 3.7 illustrates data streams produced by the Mixture Model Drift Generator. Subfigures 3.7a and 3.7c depict the initial stable concept for two different data streams; different classes are identified by different colours. These mixture models have identical parameters but the instances drawn from each are different. Subfigures 3.7b and 3.7d depict the final stable concepts for the data streams. The mixture models are separated by a Hellinger distance of 0.4 in the first case and are separated by a Hellinger distance of 0.8 in the second.

During concept drift, generating instances is based on the qualitative type of the drift. For gradual concept drift, the generator draws the instance from one of the stable concepts with the probability of selecting the original concept decreasing over time. For incremental concept drift, the generator draws instances of the same class from both concepts. These instances are weighted, with the weighting of the original concept decreasing over time, and returned as the object o.

Figure 3.7: Initial and final concepts for two data streams. $M_1 = 0.4$ and $M_2 = 0.8$.



(a) Stream 1: Initial Concept (b) Stream 1: Final Concept

(c) Stream 2: Initial Concept (d) Stream 2: Final Concept

3.6 Mixture Model One Class Generator

In order to conduct experiments using imbalanced, real-valued data streams with a sub-concept structure, a Mixture Model One Class Generator was developed based on the Mixture Model Drift Generator that was proposed in Section 3.5. The Mixture Model One Class Generator models the probabilities underlying the majority class's sub-concepts as well as the minority class(es) as a mixture models. In this thesis, MVNDs are used and are defined by a mean point and a covariance matrix.

3.6.1 Generating the Mixture Model

The generator requires the number of sub-concepts in the majority class, n_{maj}, the number of distributions in the minority class, n_{min}, the data stream's dimensionality, a, and the desired imbalance in the data stream, ι.

Although any distribution distance functions could be used at Algorithm 3.8, Step 8, the Hellinger distance was chosen because it is symmetrical and takes values between 0 and 1, inclusively [113]. Specifically, the special form of the Hellinger distance between MVNDs was used (Definition 17).

Algorithm 3.8 The Mixture Model One Class Generator

Generate M: a mixture model of $n_{maj} + n_{min}$ a-dimensional MVNDs
Label the first n_{maj} MVNDs, $P = p_{1...n_{maj}}$, as belonging to their own sub-concepts in the majority class, $C = c_{1...n_{maj}}$
Define the sub-concept assignment function $A(p_i) = c_i$
Label the remaining n_{min} MVNDs as belonging to the minority class, $Q = q_{1...n_{min}}$
Rescale the distribution probabilities of the majority class distributions, $p \in P$, such that $\sum_{p \in P} Pr(p) = \iota$
Rescale the distribution probabilities of the minority class distributions, $q \in Q$, such that $\sum_{q \in Q} Pr(q) = 1 - \iota$
for all $q_i \in Q$ **do**
  $j = \arg\min_j D(q_i, p_j)$
  Define $A(q_i) = c_j$
end for
return Mixture model M with distributions $P \cup Q$ such that $\|P\| = n_{maj}$ and $\|Q\| = n_{min}$; the surjective sub-concept assignment function $A : P \cup Q \rightarrow C$

Definition 17 (Hellinger Distance between two MVNDs)**.** For MVNDs, $N_1 \sim \mathcal{N}(\mu_1, \Sigma_1)$ and $N_2 \sim \mathcal{N}(\mu_2, \Sigma_2)$, the squared Hellinger distance between the two is:

$$H^2(N_1, N_2) := \frac{\det(\Sigma_1)^{\frac{1}{4}} \det(\Sigma_2)^{\frac{1}{4}}}{\det\left(\frac{\Sigma_1+\Sigma_2}{2}\right)^{\frac{1}{2}}} \times \exp\left\{-\frac{1}{8}(\mu_1 - \mu_2)^T \left(\frac{\Sigma_1 + \Sigma_2}{2}\right)^{-1} (\mu_1 - \mu_2)\right\} \tag{3.20}$$

3.6.2 Drawing Instances from the Generator

Sampling the mixture model provides the attributes, x, while the specific MVND that is selected, d, is used to provide the sub-concept label, c, as well as the class label, y. Together, these form the data stream object $o \in Dom(\chi)$ as introduced in Chapter 2 (Table 3.2).

3.7 The Modified Random Radial Basis Function Generator

A modified version of Massive Online Analysis (MOA)'s RandomRBFGenerator, [12], was used to generate imbalanced data streams. This modified Random radial basis function (RBF) Generator models the probabilities underlying the data stream using centroids defined by random RBFs. A centroid, R, is defined

Table 3.2: Drawing a sample m drawn from distribution d in M with the Mixture Model One Class Generator

$x \leftarrow$ sample m from M
$c \leftarrow A(d)$
$y \leftarrow \begin{cases} 0 \text{ if } d \in P \\ 1 \text{ if } d \in Q \end{cases}$
$o \leftarrow \langle x + c, y \rangle$

by a centre point, c, and a radius, r, using an RBF as an indicator function (Definition 18).

Definition 18 (Centroid)**.** A centroid R is defined by normal distributions for each attribute. Each normal distribution has a different mean value but the same standard deviation, $N_i = \mathcal{N}_i(\mu_i, \sigma) \forall i \in [1, a]$. This results in a hyper-sphere shaped centroid and is represented mathematically by the RBF ϕ.

$$\phi(x = (x_1, x_2, \ldots, x_a)) = Pr(N_1 = x_1 \wedge N_2 = x_2 \wedge \cdots \wedge N_a = x_a) \tag{3.21}$$

The centroid R is considered to have a centre point, $c = \mu$, and a radius, $r = 3 \times \sigma$.

3.7.1 Generating the Centroids

The generator requires the number of centroids to generate, n, the data stream's dimensionality, a, and the desired imbalance in the data stream, i.

Any distance function can be used to at Algorithm 3.9, Step 12. In this distance, the Euclidean distance was used to determine the distance between two centroids 19.

Definition 19 (Euclidean distance between two RBF centroids)**.** For centroids $R_1 \sim RBF(c_1, r_1)$ and $R_2 \sim RBF(c_2, r_2)$, the Euclidean distance between the two is:

$$ED(R_1, R_2) = \|c_1 - c_2\| - (r_1 + r_2) \tag{3.22}$$

3.7.2 Drawing Instances from the Generator

First, a centroid, R, is selected using the probability vector V. Then a point drawn from the centroid using its probability density function, $\rho \in R$, provides the attributes, x. The specific centroid selected is used to provide the sub-concept label, c, as well as the class label y. As with the Mixture Model One Class Generator introduced in Section 3.6, x, c, and y are taken together to form the data stream object $o \in Dom(\chi)$ (Table 3.3).

Algorithm 3.9 Modified Random RBF Generator

Generate n centroids, $R_{1...n}$
Generate probability vector of length n, V, to indicate the probability of each centroid being chosen
for all Centroids R_i **do**
  Randomly label R_i as belonging to the majority class, $R_i \in P$, or to the minority class, $R_i \in Q$
end for
Define the list of sub-concepts, $C = c_1, c_2, \ldots, c_{\|P\|}$
Define the sub-concept assignment function $A : P \cup Q \to C$
for all Centroids $p_i \in P$ **do**
  Define $A(p_i) = c_i$
end for
for all Centroids $q_i \in Q$ **do**
  $j = \arg\min_j D(q_i, p_j)$
  Define $A(q_i) = c_j$
end for
Rescale the distribution probabilities in V for the majority class centroids, $p \in P$, such that $\sum_{p \in P} Pr(p) = i$
Rescale the distribution probabilities in V for the minority class centroids, $q \in Q$, such that $\sum_{q \in Q} Pr(q) = 1 - i$
return Centroids $R_{1...n}$ with assigned to the majority class, P or to the minority class, Q; a probability vector V indicating the probability of choosing each centroid; and the surjective sub-concept assignment function $A : P \cup Q \to C$

3.8 Summary

This thesis aims to demonstrate how streaming one-class classifiers can make use of the majority class's sub-concept structure in order to achieve better performance. In this section we used Sharma's three scenarios to characterize the range of knowledge that might be available to a learner. Considered from the most informative to the least, these scenarios are characterized by complete, fuzzy or no knowledge of the sub-concept structure.

Frameworks were developed for each scenario and justification was provided for the belief that these frameworks can make use of their respective levels of knowledge. Each framework was implemented in Java for use with MOA 17.06 [12] and is available online.[4]

Theoretical contributions related to these frameworks are a proof for selecting appropriate window sizes for a stream learning algorithm and the definition a new Cluster Distance Function. Observational data was provided to reinforce the proof regarding window size, showing that the behaviour predicted by Theorem 1 does occur in actual data streams. The novel Cluster Distance Function

[4] available: https://doi.org/10.5281/zenodo.1304278

Table 3.3: Drawing a sample ρ from centroid R with the Modified Random RBF Generator

$x \leftarrow$ point ρ from R
$c \leftarrow A(R)$
$y \leftarrow \begin{cases} 0 & \text{if } R \in P \\ 1 & \text{if } R \in Q \end{cases}$
$o \leftarrow \langle x + c, y \rangle$

is valid for all clusters, no matter their shape or which DSCA produced them. Additionally, we proved that this Cluster Distance Function is a metric.

Finally, we describe three data stream generators for use in the experimental portion of this thesis. The Mixture Model Drift Generator (Section 3.5 extends Webb et al's categorical data stream generator and allows experiments to be conducted using real-valued data streams with quantitatively precise concept drift. It also allows either abrupt or extended concept drift to be present, unlike the categorical data stream generator, which is restricted to abrupt concept drift. The Mixture Model One Class Generator and the Modified Random RBF Generator are modified versions of existing data generators that allow imbalanced data streams to be produced.

Although categorical attributes present their own challenges, many of these are specific to how these attributes are defined (e.g. are they ordered or not? how many values are present for each attribute?). In order to simplify the experimental set up for this thesis and the requirement to control for each of these variables, we consider only real-valued data streams. These data streams are not simple cases, however, and do provide a robust environment in which to demonstrate the described frameworks.

In the next section we describe the experimental designs to test both DSCAs as well as the three frameworks presented in this chapter.

Chapter 4

Experimental Design

4.1 Hypotheses

There are two research questions being addressed experimentally in this thesis; each has its own hypothesis.

The first question, regarding the selection of a DSCA, is "how do the clusterings produced by different DSCAs change, relative to the ground truth, as quantitatively different types of concept drift are encountered?" The hypothesis is that different DSCAs will react better than others when faced with concept drift by continuing to produce high quality clusterings.

The second question, regarding OCC in data streams, is "how can knowledge of the majority class's sub-concept structure be used to improve one-class classifier performance in data streams?" The hypothesis is that guiding a streaming one-class classifier with the majority class's sub-concept structure will result in better classification results than using the streaming one-class classifier alone.

4.2 Software and Hardware Specification

In order to ensure that these experiment are reproducible, we provide the specifications of the hardware and software used for these experiments. All experiments were done on a laptop with 64-bit Linux Ubuntu 16.04 installed, 15.6 GiB of memory and eight 2.60 GHz processors.

All data stream clustering and data stream classification algorithms were implemented in MOA 17.06. MOA is an open source framework with the goal of being a benchmark for data stream mining research; it is implemented in Java and easily extendable [12]. It was selected because it is widely available and widely used and because it brings together the stream generation, classification, clustering and algorithm evaluation portions of data stream mining tasks.

Table 4.1: DSCA characteristics (adapted from Silva et al. [101]). The online component is summarized on top and the offline component is summarized below.

DSCA	Data Structure	Window Model	Outlier Detection
CluStream	feature vector	landmark	statistical-based
ClusTree	feature vector tree	damped	-
D-Stream	grid	damped	density-based
DenStream	feature vector	damped	density-based
StreamKM++	coreset tree	landmark	-

DSCA	Clustering Algorithm	Cluster Shape	Approach
CluStream	k-means	hyper-sphere	Partitioning
ClusTree	k-means	hyper-sphere	Hierarchical
D-Stream	DBSCAN variant	arbitrary	Grid-based
DenStream	DBSCAN variant	arbitrary	Density-based
StreamKM++	k-means++	hyper-sphere	Partitioning

4.3 Selecting a Data Stream Clustering Algorithm

Our question is "how do the clusterings produced by different DSCAs change, relative to the ground truth, as quantitatively different types of concept drift are encountered?" To answer this, we apply different DSCAs to synthetic real-valued data streams with different concept drifts. In order to confirm that these results can be applied outside of laboratory settings, we also apply these DSCAs to a real-world data stream.

4.3.1 Data Stream Clustering Algorithms

Five algorithms are chosen and are characterized in Table 4.1. Each was listed among Silva et al.'s 13 most relevant DSCAs [101] and they cover the four clustering methods identified in Sec. 2.6. In addition, they were assessed to each be representative instances of their respective method while incorporating many different components.

The MOA 17.06 [12] implementation was used for each DSCA. ClusTree was modified to properly implement the offline k-means clustering algorithm,[1] while D-Stream was modified to permit the specification of grid widths for numerical attributes.[2] For both of these algorithms, these were the only modifications that were made.

[1] available https://doi.org/10.5281/zenodo.1216189
[2] available https://doi.org/10.5281/zenodo.1213802

4.3.1.1 Data Stream Clustering Algorithm Parameters

The MOA 17.06 implementation of each algorithm was used, with slightly modified versions of ClusTree and D-Stream used, as mentioned above. The specific parameters chosen for each algorithm are given in Appendix A.

4.3.2 Data Streams

4.3.2.1 Synthetic Data Streams

Three experiments were conducted using synthetic data streams. One hundred two-dimensional data streams were produced for each experimental setting using the Mixture Model Drift Generator (described in Section 3.5). For all data streams, the initial stable concept occurs for $2,000$ instances to allow the DSCAs to achieve a stable clustering. The final stable concept occurs from the end of concept drift until the end of the data stream, again allowing a stable clustering.

The DSCA's clustering quality is the dependent variable for each experiment. Experiment A has four classes before and after concept drift and a drift duration of 1, representing abrupt concept drift. Drift magnitude is the independent variable with values of 0.4, 0.5, 0.6, 0.7, 0.8 or 0.9. Experiment B has four classes before and after concept drift and a drift magnitude of 0.6. Drift duration is the independent variable with values of $1,000$, $5,000$ or $9,000$ instances for both incremental and gradual concept drift. Experiment C has a drift duration of 1, a drift magnitude of 0.6 and four classes before concept drift. The number of post-concept drift classes is the independent variable with either 2, 3, 5 or 6 classes, representing concept evolution.

Since what the "most likely" or "most reasonable" settings are is domain dependent, the range of each experimental setting was chosen in order to provide as wide a range of conditions as possible. The dimensionality of the data streams was chosen to allow for easy visualization and to minimize the effect of the "curse of dimensionality" on the clustering task. Investigating the relationship between dimensionality and clustering quality would be worthwhile but is outside the scope of this experiment.

4.3.2.2 Real World Data Streams

Four data streams were built using the ADFA-LD Anomaly Detection dataset, introduced by Creech et al. to replace the classic KDD 1999 Network Intrusion Detection dataset [26]. Haider et al. described four features they extracted from this dataset and showed a nearest neighbour approach using them for anomaly detection [45], suggesting that sensible clusters exist in this low-dimensional feature space. Our features, based on Haider et al.'s [45], are shown in Table 4.2.

We used the dataset's validation instances as the stream's normal behaviour. Attacks begin sporadically, dominate the data stream starting at instance 2000 and continue for $250-500$ instances before returning to their original frequency. This is abrupt concept drift as the two underlying probabilities are swapped

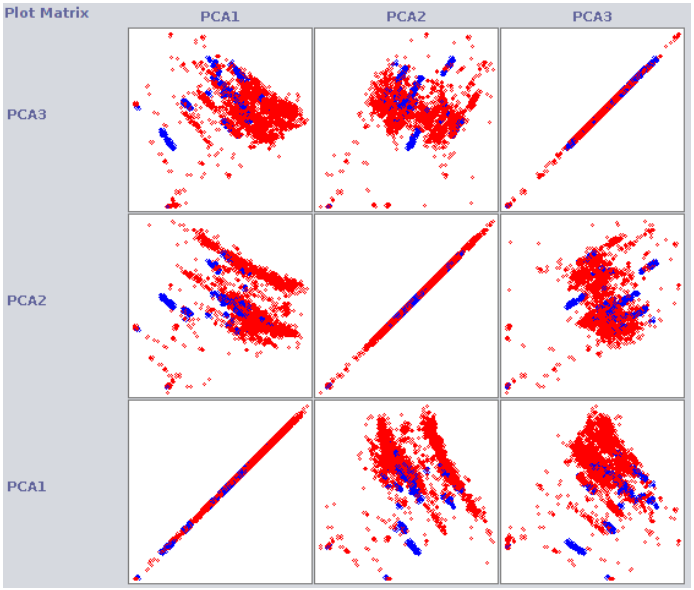
Table 4.2: ADFA-LD Anomaly Detection features (based on Haider et al. [45])

	Description
1	System call that appears with the highest frequency in the trace
2	Lowest valued system call that appears in the trace
3	Highest valued system call that appears in the trace
4	Number of distinct valued system calls that appear in the trace
5	Ratio of the number of appearances of the most repeated system call in the trace to the total number of system calls in the trace
6	Ratio between the range from lowest frequency to highest frequency of appearance in the trace to the total number of system calls in the trace

immediately. Concept evolution may also occur as the clusters that make up the normal and attack behaviours may not be present throughout.

Inspecting the first three principle components of one of the data streams (incorporating the Add User attack, Figure 4.1), we see that there the instances appeared to be grouped into a couple of clusters, confirming that cluster analysis can be attempted.

Figure 4.1: The first three principle components of the ADFA Add User data stream (majority class instances are red, minority class instances are blue)



4.3.3 Evaluation

Since we have access to the ground truth for all data streams, we use an external measure of cluster quality. Many external measures exist in the literature, including purity, Rand statistic, F-measure and Cluster Mapping Measure (CMM). CMM is chosen because it produces a value in the range $[0, 1]$, accounts for different kinds of faults and performed well in Kremer et al.'s experiments on both synthetic and real data streams [66].

CMM builds a fault set for a given clustering: the set of objects that the clustering maps to the wrong class. These faults include missed points, misassigned points and noise points that are assigned to a cluster. CMM produces a value in the range $[0, 1]$ based on each fault point's connectivity to its true and assigned clusters; 1 is the best possible clustering, 0 is the worst [66]. An object's connectivity to a cluster is the ratio of its average k-neighbourhood distance ($knhDist$, Definition 20) to the average k-neighbourhood distance of the cluster. Each object's penalty is weighted by its age through an exponential decay function [66]. Definitions 20-23 are adapted from Kremer et al. [66].

Definition 20 (average k-neighbourhood distance)**.** The average distance of point p to its k neighbours in C is:

$$knhDist(p, C) = \frac{1}{k} \sum_{o \in knh(p,C)} dist(p, o) \tag{4.1}$$

The average distance for a cluster C is:

$$knhDist(C) = \frac{1}{|C|} \sum_{p \in C} knhDist(p, C) \tag{4.2}$$

Definition 21 (Connectivity)**.** Connectivity between object o and cluster C is:

$$con(o, C) = \begin{cases} 1 & \text{if } knhDist(o, C) < knhDist(C) \\ 0 & \text{if } C = \emptyset \\ \frac{knhDist(C)}{knhDist(o,C)} & \text{else} \end{cases} \tag{4.3}$$

Definition 22 (Penalty)**.** $Cl(\cdot)$ returns the ground truth class of the argument object and $map(\cdot)$ returns the ground truth class to which the argument cluster is mapped. The penalty for an object $o \in \mathcal{F}$ assigned to cluster C_i is:

$$pen(o, C_i) = con(o, Cl(o)) \cdot (1 - con(o, map(C_i))) \tag{4.4}$$

Definition 23 (Cluster Mapping Measure)**.** Given an object set $\mathcal{O}^+ = \mathcal{O} \cup Cl_{noise}$, a ground truth $\mathcal{CL}^+ = \mathcal{CL} \cup \{Cl_{noise}\}$, a clustering $\mathcal{C} = \{C_1, ..., C_k, C_\emptyset\}$, and the fault set $\mathcal{F} \subseteq O^+$, the Cluster Mapping Measure between $\mathcal{C}$ and $\mathcal{CL}^+$ is defined using the point weight $w(o)$, overall penalty $pen(o, C)$ and connectivity $con(o, Cl(o))$ as:

$$CMM(\mathcal{C}, \mathcal{CL}) = 1 - \frac{\sum_{o \in \mathcal{F}} w(o) \cdot pen(o, C)}{\sum_{o \in \mathcal{F}} w(o) \cdot con(o, Cl(o))}$$

and if $\mathcal{F} = \emptyset$, then $CMM(\mathcal{C}, \mathcal{CL}) = 1$.

We used the implementation of CMM in MOA 17.06 [12] to evaluate the clustering produced by a DSCA every 100 instances.

4.3.4 Statistical Significance Testing

In our scenario, we wish to compare multiple algorithms across multiple domains without the use of cross-validation. Our null hypothesis for each experiment is that "each data stream clustering algorithm behaves in the same quantitative manner when faced with concept drift."

We use the Friedman test because it is non-parametric, doesn't assume the samples are drawn from a normal distribution and doesn't assume the sample variances are equal. The Friedman test compares the ranks of k algorithms on each of the n data sets and its test statistic can be computed according to equation 4.5. This test statistic can then be compared to the χ^2 distribution to determine significance if n and k are sufficiently large, or this can be done using a look-up table [57, p. 247-249].

$$\chi_F^2 = \left[\frac{12}{n \times k \times (k+1)} \times \sum_{j=1}^{k} (R_{.j})^2 \right] - 3 \times n \times (k+1) \tag{4.5}$$

If the Friedman test leads us to conclude that the difference in algorithm performance is statistically significant, we conduct post-hoc Nemenyi tests to determine which algorithm(s) are significantly different. The Nemenyi test makes use of algorithm ranks as the Friedman test did. It computes a statistic, q, relating to the difference in mean ranks[3] between each pair of algorithms and can be formulated as a "critical distance." For any two algorithms f_{j1} and f_{j2}, the q statistic is computed according to equation 4.6 [57, p. 256].

$$q = \frac{\bar{R}_{.j1} - \bar{R}_{.j2}}{\sqrt{\frac{k(k+1)}{6n}}} \tag{4.6}$$

This testing regime of Friedman's test followed by post-hoc Nemenyi tests as required is recommended by Japkowicz and Shah [57, p. 275-276] as well as Demšar [29]. Statistical testing was performed using the *scmamp* package[4] in R [18].

4.4 One-Class Classification in Data Streams

The goal of this thesis is to show how a one-class classifier can benefit from knowledge of the majority class's sub-concept structure in a data stream environment. It aims to show conditions where this idea can be profitably applied

[3] the mean rank of algorithm f_j across all n data sets is $\bar{R}_{.j} = \frac{1}{n} \sum_{i=1}^{n} R_{ij}$.

[4] available online: https://cran.r-project.org/package=scmamp

and identify challenges in areas where the benefits are not readily apparent. The research question to be answered is "how can knowledge of the majority class's sub-concept structure be used to improve one-class classifier performance in data streams?"

4.4.1 Framework and Classifier Settings

In applying the frameworks to these data streams it is important to ensure that their parameters, described in Sections 3.1.2-3.1.4, are set to values that are reasonable and consistent.

4.4.1.1 Single Classifier

We will use SAs, Streaming HS-Trees, and our streaming adaptation of NN-d as our base one-class classifiers. These represent the three approaches to OCC, which will allow us to assess the generality of our results. Each classifier is able to passively detect concept drift, as highlighted in Section 2.7, which removes the requirement for a separate concept drift detection method. This has the positive implication of simpler frameworks as well as simpler experimental design.

In the case of a single classifier, there are no additional parameters set to deal with sub-concepts. Single classifiers do not consider the sub-concept structure and they never receive concept marked instances for training or for testing. Each of the base one-class classifiers[5] was implemented for use with MOA 17.06 [12].

Streaming Autoencoder The SA's structure is an input layer with one neuron for each non-class attribute, a hidden layer of two neurons and an output layer with one neuron for each non-class attribute. Each neuron uses the logistic function as its activation function (4.7).

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4.7}$$

Since an autoencoder is attempting to reconstruct the input with its output, the squared error between input and output is used as the anomaly score (4.8). The higher an instance's anomaly score is, the more likely that instance is a member of the minority class.

$$AnomalyScore(x) = \frac{1}{2} \times \|x_{input} - x_{output}\|^2 \tag{4.8}$$

The last parameter required for the SA is its learning rate, which controls the size of the adjustments made to weights during backpropagation. In this thesis the learning rate is set to 0.5.

Streaming Half-Space Trees The Streaming HS-Trees algorithm is implemented as described by Tan et al. [108]. The values chosen for the algorithm's parameters are shown in Table 4.3.

[5] available online: https://doi.org/10.5281/zenodo.1287732

Table 4.3: Values chosen for Streaming HS-Trees' parameters

Symbol	Parameter	Value
ψ	Size of Window	500
t	Number of Trees	5
h	Maximum Tree Depth	12
-	Size Limit	0.1

Streaming HS-Trees produce an anomaly score based on the final subspace's mass for each constituent tree. In this case a smaller anomaly score is more indicative of membership in the minority class than a higher anomaly score is. In order to easily compare these scores with those produced by the other classifiers, this score is transformed to the range $[0, 1]$ where higher anomaly scores correspond to a greater likelihood of minority class membership (4.9).

$$AnomalyScore(x) = 1 - \frac{\sum_{T \in HS-Trees} Score_T(x)}{MaximumScore \times t} \tag{4.9}$$

A Streaming HS-Tree's maximum anomaly score can be calculated by considering the scenario where each tree finds all of its mass in a leaf node, maximizing both $Node^*.r$ and $Node^*.k$.

Nearest Neighbour Data Description The NN-d algorithm is implemented as described by Tax [110]. The only parameter to set for this algorithm is the size of neighbourhood to maintain, for this experiment the neighbourhood size is set to 100.

NN-d produces an anomaly score using equation 4.10. The higher an instance's anomaly score is, the more likely that instance is a member of the minority class.

$$f(x) = \frac{\|x - NN^{tr}(x)\|}{\|NN^{tr}(x) - NN^{tr}(NN^{tr}(x))\|} \tag{4.10}$$

4.4.1.2 OCComplete

The main component for the OCComplete framework is the base classifier to be used. This framework has complete knowledge of the sub-concept structure and receives concept marked instances for both training and testing. Parameters above and beyond the single classifier are the initialization window, set to 2000 instances long, and SMOTE's application by sub-concept to ensure that each has at least 1000 training instances. Any other information required by the framework, such as the number of sub-concepts, is discovered through inspection of the data stream.

4.4.1.3 OCFuzzy

The OCFuzzy framework has two key components: the base classifier to be used and the multi-class classifier to use as the concept decider. The multi-class classifier chosen for use in this thesis is the Naïve Bayes classifier found in MOA 17.06, since our testing showed that it was able to distinguish between the sub-concepts reasonably well.

This framework has some knowledge of the sub-concept structure and receives concept marked instances for the initial training period only. Parameters above and beyond the single classifier are the initialization window, set to 2000 instances long, and SMOTE's application by sub-concept to ensure that each has at least 1000 training instances. The number of sub-concepts present in the stream is determined from inspection during initialization; after initialization the framework receives no concept marked instances for training or for testing.

4.4.1.4 OCCluster

The OCCluster framework also has two main components: the base classifier to be used and the DSCA to cluster the data stream. The DSCA chosen for use in this thesis is ClusTree, in line with the results presented in Section 5.1.

This framework knows that the sub-concept structure exists but it receives no concept marked instances for training or for testing. Parameters above and beyond the single classifier are shown in Table 4.4. SMOTE is applied by sub-concept for training the classifiers, not the DSCA, to ensure each has at least 1000 training instances.

Table 4.4: Values chosen for OCCluster's parameters

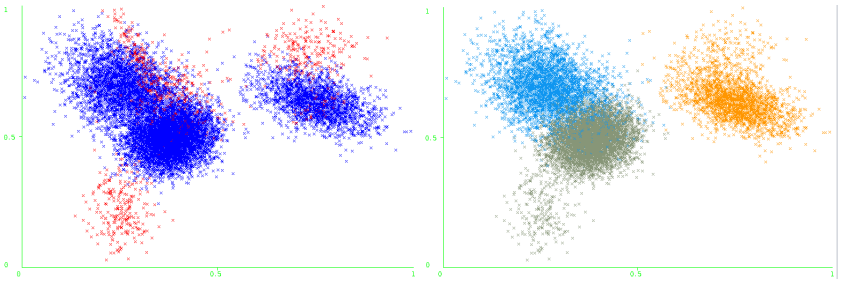
Parameter	Value
Base Classifier	-as required-
Clustering Algorithm	ClusTree
Window Size	2000
Inclusion Threshold for Training	1.0
Cluster Movement Threshold	0.2

4.4.2 Data Streams

In order for the results of this experiment to answer the research question, it is important that the data streams used be both representative and valid. Data streams were selected or synthesized to represent realistic cases of class imbalance with an underlying sub-concept structure. Each instance was therefore marked with both a class (majority/minority) and a sub-concept (e.g. Figure 4.2). An instance's sub-concept label was derived from domain knowledge or from the data stream generator's internal model.

Figure 4.2: Class markup versus sub-concept markup

(a) Class markup (b) Sub-concept markup



4.4.2.1 Synthetic Data Streams

Three families of synthetic data streams were generated using MOA based on both mixture models and random RBFs. These present a range of conditions for both the majority and minority classes; all three incorporate knowledge of a sub-concept structure. For each, the underlying sub-concept structure was assigned according to the data stream generator's internal model. Sub-concept labels were made available to classifiers in line with the framework being tested.

Table 4.5: Summary of the synthetic data streams

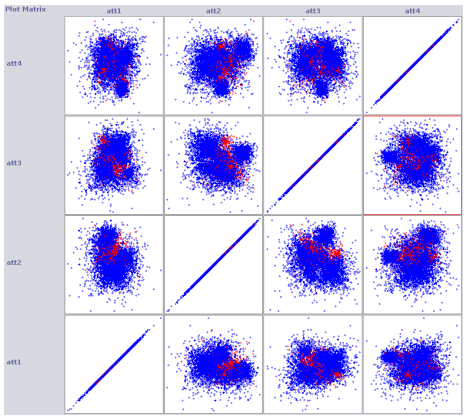
Name	# Atts.	Majority Class	Minority Class
Random RBF	4	Multiple Centroids	Multiple Centroids
Random RBF with Noise	4	Multiple Centroids	Multiple Centroids plus Uniform Noise
Mixture Model	4	Multiple MVNDs	Multiple MVNDs

Both of the Random RBF data streams produce majority class disjuncts that overlap with each other as well as the minority class to a high degree. The Random RBF data stream with noise also contains individual "outliers" of minority class instances, that belong to a less dense area of the feature space. The Mixture Model data stream presents a majority class with a number of disjuncts, some small and some big, and a similarly composed minority class. Although each disjunct is modelled by a MVND, some of them are less obviously hyper-spheres. Each data stream is generated with four attributes as a balance between presenting a realistic scenario and being able to be visualized. The synthetic data streams are summarized in Table 4.5.

Random Radial Basis Function-based Data Streams A four-dimensional data stream was generated using the Modified Random RBF Generator (de-

scribed in Section 3.7), each with thirteen centroids and a 90 : 10 class imbalance. It is visualized in Figure 4.3 with instances coloured by class.

Figure 4.3: A synthetic data stream generated using underlying centroids



Random Radial Basis Function-based Data Streams with Noise The underlying centroids were generated and assigned to sub-concepts as described in section 4.4.2.1. In this case, however, half of the minority class instances were replaced by noise from a uniform distribution over the attribute space (Definition 24).

Definition 24 (Uniform Distribution over the attribute space)**.** Consider n centroids, $R_{1...n}$, in $\mathbb{R}^a$. Define R_{ij} as the jth attribute of the ith centroid. The attribute space of these centroids is defined by the following points:

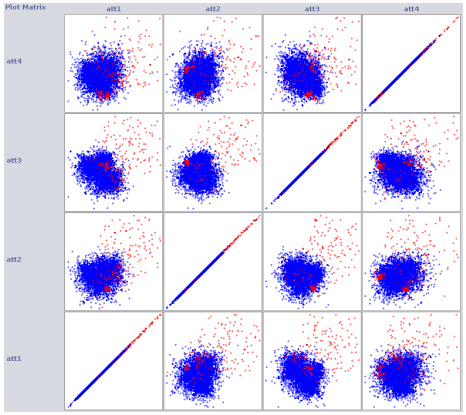
$$\begin{aligned} MinimialPoint =& (x_1, x_2, \ldots, x_a) \text{ such that} \\ & x_i \text{ is the least element of the set } \{R_{1i}, R_{2i}, \ldots, R_{ai}\} \\ MaximalPoint =& (x_1, x_2, \ldots, x_a) \text{ such that} \\ & x_i \text{ is the greatest element of the set } \{R_{1i}, R_{2i}, \ldots, R_{ai}\} \end{aligned}$$

The uniform distribution over the attribute space is then given as $\mathcal{U}(MinimalPoint, MaximalPoint)$.

A data stream was generated using this process, again with thirteen centroids and a 90 : 10 class imbalance. Noise instances were marked as belonging to the closest sub-concept as measured by the Euclidean distance. This data stream is visualized in Figure 4.4 with instances coloured by class.

Mixture Model-based Data Streams A data stream was generated by the Mixture Model One Class Generator (described in Section 3.6). A mixture

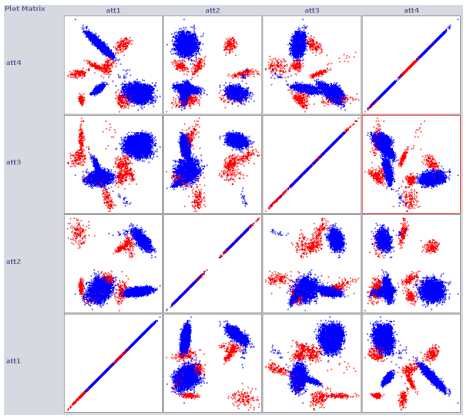
Figure 4.4: A synthetic data stream generated using underlying centroids and noise from a uniform distribution



model of four-dimensional MVNDs was used as the underlying concept. Five MVNDs were designated as belonging to the majority class and each represented a separate sub-concept. Eight MVNDs were designated as belonging to the minority class and were assigned to the sub-concept of the closest majority class.

The data stream had a 90 : 10 class imbalance and is visualized in Figure 4.5 with instances coloured by class.

Figure 4.5: A synthetic data stream generated using an underlying Mixture Model



4.4.2.2 Benchmark Data Streams

Four imbalanced data streams were constructed from benchmark data sets found in the literature. These benchmark data streams present, in some ways, more challenging tasks than the synthetic data streams. The dimensionality of each is higher and there is a high degree of overlap between the majority and minority classes. Where possible, a sub-concept structure was determined from domain knowledge. For the HTRU data stream, there is no explicit sub-concept structure and it must be inferred by the OCCluster framework. These data streams are summarized in Table 4.6.

Table 4.6: Summary of the benchmark data streams

Name	# Atts.	Majority Class	Minority Class
Wine Quality	11	Two sub-concepts	Highly overlapped
CT 2/5 vs 3/4/6	10	Two sub-concepts	Partially overlapped
CT 1/2/5 vs 3/4/6/7	10	Two sub-concepts	Partially overlapped
HTRU2 Survey	8	Possibly two sub-concepts	Little overlap

Wine Quality The Wine Quality data set[6] was introduced by Cortez et al. [25]. This data set's task is to predict the quality of the wine, as determined by the median of at least three expert ratings on a $0-10$ scale, based on physiochemical inputs.

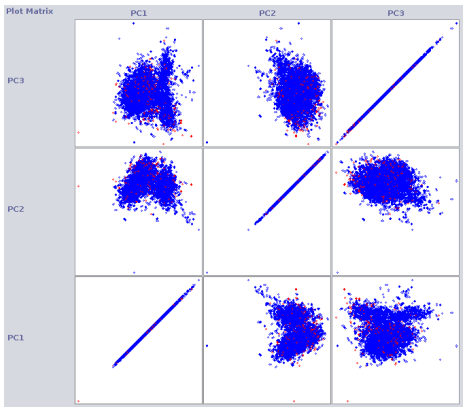
All of the predictive attributes are numeric and were normalized to the range $[0, 1]$. The class attribute is nominal with integer scores between 4 and 8 inclusively; it has been further discretized into low (score < 5), moderate (score $\in [5, 7]$), and high (score > 7) quality wines. The majority of the wines are found in the moderate quality class, which leaves two minority classes.

The Wine Quality data set was converted into a data stream by including the data set twice, given the difficulties in finding benchmark data streams with clearly defined, useful sub-concept structures. The first 2223 instances were from the majority class (moderate quality), ensuring that the frameworks in each of the 10 cross-validation folds received 2000 training instances for initialization.. The class imbalance present in the data stream is $90:10$. This data set has two clearly identified sub-concepts, the red and white variants of the *vinho verde*

[6]available from the UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Wine+Quality

wine from Portugal, and these are visible in the stream's first three principal components (Figure 4.6, instances coloured by class).

Figure 4.6: The first three principal components of the Wine Quality data stream



COVERTYPE The COVERTYPE dataset[7] is a classic benchmark data set for machine learning tasks and was introduced by Blackard and Dean [14]. The data set consists of digital spatial data from the US Geological Survey and the US Forest Service for four areas within the Roosevelt National Forest in northern Colorado. These four areas were selected because they represent forests with minimal human disturbances, ensuring that the forest covertypes are the result of natural rather than man made processes. Excerpted from the UCI Machine Learning Repository's summary:[8]

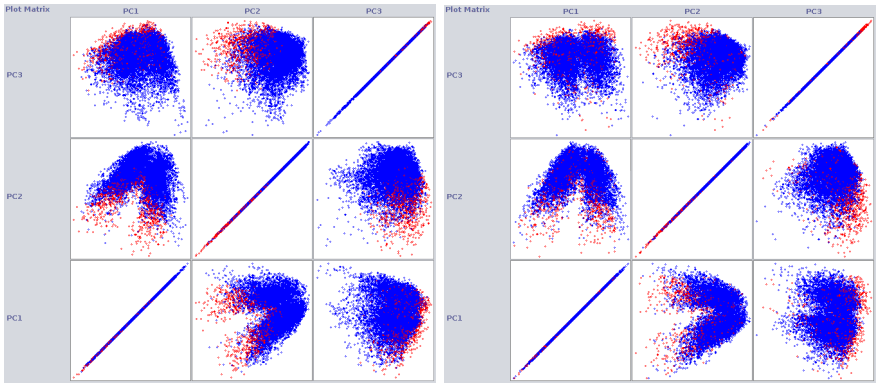
> "Neota (area 2) probably has the highest mean elevational value of the 4 wilderness areas. Rawah (area 1) and Comanche Peak (area 3) would have a lower mean elevational value, while Cache la Poudre (area 4) would have the lowest mean elevational value.
>
> As for primary major tree species in these areas, Neota would have spruce/fir (type 1), while Rawah and Comanche Peak would probably have lodgepole pine (type 2) as their primary species, followed by spruce/fir and aspen (type 5). Cache la Poudre would tend to have Ponderosa pine (type 3), Douglas-fir (type 6), and cottonwood/willow (type 4).
>
> The Rawah and Comanche Peak areas would tend to be more typical of the overall dataset than either the Neota or Cache la Poudre, due to their assortment of tree species and range of predictive variable

[7] retrieved from the MOA website: https://moa.cms.waikato.ac.nz/datasets/
[8] available online: https://archive.ics.uci.edu/ml/datasets/covertype

values (elevation, etc.) Cache la Poudre would probably be more unique than the others, due to its relatively low elevation range and species composition." [93, Data Set Information]

Figure 4.7: The first three principal components of the Covertype data streams

(a) Covertype 2/5 vs 3/4/6 (b) Covertype 1/2/5 vs 3/4/6/7



For both data streams, only numeric attributes were kept and these were normalized to the range $[0, 1]$. The attributes denoting the wilderness area were combined into a single attribute denoting the sub-concept to which the instance belonged. Both data streams begin with 2223 instances[9] from the majority class and have a class imbalance of $90 : 10$.

Covertype 2/5 vs 3/4/6 The first Covertype data stream used those tree species common to Rawah and Comanche Peak as the majority class. These areas are typical of the whole data set and both lodgepole pine (type 2) and aspen (type 5) are very common. The minority class was made up of tree species common to Cache la Poudre, the most unique of the wilderness areas. The data stream's first three principal components are shown in Figure 4.7a with instances coloured by class. Two clear sub-concepts are exhibited when comparing the first and second primary components.

Covertype 1/2/5 vs 3/4/6/7 The second Covertype data stream is an extension of the first. Here the majority class is composed of lodgepole pine, aspen and spruce/fir; these are the three most common tree species in the data set and they represent the Rawah, Comanche Peak and Neota wilderness areas. The minority class is again those tree species common to Cache la Poudre with the last tree species, krummholz, which is uncommon throughout. The data stream's first three principal components are

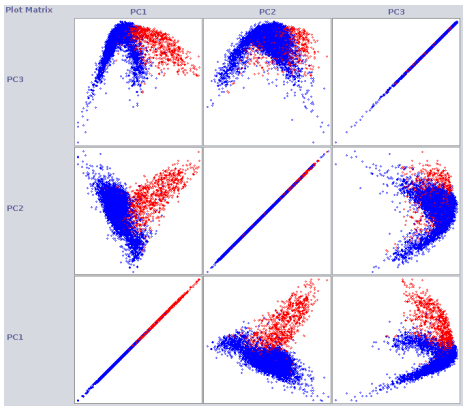
[9] for the same reason as the Wine Quality data stream, to ensure that the framework in each of the 10 cross-validation folds receives 2000 training instances for initialization.

shown in Figure 4.7b with instances coloured by class. There are again two clearly exhibited sub-concepts when comparing the first two primary components.

High Time Resolution Universe Survey The HTRU2 data set[10] was introduced by Lyon et al. [79]. It describes "a sample of pulsar candidates collected during the High Time Resolution Universe Survey (South)" with the task of identifying the true pulsars, which are of scientific interest, from the spurious examples, caused by radio frequency interference and noise.

The eight attributes are all numeric and contain two different descriptions of the received signal; they have been normalized to the range $[0, 1]$. No information is available to define an explicit sub-concept structure, so none is assumed. The data stream's first three principal components are shown in Figure 4.8 with instances coloured by class. It can be seen that the minority and majority classes are very separable and that the majority class can likely be broken into two sub-concepts.

Figure 4.8: The first three principal components of the High Time Resolution Universe Survey data stream



4.4.3 Evaluation

4.4.3.1 Performance Measures

We consider performance measures based on two classic paradigms for evaluating classifier performance: the confusion matrix and the Receiver Operating Characteristic (ROC) curve. Each performance measure is assessed for its ability to convey useful information about a classifier's discriminating ability with a specific emphasis on imbalanced datasets.

[10] available from the UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/HTRU2

Confusion Matrix The confusion matrix is a widely used method of analyzing the performance of a classifier [16, 46, 48, 51, 55, 60, 69, 91, 102]. The general form of the binary classification confusion matrix is shown in Table 4.7. The confusion matrix can be extended to the multi-class classification problem, however we consider only the binary classification problem in this thesis.

Table 4.7: Confusion matrix for a binary classification problem

	Actual Positive	**Actual Negative**
Predicted Positive	True positive (TP)	False positive (FP)
Predicted Negative	False negative (FN)	True negative (TN)

Simple Measures based on the Confusion Matrix Many performance measures for binary classification problems can be derived from the confusion matrix, including: accuracy (4.11); precision (4.12); recall or sensitivity (4.13); and specificity (4.14) [16, 55],[36, Ch. 5].

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{4.11}$$

$$precision = \frac{TP}{TP + FP} \tag{4.12}$$

$$recall/sensitivity = \frac{TP}{TP + FN} \tag{4.13}$$

$$specificity = \frac{TN}{TN + FP} \tag{4.14}$$

Each of these measures suffers from three problems: unsuitability for imbalanced data sets; bias towards a specific kind of example; and threshold sensitivity.

For the first problem, it is possible to imagine a data stream with a $99:01$ class imbalance. In such a case, a classifier that labels all instances as the majority (positive) class would have an accuracy of 0.99 while having no discriminating ability [10, 51]. As well, this classifier would have a precision of 0.99 and a recall of 1.0, both suggesting excellent performance. Although specificity does report a value of 0.0 in this scenario, reflecting the classifier's unhelpful performance, a similar classifier that labels all instances as the minority (negative) class would have a specificity of 1.0.

Measures which combine Simple Confusion Matrix Measures The use of appropriate evaluation metrics is Weiss's first recommendation for addressing class imbalance at the problem-definition level [114]. Performance measures such as accuracy are not appropriate for imbalanced data sets because they fail to take into account biases in the user's levels of interest and in the data set

itself [16]. As identified by Powers, the weaknesses exhibited by these simple measures are due to the fact that they take into account only one class of example (positive or negative) and contain no information about the other [91]. Each does contain information, however, which suggests that they could be combined into a single measure [16]. Combining measures must done with Powers's bias in mind, otherwise it will still be present in the combined measure, e.g. F score (4.15) [48].

$$F_\beta = \frac{(1+\beta)^2 \times recall \times precision}{(\beta^2 \times recall) + precision} \tag{4.15}$$

Combinations that do consider both classes of examples are possible to construct and are robust to class imbalance. The geometric mean (g-mean) (4.16) is independent of class distribution and its non-linearity scales the cost of misclassification by the number of examples in that class that have been misclassified [69]. Informedness (4.17) is based on Powers's Bookmaker algorithm and sets the cost of misclassifications based on the odds that a fair bookmaker would provide [91].

$$g - mean = \sqrt{sensitivity \times specificity} \tag{4.16}$$

$$Informedness = sensitivity + specificity - 1 \tag{4.17}$$

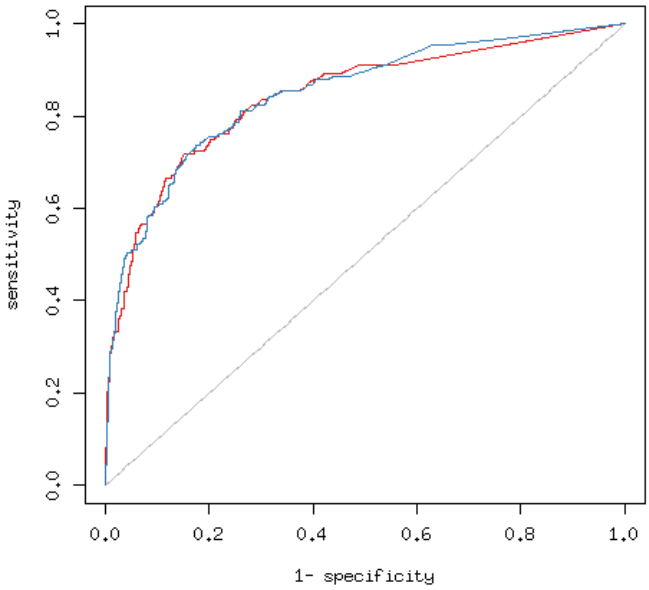
Japkowicz assessed that the g-mean is an appropriate threshold-based measure for assessing classifier performance in imbalanced data sets, noting that it gives gives equal weight to both classes [55]. Considering this as well as the theoretical justifications above, we will use the g-mean as a measure of classifier performance for our experiments.

All of these confusion matrix-based measures, however, are sensitive to the decision threshold used by the classifier. This is a problem as the final labels may have been assigned according to a sub-optimal threshold, masking the classifier's discriminating ability [55]. This is especially possible in data streams, where the appropriate threshold may change throughout the data stream. To address this we will make use of another paradigm for classifier evaluation.

Receiver Operating Characteristic Curve Instead of the confusion matrix, the ROC curve is often used when assessing machine learning algorithms. The ROC curve has the advantage of illustrating an algorithm's discriminating ability for all possible threshold values by graphing the proportion of actual positive instances (true positive rate or *sensitivity*) correctly identified as positive versus the proportion of actual negative instances incorrectly identified as positive (false positive rate or $1 - specificity$) [17, 48]. As an example, see Figure 4.9.

With the ROC curve paradigm, a random classifier with no discriminating ability will have a curve that cuts directly from $(0, 0)$ to $(1, 1)$ as any change in the threshold which allows more actual positive instances to be identified

Figure 4.9: Two example ROC curves



will also incorrectly identify an equal number of actual negative instances (Figure 4.10a) [36, Ch. 5]. Conversely, an ideal classifier will have a curve that moves from $(0, 0)$ to $(0, 1)$ to $(1, 1)$ as it will correctly identify all positive instances before incorrectly identifying any negative instances (Figure 4.10b).

Use in Threshold Selection As the ROC curve displays classifier performance for all possible threshold values, it can also be used to select the optimal threshold value. Sharma identified two ways that this can be done [100]:

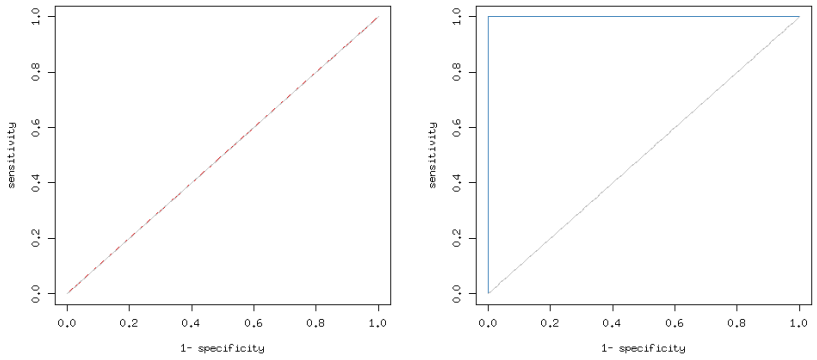
Informedness (Youden's Index) Maximizing Informedness results from maximizing both sensitivity and specificity. This is represented by the point on the ROC curve that is furthest away from the diagonal (i.e. the threshold which produces the largest gap between the classifier's performance and that of a random classifier); and

Zero-One Minimizing the distance to the point $(0, 1)$ in the ROC space is equivalent to picking the threshold which produces the smallest gap between the classifier's performance and that of an ideal classifier.

These two methods are illustrated in Figure 4.11. The trade-off between the two methods is that Informedness leads to a higher number of FP while the Zero-One method leads to a lower number of TP [100].

Figure 4.10: The ROC curves depicting ideal and random classifiers

(a) Random classifier (b) Ideal classifier



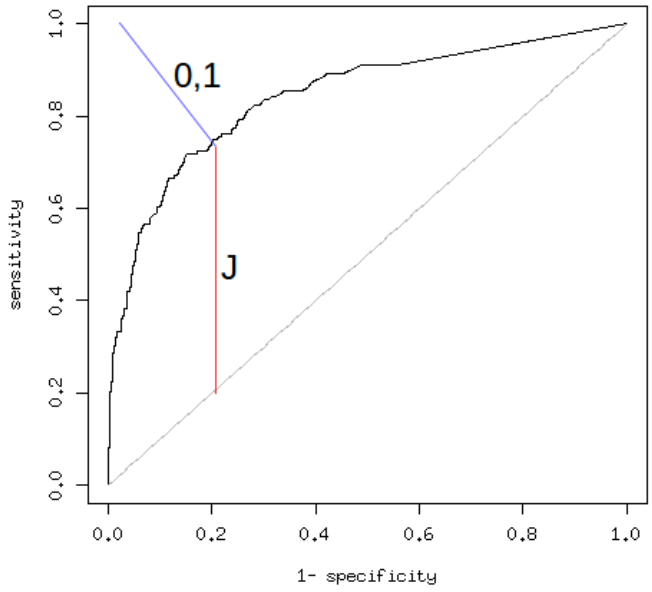
Area Under the Curve Although a ROC curve is useful to visualize a classifier's performance for the full range of possible thresholds, it is difficult to compare these for two (or more) classifiers. One way of summarizing a ROC curve as a single number is to calculate the area under the ROC curve. This measure, known as area under the curve (AUC), falls within the range of $[0, 1]$ where 1 represents the performance of an ideal classifier, 0.5 represents the performance of a random classifier and any value less than 0.5 represents a perverse classifier[11] [48].

Of course, condensing the ROC curve into a single measure does cause information to be lost. For example, a classifier with a higher AUC may perform worse in certain ROC space regions than a classifier with a lower AUC [48]. With that in mind, however, the AUC can be used as one way of comparing different classifiers' discriminating ability throughout a data stream, as done by Chen and He [22].

Prequential Area Under the Curve AUC was originally developed for evaluating classifier performance on static data sets. As is usual with adapting techniques from static data sets to data streams, there are additional complexities that need to be considered. Foremost for AUC is how to calculate the ROC curve. Possible approaches include computing the ROC curve for the whole data stream, computing it for a hold out set(s), or computing it for a series of sliding windows. Computing the ROC curve over the whole data stream is undesirable as it will not adapt to concept drift; similarly the use of a hold out set may leave important events out of the test set [17].

[11] All that needs to be done to rectify the situation of a perverse classifier, C with performance $AUC < 0.5$, is to invert the labelling method, giving C' with performance $AUC' = 1 - AUC$ [48].

Figure 4.11: For threshold selection from a ROC curve, using Informedness to guide threshold selection amounts to maximizing J and the Zero-One method amount to minimizing $0, 1$



Prequential AUC, as described by Brzezinski and Stefanowski [17], makes use of the last approach: a sliding window. Prequential ("test-then-train") evaluation, a common evaluation approach for data streams, provides as large a test set as possible. The sliding window, where the ROC curve considers only the last number of instances allows classifier performance to be tracked accurately throughout the data stream.

The results of their experiments led Brzezinski and Stefanowski to conclude that prequential AUC is "statistically consistent and comparably discriminant with AUC calculated on stationary data" and that it performed well on a range of synthetic and real world data streams exhibiting varying imbalances and concept drifts [17]. We will therefore use Prequential AUC as a performance measure alongside the g-mean. Calculation of the prequential AUC was done using the AUC package[12] in the R statistical software [7].

[12]available online: https://cran.r-project.org/package=AUC

4.4.3.2 Cross-Validation

Ten-fold cross validation was also used for all tasks, meaning that ten parallel models were constructed on the data stream. Although all models were tested on all test instances, each model had one fold of the data stream withheld from its training set throughout the data stream (4.19). For n-fold cross validation of a data stream, $\mathcal{DS}$, the data stream's instances, $x_0, x_1, x_2, \ldots$, are divided into n folds, $F_{1\ldots n}$ (4.18).

$$F_i := \{x_i | \, (i \bmod n = 0)\} \tag{4.18}$$

Using these n folds, n models are trained on $n-1$ folds each. The training set for each model, T_i is constructed as shown in (4.19).

$$T_i := \bigcup_{j \in 1\ldots n, j \neq i} F_j \tag{4.19}$$

For all tasks, each model was given 2000 initial training instances to initialize itself.

4.4.4 Statistical Significance Testing

Once the experiments have been run and the performance measures have been calculated, the last step is to assess whether the differences in performance observed are due to the inherent superiority of one approach compared to another or if they are simply due to random chance. This is normally done using null hypothesis significance testing (NHST), but recently there have been a number of authors recommending that Bayesian analysis be used instead [11, 24, 68].

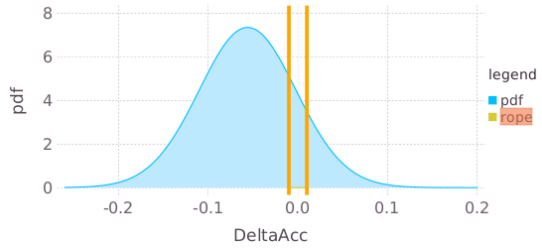
Benavoli et al. recommend using Bayesian analysis for statistical significance testing over the frequentist NHST for a variety of reasons [11]. As the authors note, NHST makes incorrect assumptions: first, that the p-value contains sufficient information about how probable the null hypothesis is; and second, that practical significance follows on from statistical significance. What's more, Bayesian analysis more naturally answers the central question of interest for significance testing in machine learning: "is method A better than method B?" [11].

4.4.4.1 The Region of Practical Equivalence

One observation made by Benavoli et al. is that methods should not require a difference in performance of exactly 0 to be considered equivalent; instead they adopt Kruschke's concept of a region of practical equivalence (ROPE), which introduces the idea that values can be close enough to the null value to be equivalent for practical purposes [68]. Mathematically a ROPE is an interval $[-r, r]$ centred on zero, as shown in Figure 4.12. For classifiers, the interval $[-0.01, 0.01]$ is likely appropriate, though this may vary by domain [11, 24].

The ROPE is used to define regions in the pdf for the difference in performance between two classifiers. The region of the pdf that is above the ROPE

Figure 4.12: An illustration of the ROPE for the difference in accuracy between two classifiers (from Benavoli et al. [11])



indicates how likely it is that method A is better than method B, the region of the pdf below the ROPE indicates how likely it is that the converse is true. The region of the pdf within the ROPE indicates how likely it is that method A and method B are *practically equivalent* [11].

4.4.4.2 Correlated Bayesian t-test

The correlated Bayesian t-test (CBTT) analyses cross-validation results for a single data set, accounting for the correlation between the data sets and based on the generative model in (4.20), where x is a vector of differences in performance between the two classifiers, $1_{n\times 1}$ is a vector of ones, μ is the mean difference in performance, and v is noise drawn from a MVND, $MVN(0, \Sigma_{n\times n})$. The covariance matrix $\Sigma_{n\times n}$ is defined according to (4.21).

$$x_{n\times 1} = 1_{n\times 1}\mu + V_{n\times 1} \tag{4.20}$$

$$\Sigma_{n\times n} = \begin{bmatrix} \sigma^2 & \sigma^2\rho & \sigma^2\rho & \cdots & \sigma^2\rho \\ \sigma^2\rho & \sigma^2 & \sigma^2\rho & \cdots & \sigma^2\rho \\ \sigma^2\rho & \sigma^2\rho & \sigma^2 & \cdots & \sigma^2\rho \\ \vdots & \vdots & & \ddots & \vdots \\ \sigma^2\rho & \sigma^2\rho & \sigma^2\rho & \cdots & \sigma^2 \end{bmatrix} \tag{4.21}$$

In order to estimate μ, the parameter of interest, Benavoli et al. derived the posterior distribution of μ as a Student distribution (Equation 4.22) This distribution can be plotted along with the ROPE and analysed to determine how likely it is the one classifier is superior to the other or how likely it is that they are practically equivalent [11].

$$Pr(\mu|x, \mu_0, k_0, a, b) = St\left(\mu; n-1, \bar{x}, \left(\frac{1}{n} + \frac{\rho}{1-\rho}\right)\hat{\sigma}^2\right) \text{ see } [13] \tag{4.22}$$

Statistical testing was done using Benavoli et al.'s implementation of the CBTT [11]. This code implementation is available in both the R programming language as well as in Python at:

https://github.com/BayesianTestsML/tutorial/

4.4.4.3 Visualizing the Results of the Correlated Bayesian t-test

We will use the CBTT to assess the statistical significance of the difference in performance between each of the frameworks and the associated single classifier for each of the data stream's evaluation windows. Although figures such as Figure 4.12 are handy for visualizing the results of a CBTT, it is clear that this is not practical when visualizing the results throughout an entire data stream.

Instead, we will plot the evolution of the three values, $Pr(\mu > rope)$, $Pr(\mu \in [-rope, rope])$, and $Pr(\mu < -rope)$, for each evaluation window. This is inspired by the work of Gama et al. concerning the evaluation of stream learning algorithms; in their paper the authors graph the evolution of statistical tests such as the ratio of prequential accumulated loss, the Signed McNemar's Test, and the Page-Hinkley Statistic [39]. This method provides an intuitive way to understand the evolution of a statistical test over time and allows conclusions to be drawn about specific periods within the data stream, e.g. the beginning of the stream or around an instance of concept drift.

4.4.5 Procedure

Complete knowledge of the sub-concept structure is available for all three synthetic data streams. Experiments are therefore conducted with a single classifier as well as all three frameworks. Since our question of interest concerns how knowledge of the sub-concept structure can be used to improve one-class classifier performance, we restrict our comparisons to approaches using the same base classifier. The results are presented in Figures 5.7-5.9; AUC scores and g-mean for the same data stream are reported side-by-side.

Of the benchmark data streams, Wine Quality and the two Covertype streams had an available sub-concept structure. Results from these data stream are provided for a single classifier as well as all three frameworks (Figures 5.10-5.12).

For the High Time Resolution Universe survey data stream, although likely sub-concepts could be seen during the data exploration phase, no explicitly defined sub-concept structure was available. As such, results from this data stream are provided for only a single classifier and the OCCluster framework (Figure 5.13).

The performance of each framework will be evaluated using the g-mean and AUC as measures of performance. The Bayesian hierarchical correlated t-test will be used to infer the significance of differences in classifier performance or whether their performance is practically equivalent.

The g-mean for all data streams is calculated by selecting the average optimal threshold for all of the evaluation windows as determined by Informedness. A single threshold value was calculated in this way, which was deemed to be more

[13] $\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$, $\hat{\sigma}^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$, and ($\mu_0 = 0$, $k_0 \to \infty$, $a = -\frac{1}{2}$, $b = 0$) are the parameters of the prior.

realistic than calculating the (potentially different) optimal threshold value for each evaluation windows.

4.5 Summary

The experimental design has been conducted with the goal of answering two research questions: "how do the clusterings produced by different DSCAs change, relative to the ground truth, as quantitatively different types of concept drift are encountered?" and "how can knowledge of the majority class's sub-concept structure be used to improve one-class classifier performance in data streams?"

The first question will be answered by generating real-valued synthethic data streams with precise quantitative concept drift and applying five different DSCAs to each. These five DSCAs have been selected as being representative of the range of approaches used in the literature. The clusterings produced by each DSCA are evaluated compared to the ground truth using the CMM.

To answer the second question we have described a range of synthetic and real world data sets that exhibit a variety of aspects of the class imbalance problem. Three streaming one-class classifiers have been selected, each representing a different approach to the OCC problem. These streaming one-class classifiers are applied singly and as part of the the three frameworks to each of the data streams with their performance measure using prequential AUC and the g-mean.

The results of both experiments are presented in the next section. These results are discussed on an experiment-by-experiment basis as well as holistically with the aim of answering the respective research questions.

Chapter 5

Experiment Results and Discussion

5.1 Selecting a Data Stream Clustering Algorithm

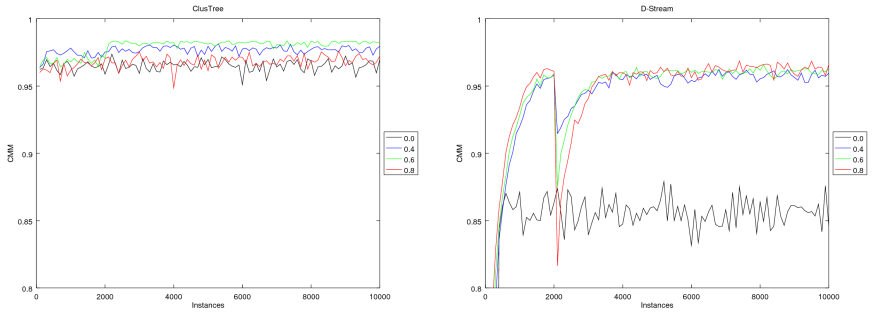
The results show the average CMM across the 100 data streams for each setting. For algorithms that require k, the number of clusters, that parameter was set to 4. Representative results for ClusTree and D-Stream are shown in this chapter; additional results are shown in Appendix C.1. Post-hoc Nemenyi test results are shown graphically; algorithms that are not significantly different (at $p = 0.05$) are linked.

5.1.1 Experiment A - Abrupt Concept Drift

These data streams exhibited abrupt concept drift: one stable concept immediately replaced by another. Each algorithm's maximum change in cluster quality for a given setting was calculated using the 1500 instances after concept drift. This change was compared to the algorithm's change in quality for the baseline data streams, controlling for changes in cluster quality not due to concept drift, e.g. StreamKM++'s characteristic decrease in quality. For ease of interpretation, only magnitudes 0.4, 0.6 and 0.8 are shown along with the baseline (0.0).

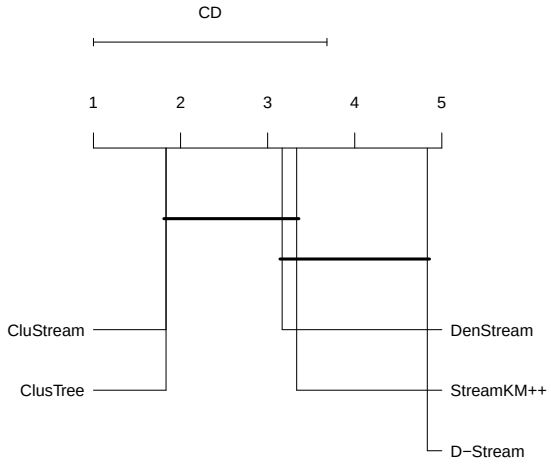
The algorithms' results, shown in Figure 5.1, divide into two qualitative groups. CluStream and ClusTree are largely invariant to abrupt concept drift for all magnitudes. The other three algorithms' cluster quality changes due to concept drift, with DenStream and StreamKM++ sensitive when the magnitude of the concept drift is larger. All three algorithms' results behave the same, however: the abrupt concept drift is met with a decrease in cluster quality, an extreme cluster quality is reached and then a new stable cluster quality is established for the remainder of the data stream. Larger decreases in cluster

Figure 5.1: Experiment A - Data streams with abrupt concept drift



(a) (b)

Figure 5.2: Experiment A - Nemenyi test results



quality are associated with larger magnitude concept drifts for all three algorithms. Also noteworthy is that the magnitude of the concept drift does not affect the stable cluster quality for the second concept.

Using Friedman's test we conclude that among the five algorithms there is a significant difference in the change of cluster quality due to concept drift ($p < 0.01$). Post-hoc Nemenyi test results are shown in Figure 5.2 where the highest ranked algorithm had the highest (most positive) change in cluster quality and the lowest ranked algorithm had the lowest (most negative) change.

5.1.2 Experiment B - Extended Concept Drift

These data streams exhibited either gradual concept drift, where the data stream alternated between two stable concepts before settling on the second,

or incremental concept drift, where the data stream moved through a series of intermediate concepts before reaching the second stable concept. Each algorithm's time to reach and time to recover from its extreme CMM value was determined using average performance across all data streams. Results are shown in Fig. 5.3.

Qualitatively, the algorithms' results are divided into two groups. CluStream and StreamKM++ continue to be invariant for all durations and for both types. ClusTree, DenStream and D-Stream show changes in cluster quality that are affected by the concept drift's duration and type. DenStream and D-Stream exhibit the same general behaviour from Experiment A: concept drift resulting in a decrease in cluster quality, an extreme cluster quality is reached and then a new stable cluster quality is established for the remainder of the data stream. Longer concept drift durations soften this effect, as seen when comparing the $1,000$ duration drift with the $9,000$ duration drift for both DenStream and D-Stream; this effect is also dampened when facing incremental concept drift. In contrast, ClusTree's small changes in cluster quality take longer to reach the new stable cluster quality during longer concept drifts and for incremental compared to gradual drift.

Using Friedman's test we conclude that there is a significant difference among the five algorithms in the time to reach an extreme value due to concept drift ($p < 0.01$) and in the time to recover from that extreme value to a new stable quality ($p < 0.01$); post-hoc Nemenyi test results are shown in Fig. 5.4.

5.1.3 Experiment C - Concept Evolution

These data streams exhibited concept evolution. That is, concept drift caused the number of classes present in the data stream to either increase or decrease. The change in cluster quality was measured the same way as for Experiment A. Results are shown in Figure **??**.

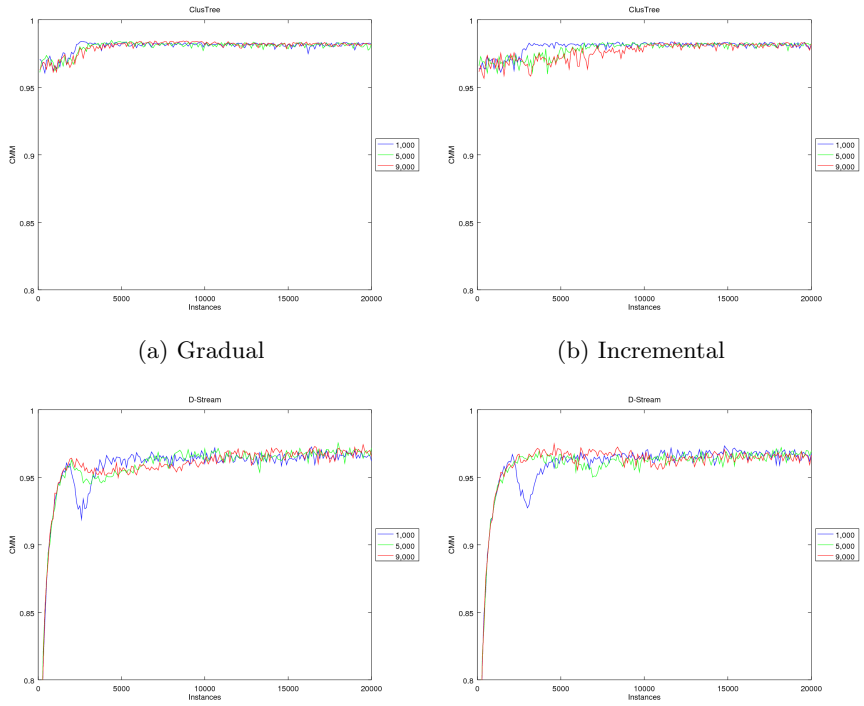
CluStream and StreamKM++, the algorithms with a k parameter, suffered from increasing the number of classes, though neither suffered from decreasing the number of classes, i.e. representing two classes by splitting them across four clusters still resulted in better clusters than attempting to merge six classes into four clusters. DenStream and D-Stream, both algorithms without a k parameter, were also affected by concept evolution.

Compared to Experiment A, the change in cluster quality was decreased when the number of classes present decreased ($\mu_{\Delta\text{CMM}} = -0.01169$) and increased when the number of classes present increased ($\mu_{\Delta\text{CMM}} = -0.05016$). Using Welch's unequal variances t-test, the difference in cluster quality changes between decreasing the number of classes and increasing the number of classes was found to be statistically significant ($p = 0.04278$).

5.1.4 Real World Data Streams

Algorithms that required the number of clusters had the parameter k set to 2 by inspection. Cluster quality was generally similar to the quality obtained for

Figure 5.3: Experiment B - Data streams with extended concept drift



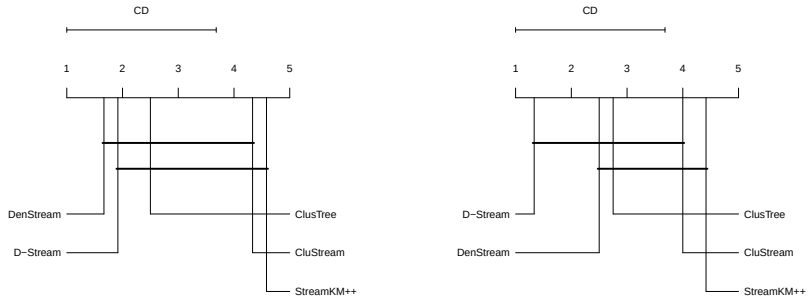
(a) Gradual

(b) Incremental

(c) Gradual

(d) Incremental

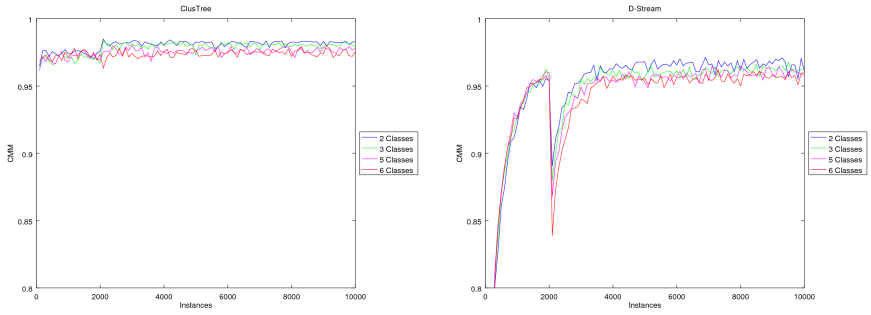
Figure 5.4: Experiment B - Nemenyi test results.



(a) Time to reach extreme cluster quality.

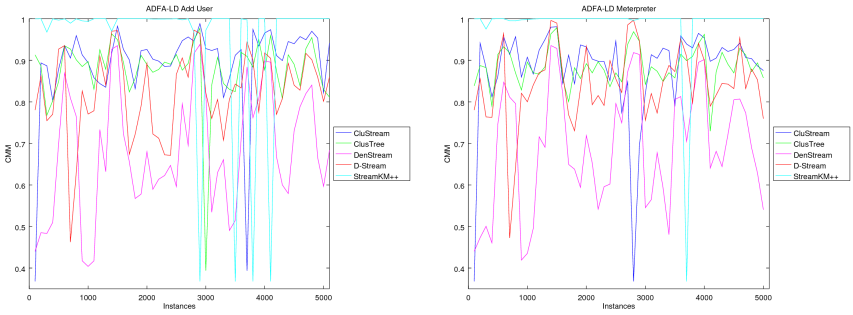
(b) Time to recover.

Figure 5.5: Experiment C - Data streams exhibiting concept evolution



(a) (b)

Figure 5.6: ADFA-LD Intrusion Detection data streams



(a) (b)

the synthetic data streams, with the exception of DenStream. Though more volatile, each algorithm's overall performance is similar to its performance on synthetic data streams with abrupt drift, in line with how we described these data streams in Section 4.3.2.2. ClusTree, CluStream and StreamKM++ are all stable in the face of the first concept drift, although each becomes more volatile after the return to normal behaviour. D-Stream exhibits some volatility throughout and produces, on average, slightly lower quality clusters. DenStream produces consistently lower quality clusters than the other algorithms, though its volatility makes further interpretation difficult.

We note that DSCA performance is largely the same for all types of attack. This is a promising characteristic as an Intrusion Detection System must be robust against a wide range of known and unknown attacks.

5.1.5 Summary

As mentioned previously, we are motivated to use clustering to uncover a majority class's sub-concept structure for the OCC task in data streams. Clustering in data streams faces challenges, however, which have not been investigated in the literature. In order to ensure that a reliable DSCA is selected, we asked 'how do the clusterings produced by different DSCAs change, relative to the ground truth, as quantitatively different types of concept drift are encountered?" This chapter proposed a method for generating real-valued data streams with precise quantitative concept drift and used these to conduct an experimental study to answer this question.

We observe that concept drifts with larger magnitudes and shorter durations were the most difficult for DSCAs, with each resulting in a larger decrease in cluster quality. We also observe that the key factor in determining how a DSCA will respond to concept drift is the algorithm it uses in the offline component. DSCAs that incorporate partitioning-based clustering methods are more robust to concept drift while those that incorporate density-based clustering methods are more sensitive. We also observe that an increase in the number of classes present in a data stream is more challenging for DSCAs than a decrease and that this was true even for DSCAs that did not require an explicit k parameter.

Applying these DSCAs to a real world data stream in the domain of intrusion detection, we observed similar behaviour as in our experiments with synthetic real-valued data streams. This provides evidence that these findings can be taken from laboratory settings and used to predict behaviour in real world domains. Based on these results, we decide to incorporate ClusTree as the DSCA chosen for use with OCCluster. ClusTree consistently produces high quality clusters, does not require k, and is generally invariant to concept drift.

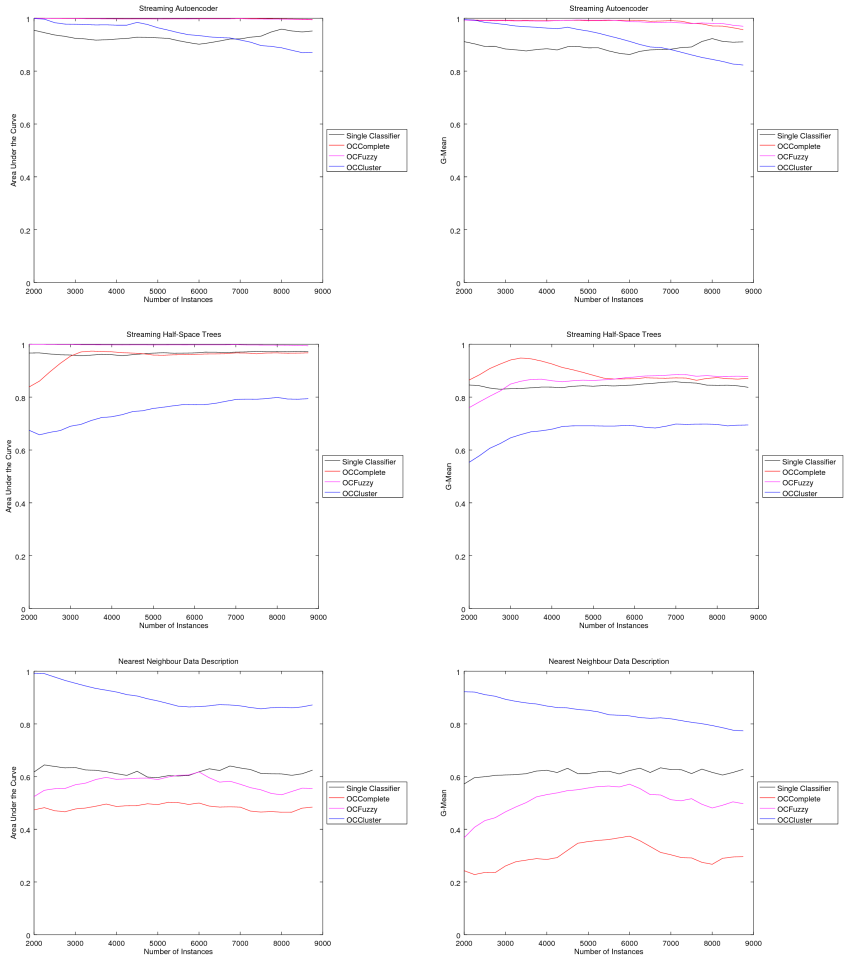
5.2 One-Class Classification in Data Streams

In considering the experiment results, we recall that our research question is "how can knowledge of the majority class's sub-concept structure be used to improve one-class classifier performance in data streams?"

We first test the frameworks on the synthetic data streams that were generated as described in Section 4.4.2.1. From these experiments we seek to validate our belief that using knowledge of the majority class's sub-concept structure will improve classifier performance. We also seek to characterize the performance of each framework.

Following this, we continue by testing the frameworks on the benchmark data streams that were described in Section 4.4.2.2 to determine whether our observations can be transferred from laboratory conditions to real-world problems of interest.

Figure 5.7: Results for the Mixture Model data stream



5.2.1 Synthetic Data Streams

Mixture Model Data Stream For the Mixture Model data stream (Figure 5.7) although the single SA performs well, it is clear that using the sub-concept structure benefits the classifier. Both the OCComplete and OCFuzzy frameworks dominate the single classifier in terms of AUC throughout the data stream; the difference is such that the CBTT assesses the probability of the respective frameworks being superior at 1.[1] The OCCluster framework begins with a higher AUC and is assessed by the CBTT as being superior to the single classifier. This performance degrades over the data stream, however, and the stream ends with the single classifier being rated superior by the CBTT and OCCluster recording the lowest AUC of the four approaches. This decrease in performance is related to a decrease in sensitivity,[2] which only OCCluster experiences. This means that OCCluster loses the ability to identify minority classes over time.

For the Streaming HS-Trees, the single classifier performs very well. In terms of the frameworks, both OCComplete and OCFuzzy perform as well or better than the single classifier, but the OCCluster framework struggles to find useful representations for learning. When assessed for significance by the CBTT, each framework comes out differently: OCFuzzy is assessed as likely superior; OCCluster is assessed as being worse; and OCComplete is hard to distinguish from the single classifier, though the latter is slightly more likely to be superior at the end of the data stream. When considering the optimal threshold, however, both OCComplete and OCFuzzy do provide higher g-mean scores than the single classifier.

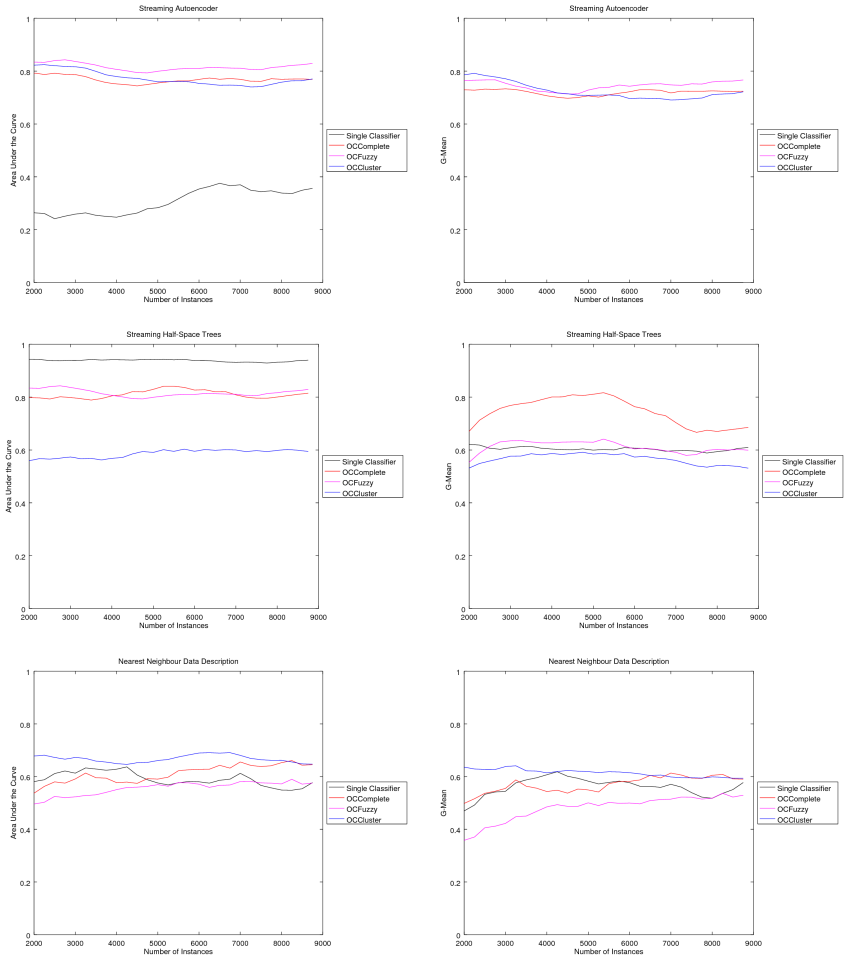
Finally, the NN-d performs poorly as a single classifier and as part of both the OCComplete and OCFuzzy frameworks. The CBTT assess the single classifier as superior to each, though. Interestingly, much higher AUCs are seen with the OCCluster framework – the NN-d is the only base classifier for which OCCluster produced the best results. This results in performance superior to the single classifier throughout the data stream. Although OCCluster experiences a decrease in sensitivity similar to its performance with the SA, it provides far an away the highest specificity of the four approaches throughout the data stream.

Random RBF Data Stream For the first Random RBF data stream, without noise - Figure 5.8, also shows a clear advantage for the SA in using the sub-concept structure. As was seen in Figure 4.3, the minority class centroids are essentially surrounded by the majority class centroids. This leads to the single SA essentially learning to recognize the minority class and performing worse than even a random guessing classifier would. The three frameworks all perform well throughout the data stream. All three frameworks are assessed by the CBTT as being superior to the single classifier with probability 1 throughout

[1] Graphics showing the results of the CBTTs used in determining the statistical significance of differences in AUC scores for the synthetic data streams are shown in Appendix D.1.

[2] The sensitivity and specificity values calculated for the evaluation windows are shown in Appendix C.2.

Figure 5.8: Results for the Random RBF data stream



the data stream.

The Streaming HS-Trees performs very well again as a single classifier and has significantly lower AUC scores with all three frameworks. As with the Mixture Model data stream, the OCCluster framework results in the Streaming HS-Trees' worst performance. The single classifier is assessed by the CBTT as being superior to each framework with probability greater than 0.9 throughout the data stream. OCComplete's best threshold does provide a higher g-mean than the single classifier's best threshold, though, for the entirety of the data stream.

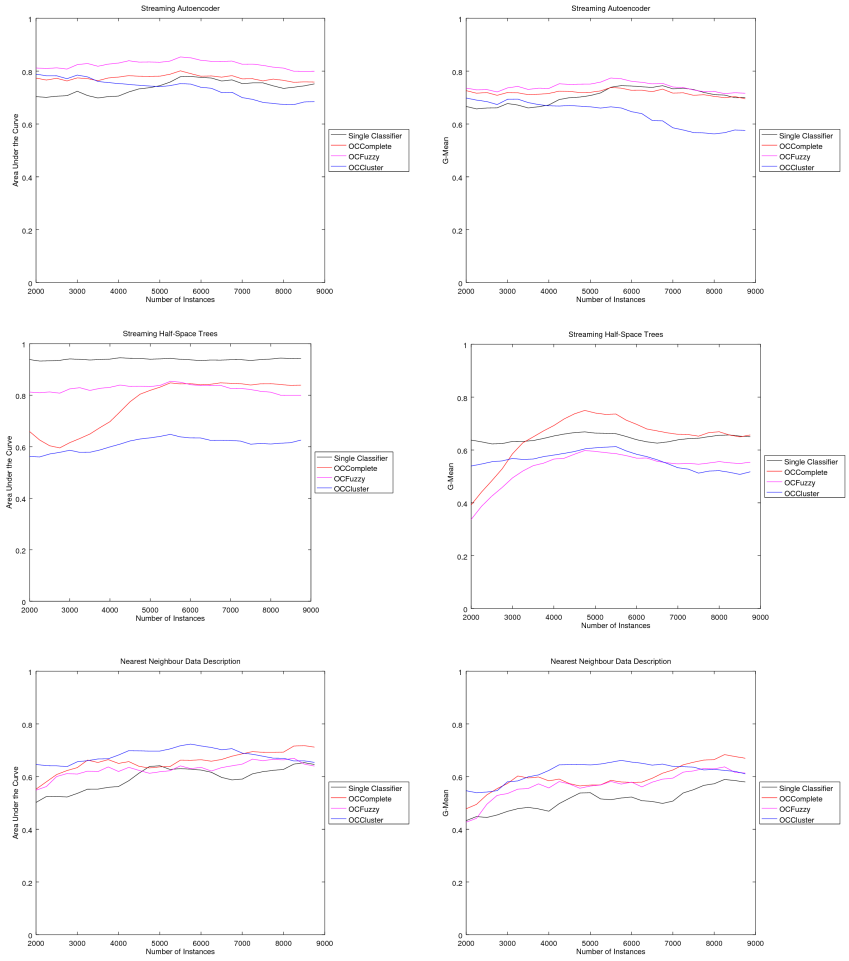
Compared to the other two classifiers, the NN-d again performs worse as a single classifier. The long term performance of the OCComplete framework is higher than that of the single classifier; both the OCComplete and OCFuzzy frameworks finish the data stream with the CBTT assessing their performance to be superior to the single classifier. Meanwhile, the OCCluster framework's performance again dominates the single classifier setting. The CBTT assesses that the OCCluster framework being superior with probability greater than 0.5 throughout.

Random RBF Data Stream with Noise The second Random RBF data stream, with noise - Figure 5.9, is the third example of the advantage accrued by the SA when using knowledge of the sub-concept structure. Both the OCComplete and OCFuzzy frameworks dominate the single classifier; while the CBTT confidently states that OCFuzzy is superior throughout the data stream, it becomes unsure about OCComplete towards the end. The OCCluster framework begins by performing better than the single classifier, but then sees its AUC degrade as the data stream continues and the single classifier is preferred at the end.

AUC scores for the Streaming HS-Trees behave very similarly to the AUC scores for the Random RBF data stream without noise. Again the single classifier performs very well and dominates the three frameworks; again the OCCluster framework results in markedly worse performance than the other three approaches. The CBTT comfortably assesses the single classifier as superior to each framework for the whole data stream. Similar to the Random RBF data stream, however, OCComplete's best threshold does provide higher g-mean scores at points during the data stream.

The situation for the NN-d is similar as well: the single classifier again performs poorly throughout the data stream. This time OCComplete and OCFuzzy both lead to AUC scores that are generally higher than the single classifier's while the OCCluster framework again dominates the single classifier. All three frameworks are generally assessed as superior to the single classifier by the CBTT, though there are parts of the data stream around instance 4000 where the single classifier is thought to be superior to OCComplete and OCFuzzy.

Figure 5.9: Results for the Random RBF data stream with Noise



5.2.1.1 Discussion

All three synthetic data streams show consistent results for each of the base classifiers. The performance of both the SA and the NN-d can be improved upon using knowledge of the sub-concept structure; the case of the Streaming HS-Trees is one where the incorporation of sub-concept knowledge repeatedly degrades classifier performance.

For the SA, the OCComplete and OCFuzzy frameworks, which make explicit use of the sub-concept structure, both dominate the single classifier throughout all three data streams. Their AUC scores are relatively consistent throughout the data streams, although this is certainly aided by the streams' synthetic nature. Interestingly, the OCFuzzy framework generally outperforms the OC-Complete framework. This suggests that, once the sub-concept structure is defined, it is better to adapt to the data stream's characteristics than to accept the formal definition of the sub-concepts as the best way of identifying the majority class. The OCCluster framework, which assumes a sub-concept structure but is given no more information regarding it, shows promise throughout the three data streams. Its performance is susceptible to a decline throughout the data stream, however, which leads it to perform worse than the single classifier after a certain point. A possible cause for this is that the clusters representing the sub-concepts are used to screen training instances for the SA. If any of the clusters begin to incorporate the minority class then this would directly impact the SA's ability to discriminate them.

Next, the NN-d method also benefits from the sub-concept structure. Although the single NN-d classifier reliably produced poor AUC scores, these were reliably increased by using the OCCluster framework. Interestingly, the OC-Complete and OCFuzzy frameworks only occasionally resulted in higher AUC scores while the OCCluster framework produced AUC scores that dominated the single classifier for all three data streams. It is worth noting that the decision boundary produced by the NN-d method depends only on two points, both of which are very close to the test instance. This suggests that the ClusTree algorithm used by OCCluster is able to find "local groupings" of points that are more informative for the NN-d than the formally defined concepts.

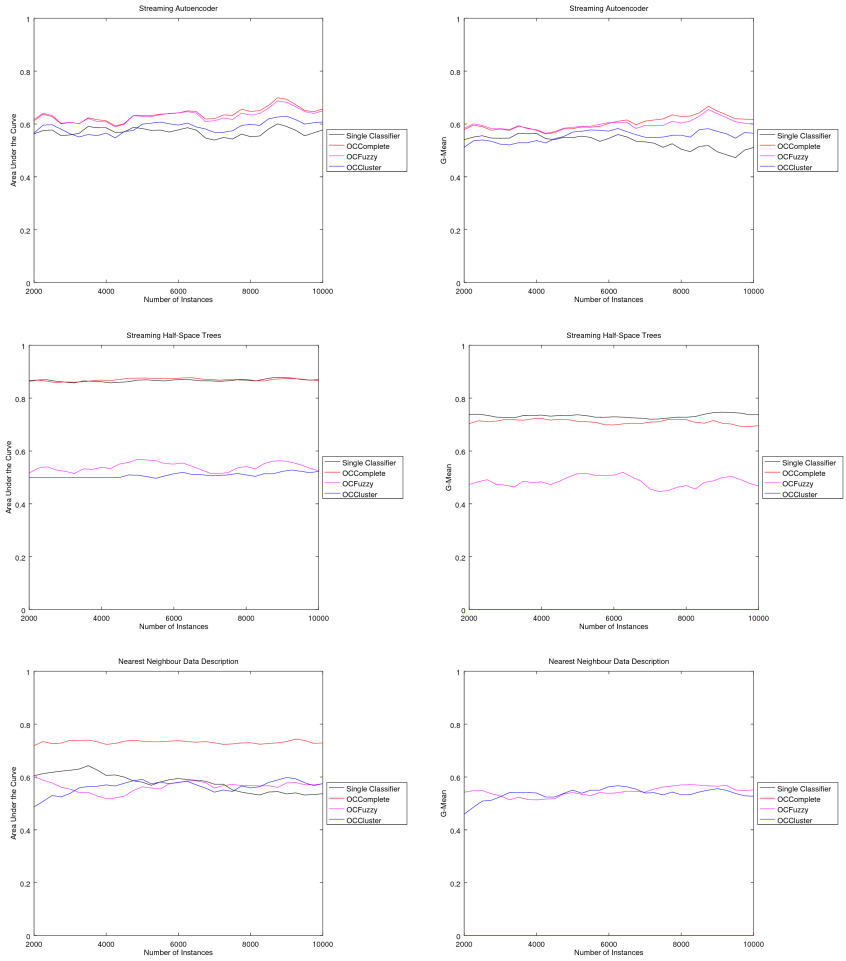
The Streaming HS-Trees result in the most disappointing performance. Only the Mixture Model data stream showed any improved performance as a result of using the sub-concept structure. Even this is muted by the fact that the single classifier performs very well and the increase in performance from the OCFuzzy framework is of a much smaller magnitude than the decreases in performance seen in the other two data streams. Interestingly, however, when performance using the optimal threshold is considered, the single classifier is generally beaten by OCComplete and occasionally by OCFuzzy as well. For each of the data streams the frameworks see increasing sensitivity (recognition of the minority class) and decreasing specificity (recognition of the majority class) while the single classifier sees both measures stay stable.

Two observations regarding the Streaming HS-Trees are that they embody an ensemble approach and that each HS-Tree's method of recognition involves

partitioning the feature space in an "instance-independent" manner. This suggests that the effect of training separate HS-Tree ensembles on each sub-concept actually has the effect of reducing the information available to each and results in poorer discriminating power.

5.2.2 Benchmark Data Streams

Figure 5.10: Results for the Wine Quality data stream



Wine Quality As was seen for the SA on the synthetic data streams, using the sub-concept structure improved classification for the Wine Quality data stream. The OCComplete and OCFuzzy frameworks have the highest AUC scores again;

the OCCluster framework exchanges positions with the single classifier a couple of times but generally performs better. These observations are true for the g-mean as well. The CBTT assesses OCComplete and OCFuzzy as being superior to the single classifier with probability 1 throughout the data stream while OCCluster is most likely superior in the stream's second half.[3]

The Streaming HS-Trees again sees excellent performance from the single classifier; here though the OCComplete framework results in AUC scores which are just as high and the CBTT is unable to clearly choose one approach as superior. Both the OCFuzzy and OCCluster frameworks, however, result in very poor AUC scores and are assessed as being superior with probability 0.

Interestingly for the NN-d method, the OCComplete framework produces the highest AUC scores throughout the data stream and the CBTT assesses its superiority over the single classifier as the most likely possibility throughout the data stream. Over the long term,OCFuzzy and OCCluster approaches resulted in similar performance. The CBTT assesses each as likely superior to the single classifier by the end of the data stream. This is the first time that the OCCluster framework did not dominate the single classifier and the first time that the OCComplete framework did so well for the NN-d. Inspecting the g-mean results, however, it is clear that optimal thresholds are more easily found for the OCFuzzy and OCCluster frameworks.[4]

Covertype With the SA as the base classifier, the frameworks that make explicit use of the sub-concept structure performed poorly on both Covertype data streams. Nonetheless, the OCCluster framework has the ability to produce better performance for both the SA and the NN-d. The first Covertype data stream, 2/5 vs 3/4/6 - Figure 5.11, does see OCCluster's performance degrade for the SA over the length of the data stream (as was seen on some of the synthetic data streams), but OCCluster does produce AUC scores that dominate the single classifier for the other classifier/data stream pairs.

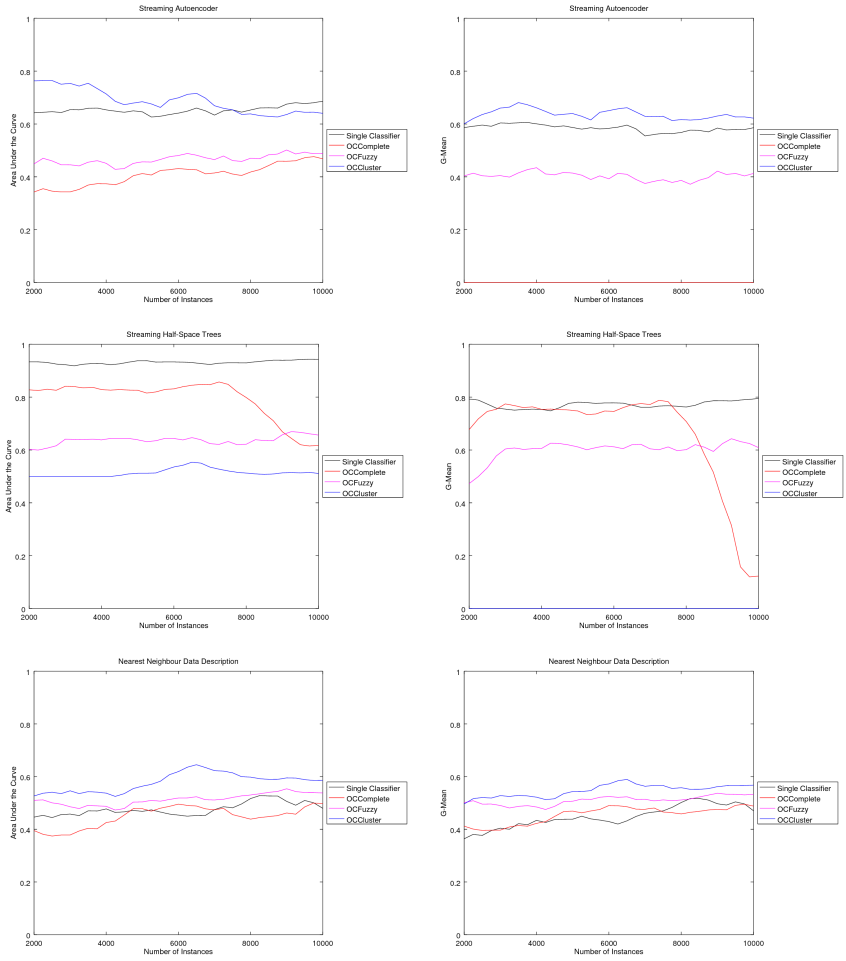
The single SA is assessed as superior to OCComplete and OCFuzzy for both Covertype data streams, but OCCluster is likely superior to the single classifier excepting the end of the 2/5 vs 3/4/6 data stream. For NN-d, the CBTT believes the OCCluster framework is superior to the single classifier for both data streams and OCCluster does produce the highest g-mean scores when using the optimal threshold. The situation is less clear for OCComplete and OCFuzzy, both of which produce AUC and g-mean scores similar to the single classifier.

As usual, the story is different for the Streaming HS-Trees. The single classifier approach easily dominates the performance produced by the OCCluster framework for both data streams; in fact the OCCluster performance begins each of the streams able to do no better than a randomly guessing classifier.

[3]Graphics showing the results of the CBTTs used in determining the statistical significance of differences in AUC scores for the benchmark data streams are shown in Appendix D.2.

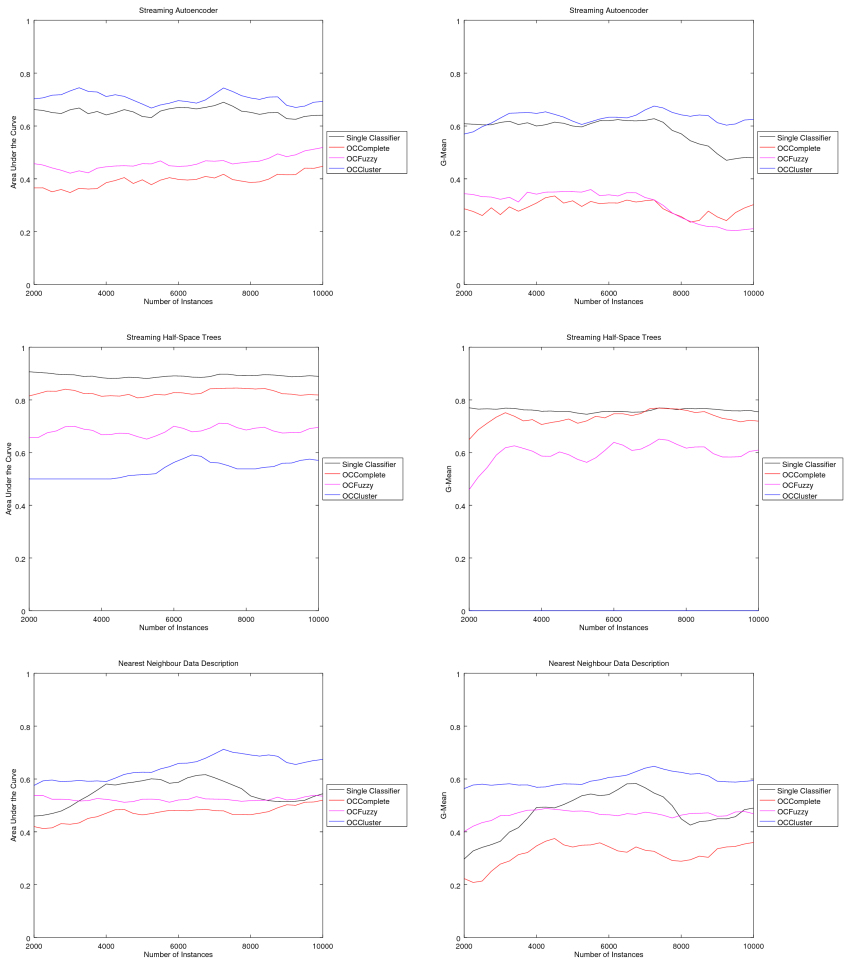
[4]The sensitivity and specificity values for each evaluation window are shown in Appendix C.2.

Figure 5.11: Results for the Covertype 2/5 vs 3/4/6 data stream



The CBTT is confident that the single classifier outperforms each framework for both data streams.

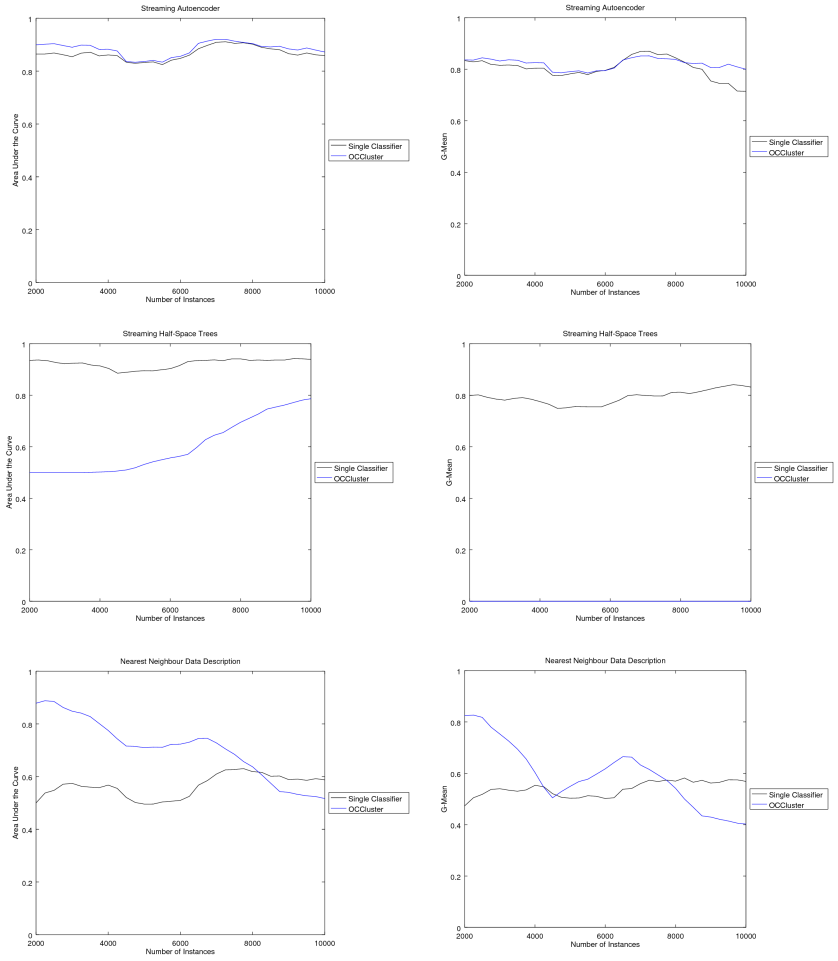
Figure 5.12: Results for the Covertype 1/2/5 vs 3/4/6/7 data stream



High Time Resolution Universe Survey The data stream representing the High Time Resolution Universe survey, Figure 5.13, lacks an explicitly defined sub-concept structure. Unlike the other data streams, however, the minority class is fairly well separated from the majority class in terms of principal components.

For the SA, the OCCluster framework dominates the single classifier in terms of AUC score for the entire data stream. This is done in a very tightly bound way: both approaches experiences dips and peaks in performance at the same

Figure 5.13: Results for the High Time Resolution Universe survey data stream



points during the data stream. The CBTT alternates between believing that OCCluster is superior to the single classifier and believing that they are practically equivalent.

For the NN-d method, the OCCluster framework begins with higher AUC scores than the single classifier and is assessed as superior by the CBTT. Its AUC and g-mean scores do degrade over time, however – a new phenomenon for the NN-d method, and the CBTT prefers the single classifier at the end of the stream.

As seen with most of the other data streams, the single classifier approach results in very good AUC scores for the Streaming HS-Trees and these dominate the scores produced when using the OCCluster framework. The CBTT is very confident that the single classifier is superior.

5.2.2.1 Discussion

The Wine Quality data stream is the only one of the benchmark data stream whose explicitly defined sub-concept structure proved useful. Similar results were observed as compared to the synthetic data streams: using of the sub-concept structure helped both the SA and NN-d classifiers while the single classifier approach to the Streaming HS-Trees performed very well. The most notable aspect of the Wine Quality data stream's results is the excellent performance of the OCComplete framework, which was the best or equal best approach for all three base classifies. Considering this, it seems that the sub-concepts for the Wine Quality data stream (red wine vs. white wine) are hard to discover but very helpful in guiding recognition.

For the Covertype data streams, although OCComplete and OCFuzzy (the two frameworks that use the explicit sub-concepts) do not perform well, OCCluster is able to find useful ways of breaking the majority class down for the SA and NN-d classifiers, resulting in superior performance. The Streaming HS-Trees classifiers, however, saw higher AUC scores as a single classifier than as part of any framework.

For the High Time Resolution Universe survey data stream, no explicit sub-concepts were available. Nonetheless, both the SA and NN-d classifiers were able to improve their AUC scores with the use of the OCCluster framework. This supports the idea that sub-concept structures discovered through inspection (i.e. Figure 4.8) can be profitably used by a one-class classifier without needing to be explicitly defined.

For the Streaming HS-Trees, the single classifier approach was again the superior approach. This is inline with the results obtained on the synthetic data streams and again suggests that the HS-Tree's design is not conducive to dividing instances up into sub-concepts.

5.2.3 Summary

The experimental results support the case that knowledge of the sub-concept structure can be used by streaming one-class classifiers to achieve superior per-

formance, though they also highlight a variety of challenges and situations where this is not the case.

Beginning with the positive findings, both the SA and NN-d classifiers regularly saw improved performance when knowledge of the sub-concept structure was incorporated. In the case of the SA, the explicit sub-concept knowledge used by the OCComplete and OCFuzzy frameworks led to the best performance, though the OCCluster framework generally outperformed the single classifier as well. For the NN-d method we saw a very different trend: the OCCluster framework generally outperformed the other three approaches and did dominate the single classifier for most of the data streams.

Also positive was that classifier performance could be improved using sub-concept knowledge, even if no sub-concept structure was explicitly defined for the data stream. This opens the door to applying these techniques to cases where data exploration suggests a sub-concept structure exists and does restrict them to data streams for which domain knowledge can explicitly enumerate and identify sub-concepts.

In terms of limitations, the major one was seen in the performance of the Streaming HS-Trees. This classifier only rarely showed an improvement for the frameworks over the single classifier; more distressingly the frameworks generally performed significantly worse. This was especially true for the OCCluster framework, which was never able to produce useful representations for the HS-Trees to learn from. As noted in the earlier discussions, there are two possible explanations for this. First, Streaming HS-Trees use a density-based method for OCC. From the review of the taxonomies in Section 2.2, it is reasonable to believe that density-based methods are more dependent on a global representation of the majority class than either reconstruction-based or boundary-based approaches. Density-based approaches may benefit more from a global picture of the data stream and suffer more from seeing only a sub-space of the data stream's feature space. Second, Streaming HS-Trees is an ensemble method made up of individual HS-Trees. As reviewed in Section 2.3, ensembles inherently incorporate the idea of sub-dividing a problem and it possible that there is a limit to which a problem can be usefully sub-divided, limiting the ability of the three frameworks to produce better performance.

Another identified limitation is that the OCCluster framework is susceptible to decreasing AUC scores as the data stream progresses. This obviously limits the usefulness of the technique for data streams, where steady-state behaviour over the long term is important. As discussed earlier in this chapter, the deciding factor for this is likely whether or not the clustering algorithm's clusters are able to track the majority class, whether they track the majority class tightly, and whether they begin to model concentrations of minority class instances as well.

Considering these findings as a whole, we note that there are clear areas for future research. These are discussed in the next chapter. With the caveats acknowledged, however, it is still the case that knowledge of the sub-concept structure can be employed to improve one-class classifier performance in data streams. Although the available knowledge and the classifier's characteristics need to be considered, both the SA and NN-d classifiers did obtain higher AUC

and g-mean scores when incorporated into the frameworks proposed in this work.

Chapter 6

Conclusion

This thesis identified that the idea of using sub-concept knowledge to improve one-class classifier performance has not been explored for the data stream environment. Although the idea has been demonstrated for static data sets, transitioning the idea to the data stream environment uncovered a number of challenges. These challenges were addressed by consulting the literature (e.g. subcomponent-based oversampling, clustering a large class to discover sub-concepts that are more easily learned), by proposing new theoretical ideas (e.g. the Cluster Distance Function), and by developing new experimental evidence (e.g. the behaviour of DSCAs in concept drifting data streams).

In all, this thesis made five contributions to the literature. The main contribution is three novel frameworks, described in Chapter 3.1, that use knowledge of the majority class's sub-concept structure in a OCC problem. These were presented with theoretical justification for their use. An experiment was conducted and its results discussed in Section 5.1.

Considering the experimental results presented in Section 5.2, we found that there was justification for using knowledge of the majority class's sub-concept structure for streaming one-class classifiers. Both the SA and NN-d classifiers saw higher AUC and g-mean scores when incorporated as part of one of the three frameworks as compared to when used as a single classifier. The success of the OCCluster framework, especially on the High Time Resolution Universe survey data stream, suggests that sub-concept structure can be profitably incorporated into learning frameworks for data streams without being explicitly defined.

The frameworks for this thesis also resulted in other contributions; the OCCluster framework in particular required the generation of novel theoretical and experimental results. The second contribution was a proof presented regarding the size of window necessary for a streaming algorithm to be assured, to a given degree of confidence, that there will be a minimum number of instances from a rare sub-concept. This proof was also demonstrated with experimental observation. Although this result was used in this thesis for selecting the frameworks' window size parameter, it is equally applicable to all stream learning algorithms.

The third contribution was the development of a novel Cluster Distance

Function based on CIP that measures the distance between any two clusters, regardless of shape or the DSCA that produced them. This function was also proven to be a metric and sample calculations were provided for the motivating scenarios.

The fourth contribution was an in-depth study of DSCAs and their behaviour in challenging data stream environments. This study surveyed the literature surrounding the data stream clustering task and experimentally compared five different DSCAs, allowing an informed choice to be made regarding the DSCA to incorporate into the OCCluster framework.

Finally, as part of the study of DSCA behaviour, a novel method was proposed to produce synthetic data streams with precise quantitative concept drift. This method, the Mixture Model Drift Generator, was an extension of a decision tree-based categorical data stream generator that could exhibit precise abrupt concept drift. The Mixture Model Drift Generator, however, was based on mixture models of multivariate distributions and could produce precise abrupt and extended concept drift. Extended concept drift could be done using either incremental or gradual concept drift. This method was also adapted to produce synthetic data streams consisting of a majority class with a sub-concept structure.

Alongside these contributions, two outstanding challenges and difficulties were identified as well. The clearest of these was the difficulty in incorporating the Streaming HS-Trees into the proposed frameworks. As discussed in Section 5.2, some of the Streaming HS-Trees' characteristics make it less likely to benefit from the sub-concept structure. The second identified challenge was the OCCluster framework's tendency for its AUC scores to decrease as the data stream went on.

To summarize, although there is evidence that this sub-concept knowledge can be made use of by one-class classifiers in the data stream environment, future research is required to fully develop this idea. Research directions to continue the work in the thesis include:

1. Expanding the evaluation of DSCAs performance when faced with heterogeneous forms of concept drift;

2. Further development of the Cluster Distance Function to better capture non-overlapped clusters, perhaps based on the concept of the Wasserstein metric (the earth mover's distance), and to formalize its application to categorical data;

3. Further investigation into the kinds of one-class classifiers for data streams that are able to benefit from sub-concept knowledge; and

4. A method to avoid the degradation in classifier performance that occurs in some data streams for the OCCluster framework.

Appendix A

Data Stream Clustering Algorithm Parameters

The MOA 17.06 implementation of each algorithm was used, with slightly modified versions of ClusTree and D-Stream used (these modifications are discussed below in the respective sections). We chose the parameters to be consistent across all experiments. In all cases, algorithms with a parameter k were given the number of classes at the beginning of the stream.

A.1 CluStream

The implementation CluStream with K-means was used.

Symbol	Description	Value
h	time-horizon	100
k	the number of clusters to find	*
q	the maximum number of micro-clusters	100
t	parameter controlling the maximum boundary of micro-clusters	2

Table A.1: Parameters for CluStream

A.2 ClusTree

As mentioned in Section 4.3.1, a slightly modified version of ClusTree[1] was used to correctly implement the offline k-means clustering algorithm and generate proper macroclusters.

Symbol	Description	Value
h	horizon, which determines the decay rate λ	100
H	the maximal height of the tree	5
-	whether to implement a breadth-first strategy instead of a depth-first strategy	FALSE
-	which offline clustering algorithm to use	Silhouette k-means

Table A.2: Parameters for ClusTree

A.3 D-Stream

As mentioned in Section 4.3.1, a slightly modified version of D-Stream[2] was used to permit the user to specify the width of grid cells for numerical attributes.

Symbol	Description	Value
λ	decay factor	0.998
C_m	parameter controlling the lower threshold for considering a grid cell to be dense	3
C_l	parameter controlling the upper threshold for considering a grid cell to be sparse	0.8
β	parameter controlling the length of time to wait before assessing whether a sparse grid cell is sporadic	0.3
-	Grid width (Synthetic real-valued data streams)	0.035
-	Grid width (ADFA-LD data streams)	0.5

Table A.3: Parameters for D-Stream

[1] available https://doi.org/10.5281/zenodo.1216189
[2] available https://doi.org/10.5281/zenodo.1213802

A.4 DenStream

Symbol	Description	Value
H	horizon	100
ϵ	the upper threshold for the radius of microclusters	0.065
β	determine the threshold of outlier relative to c-micro-clusters	0.2
μ	the lower threshold for the weight of microclusters	1
-	number of points to use for initialization	100
-	offline multiplier for ϵ	2
λ	decay factor	0.25
-	number of incoming points per time unit	10

Table A.4: Parameters for DenStream

A.5 StreamKM++

StreamKM++ was given the length of the data stream rounded up to the nearest 100, denoted by **.

Symbol	Description	Value
m	the size of the coreset	100
k	the number of clusters to find	*
n	the number of points in the data stream	**
-	random seed	1

Table A.5: Parameters for StreamKM++

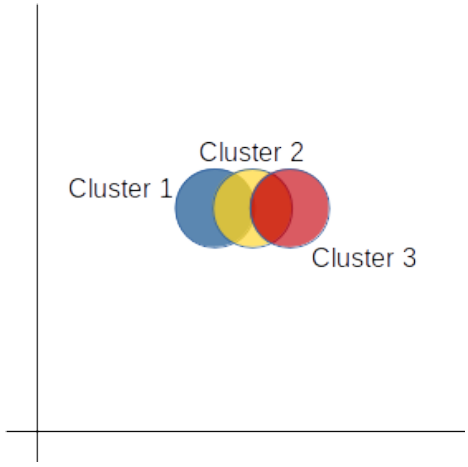
Appendix B

Sample Calculations for the Cluster Distance Function

The scenarios in this appendix are those that were introduced in Section 3.3. In all four scenarios, we expect Clusters 1 and 3 to be further apart from each other than either is from Cluster 2

B.1 Scenario A

Figure B.1: Clusters in scenario A



$$
\begin{aligned}
CD(C_1, C_2) = &\int_{\mathbb{F}} |IP_{C_1}(x) - IP_{C_2}(x)| \\
= &\int_{C_1 \setminus C_2} |IP_{C_1}(x) - IP_{C_2}(x)| + \int_{C_2 \setminus C_1} |IP_{C_1}(x) - IP_{C_2}(x)| \\
&+ \int_{C_1 \cup C_2} |IP_{C_1}(x) - IP_{C_2}(x)| + \int_{\mathbb{F} \setminus C_1 C_2} |IP_{C_1}(x) - IP_{C_2}(x)| \\
= &\int_{C_1 \setminus C_2} 1 + \int_{C_2 \setminus C_1} 1 + \int_{C_1 \cup C_2} 0 + \int_{\mathbb{F} \setminus C_1 C_2} 0 \\
= &|C_1 \setminus C_2| + |C_2 \setminus C_1| \\
\approx &5.05
\end{aligned}
$$

$$
\begin{aligned}
CD(C_1, C_3) = &\int_{\mathbb{F}} |IP_{C_1}(x) - IP_{C_3}(x)| \\
= &\int_{C_1} |IP_{C_1}(x) - IP_{C_3}(x)| + \int_{C_3} |IP_{C_1}(x) - IP_{C_3}(x)| \\
&+ \int_{\mathbb{F} \setminus C_1 C_3} |IP_{C_1}(x) - IP_{C_3}(x)| \\
= &\int_{C_1} 1 + \int_{C_3} 1 + \int_{\mathbb{F} \setminus C_1 C_3} 0 \\
= &|C_1| + |C_3| \\
= &2\pi
\end{aligned}
$$

$$
\begin{aligned}
CD(C_2, C_3) = &\int_{\mathbb{F}} |IP_{C_2}(x) - IP_{C_3}(x)| \\
= &\int_{C_2 \setminus C_3} |IP_{C_2}(x) - IP_{C_3}(x)| + \int_{C_3 \setminus C_2} |IP_{C_2}(x) - IP_{C_3}(x)| \\
&+ \int_{C_2 \cup C_3} |IP_{C_2}(x) - IP_{C_3}(x)| + \int_{\mathbb{F} \setminus C_2 C_3} |IP_{C_2}(x) - IP_{C_3}(x)| \\
= &\int_{C_2 \setminus C_3} 1 + \int_{C_3 \setminus C_2} 1 + \int_{C_2 \cup C_3} 0 + \int_{\mathbb{F} \setminus C_2 C_3} 0 \\
= &|C_2 \setminus C_3| + |C_3 \setminus C_2| \\
\approx &5.05
\end{aligned}
$$

Figure B.2: Clusters in scenario B

B.2 Scenario B

$$
\begin{aligned}
CD(C_1, C_2) = & \int_{\mathbb{F}} |IP_{C_1}(x) - IP_{C_2}(x)| \\
= & \int_{C_1} |IP_{C_1}(x) - IP_{C_2}(x)| + \int_{C_2} |IP_{C_1}(x) - IP_{C_2}(x)| \\
& + \int_{\mathbb{F} \setminus C_1 C_2} |IP_{C_1}(x) - IP_{C_2}(x)| \\
= & \int_{C_1} 1 + \int_{C_2} 1 + \int_{\mathbb{F} \setminus C_1 C_2} 0 \\
= & |C_1| + |C_2| \\
= & 2\pi
\end{aligned}
$$

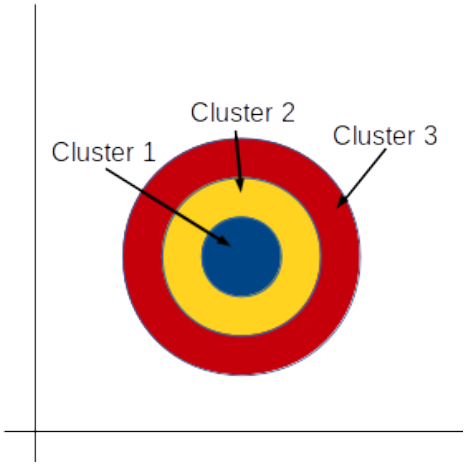
$$
\begin{aligned}
CD(C_1, C_3) &= \int_{\mathbb{F}} |IP_{C_1}(x) - IP_{C_3}(x)| \\
&= \int_{C_1} |IP_{C_1}(x) - IP_{C_3}(x)| + \int_{C_3} |IP_{C_1}(x) - IP_{C_3}(x)| \\
&\quad + \int_{\mathbb{F}\setminus C_1 C_3} |IP_{C_1}(x) - IP_{C_3}(x)| \\
&= \int_{C_1} 1 + \int_{C_3} 1 + \int_{\mathbb{F}\setminus C_1 C_3} 0 \\
&= |C_1| + |C_3| \\
&= 2\pi
\end{aligned}
$$

$$
\begin{aligned}
CD(C_2, C_3) &= \int_{\mathbb{F}} |IP_{C_2}(x) - IP_{C_3}(x)| \\
&= \int_{C_2} |IP_{C_2}(x) - IP_{C_3}(x)| + \int_{C_3} |IP_{C_2}(x) - IP_{C_3}(x)| \\
&\quad + \int_{\mathbb{F}\setminus C_2 C_3} |IP_{C_2}(x) - IP_{C_3}(x)| \\
&= \int_{C_2} 1 + \int_{C_3} 1 + \int_{\mathbb{F}\setminus C_2 C_3} 0 \\
&= |C_2| + |C_3| \\
&= 2\pi
\end{aligned}
$$

B.3 Scenario C

$$
\begin{aligned}
CD(C_1, C_2) &= \int_{\mathbb{F}} |IP_{C_1}(x) - IP_{C_2}(x)| \\
&= \int_{C_1} |IP_{C_1}(x) - IP_{C_2}(x)| + \int_{C_2\setminus C_1} |IP_{C_1}(x) - IP_{C_2}(x)| \\
&\quad + \int_{\mathbb{F}\setminus C_2} |IP_{C_1}(x) - IP_{C_2}(x)| \\
&= \int_{C_1} 0 + \int_{C_2\setminus C_1} 1 + \int_{\mathbb{F}\setminus C_2} 0 \\
&= |C_2| - |C_1| \\
&= 3\pi
\end{aligned}
$$

Figure B.3: Clusters in scenario C

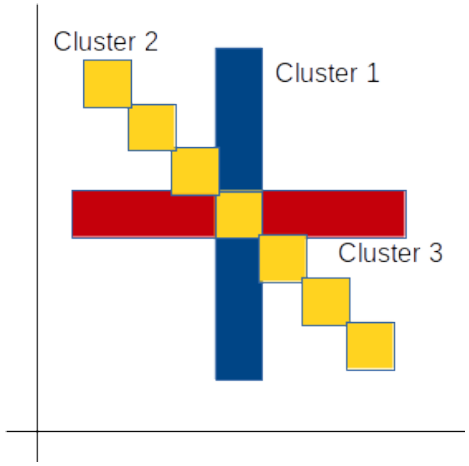


$$
\begin{aligned}
CD(C_1, C_3) = & \int_{\mathbb{F}} |IP_{C_1}(x) - IP_{C_3}(x)| \\
= & \int_{C_1} |IP_{C_1}(x) - IP_{C_3}(x)| + \int_{C_3 \setminus C_1} |IP_{C_1}(x) - IP_{C_3}(x)| \\
& + \int_{\mathbb{F} \setminus C_3} |IP_{C_1}(x) - IP_{C_3}(x)| \\
= & \int_{C_1} 0 + \int_{C_3 \setminus C_1} 1 + \int_{\mathbb{F} \setminus C_3} 0 \\
= & |C_3| - |C_1| \\
= & 8\pi
\end{aligned}
$$

$$
\begin{aligned}
CD(C_2, C_3) = & \int_{\mathbb{F}} |IP_{C_2}(x) - IP_{C_3}(x)| \\
= & \int_{C_2} |IP_{C_2}(x) - IP_{C_3}(x)| + \int_{C_3 \setminus C_2} |IP_{C_2}(x) - IP_{C_3}(x)| \\
& + \int_{\mathbb{F} \setminus C_3} |IP_{C_2}(x) - IP_{C_3}(x)| \\
= & \int_{C_2} 0 + \int_{C_3 \setminus C_2} 1 + \int_{\mathbb{F} \setminus C_3} 0 \\
= & |C_3| - |C_2| \\
= & 5\pi
\end{aligned}
$$

B.4 Scenario D

Figure B.4: Clusters in scenario D



$$
\begin{aligned}
CD(C_1, C_2) &= \int_{\mathbb{F}} |IP_{C_1}(x) - IP_{C_2}(x)| \\
&= \int_{C_1 \setminus C_2} |IP_{C_1}(x) - IP_{C_2}(x)| + \int_{C_2 \setminus C_1} |IP_{C_1}(x) - IP_{C_2}(x)| + \\
&\quad \int_{C_1 \cup C_2} |IP_{C_1}(x) - IP_{C_2}(x)| + \int_{\mathbb{F} \setminus C_2} |IP_{C_1}(x) - IP_{C_2}(x)| \\
&= \int_{C_1 \setminus C_2} 1 + \int_{C_2 \setminus C_1} 1 + \int_{C_1 \cup C_2} 0 + \int_{\mathbb{F} \setminus C_1 C_2} 0 \\
&= |C_1| + |C_2| - 2|C_1 \cup C_2| \\
&= 12
\end{aligned}
$$

$$
\begin{aligned}
CD(C_1, C_3) = & \int_{\mathbb{F}} |IP_{C_1}(x) - IP_{C_3}(x)| \\
= & \int_{C_1 \setminus C_3} |IP_{C_1}(x) - IP_{C_3}(x)| + \int_{C_3 \setminus C_1} |IP_{C_1}(x) - IP_{C_3}(x)| + \\
& \int_{C_1 \cup C_3} |IP_{C_1}(x) - IP_{C_3}(x)| + \int_{\mathbb{F} \setminus C_3} |IP_{C_1}(x) - IP_{C_3}(x)| \\
= & \int_{C_1 \setminus C_3} 1 + \int_{C_3 \setminus C_1} 1 + \int_{C_1 \cup C_3} 0 + \int_{\mathbb{F} \setminus C_1 C_3} 0 \\
= & |C_1| + |C_3| - 2|C_1 \cup C_3| \\
= & 12
\end{aligned}
$$

$$
\begin{aligned}
CD(C_2, C_3) = & \int_{\mathbb{F}} |IP_{C_2}(x) - IP_{C_3}(x)| \\
= & \int_{C_2 \setminus C_3} |IP_{C_2}(x) - IP_{C_3}(x)| + \int_{C_3 \setminus C_2} |IP_{C_2}(x) - IP_{C_3}(x)| + \\
& \int_{C_2 \cup C_3} |IP_{C_2}(x) - IP_{C_3}(x)| + \int_{\mathbb{F} \setminus C_3} |IP_{C_2}(x) - IP_{C_3}(x)| \\
= & \int_{C_2 \setminus C_3} 1 + \int_{C_3 \setminus C_2} 1 + \int_{C_2 \cup C_3} 0 + \int_{\mathbb{F} \setminus C_2 C_3} 0 \\
= & |C_2| + |C_3| - 2|C_2 \cup C_3| \\
= & 12
\end{aligned}
$$

Appendix C

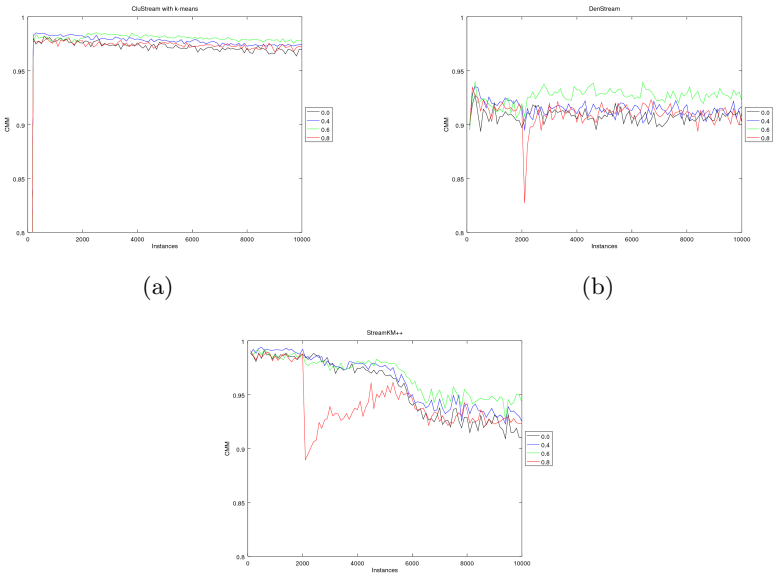
Additional Results

C.1 Selecting a Data Stream Clustering Algorithm

C.1.1 Synthetic Data Streams

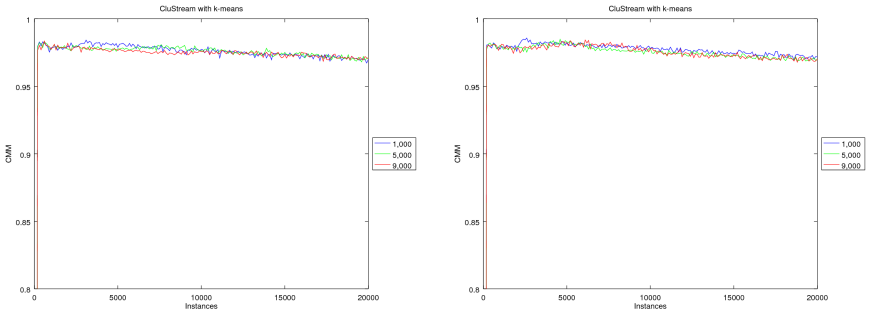
These are the results for cases that were not included in Chapter 4.3. Figures C.1, C.2 and C.3 show additional results for DSCA experiments A, B, and C.

Figure C.1: Additional results for experiment A



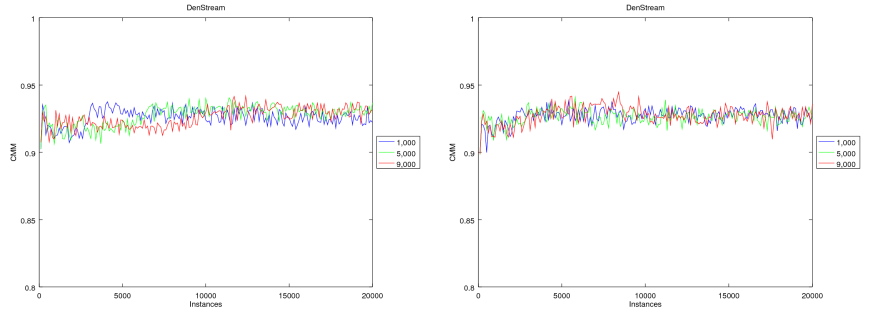
(a) (b)

(c)



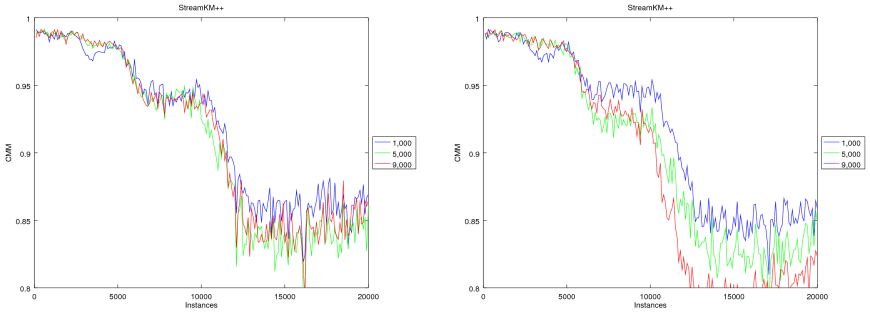
(a) Gradual

(b) Incremental



(c) Gradual

(d) Incremental

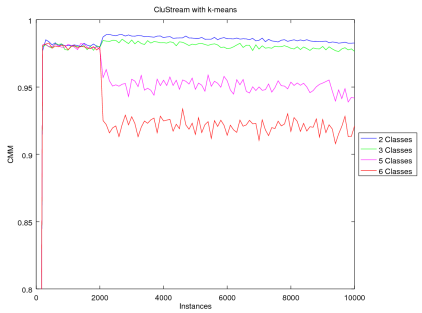


(e) Gradual

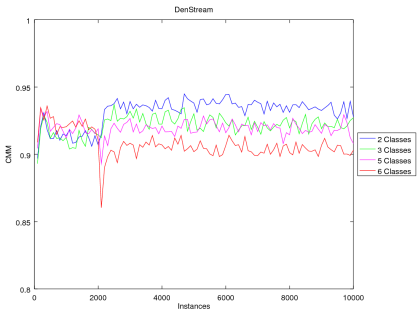
(f) Incremental

Figure C.2: Additional results for experiment B

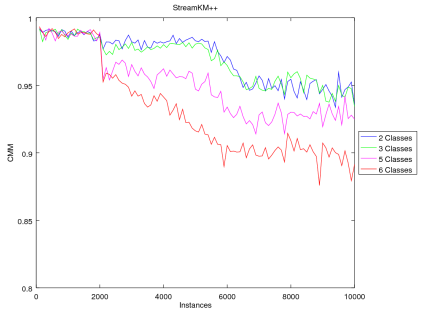
Figure C.3: Additional results for experiment C



(a)



(b)



(c)

C.1.2 Real-World Data Streams

Figure C.4 shows the additional results for the ADFA-LD data streams. The behaviour of the DSCAs for these data streams was considered very similar to their behaviour in the two reported cases.

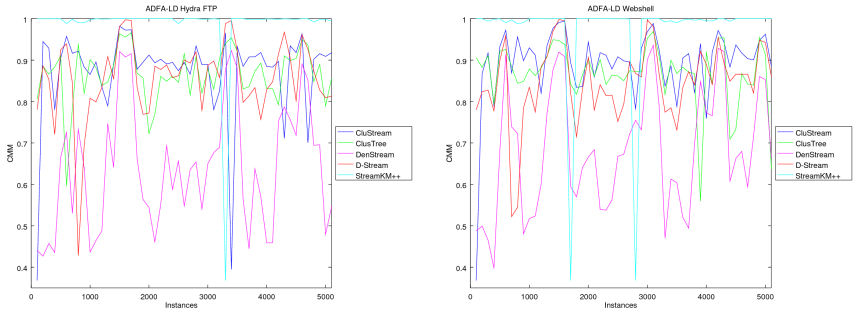


Figure C.4: Additional results for the ADFA-LD data streams

C.2 One-Class Classification in Data Streams

These are the sensitivity and specificity graphs for each of the algorithm/approach pair reported on in Chapter 5.

C.2.1 Synthetic Data Streams

Figure C.5: Sensitivity and Specificity for the Mixture Model data stream

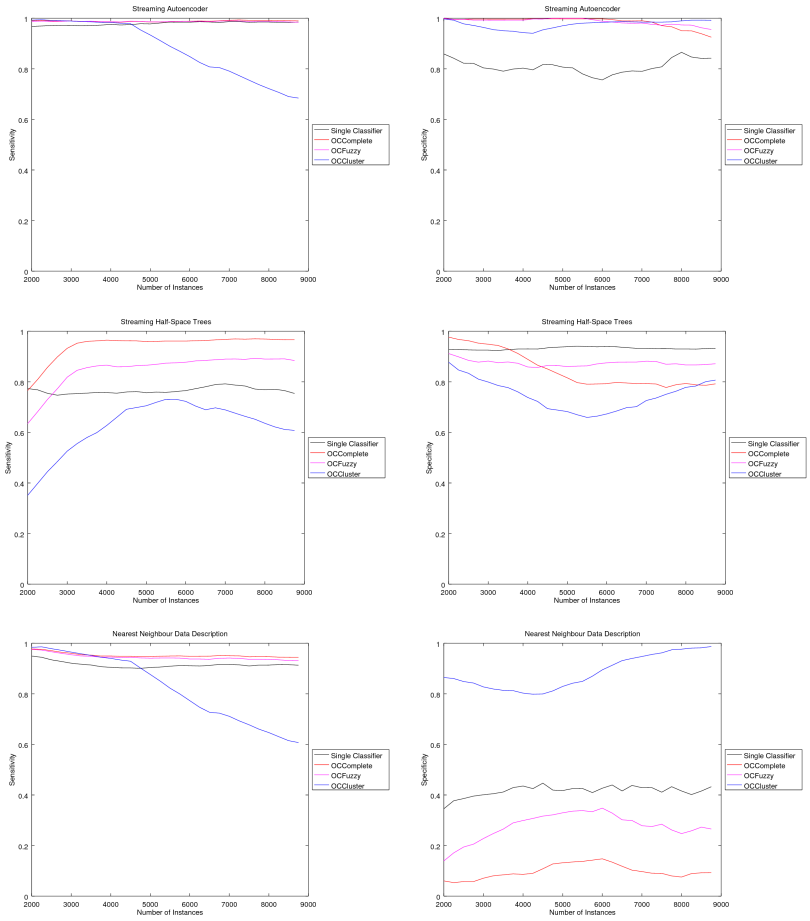


Figure C.6: Sensitivity and Specificity for the Random RBF data stream

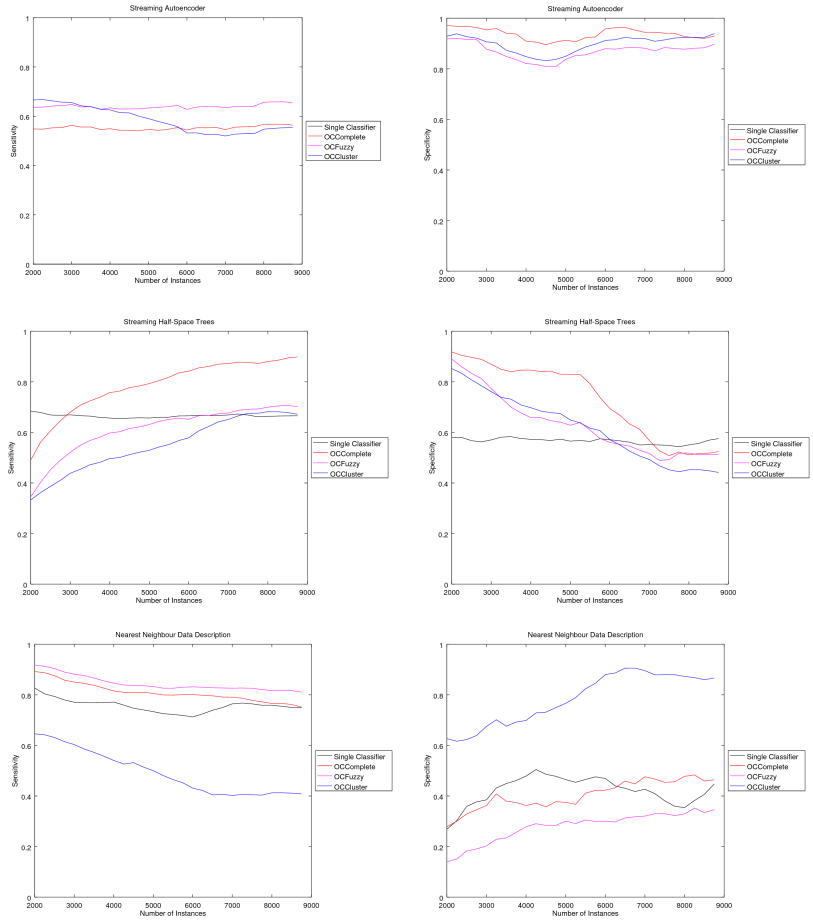
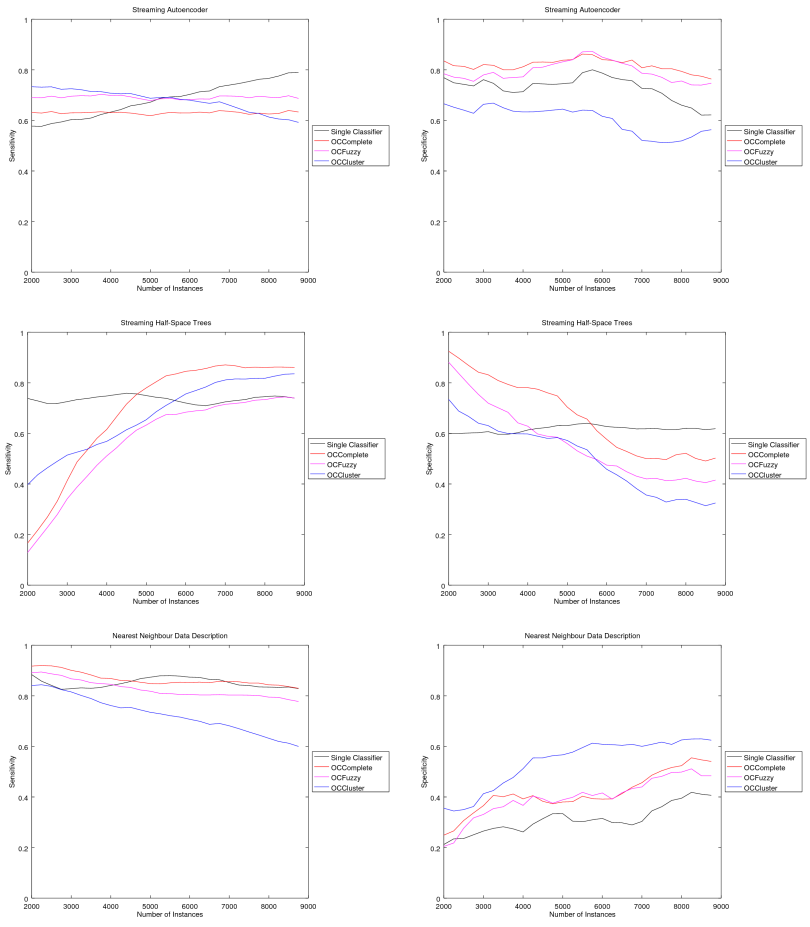


Figure C.7: Sensitivity and Specificity for the Random RBF data stream with Noise



C.2.2 Benchmark Data Streams

Figure C.8: Sensitivity and Specificity for the Wine Quality data stream

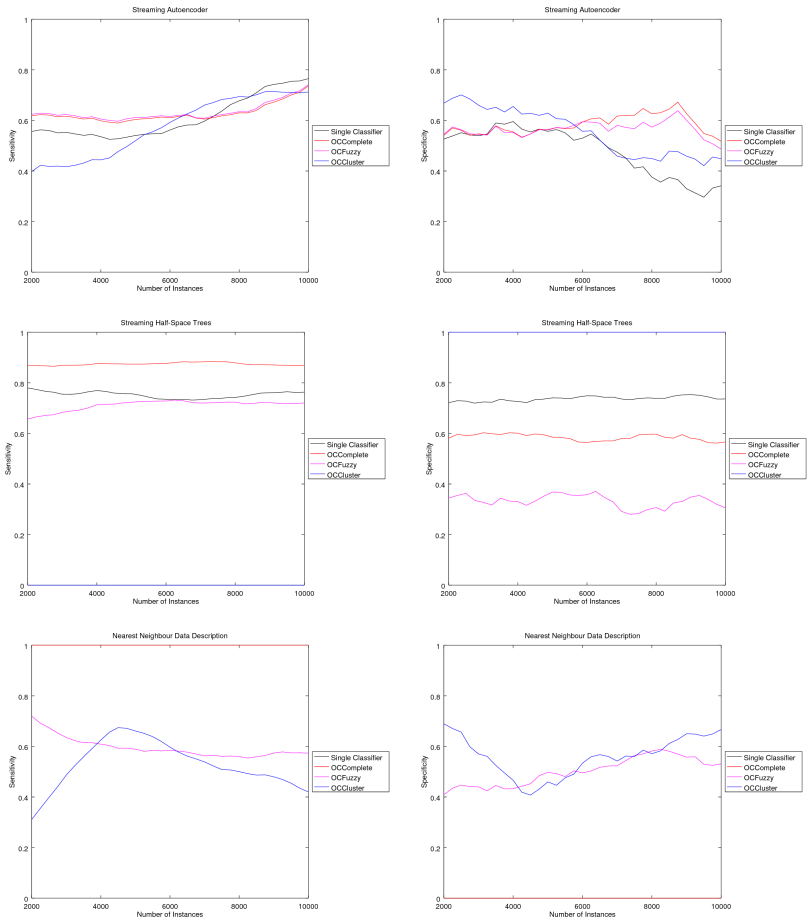


Figure C.9: Sensitivity and Specificity for the Covertype 2/5 vs 3/4/6 data stream

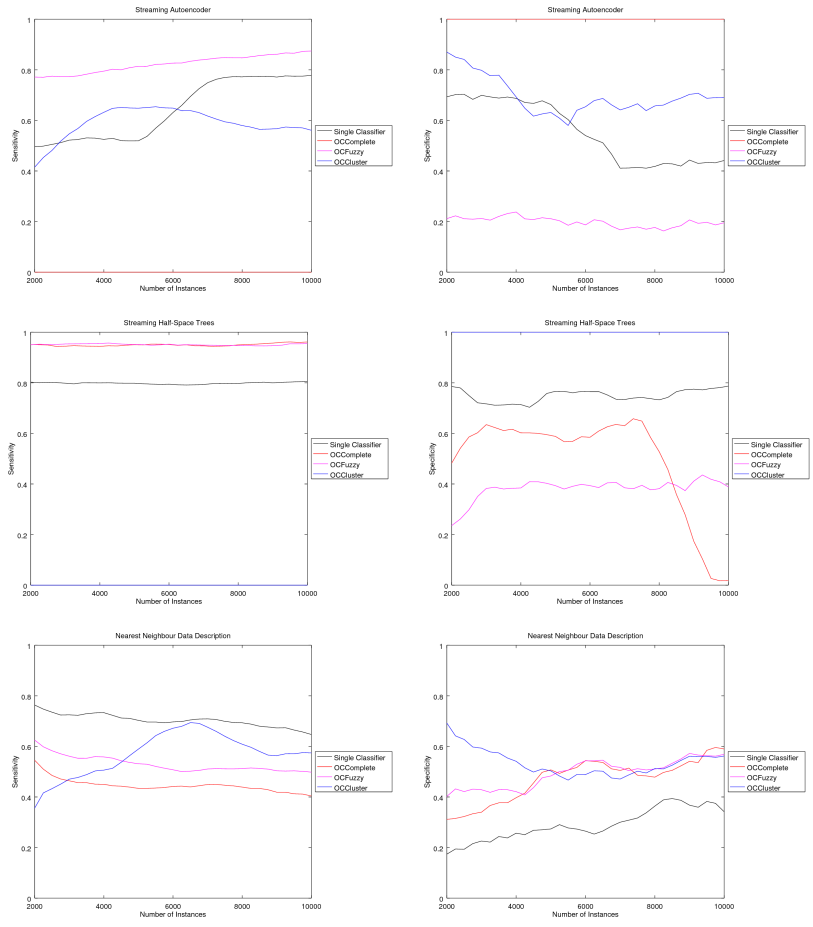


Figure C.10: Sensitivity and Specificity for the Covertype 1/2/5 vs 3/4/6/7 data stream

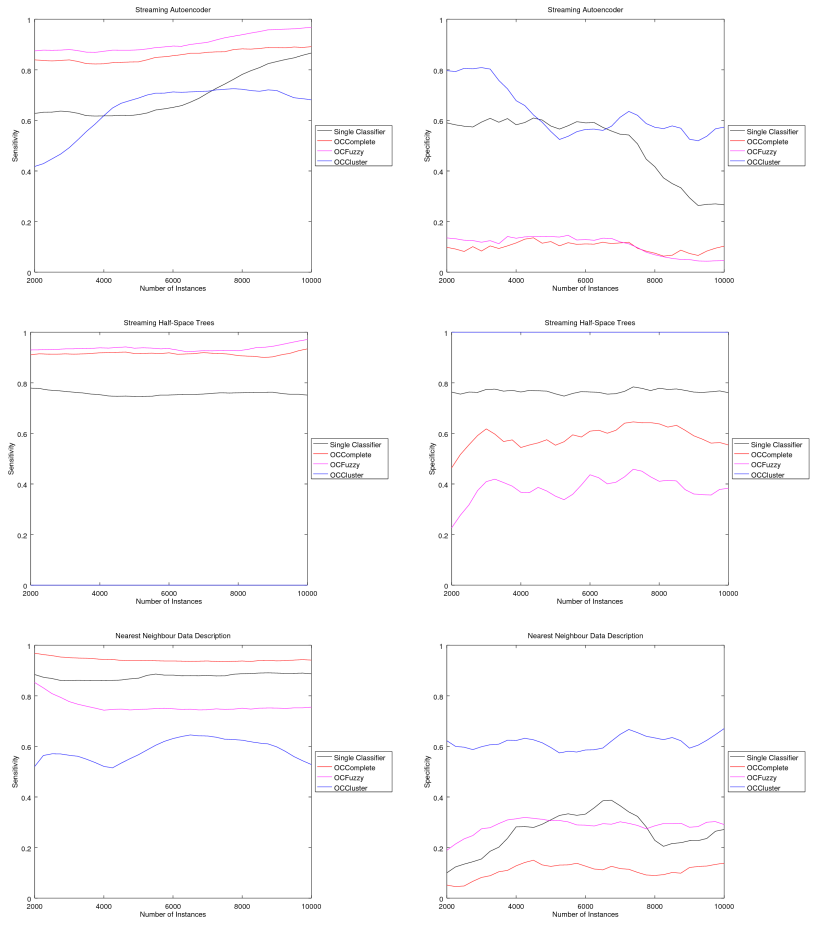
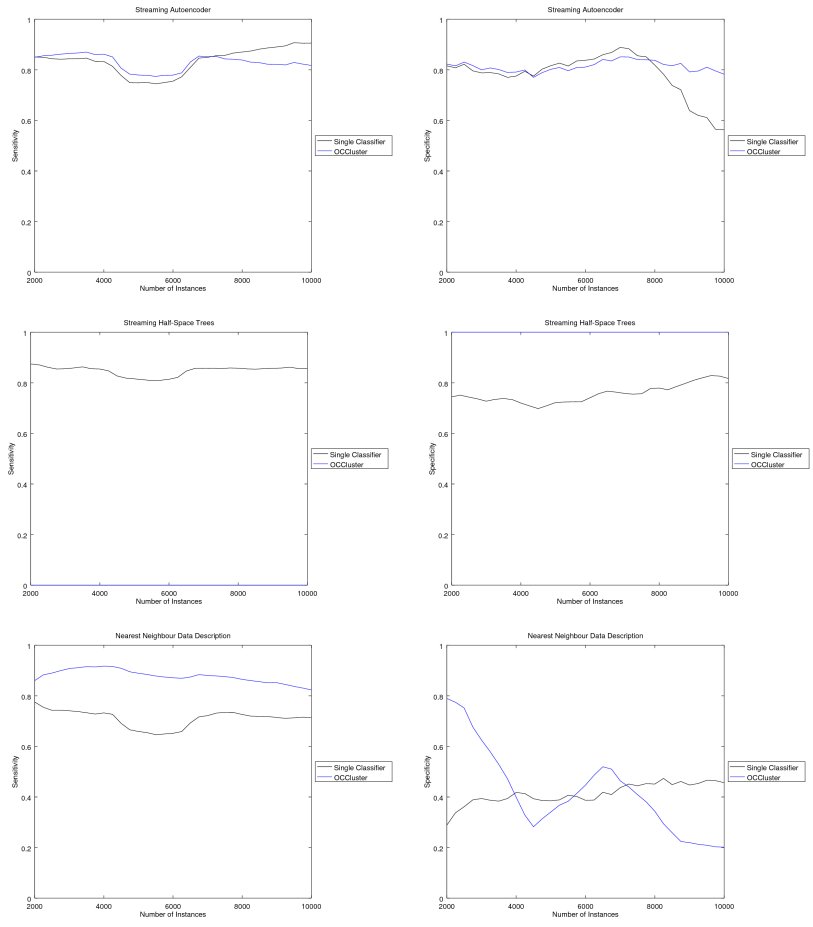


Figure C.11: Sensitivity and Specificity for the High Time Resolution Universe survey data stream



Appendix D

Statistical Significance Results

These graphics display the graphical results of the correlated Bayesian t-tests for each framework's AUC scores compared to the single classifier's AUC scores.

The graphs are organized by data stream (rows) and by framework being compared to the single classifier (columns).

In each graph, the blue/cyan/green line represents the probability that the framework being compared is superior, the red line represents the probability that the single classifier is superior, and the black line represents the probability that the two are practically equivalent.

D.1 Synthetic Data Streams

Figure D.1: Statistical significance of difference in Area Under the Curve scores between the proposed frameworks and the single classifier (synthetic data streams, autoencoder as base classifier).

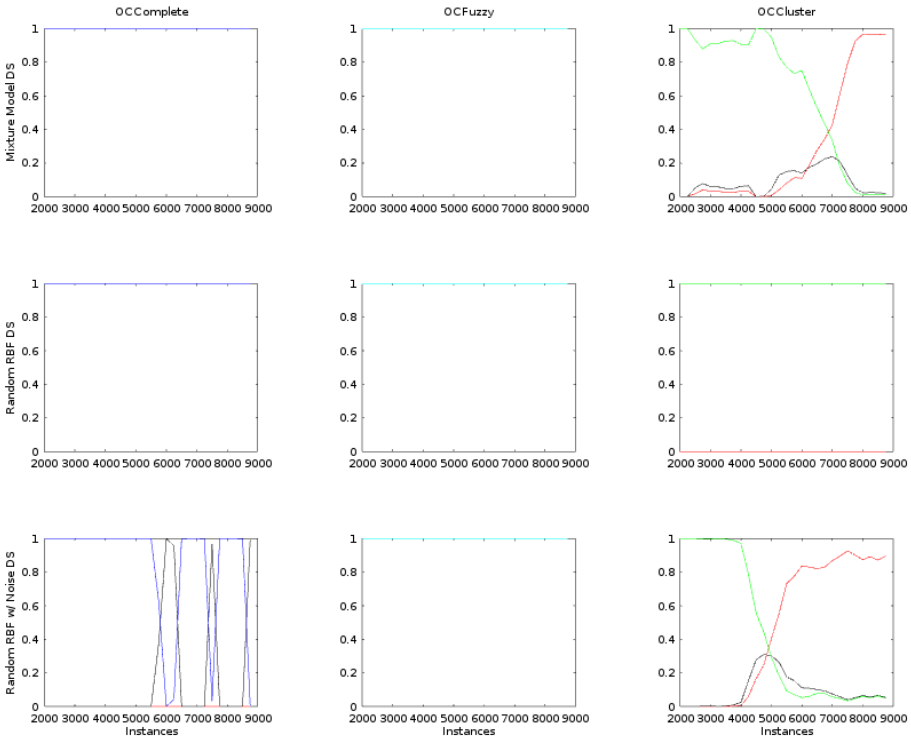


Figure D.2: Statistical significance of difference in Area Under the Curve scores between the proposed frameworks and the single classifier (synthetic data streams, Streaming Half-Space Trees as base classifier).

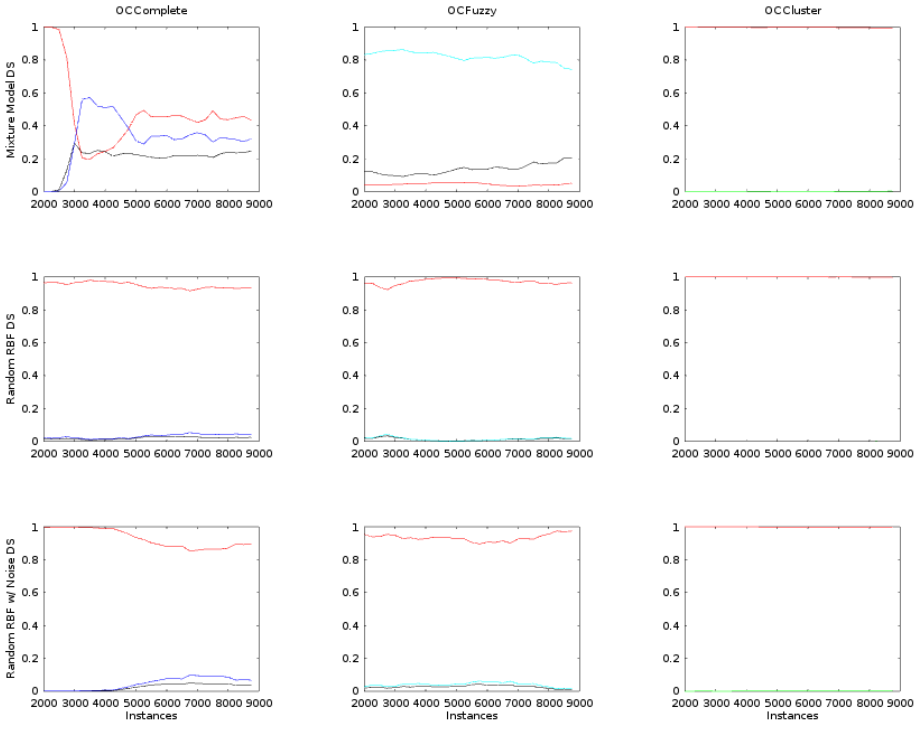
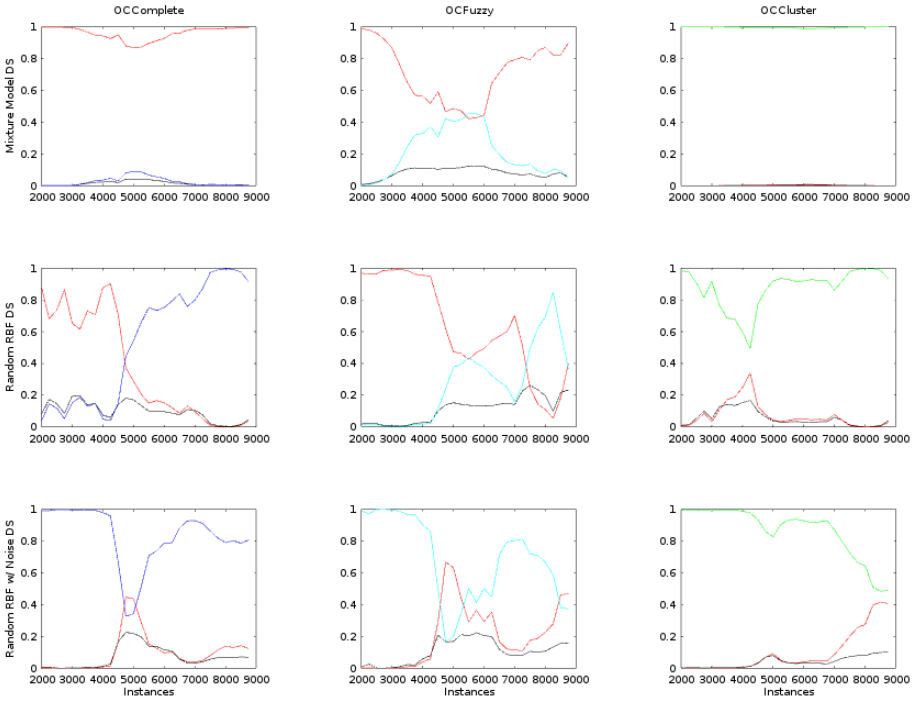


Figure D.3: Statistical significance of difference in Area Under the Curve scores between the proposed frameworks and the single classifier (synthetic data streams, Nearest Neighbour Data Description as base classifier).



D.2 Benchmark Data Streams

Figure D.4: Statistical significance of difference in Area Under the Curve scores between the proposed frameworks and the single classifier (benchmark data streams, autoencoder as base classifier).

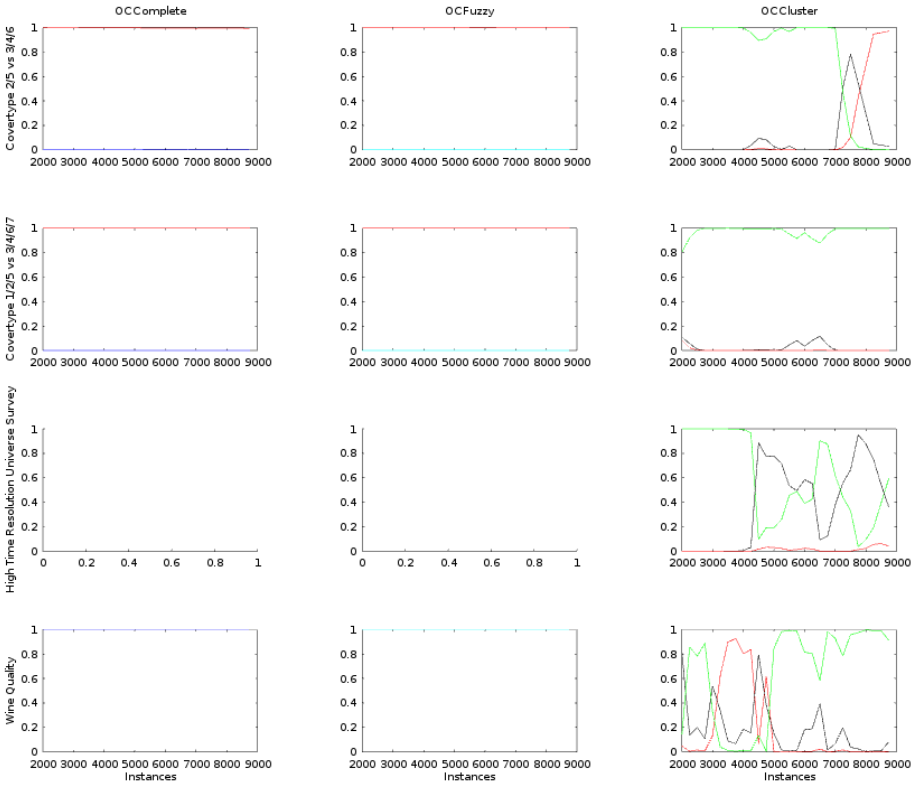


Figure D.5: Statistical significance of difference in Area Under the Curve scores between the proposed frameworks and the single classifier (benchmark data streams, Streaming Half-Space Trees as base classifier).

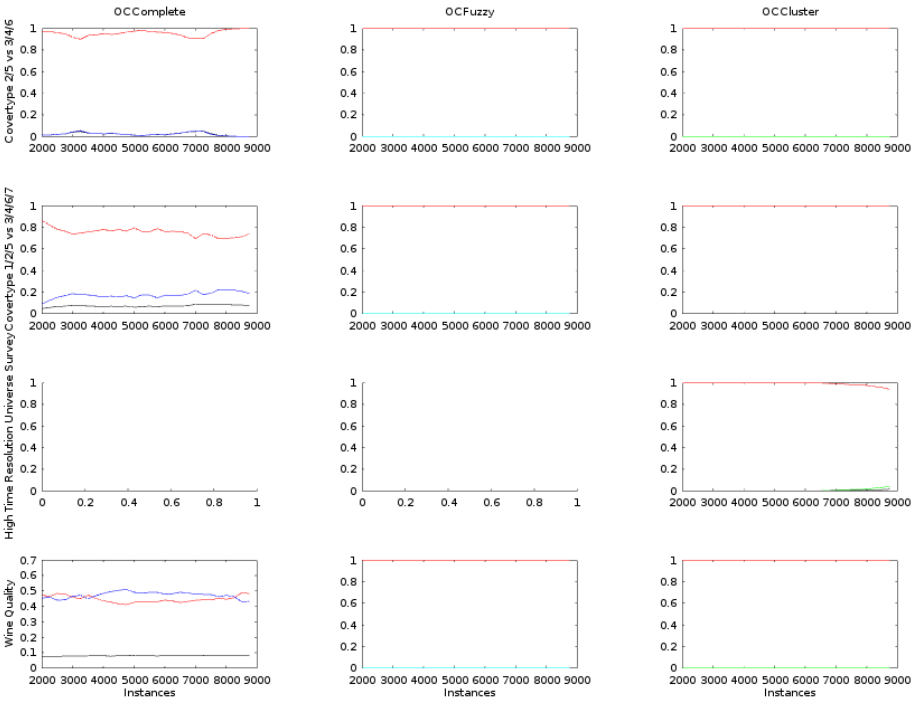
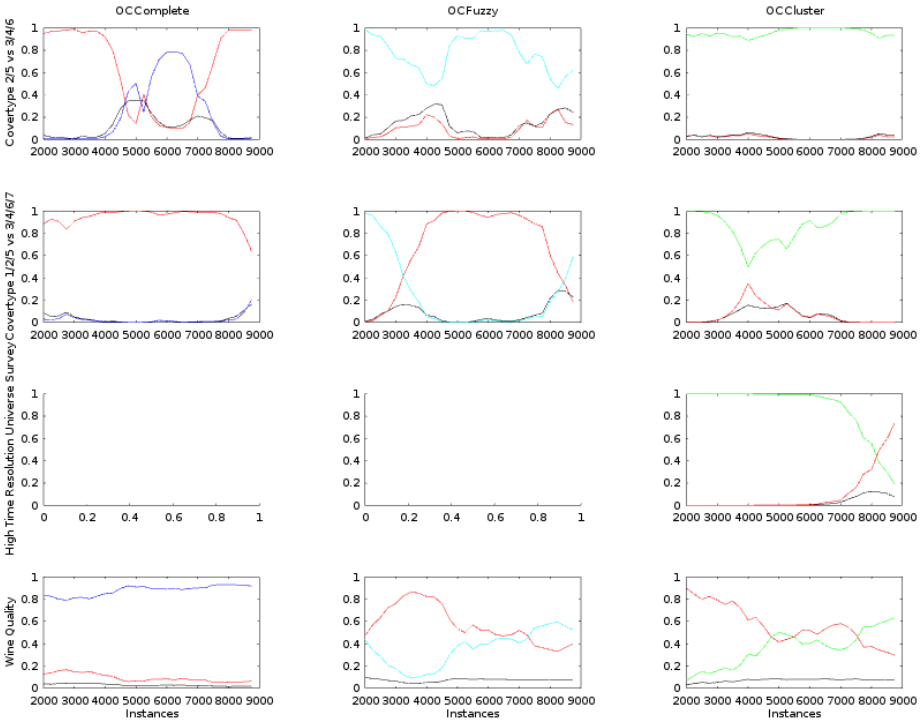


Figure D.6: Statistical significance of difference in Area Under the Curve scores between the proposed frameworks and the single classifier (benchmark data streams, Nearest Neighbour Data Description as base classifier).



Bibliography

[1] Zahraa S. Abdallah, Mohamed Medhat Gaber, Bala Srinivasan, and Shonali Krishnaswamy. AnyNovel: detection of novel concepts in evolving data streams. *Evolving Systems*, 7(2):73–93, Jun 2016.

[2] Marcel R. Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. StreamKM++: A Clustering Algorithm for Data Streams. *ACM Journal of Experimental Algorithmics*, 17(2):30, Jul 2012.

[3] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A Framework for Clustering Evolving Data Streams. In *29th Very Large Database Conference*, page 12, Berlin, 2003.

[4] Omar Y. Al-Jarrah, Yousof Al-Hammdi, Paul D. Yoo, Sami Muhaidat, and Mahmoud Al-Qutayri. Semi-Supervised Multi-Layered Clustering Model for Intrusion Detection. *Digital Communications and Networks*, (September):1–10, Sep 2017.

[5] David Arthur and Sergei Vassilvitskii. K-Means++: the Advantages of Careful Seeding. *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 8:1027–1025, 2007.

[6] Stephen H. Bach and Marcus A. Maloof. Paired learners for concept drift. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 23–32, 2008.

[7] Michel Ballings and Dirk Van den Poel. *AUC: Threshold independent performance measures for probabilistic classifiers.*, 2013.

[8] Daniel Barbará. Requirements for clustering data streams. *ACM SIGKDD Explorations Newsletter*, 3(2):23, 2002.

[9] Daniel Barbará and Ping Chen. Using the fractal dimension to cluster datasets. *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '00*, pages 260–264, 2000.

[10] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria C. Monard. Balancing Strategies and Class Overlapping. *Advances in Intelligent Data Analysis VI*, pages 24–35, 2005.

[11] Alessio Benavoli, Giorgio Corani, Janez Demšar, and Marco Zaffalon. Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis. *Journal of Machine Learning Research*, 18(1):2653–2688, Jun 2017.

[12] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *Journal of Machine Learning Research*, 11(May):1601–1604, 2010.

[13] Christopher M. Bishop. Mixture Density Networks. Technical report, Neural Computing Research Group, Department of Computer Science and Applied Mathematics, Aston University, Birmingham, UK, 1994.

[14] Jock A. Blackard and Denis J. Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24(3):131–151, 1999.

[15] Abdelhamid Bouchachia. Fuzzy classification in dynamic environments. *Soft Computing*, 15(5):1009–1022, May 2011.

[16] Paula Branco, Luís Torgo, and Rita P. Ribeiro. A Survey of Predictive Modeling on Imbalanced Domains. *ACM Computing Surveys*, 49(2):1–50, Aug 2016.

[17] Dariusz Brzezinski and Jerzy Stefanowski. Prequential AUC: properties of the area under the ROC curve for data streams with concept drift. *Knowledge and Information Systems*, 52(2):531–562, 2017.

[18] Borja Calvo and Guzman Santafe. *scmamp: Statistical Comparison of Multiple Algorithms in Multiple Problems*, 2015.

[19] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-Based Clustering over an Evolving Data Stream with Noise. In Joydeep Ghosh, Diane Lambert, David Skillicorn, and Jaideep Srivastava, editors, *Proceedings of the 2006 SIAM International Conference on Data Mining*, pages 328–339, Bethesda, 2006.

[20] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[21] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[22] Sheng Chen and Haibo He. Nonstationary Stream Data Learning with Imbalanced Class Distribution. In Haibo He and Yunqian Ma, editors, *Imbalanced Learning: Foundations, Algorithms, and Applications*, chapter 7, pages 151–186. John Wiley & Sons, Inc., first edition, 2013.

[23] Yixin Chen and Li Tu. Density-Based Clustering for Real-Time Stream Data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, San Jose, USA, 2007.

[24] Giorgio Corani, Alessio Benavoli, Janez Demšar, Francesca Mangili, and Marco Zaffalon. Statistical comparison of classifiers through Bayesian hierarchical modelling. *Machine Learning*, 106(11):1817–1837, 2017.

[25] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, Nov 2009.

[26] Gideon Creech and Jiankun Hu. Generation of a new IDS test dataset: Time to retire the KDD collection. In *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, pages 4487–4492. IEEE, 2013.

[27] David L. Davies and Donald W. Bouldin. A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, Apr 1979.

[28] Elaine Ribeiro de Faria, André Carlos Ponce de Leon Ferreira Carvalho, and João Gama. MINAS: multiclass learning algorithm for novelty detection in data streams. *Data Mining and Knowledge Discovery*, 30(3):640–680, 2015.

[29] Janez Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[30] Michel Marie Deza and Elena Deza. *Encyclopedia of Distances*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[31] Gregory Ditzler and Robi Polikar. Hellinger distance based drift detection for nonstationary environments. In *2011 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE)*, pages 41–48. IEEE, Apr 2011.

[32] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, Oct 2012.

[33] Yue Dong and Nathalie Japkowicz. Threaded Ensembles of Supervised and Unsupervised Neural Networks for Stream Learning. *Lecture Notes in Computer Science*, 9673:304–315, 2016.

[34] Elaine R. Faria, Isabel J.C.R. Gonçalves, André C.P.L.F. de Carvalho, and João Gama. Novelty detection in data streams. *Artificial Intelligence Review*, 45(2):235–269, 2016.

[35] B Fritzke. A Growing Neural Gas Learns Topologies. *Advances in Neural Information Processing Systems*, 7:625–632, 1995.

[36] João Gama. *Knowledge Discovery from Data Streams*. Chapman and Hall/CRC, 2010.

[37] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with Drift Detection. In Ana L C Bazzan and Sofiane Labidi, editors, *Proceedings of the 17th Brazilian Symposium on Artificial Intelligence*, pages 286–295, Sao Luis, Brazil, 2004. Springer Berlin Heidelberg.

[38] João Gama, Pedro Pereira Rodrigues, and Luís Lopes. Clustering distributed sensor data streams using local processing and reduced communication. *Intelligent Data Analysis*, 15(1):3–28, 2011.

[39] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346, 2013.

[40] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):1–37, Mar 2014.

[41] Mohammed Ghesmoune, Hanene Azzag, and Mustapha Lebbah. G-Stream : Growing Neural Gas over Data Stream. *Lecture Notes in Computer Science*, 8834:207–214, 2014.

[42] Mohammed Ghesmoune, Mustapha Lebbah, and Hanene Azzag. State-of-the-art on clustering data streams. *Big Data Analytics*, 1(1):13, 2016.

[43] Heitor Murilo Gomes, Jean Paul Barddal, Fabrício Enembreck, and Albert Bifet. A Survey on Ensemble Learning for Data Stream Classification. *ACM Computing Surveys*, 50(2):1–36, 2017.

[44] R.A. Gopinath. Maximum likelihood modeling with Gaussian distributions for classification. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98*, volume 2, pages 661–664, Seattle, USA, 1998. IEEE.

[45] Waqas Haider, Jiankun Hu, and Miao Xie. Towards reliable data feature retrieval and decision engine in host-based anomaly detection systems. *Proceedings of the 2015 10th IEEE Conference on Industrial Electronics and Applications, ICIEA 2015*, pages 513–517, 2015.

[46] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann Publishers, third edition, 2011.

[47] Morteza Zi Hayat and Mahmoud Reza Hashemi. A DCT based approach for detecting novelty and concept drift in data streams. *Proceedings of the 2010 International Conference of Soft Computing and Pattern Recognition, SoCPaR 2010*, pages 373–378, 2010.

[48] Haibo He and Edwardo A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009.

[49] Jiazhen He, Yang Zhang, Xue Li, and Peng Shi. Learning naive Bayes classifiers from positive and unlabelled examples with uncertainty. *International Journal of Systems Science*, 43(10):1805–1825, 2012.

[50] Kathryn Hempstalk, Eibe Frank, and Ian H. Witten. One-class classification by combining density and class probability estimation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5211 LNAI(PART 1):505–519, 2008.

[51] T. Ryan Hoens and Nitesh V. Chawla. Imbalanced Datasets: From Sampling to Classifiers. In Haibo He and Yunqian Ma, editors, *Imbalanced Learning: Foundations, Algorithms, and Applications*, chapter 3, pages 43–59. John Wiley & Sons, Inc., first edition, 2013.

[52] Mohammad Javad Hosseini, Ameneh Gholipour, and Hamid Beigy. An ensemble of cluster-based classifiers for semi-supervised classification of non-stationary data streams. *Knowledge and Information Systems*, 46(3):567–597, 2016.

[53] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive Mixtures of Local Experts. *Neural Computation*, 3(1):79–87, 1991.

[54] Nathalie Japkowicz. Supervised versus unsupervised binary-learning by feedforward neural networks. *Machine Learning*, 42(1-2):97–122, 2001.

[55] Nathalie Japkowicz. Assessment Metrics for Imbalanced Learning. In Haibo He and Yunqian Ma, editors, *Imbalanced Learning: Foundations, Algorithms, and Applications*, chapter 8, pages 187–206. John Wiley & Sons, Inc., first edition, 2013.

[56] Nathalie Japkowicz, Catherine Myers, and Mark Gluck. A novelty detection approach to classification. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 518–523, 1995.

[57] Nathalie Japkowicz and Mohak Shah. *Evaluating learning algorithms: a classification perspective.* Cambridge University Press, New York, 2011.

[58] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429 – 449, 2002.

[59] Taeho Jo and Nathalie Japkowicz. Class Imbalances versus Small Disjuncts. *SIGKDD Explorations*, 6(1):40–49, 2004.

[60] Shehroz S. Khan and Michael G. Madden. One-class classification: taxonomy of study and review of techniques. *The Knowledge Engineering Review*, 29(03):345–374, Jun 2014.

[61] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982.

[62] Philipp Kranen, Ira Assent, Corinna Baldauf, and Thomas Seidl. The ClusTree: indexing micro-clusters for anytime stream mining. *Knowledge and Information Systems*, 29(2):249–272, Nov 2011.

[63] Bartosz Krawczyk, Leandro L. Minku, João Gama, Jerzy Stefanowski, and Michał Woźniak. Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132–156, Sep 2017.

[64] Bartosz Krawczyk and Michał Woźniak. Incremental Learning and Forgetting in One-Class Classifiers for Data Streams. In *Advances in Intelligent Systems and Computing*, pages 319–328. 2013.

[65] Bartosz Krawczyk and Michał Woźniak. One-class classifiers with incremental learning and forgetting for data streams with concept drift. *Soft Computing*, 19(12):3387–3400, Dec 2015.

[66] Hardy Kremer, Philipp Kranen, Timm Jansen, Thomas Seidl, Albert Bifet, and Geoff Holmes. An Effective Evaluation Measure for Clustering on Evolving Data Streams. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'11)*, pages 868–876, New York, 2011. ACM Press.

[67] Georg Krempl, Myra Spiliopoulou, Jerzy Stefanowski, Indrė Žliobaitė, Dariusz Brzeziński, Eyke Hüllermeier, Mark Last, Vincent Lemaire, Tino Noack, Ammar Shaker, and Sonja Sievi. Open challenges for data stream mining research. *ACM SIGKDD Explorations Newsletter*, 16(1):1–10, Sep 2014.

[68] John K. Kruschke. Bayesian assessment of null values via parameter estimation and model comparison. *Perspectives on Psychological Science*, 6(3):299–312, 2011.

[69] Miroslav Kubat, Robert C. Holte, and Stan Matwin. Machine learning for the detection of oil spills in satellite radar images. *Machine Learning*, 30(2-3):195–215, 1998.

[70] Miroslav Kubat and Stan Matwin. Addressing the Curse of Imbalanced Training Sets: One Sided Selection. In Douglas H. Fisher, editor, *Proceedings of the Fourteenth International Conference on Machine Learning*, volume 97, pages 179–186, Nashville, USA, 1997. Morgan Kaufmann.

[71] Ludmila I. Kuncheva. Switching between selection and fusion in combining classifiers: An experiment. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 32(2):146–156, 2002.

[72] Ludmila I. Kuncheva. Using control charts for detecting concept change in streaming data. Technical report, Bangor University, Bangor, United Kingdom, 2009.

[73] Jorma Laurikkala. Improving Identification of Difficult Small Classes by Balancing Class Distribution. In S. Quaglini, P. Barahona, and S. Andreassen, editors, *Artificial Intelligence in Medicine. AIME 2001. Lecture Notes in Computer Science, vol 2101*, pages 63–66. Springer Berlin Heidelberg, 2001.

[74] Chen Li, Yang Zhang, and Xue Li. OcVFDT: One-class Very Fast Decision Tree for One-class Classification of Data Streams. *Proceedings of the Third International Workshop on Knowledge Discovery from Sensor Data - SensorKDD '09*, pages 79–86, 2009.

[75] Larry S. Liebovitch and Tibor Toth. A fast algorithm to determine fractal dimensions by box counting. *Physics Letters A*, 141(8-9):386–390, 1989.

[76] Patrick Lindstrom, Brian Mac Namee, and Sarah Jane Delany. Drift Detection Using Uncertainty Distribution Divergence. In *2011 IEEE 11th International Conference on Data Mining Workshops*, pages 604–608. IEEE, Dec 2011.

[77] Nedim Lipka, Benno Stein, and Maik Anderka. Cluster-based one-class ensemble for classification problems in information retrieval. *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval - SIGIR '12*, page 1041, 2012.

[78] Viktor Losing, Barbara Hammer, and Heiko Wersing. Tackling heterogeneous concept drift with the Self-Adjusting Memory (SAM). *Knowledge and Information Systems*, 54(1):171–201, Jan 2018.

[79] R. J. Lyon, B. W. Stappers, S. Cooper, J. M. Brooke, and J. D. Knowles. Fifty years of pulsar candidate selection: from simple filters to a new principled real-time classification approach. *Monthly Notices of the Royal Astronomical Society*, 459(1):1104–1123, Jun 2016.

[80] Wolfgang Maass. On-line Learning with an Oblivious Environment and the Power of Randomization. In Manfred K. Warmuth and Leslie G. Valiant, editors, *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, page 9, San Mateo, USA, 1991. Morgan Kaufmann Publishers.

[81] Saeed Masoudnia and Reza Ebrahimpour. Mixture of experts: A literature survey. *Artificial Intelligence Review*, 42(2):275–293, 2014.

[82] Mohammad Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani M. Thuraisingham. Classification and Novel Class Detection in Concept-Drifting Data Streams under Time Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 23(6):859–874, Jun 2011.

[83] Oleksiy Mazhelis. One-Class Classifiers : A Review and Analysis of Suitability in the Context of Mobile-Masquerader Detection. *South African Computer Journal*, 2006(36):29–48, 2006.

[84] Marina Meilă. Comparing clusterings-an information based distance. *Journal of Multivariate Analysis*, 98(5):873–895, 2007.

[85] Adam Nickerson, Nathalie Japkowicz, and Evangelos E. Milios. Using Unsupervised Learning to Guide Resampling in Imbalanced Data Sets. In *AISTATS*, 2001.

[86] Irene Ntoutsi, Myra Spiliopoulou, and Yannis Theodoridis. Tracing cluster transitions for different cluster types. *Control and Cybernetics*, 38(1):239–259, 2009.

[87] Liadan O'Callaghan, Nina Mishra, Adam Meyerson, Sudipto Guha, and Rajeev Motwani. Streaming-Data Algorithms For High-Quality Clustering. *Proceedings 18th International Conference on Data Engineering*, pages 1–25, 2002.

[88] Márcia Oliveira and João Gama. A framework to monitor clusters evolution applied to economy and finance problems. *Intelligent Data Analysis*, 16(1):93–111, 2012.

[89] E. S. Page. Continuous Inspection Schemes. *Biometrika*, 41(1/2):100, Jun 1954.

[90] Marco A.F. Pimentel, David A. Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, Jun 2014.

[91] David M W Powers. Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation. Technical Report December, Flinders University, Adelaide, Australia, 2007.

[92] Ronaldo C. Prati, Gustavo E. A. P. A. Batista, and Maria Carolina Monard. Class Imbalances versus Class Overlapping: An Analysis of a Learning System Behavior. In *MICAI 2004: Advances in Artificial Intelligence*, pages 312–321. 2004.

[93] Colorado State University Remote Sensing and GIS Program, Department of Forest Sciences, College of Natural Resources. Covertype Data Set, 1998.

[94] Jiangtao Ren, Sau Dan Lee, Xianlu Chen, Ben Kao, Reynold Cheng, and David Cheung. Naive Bayes Classification of Uncertain Data. In *2009 Ninth IEEE International Conference on Data Mining*, pages 944–949, Miami, USA, Dec 2009. IEEE.

[95] Gordon J. Ross, Niall M. Adams, Dimitris K. Tasoulis, and David J. Hand. Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters*, 33(2):191–198, Jan 2012.

[96] Jeffrey C Schlimmer and Richard H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986.

[97] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the Support of a High-Dimensional Distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[98] Tegjyot Singh Sethi and Mehmed Kantardzic. On the reliable detection of concept drift from streaming unlabeled data. *Expert Systems with Applications*, 82:77–99, Oct 2017.

[99] Ammar Shaker and Eyke Hüllermeier. IBLStreams: A system for instance-based classification and regression on data streams. *Evolving Systems*, 3(4):235–249, 2012.

[100] Shiven Sharma. *Learning in the Context of Concepts : A Framework for One-Class Classification*. Doctoral, University of Ottawa, 2014.

[101] Jonathan A. Silva, Elaine R. Faria, Rodrigo C. Barros, Eduardo R. Hruschka, André C.P.L.F. De Carvalho, and João Gama. Data Stream Clustering: A Survey. *ACM Computing Surveys*, 46(1):1–31, 2013.

[102] Parinaz Sobhani, Herna Viktor, and Stan Matwin. Learning from Imbalanced Data Using Ensemble Methods and Cluster-Based Undersampling. In *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, volume 8983, pages 69–83, 2015.

[103] Myra Spiliopoulou, Irene Ntoutsi, Yannis Theodoridis, and Rene Schult. MONIC - Modeling and Monitoring Cluster Transitions. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and DataMining (KDD'06)*, page 6, Philadelphia, PA, 2006.

[104] Eduardo J. Spinosa, André Ponce De Leon F. de Carvalho, and João Gama. Novelty detection with application to data streams. *Intelligent Data Analysis*, 13(3):405–422, 2009.

[105] Eduardo J. Spinosa, André Ponce de Leon F. de Carvalho, and João Gama. OLINDDA: a cluster-based approach for detecting novelty and concept drift in data streams. In *Proceedings of the 2007 ACM Symposium on Applied Computing - SAC '07*, number November 2015, page 448, Seoul, Korea, 2007. ACM Press.

[106] Jerzy Stefanowski. Overlapping, Rare Examples and Class Decomposition in Learning Classifiers from Imbalanced Data. In *Emerging Paradigms in Machine Learning. Smart Innovation, Systems and Technologies, vol 13*, volume 13, pages 277–306. 2013.

[107] Jerzy Stefanowski and Szymon Wilk. Selective pre-processing of imbalanced data for improving classification performance. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5182 LNCS:283–292, 2008.

[108] Swee Chuan Tan, Kai Ming Ting, and Tony Fei Liu. Fast anomaly detection for streaming data. In *Proceedings of the Twenty Second International Joint Conference on Artificial Intelligence*, volume 2, pages 1511–1516, 2011.

[109] Dimitris K. Tasoulis, Gordon Ross, and Niall M. Adams. Visualising the cluster structure of data streams. In Michael R. Berthold, John Shawe-Taylor, and Nada Lavrač, editors, *Advances in Intelligent Data Analysis VII*, pages 81–92, Ljubljana, Slovenia, 2007. Springer, Berlin, Heidelberg.

[110] David Martinus Johannes Tax. *One-Class Classification: Concept-learning in the absence of counter-examples*. Doctoral, Delft University of Technology, 2001.

[111] David M.J. Tax and Robert P.W. Duin. Combining One-Class Classifiers. *Lecture Notes in Computer Science*, 1032:299–308, 2001.

[112] David M.J. Tax and Robert P.W. Duin. Support Vector Data Description. *Machine Learning*, 54(1):45–66, 2004.

[113] Geoffrey I. Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and François Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994, Jul 2016.

[114] Gary M. Weiss. Foundations of Imbalanced Learning. In Haibo He and Yunqian Ma, editors, *Imbalanced Learning: Foundations, Algorithms, and Applications*, chapter 2, pages 13–41. John Wiley & Sons, Inc., first edition, 2013.

[115] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, Apr 1996.

[116] David H. Wolpert. The Existence of A Priori Distinctions Between Learning Algorithms. *Neural Computation*, 8(7):1391–1420, 1996.

[117] Alexander Ypma and Robert P.W. Duin. Support objects for domain approximation. In L. Niklasson, M. Bodén, and T. Ziemke, editors, *ICANN 98*, pages 719–724. Springer, London, United Kingdom, 1998.

[118] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An Efficient Data Clustering Databases Method for Very Large Databases. *ACM SIGMOD International Conference on Management of Data*, 1:103–114, 1996.

[119] Wenbin Zhang and Jianwu Wang. A Hybrid Learning Framework for Imbalanced Stream Classification. In George Karypis and Jia Zhang, editors, *2017 IEEE International Congress on Big Data*, pages 480–487, Honolulu, USA, Jun 2017. IEEE.

[120] Indrė Žliobaitė. Learning under Concept Drift: an Overview. Technical report, Vilnius University, 2010.

[121] Indrė Žliobaitė, Albert Bifet, Bernhard Pfahringer, and Geoffrey Holmes. Active Learning With Drifting Streaming Data. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):27–39, Jan 2014.

[122] Indrė Žliobaitė and Ludmila I. Kuncheva. Determining the Training Window for Small Sample Size Classification with Concept Drift. In *2009 IEEE International Conference on Data Mining Workshops*, pages 447–452. IEEE, Dec 2009.