Fine-Grained, Unsupervised, Context-based Change Detection and Adaptation for Evolving Categorical Data

by

Sarah D'Ettorre

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of Master of Computer Science in

Electrical Engineering and Computer Science
University of Ottawa
Ottawa, Ontario, Canada

September 2016

Abstract

Concept drift detection, the identification of changes in data distributions in streams, is critical to understanding the mechanics of data generating processes and ensuring that data models remain representative through time [2]. Many change detection methods utilize statistical techniques that take numerical data as input. However, many applications produce data streams containing categorical attributes. In this context, numerical statistical methods are unavailable, and different approaches are required. Common solutions use error monitoring, assuming that fluctuations in the error measures of a learning system correspond to concept drift [4]. There has been very little research, though, on context-based concept drift detection in categorical streams. This approach observes changes in the actual data distribution and is less popular due to the challenges associated with categorical data analysis. However, context-based change detection is arguably more informative as it is data-driven, and more widely applicable in that it can function in an unsupervised setting [4].

This study offers a contribution to this gap in the research by proposing a novel context-based change detection and adaptation algorithm for categorical data, namely Fine-Grained Change Detection in Categorical Data Streams (FG-CDCStream). This unsupervised method exploits elements of ensemble learning, a technique whereby decisions are made according to the majority vote of a set of models representing different random subspaces of the data [5]. These ideas are applied to a set of concept drift detector objects and merged with concepts from a recent, state-of-the-art,

context-based change detection algorithm, the so-called Change Detection in Categorical Data Streams (CDCStream) [4]. FG-CDCStream is proposed as an extension of the batch-based CDCStream, providing instance-by-instance analysis and improving its change detection capabilities especially in data streams containing abrupt changes or a combination of abrupt and gradual changes. FG-CDCStream also enhances the adaptation strategy of CDCStream producing more representative post-change models.

Acknowledgments

I would like to thank my thesis supervisor, Dr. Herna Viktor, for her unwavering patience and support through the duration of this study. Her knowledge and guidance are ever appreciated. I would also like to thank my fantastic family and friends for their love and support throughout this adventure.

Many thanks to each of you.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Acronyms and Abbreviations

ADM Adaptive Dissimilarity Matrices

ADWIN Adaptive Windowing

AI Artificial Intelligence

CDCStream Change Detection in Categorical Data Streams

CDDA Concept Drift Detecting Algorithm

CSV Comma Separated Values

CUSUM Commulative Sum

DDM Drift Detection Method

DILCA Distance Learning for Categorical Attributes

EDDM Early Drift Detection Method

FCBF Fast Correlation-Based Filter

FG-CDCStream Fine-Grained Change Detection in Categorical Data Streams

GB Gigabyte

GUI Graphical User Interface

IbK Instance-based K-Nearest Neighbour

IEEE Institute of Electrical and Electronics Engineers

IOF Inverse Occurrence Frequency

IVDM Interpolated Value Difference Metric

KDD99 Knowledge Discovery in Databases, 1999

KNN K-Nearest Neighbour

LED Light Emitting Diode

MDR Missed Detection Rate

MOA Massive Online Analysis

MTA Mean Time between Alarms

MTD Mean Time to Detection

MTFA Mean Time between False Alarms

MTR Mean Time Ratio

MVDM Modified Value Difference Metric

NHST Null-hypothesis Statistical Testing

NMI Normalized Mutual Information

OF Occurrence Frequency

RAHCA Rough Set-Based Agglomeration Hierarchy Clustering Algorithm

RAM Random-Access Memory

ROCK Robust Clustering using Links

SDriftClassif Supervised Drift Detection by Classification

SPC Statistical Process Control

SPRT Sequential Probability Ratio Test

STIRR Sieving Through Iterated Relational Reinforcement

SU Symmetric Uncertainty

TCP Transmission Control Protocol

UCI University of California, Irvine

VBS Visual Basic Scripting Edition

VDM Value Difference Metric

WEKA Waikato Environment for Knowledge Analysis

WVDM Windowed Value Difference Metric

Chapter 1

Introduction

For decades, researchers have been studying and developing techniques to extract knowledge from collections of data using analytical models [1]. Over the years, however, data analytics has had to keep up with the changing trends in data collection. As computerization became ubiquitous, data collection transformed from retaining sets of relatively small amounts of finite homogeneous data to large, potentially infinite and potentially heterogeneous data [8]. This new data format is referred to as a data stream and is characterized by a continuous flow of data, often at high speeds [1] [8]. These data streams commanded analytical models that were able to update as more data became available and to adapt appropriately as underlying concepts in the data evolve over time. Explicit detection of this evolution, known as concept drift, is beneficial in ensuring that data models remain accurate and providing insights into the mechanics of the data generating process [2]. Thus, concept drift detection has been of continuous interest to Machine Learning researchers.

A multitude of change detection and adaptation techniques have been proposed in the literature, each with its own set of strengths and weaknesses. This depends on various parameters, such as the types of data and concept changes it is applied to. For example, some change detection techniques might perform well on binary data but poorly in multi-class scenarios. Others might perform well when changes occur

suddenly, but falter when changes are more gradual. A majority of these techniques utilize statistical methods requiring numerical input [9]. Real world data attributes, however, are often categorical [4]. For example, point of sale data might include categorical attributes like colour {red, green, blue} or size {small, medium, large, x-large}. Environmental data attributes might contain categorical attributes like predators {eagle, owl, fox} or land cover {desert, forest, tundra}. This prevalence of categorical data poses a further challenge for change detection researchers.

This thesis focuses on improving the quality of knowledge extraction from evolving streams of categorical data through the use of change detection and adaptation strategies. Specifically, this thesis extends the ideas of CDCStream, a recent method proposed in [4]. The remainder of this chapter details the specific motivations and defines the precise objectives of our research.

1.1 Motivation

Currently, the majority of research on change detection in categorical data streams is learning-based, thus utilizing error changes in the learning system as an indicator of concept drift [4]. While these techniques have proven to be reasonably successful, it remains that fluctuations in error measures can not be definitively attributed to concept drift alone. This results in a lack of precision, limiting the information available regarding the data generator.

Relatively few studies in the literature have examined context-based change detection in categorical data streams [4]. In this case, concept drift is detected when changes in the actual data distribution are observed, providing more precise information about particular changes. This opens the door to unsupervised change detection [4], a non-trivial task [10] [11] [12]. Since class information is not always available, facilitating unsupervised change detection broadens the spectrum of categorical data

that may be analyzed.

1.2 Objective

This thesis aids to narrow the research gap surrounding context-based change detection for evolving categorical data. It extends a recent state-of-the-art change detection system, CDCStream, to improve its change detection and adaptation capabilities, especially in the abrupt drift scenario where theoretical analysis and preliminary testing suggest CDCStream falters most.

In light of this goal, the FG-CDCStream algorithm was introduced which merges concepts from CDCStream with some elements of ensemble learning to provide both new and improved change detection capabilities. A goal of this new algorithm is to improve the change detection capacity of CDCStream. Specifically, the new algorithm intends to increase Mean Time between False Alarms (MTFA) and Mean Time Ratio (MTR) and reduce Mean Time to Detection (MTD) and Missed Detection Rate (MDR). These statistics will be defined in Section 4.1.1.1. This is done by reducing the grain size of CDCStream, providing instance-by-instance rather than batch-by-batch analysis, in conjunction with the employment of ensemble-type change detection.

Adaptation is improved by tightening the bounds on the amount of historical data a post-change classifier is trained on. This ensures that the classifier is more representative of the post-change concept. Improvements to CDCStream's adaptation can be measured by accuracy-type statistics surrounding change points.

This thesis intends to provide a novel fine-grained, unsupervised and context-based change detection algorithm for categorical data with capabilities previously undocumented in Machine Learning research. This algorithm serves as a contribution to narrowing the research gap that exists in this area.

1.3 Organization

The remainder of this thesis is structured as follows. Chapter Two outlines relevant background information and previous work related to this thesis in order to prime the context for the reader. It emphasizes concept drift detection methods, the challenges and proposed solutions to categorical data analysis and finally details the CDCStream algorithm. Chapter Three discusses the limitations of CDCStream, relating to both change detection and adaptation, and proposes an improved variation, FG-CDCStream. It presents the methods by which FG-CDCStream combats the limitations of CDCStream and discusses the intimate mechanics of the new algorithm. Chapter Four presents and justifies the evaluation strategy and experimental framework designed to meaningfully assess and compare the effectiveness of the two algorithms. Chapter Five presents the results of the proposed experiments and analyzes their significance. Finally, Chapter Six provides a conclusion to this research and provides suggestions for future work.

Chapter 2

Background and Related Work

This chapter outlines relevant background information and previous work pertinent to this thesis. It begins with a brief refresher on machine learning techniques and details how these techniques evolved in the face of societal shifts resulting in the collection of more abundant data and data streams. It introduces the phenomenon of concept drift and the value of explicit concept drift detection and adaptation in modern Machine Learning. In this chapter, the particular challenges of categorical data analysis are discussed and various solutions from the literature are presented. The chapter concludes by detailing the CDCStream algorithm, a context-based change detection algorithm for categorical data, upon which this study is largely based.

2.1 Machine Learning Fundamentals

Machine Learning is a sub-field of Artificial Intelligence (AI) which encapsulates a collection of techniques used to automatically extract knowledge from data [1]. This is accomplished by building analytical models based on data characteristics and using these models to perform knowledge extraction. For example, a function performed might be making predictions about some data and/or learning about relationships between data. Classification, clustering and frequent pattern mining are among the

most prominent methods [13].

Classification is a common task that has been extensively researched. It is a supervised learning method, namely, the instances in the training set have pre-defined nominal class labels. This labelled training set is used to create a model of the relationship between the feature attributes and the class [14]. Once the model is complete, it can be used to predict class labels of an un-labelled test set. Regression modelling is a sub-type of classification that is used when class labels are quantitative rather than nominal. In this case, the regressor uses the relationships between the training attributes to build a model which predicts class labels of the test set [14].

Clustering resembles classification, but operates in an unsupervised scenario where class labels are not present in the data set. The goal of a clustering algorithm is: given a multidimensional data set, determine a partition of the points into clusters [14] such that there is high intra-cluster similarity and low inter-cluster similarity [13]. Researchers have developed numerous methods of measuring similarity as discussed in Section 2.4. Clustering essentially infers the classes from the data characteristics.

Rather than placing focus on similarities among instances, as in classification and clustering, frequent pattern mining seeks out recurring patterns and frequent associations between attributes [13]. Depending on the task at hand and the particular data sets, these patterns can take the form of itemsets, sequences, subtrees or subgraphs [15]. A popular example of frequent pattern mining is association rule mining dealing with itemsets. Rules of itemset associations are computed with various support and confidence levels [13]. This type of knowledge discovery emerged in business applications whereby knowledge of, for example, which items are often purchased together, may be exploited to increase profits [16].

Classical machine learning dealt with small, homogenous, identically distributed and finite data [8]. In this context, the models created to extract information from these data were static; once they were created they were not updatable. Later,

computerization became ubiquitous and larger amounts of data became available for collection. At this stage, data sets remained static. However the sheer volume of data placed new demands on information extraction algorithms and traditional Machine Learning techniques were no longer entirely appropriate. Computation time and memory usage had to be reduced by prioritizing linear time efficiency and using sampling or temporary external storage. Under these constraints, various novel methods for information extraction emerged while some of the classical approaches were adapted.

More recently, another societal shift occurred where large amounts of continuously flowing and potentially infinite data are prevalent. This type of data is known as a data stream and examples include sensor data or search engine data. Data streams present many new challenges to classical Machine Learning algorithms. The properties of data streams and how the field of Machine Learning has adapted to the new challenges posed by them, are discussed in detail in the following section.

2.2 Data Streams

Streaming data are often diverse, transient, and can flow at high speeds [1] [8]. To learn from data streams, models must update incrementally as instances arrive (in less than instance arrival time for high speed streams) and use only a small, fixed amount of memory [2]. Data stream environments are dynamic, temporally ordered and potentially infinite settings where the distribution of the data often changes over time [2]. The phenomenon of data evolution is referred to as concept drift and is one of the major challenges of data stream mining algorithms. In order to remain relevant in the face of evolving data, models must be capable of effectively adapting to represent the current distributions. In this environment, results are approximate and ensuring the quality of approximate results is a challenge in itself. These challenges

are non-trivial and a great deal of research has been conducted to achieve appropriate solutions.

Data stream classification evolved various new paradigms under the requirements posed by data streams. Some algorithms are one-pass adaptations of their classical counterparts. These approaches naturally adapt to drifting concepts by forgetting old information and updating with new information as it arrives [2]. This method, however, is not always satisfactory, especially in the face of abrupt changes. Other methodologies do employ designed adaptation strategies, some accompanied by explicit concept drift detection [17]. Data stream classification is further discussed in section 2.3.3.

The transition to dynamic data clustering was more complicated than that of classification. Obstacles arose due to one-pass constraints and the temporal nature of data streams [17] [18]. Early data stream clustering algorithms, such as [19], consider all of the data in the stream to define the clusters [18]. In some cases, however, this approach is unsuitable. This approach is inappropriate, for example, in cases where the data clusters change substantially through time. It is also ill fit to scenarios where users require the ability to explore the clusters at specific points in time [18]. In consideration of such cases, the authors of [20] designed an approach consisting of both an online and offline phase. The online phase considers all of the data in the stream. As data points arrive, they are either absorbed by an existing cluster or become a new cluster. Micro-clusters are saved on time intervals which are used in the offline phase. In the offline phase, macro-clusters are created on a time interval by considering the micro-cluster state at that time and subtracting the previous micro-cluster states [18] providing snapshots of the data distribution through time.

The mining of frequent patterns in a streaming environment faces its own unique challenges. Given that the algorithms must search a space with an exponential number of patterns where the cardinality of the result can be incredibly large, they are

required to be very memory efficient. Pruning infrequent patterns requires great computational power which is challenging in high-speed streaming scenarios. Numerous methods have been proposed to cope with these issues. Some are based on mining the entire data stream. In this case, the stream is divided into batches from which frequent itemsets are mined. The itemsets are then aggregated and further pruned to represent the entire stream. Another approach is based on a sliding window whereby the boundary between the closed frequent itemsets and the others is monitored. A damped window stores potential itemsets with their counts, reducing their weights as they age [15].

Classification, clustering and frequent pattern mining algorithms adapted in various ways to cope with the new time and memory constraints, as well as the issue of drifting concepts, posed by data streams. Research efforts continue, seeking new methods and improving current methods of extracting knowledge from data streams effectively and efficiently.

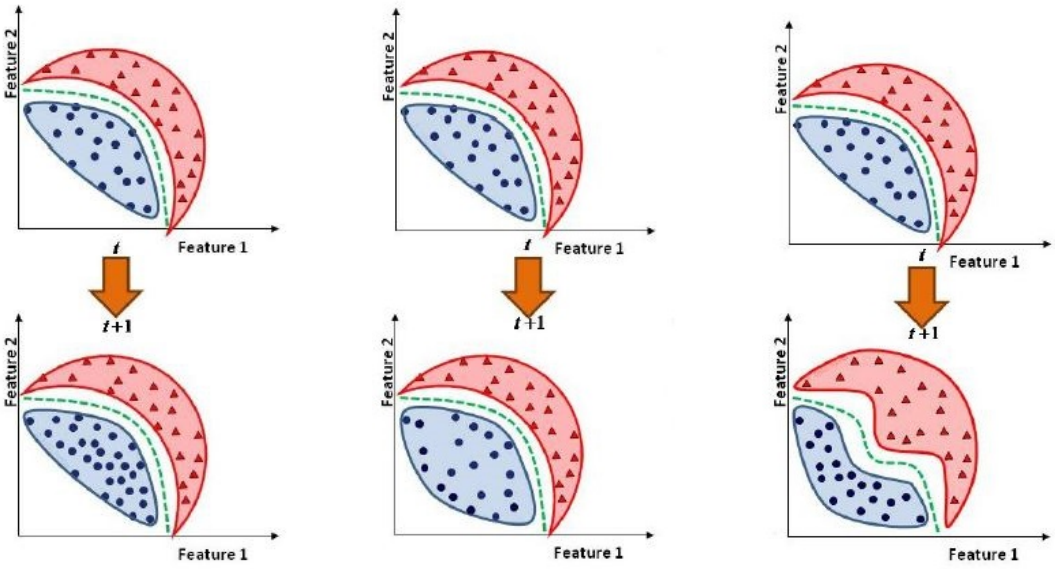
2.3 Evolving Concepts

As discussed above, a major obstacle associated with knowledge discovery in data streams is the fact that the concepts in those streams may drift and change over time. The most time and memory efficient algorithms are futile if they have no ability to adapt. This section details concept drift and the methods used to of manage it.

2.3.1 Concept Drift Preliminaries

Concept drift is defined as change in data distribution in a stream over time [2]. In [21], the authors defined three types of concept drift based on Bayes' theorem:

1. Change in the probabilities of class priors



(a) Change in class priors (b) Change in class distrib. (c) Change in class boundary

Figure 1: Illustration of the three types of concept drift based on the Bayesian theorem, as shown in [6]

2. Change in the distributions of the classes
3. Posterior probabilities of class membership

Each of these subsets is illustrated in Figure 1 [6]. In Figure 1a, the circle class occurs more frequently after the change demonstrating and increase in circle class prior probability. Figure 1b shows a change in the distribution of the circle class while the class boundary remains the same. Finally, Figure 1c shows a drastic change in the boundary between the circle and triangle classes.

Concept drift is often further classified into two types: virtual concept drift and real concept drift [6]. Virtual concept drift is characterized by changes in the distribution of instances. It corresponds to the first two types discussed above where the class boundary (the underlying concept) remains stationary while changes in class priors of class distribution occur. Virtual concept drift may cause issues for a trained learner if, for example, instances that become more prevalent after a change contain elements of the concept that the learner originally missed. Therefore, the model must

(a) Abrupt (b) Incremental (c) Dispersed (d) Recurring

Figure 2: Concept drift patterns over time [2]

adapt to retain accuracy.

Real concept drift coincides with the third type of drift, as discussed above. It is characterized by a change in the posterior distribution or the class boundary and implies a more elemental change [6]. This type of concept drift significantly impacts the relevance of the model and thus adaptation is required in this scenario.

Concept drift is also classified based on how changes occur over time: abruptly or gradually. Figure 2 [2] illustrates these patterns. Abrupt change, as shown in Figure 2a, occurs when concepts change from one to another over a very short time period. Gradual change occurs more slowly and less radically. Gradual drift can be incremental (Figure 2b) whereby there are many intermediate concepts in between the extremes [2], or more dispersed (Figure 2c) whereby the new concept appears in increasingly more instances. It is also possible for previous concepts to reoccur through time (Figure 2d). For example, seasonal concepts might reoccur each year, but not necessarily at exactly the same time.

It is important to note that not all changes in data distribution necessarily indicate concept drift. Some objects can have behaviours that vary significantly from the expectation; they are called outliers or anomalies [13]. This noise occurs when random error or variance appears in a feature [13]. When noise is present in a data stream, it may cause changes in the data mean that could be mistaken for concept drift. Differentiating between noise and concept drift, and adapting appropriately, is an ongoing challenge in concept drift research.

Concept drift, regardless of the type, creates the requirement that models adapt in order to remain relevant. Different adaptation strategies, however, can be better suited to different types of concept drift. The following section discusses the basic elements involved in model adaptation: remembering and forgetting.

2.3.2 Remembering and Forgetting

In order to account for concept drift, data stream models must be updatable with relevant information and forget stale information. For many applications, recent data are the most informative relative to the current prediction, and should therefore be considered by the current version of the model. Older data, which presumably contain outdated concepts, are often forgotten [2].

There are two general methods of managing new data from a stream, either analyzing one instance at a time or multiple recent instances at a time. With the former, the learner examines a single example at a time as examples arrive and has no access to previous examples. Most such algorithms do not employ forgetting mechanisms so adaptation occurs slowly as old concepts become outnumbered by new ones. While adequate in a gradual drift scenario, this type of adaptation does not respond quickly enough to abrupt drift [2]. The latter approach trains a learner from a buffer of most recent examples. Most often, this buffer is in the form of a sliding window. Sliding windows of fixed size gain the most recent instance and drop the oldest instance as each instance arrives. The model updates and forgets to remain consistent with the sliding window. Window size is largely responsible for the efficiency of the model's responses to gradual and abrupt concept drift. Small windows allow the model to respond faster to abrupt drift, but hinder performance in periods of stability. Large windows respond slowly to drift and perform well in periods of stability. Thus, there is always a trade off when using a fixed size window. Using variable sized windows, however, can yield good performance in periods of both change and stability. In this

scenario, window size is shrunk during periods of change and grown during periods of stability [2] [22].

When a window shrinks, the model forgets the information that the window shed. In this context, forgetting is abrupt. When an instance or set of instances is dropped from the window frame it will never again be considered by the model. Abrupt forgetting can also occur when, rather than dropping the oldest data sequentially, a sample of the oldest data is dropped [2].

With gradual forgetting, no instances are completely forgotten. Aggregated statistics are stored in memory such that each time an instance arrives, the fading factor of earlier statistics increases, decreasing their influence on the model. Both linear and exponential gradual forgetting have been proposed [2] [23].

This section outlined the basics of model adaptation: remembering and forgetting. The various methods used to carry out said remembering and forgetting were presented. The intricacies of how these processes contribute to an adaptive learning system are discussed in the following section.

2.3.3 Learning and Adaptation Strategies

In order for models to adapt to continuously evolving data streams, models in the data stream context must be updated. Two schools of model update methods have emerged in the literature: retraining and incremental [2]. Retraining algorithms correspond to a batch scenario whereby some buffer of the data is held in memory [24]. These algorithms will train a learner on a batch, then wait for the next batch to arrive. The data are merged and an entirely new model is learned from scratch on the merged data. If explicit change detection is employed, a new model might be learned on only a batch of recent data that reflects the current concept. In this case, the adaptation following the change detection is referred to as informed [25]. Informed model adaptation can also occur based on triggers from data descriptors.

Contrarily, blind adaptation strategies do not use triggers. In the context of retraining algorithms, blind adaptation would periodically retrain on recent windows [2].

Incremental learning algorithms update the model upon the arrival of each instance. They may update blindly by periodical abrupt forgetting or gradual forgetting. If informed adaptation is used, the algorithm may forget the old concept using global or local replacement. Global replacement involves discarding the model and constructing a new incremental learner from scratch. With some learners, like Naive Bayes, global replacement is the only option. Local replacement is available to granular models like decision trees. In this case, irrelevant subsections of the model may be discarded and replaced with subsections trained on recent data [2] [24].

Inactive versions of a model may also be stored rather than completely discarded if concept re-occurrence is expected in the stream. With a repository of models tailored to certain concepts, the algorithm may select the most appropriate model for the incoming data without having to retrain. This may be done in either single model or ensemble scenarios [2].

With ensemble learning, a collection of models is used to make a combined decision [5]. The idea of ensemble learning is that each model, trained on a different random subspace, represents a slightly different distribution and with a large enough ensemble, the combined prediction converges to a reasonably accurate result [5]. This is based on the Strong Law of Large Numbers [26]. Ensembles may adapt in various ways. A weight may be placed on each model which updates dynamically to represent the suitability of the model to the current concept. This allows more relevant models to have more of an affect on the overall decision [2]. Another means by which ensembles adapt is by systematically discarding a number of models (typically the worst performing or the lowest weighted) and replacing them with new models [2]. The proportion of models to replace may be adjusted to reflect gradual or abrupt forgetting to varying degrees [27].

2.3.4 Concept Drift Detection Methods

Although explicit concept drift detection is not necessary for algorithms to adapt to drifting concepts, as they often do naturally by continually updating and forgetting, it does afford several advantages. For example, if concept drift occurs abruptly, the model can detect and adapt to it more quickly. Concept drift detection also provides insight into the mechanics of the generating process. Various schools of change detection have emerged. They include the following non-mutually exclusive methods: sequential analysis, control charts, monitoring differences in two distributions and heuristic methods [2].

Sequential analysis, based on Sequential Probability Ratio Test (SPRT) [28], aims to minimize detection delay time while maintaining an acceptable false positive rate [29]. It examines the data stream instance by instance as examples arrive. A change point occurs when the probability of observing certain subsequences under the historical distribution compared to that of the recent distribution is beyond a specified threshold [2]. Commulative Sum (CUSUM) based approaches [30] to concept drift detection are examples of sequential analysis. CUSUM monitors the mean of the data stream and reports a change when it significantly deviates from zero [2] based on an adaptive threshold [3].

Statistical Process Control (SPC) techniques monitor the error rate of the model and update the model when the error rate exceeds a threshold. For each instance in the sequence, the error rate is calculated based on the probability of incorrect prediction with a certain standard deviation [2]. The change detector describes the system in three states:

1. ***In Control.*** In this state the distribution is stable. The model is not updated.
2. ***Warning.*** The error is increasing, but more information is required to confirm if this increase is caused by concept drift or noise. No model update is required.

3. ***Out of Control.*** In this state, the error has increased significantly enough for the cause to be deemed concept drift. The model must be updated.

Interestingly, SPC inherently measures the rate of change of the system as time time between warning and out of control states [2].

Popular examples of the SPC approach include Drift Detection Method (DDM) [24] and Early Drift Detection Method (EDDM) [31]. DDM monitors the classification error rates of the system. Two error thresholds are considered: one corresponding to the warning state and the other to the out of control or confirmed drift state. When the drift state is reached, the model adapts using only instances that arrived after the warning state was declared. While DDM performs well in the face of abrupt change, it falters when tasked with detecting gradual change. EDDM succeeds in outperforming DDM in gradual change detection by considering not only the error rate but also average distances between errors and standard deviation.

Change detection techniques that monitor two different distributions typically employ a reference window containing older information and a detection window storing more current instances. Windows can monitor raw data or model parameters. They can have equal or unequal, static or dynamic sizes [2]. It isn't necessary that detection and training windows be identical. Null hypothesis statistical tests are used to determine a change in distribution where N_0 states that the two distributions are equal. A change is flagged when the null hypothesis is rejected [2]. Statistical tests that have been used in the literature include accuracy measures [32] [33], entropy measures [34] and concentration inequalities such as Chebychev's inequality [4], Chernoff's bounds [35], the Hoeffding bound [25] and the Bernstein inequality [3].

In [32] concept drift detection is accomplished by using a pair of stable and reactive learners. The reactive learner classifies based on a short time window and the stable learner predicts based on a potentially long history. If the reactive learner correctly classifies instances that the stable learner incorrectly classifies enough times (based on

a threshold) within a certain time frame, the stable learner is replaced with another stable learner with an initial concept description identical to that of the reactive learner. The reactive learner is used only for concept drift detection and never for making actual predictions for the system. Similarly in [33] the accuracy estimated over the entire stream and the accuracy over a short recent window are monitored and concept drift is indicated by a significant decrease in accuracy over the recent window [2].

Using entropy in conjunction with sliding windows is a computationally inexpensive, reasonably noise tolerant and fast-reacting method of detecting concept drift [34]. The metric proposed in [34] measures the differences in distributions between historical and current sliding windows. The entropy measure is bounded by [0,1] with 0 representing entirely different distributions and 1 representing exactly equal distributions. If this value falls below a threshold, concept drift is signalled.

Concentration inequalities have proved extremely useful in the sliding window setting. Concentration inequalities give error bounds with associated probabilities on how a random variable deviates from its expected value. Thus, concentration inequalities can be used to determine whether or not the distributions of two windows are different enough to flag a concept shift. In Machine Learning, various concentration inequalities have been used in this manner [36] [3].

Chebychev's inequality is used to indicate the proportion of values in a set that are close to the mean. It is incredibly versatile as it can be used on streams of positive random variables, independent or dependant, for most distributions [4]. The trade-off, however, is that the bound is fairly coarse [37] which can increase the reaction delay in change detection. The incremental nature of this inequality also provides an opportunity to borrow techniques from SPC. In [4], sliding windows were analyzed using Chebychev's inequality to determine if historical and current distributions did not significantly differ (in control), differed somewhat (warning) or

differed significantly (out of control). In this thesis, we use Chebychev's inequality extensively, as will be detailed in Section 2.6.

Tighter bounds than those of Chebychev's inequality can be obtained when the data distribution is known [37]. The Chernoff inequalities consider several popular data distributions in order to achieve this [38]. Generally the context is a zero mean set of independent Bernouli random variables[1]. The version of the Chernoff bound used depends upon the data distribution, for example Gaussian or Poisson. In [35], a normal distribution is assumed and a corresponding Chernoff bound is used in conjunction with a balanced Kolmogorov Smirnoff tree to compare the distributions of the rear (historical) and front (current) windows.

The Hoeffding bound is a generalization of the Chernoff bounds. Rather than being restricted to Bernouli random variables, the Hoeffding inequality applies to a set of bounded independent random variables [37]. The original version of Adaptive Windowing (ADWIN), as proposed in [25], continually adds new instances with values bounded by [0,1] to the encompassing window until change is detected. The change detection module uses the Hoeffding bound to determine when two subwindows are both large enough and distinct enough from each other. This decision is based on a threshold computed using the harmonic mean of the subwindows and a user defined confidence threshold. Once change is flagged, the oldest instance is dropped and the process repeats. This allows the window to grow in times of stability and shrink in times of change in support of model accuracy, though it is computationally expensive. ADWIN is also attractive in that its use of the Hoeffding bound allows it to provide rigorous guarantees on false positive and false negative rates.

Bernstein's inequality also provides the tools for such rigorous guarantees and can provide a much tighter bound (providing shorter detection delay) than the Hoeffding inequality. This is achieved by considering population variance. Since population

[1] variables with exactly two possible values

variance is generally unknown in real world data streams, change detectors often compensate by using sample variance which has had good empirical results [3]. In [3], reservoir sampling is used to construct a windowed sequential change detection model which uses Bernstein's inequality for change detection. Results were comparable to ADWIN with minimized computational complexity.

Heuristic methods may be used for concept drift detection as well. For example, in [39] the Splice algorithm uses the time stamp of instances as a feature. Concepts are detected in two stages. In the first stage a decision tree is trained on a batch of instances. If splits occur on the time-stamp attribute, the resulting sub-trees represent different contexts. The second phase trains a decision tree on each of the partitions to determine temporal concepts [2].

Many concept drift detection methods have been proposed in the literature. A great number of these approaches rely on statistical analysis techniques requiring features to be numerical. Categorical data introduce further change detection challenges.

2.4 Categorical Data Analysis

Categorical variables are abundant in modern real world data. For example, they occur frequently in social media [40] behavioural sciences, epidemiology, ecology, genetics, botany, marketing and a plethora of other fields [41]. With the recent explosion of data collection, analyzing these data has become an important topic of interest.

A categorical variable is one that may take a single discrete value out of a limited set of possible values [12] [41]. An example of a categorical variable is animal {dog, cat, rabbit, horse}. In this case, the variable "animal" may take one of the four values indicated. In this example, the "animal" attribute has no inherent ordering; it is nominal. If the attribute does have ordered categories, like size {small, medium, large}, it is called ordinal. Note that, even though an order is present in ordinal

values, a distance between the values remains unknown [41]. There is no quantity given to represent how much larger large is from small. Categorical variables may also be classified as interval variables, for example age {0-2, 3-5, 6-13, 14-20, 21-30, 31-50, 50+}. Distances between two values of an interval variable do have numerical values. When continuous variables are discretized, the result is often an interval variable.

Many Machine Learning tasks, like classification and clustering, involve computing distances between two objects [12]. When variables are continuous, this task is reasonably straightforward using distance measures such as Euclidean, Manhattan or Mahalanobis distances [11]. In the case of categorical variables, especially nominal ones that are most lacking in structure, methods of computing distances are more complex. This has lead to a large and diverse collection of proposed distance measures spanning various fields of study [41], many of which arising in the context of categorical data clustering. Similarity measures can be context-free or context-sensitive, supervised or unsupervised [12]. Context-free measures do not depend on relationships between instances while context-sensitive measures do. Supervised methods use the class attribute to determine distance measures whereas un-supervised methods do not. Based on [10], [11] and [12], for this study, distance measuring techniques for categorical values will be classified into six groups that are not necessarily mutually exclusive: simple matching approaches, frequency-based approaches, learning approaches, co-occurrence approaches, probabilistic approaches, information theoretic approaches and rough set theory approaches.

Simple matching approaches involve assigning a distance of zero (or similarity of one) when two categorical values are identical and a distance of one (or similarity of zero) otherwise [10]. The simplest similarity measure for categorical data, the overlap metric, considers multivariate data points assigning a value of one where attribute values are identical and zero where they are not [11]. The distance between two data points is described by the sum of assigned values for each attribute. The downfall

of this method is that it assumes all attribute values are equidistant and treats all features with equal importance. The Jaccard coefficient [42] is a normalized version of the overlap metric. The authors of [43] proposed an improved, supervised, non-learning version of the overlap metric, Value Difference Metric (VDM). This method gives higher weights to attribute values that have more influence on determining the class [12]. VDM was later modified in several different ways. Modified Value Difference Metric (MVDM) [44] revised VDM to produce a symmetric distance. Interpolated Value Difference Metric (IVDM) [45] discretizes continuous values and interpolates probabilities from the midpoints of each interval whereas Windowed Value Difference Metric (WVDM) [45] interpolates probabilities from adjacent values [12]. Various clustering methods employ simple matching methods including the k-modes algorithm and its derivatives [46], [47], [48], [49], [50], [51], [52]. K-modes is an extension of the k-means algorithm for categorical data. Each cluster is effectively represented by a single instance whose attributes take the most frequent values of those in the cluster. Similarity is calculated using the overlap method [53]. The resulting clusters, however, often have weak intrasimilarity, which can be improved by also considering the frequency of mode components in a cluster. Simple matching methods are attractive for their simplicity, but do not consider any degree of difference beyond the binary. All matches and all mismatches are treated equally [11].

Frequency-based unsupervised methodologies are based on frequency distributions of attribute values [11]. The Eskin [54] measure takes into account the cardinality of the attributes and assigns greater similarity to matches that occur on attributes that take many values. The Occurrence Frequency (OF) method [55] assigns higher weight to mismatches on frequent feature values. Inverse Occurrence Frequency (IOF) [55] does the opposite, giving lower weight to mismatches on frequent values. Interestingly, if one performs well on a particular data set the other performs poorly and vice versa [11]. Frequency-based similarity measures use more of the information in the

data than simple matching measures, but they are based on heuristics that may not necessarily be fitting to a particular data set (as evident in the complimentary results of OF and IOF).

Learning methods use learning techniques and update based on some change in a monitored parameter, such as error. The authors of [56] present Adaptive Dissimilarity Matrices (ADM), a supervised, learning approach to dissimilarity computation. This method computes real-valued pairwise distances between attribute values of a training set. The training set is split into two parts, the first used to build the classifier and the second to minimize the classifier's error based on the dissimilarity matrices. The benefit to learning the dissimilarities together is that any feature correlation will be accounted for. Another supervised dissimilarity learning approach [57] is quite similar to ADM. This method maps each categorical feature to a corresponding continuous real-valued feature. On each iteration of the algorithm, the assignment of categorical attribute values to their real valued counterparts is updated based on the classification error of a nearest neighbour learner. Learning methods learn the relationships between categorical values by making assumptions about their effect on error. Although these assumptions may not always be entirely accurate, learning methods still tend to outperform many other similarity measures [12].

Co-occurrence techniques assume that similarity is based on two categorical values' co-occurrence with a common value or set of values [10]. For example, Robust Clustering using Links (ROCK) [58] assigns a link to common neighbours between two patterns. Patterns with large numbers of links are grouped together. Other co-occurrence approaches include CACTUS [59] which views similarity with respect to the frequency of two values appearing in patterns together. Sieving Through Iterated Relational Reinforcement (STIRR) [60] constructs a graph where each attribute value is a weighted node (initially arbitrary and small) which propagates to items with which it co-occurs frequently. The result is that items with common co-occurring

items have similar weights. Similarly CLICKS [61] partitions graphs/hypergraphs by strongly connected components: pairs of feature values with co-occurrence surpassing a defined threshold [53]. A strong advantage of co-occurrence approaches is that they are data driven.

Probabilistic methods consider the probabilities of certain matches occurring. Goodall [62] suggests a probabilistic approach whereby higher similarity is assigned if the value is infrequent while taking inter-attribute dependencies into account [11]. In other words, if the probability of observing the match in a random sample is low, that match is considered more informative than one with high probability. Smirnov [63] similarity gives more weight to matches where the frequency of the matching value is low and the other values occur frequently [11]. Anderberg [64] also gives greater weight to rare matches but also penalizes rare mismatches. In [65], a context-based probabilistic method is presented where the dissimilarity between two attribute values is a result of a combination of dissimilarities between conditional probability distributions of different attributes given the two values in question. This approach is unsuitable for data sets with independent attributes. In the clustering context, probabilistic approaches consider conditional probability to define distances between groups or clusters [10]. For example COBWEB [66] uses the category utility measure to measure the usefulness of class in predicting attribute values. AUTOCLASS [67] uses a Bayesian approach to deduce the more probably class distribution.

Information-theoretic approaches are typically based on entropy and entropy derivatives. Gambaryan [68] uses entropy to give greater weight to matches that are in between frequent and rare. The Lin measure [69] is an unsupervised measure that assigns higher similarity to matches of frequent values and lower similarity to mismatches on infrequent values. Burnaby [70] does the same using information weighted measures. The context-based Distance Learning for Categorical Attributes (DILCA)

methodology presented in [53] is made up of two steps. In the first step, the context i.e. a set of attributes highly related to the target attribute, is selected using the Symmetric Uncertainty (SU) measure [71] derived from entropy. If the context attributes are similarly distributed in the instances containing both attribute values in question, a low distance is assigned and vice versa. This approach is discussed in detail in Section 2.5. The method proposed in [72] similarly contains context selection and distance calculation steps, but uses different methods for each. Rather than using the SU measure, the Dependency Score (also derived from entropy) is used to select the context. In the second step, the KL divergence method is used compute the distance measure. COOLCAT [73] attempts to minimize entropy within clusters. Initial clusters are created from a bootstrap sample and the remaining instances are added incrementally making this approach heavily dependent on the order in which points are selected [53]. The authors of [74] resolve this by randomly swapping points between clusters and only preserving swaps that reduce entropy. LIMBO [75] uses the information bottleneck framework to summarize the data in k clusters and then assigns data points to clusters with similar statistics. [40] uses random matrix theory to compute similarities between categorical values. A similarity matrix (similar to the Jaccard method [42]) is compared to random matrix results to determine the presence of non-random correlations. A strength of information-theoretic approaches is that they utilize the information content of a variable with respect to the data set and are thus highly data driven. However, these approaches must often employ heuristics which may be inappropriate for certain data sets [11].

Rough set theory is a non statistical method concerned with the analysis of imprecise, uncertain or incomplete information [76] and is thus well suited to categorical data clustering. For example, Rough Set-Based Agglomeration Hierarchy Clustering Algorithm (RAHCA) [77] uses rough set theory to present a similarity measure based

on Euclidean distance. Cao [78] presented a rough-membership approach to categorical streams using a sliding window. Similarity measures based on rough set theory have proven quite competitive, however they do require a threshold parameter that is difficult to properly set [4].

An abundance of approaches have been proposed in order to manage the unique challenges posed by categorical data due to its inherent uncertainty regarding similarity measures between attribute values. Given the volume of categorical data collected on a daily basis, many categorical data analysis techniques have been proposed with the key task of defining a distance measure for feature values. These methods range from simple to complex and span various fields of study and are based upon a multitude of different frameworks. Continued research and development of categorical data analysis techniques is paramount moving forward in the field of Machine Learning.

2.5 DILCA

The previous section briefly discussed DILCA [53], an information-theoretic similarity measure for categorical attribute values. This section discusses DILCA in more intricate detail, as understanding its mechanics is elemental to this thesis. DILCA is a recent state-of-the-art similarity measure that is purely context-based which makes no assumptions about data distribution and does not depend on heuristics to determine inter-attribute relationships. These properties make DILCA attractive as it minimizes bias and can be applied to a great range of data sets with varying characteristics.

The DILCA similarity measure is computed in two steps, namely context selection (Algorithm 1) and distance computation (Algorithm 2). Note that attribute values can have different degrees of similarity in different contexts. For example, the article [53] presents a sample data set $Person$ with two categorical attributes: $City\{Milan, Turin, Florence\}$ and $Gender\{Male, Female\}$. Figure 3 displays the

Gender	City
M	Turin
F	Florence
M	Turin
M	Florence
F	Milan

(a) Person Dataset

	Turin	Florence	Milan
F	0	1	1
M	2	1	0

(b) Contingency Table

Figure 3: The *Person* data set and its corresponding contingency table

Algorithm 1 DILCA Context Selection

Require: $matrixSU$
Require: Y
1: $VectorSYU_Y = MatrixSU[Y]$
2: $context(Y) = \{X \in F | X \neq Y\}$
3: //Relevance step
4: sort $VectorSU_Y$ in descending order
5: //Redundancy step
6: **for all** $VectorSU_Y[X]$ starting from the top position and s.t. $X \neq Y$ **do**
7: $\quad VectorSU_X = MatrixSU[X]$
8: $\quad$ **for all** $VectorSU_Y[K]$ s.t. $VectorSU_Y[K] \leq VectorSU_Y[X]$ **do**
9: $\quad\quad$ **if** $(VectorSU_X[K] \geq VectorSU_Y[K]$ **then**
10: $\quad\quad\quad$ erase attribute K from $context(Y)$
11: $\quad\quad\quad$ erase $VectorSU_Y[K]$)
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: **end for**
15: $DistMatrix_Y$ = DistanceComputation(Y, $context(Y)$)
16: **return** $DistMatrix_Y$

Algorithm 2 DILCA Distance Computation

Require: $context(Y)$
Require: Y
1: **for all** $y_i, y_j \in Y | y_i \neq y_j$ **do**
2: $\quad DistanceMatrix[y_i][y_j] = \sqrt{\frac{\Sigma_{X \in context(Y)} \Sigma_{x_k \in X} (P(y_i|x_k) - P(y_j|x_k))^2}{\Sigma_{X \in context(Y)} |X|}}$
3: **end for**
4: **return** $DistMatrix_Y$

data set[2] and its corresponding contingency table. Assume $City$ is the target attribute (i.e. the attribute for which the distance between values is to be measured currently). Notice that $City = Milan$ only occurs with $Gender = F$ and $City = Turin$ occurs only with $Gender = M$ while $City = Florence$ occurs with both $Gender = F$ and $Gender = M$. It can be inferred from the data distribution that $Milan$ is more similar to $Florence$ than $Turin$ since the probability of observing a person of a given gender is similar. These similarities are as such in the context of this data set. However, in a different context, for example a geographical one, $Milan$ would be more similar to $Turin$ than $Florence$. It is clear that selecting a proper context is imperative for computing a good similarity measure.

Informative context selection is non-trivial, especially for data sets with many attributes. Consider the following set of m categorical attributes: $F = \{X_1, X_2, ..., X_m\}$. The context of the target attribute Y is defined as a subset of the remaining attributes: $context(Y) \subseteq F/Y$. DILCA uses SU [71] to select a relevant, non-redundant set of attributes which are correlated to the target attribute as the context. SU is a feature selection method derived from entropy. Entropy $H(Y)$ measures the disorder of a random variable Y and is defined by Equation 2.5.1 where $P(y_i)$ is the probability of the i^th value of Y.

$$H(Y) = -\sum_i P(y_i) log_2(P(y_i)) \tag{2.5.1}$$

The conditional entropy of Y when the values of X are known is defined in Equation 2.5.2 where $P(y_i|x_i)$ is the probability that $Y = y_i$ knowing that $X = x_i$

$$H(Y|X) = -\sum_j P(x_j) \sum_i P(y_i|x_i) log_2(P(y_i|x_i)) \tag{2.5.2}$$

Information Gain [79] is a commonly used method of measuring feature correlation. A

[2]This data set has been altered slightly from the one presented in [53] for this example

higher Information Gain value of the target attribute with respect to another indicates a stronger correlation. Information Gain constitutes the information that X provides about Y and is defined in Equation 2.5.3.

$$IG(Y|X) = H(Y) - H(Y|X) \tag{2.5.3}$$

Finally, SU, as defined in Equation 2.5.4, like Information Gain determines the correlation between two attributes but, unlike Information Gain, is not biased towards features of greater cardinality and is normalized on [0,1].

$$SU(Y, X) = 2 * \frac{IG(Y|X)}{H(Y) + H(X)} \tag{2.5.4}$$

DILCA selects a set of attributes with high SU with respect to the target Y. Determining a satisfactory number of attributes for the context is yet another challenge. The authors offer a parametric and an automatic approach. Only the automatic approach will be discussed. The automatic approach is based on the Fast Correlation-Based Filter (FCBF) [71], feature reduction technique which identifies relevant attributes and any redundancy among them without pairwise analysis. DILCA ranks relevance according to SU where features with higher SU with respect to the target attribute are more relevant. Once relevance is ranked, redundant features are removed. A feature X_j is redundant with respect to X_i if X_i is more relevant with respect to Y than X_j and the relevance of X_i with respect to X_j is higher than the relevance of X_j with respect to Y. Following the removal of redundant features, a context containing only relevant and non-redundant attributes remains. See Definition 1 for a concise definition of the resulting context from the context selection step.

Definition 1 *Context(Y) refers to a set of relevant and non-redundant attributes which are highly correlated to the target attribute Y.*

Once the context has been extracted, the distances between attribute values of the target attribute may be computed using Equation 2.5.5, an application of the Euclidean distance. The advantage of using Euclidean distance is that this definition of distance is a metric bounded, in this case, by [0,1].

$$d(y_i, y_j) = \sqrt{\frac{\Sigma_{X \in context(Y)} \Sigma_{x_k \in X} (P(y_i|x_k) - P(y_j|x_k))^2}{\Sigma_{X \in context(Y)} |X|}} \tag{2.5.5}$$

For each value of each context attribute, the Euclidean distance between the conditional probabilities for both values of Y is computed and summed. This value is then normalized by the total number of values in X. The pairwise distances computation between each of the values of Y results in a symmetric and hollow (where diagonal entries are all equal to 0) $|Y|$ x $|Y|$ matrix. The DILCA algorithm discussed is visualized in Algorithms 1 and 2.

The authors of [53] tested DILCA in a hierarchical clustering scenario. It was compared to other various similarity measures of different types (probabilistic, information-theoretic, co-occurrence, rough set theoretic and simple matching) in terms of purity and Normalized Mutual Information (NMI). Purity is a measure of the proportion of data points that were clustered correctly based on actual class values. NMI measures the average mutual information shared between a pair of clusters. Compared with the other similarity measures, DILCA either outperformed them in terms of purity and NMI or performed similarly. In cases where DILCA outperformed other similarity measures, the differences in performance indicators were high. In cases where the other measures outperformed DILCA, performance indexes were close.

This section detailed the characteristics and computation of the DILCA similarity measure. The authors of DILCA noted that its effective quantification of distances

between categorical attribute values made it an attractive component to change detection in categorical data [53]. A subsequent study proposed the CDCStream algorithm for change detection and adaptation in categorical data streams which utilizes DILCA as its similarity measure. The CDCStream algorithm is essential to this thesis, which proposes an improved variation of it. CDCStream is detailed in the following section.

2.6 CDCStream

In [4] change detection in categorical streams based on data distribution is explored. This type of change detection has not been studied extensively, contrary to the typical approach for categorical data which is learning-based, detecting change based on error monitoring. The authors propose a minimal parameter, batch incremental algorithm called CDCStream, which compresses and numerically summarizes batches of categorical data and applies a concentration inequality to both issue warnings and detect changes, as in SPC. The full algorithm pseudocode is shown in Algorithm 3.

Consider an infinite data stream S whereby each attribute X is categorical. A buffer is used to segment the stream into batches: $S = \{S_1, S_2, ..., S_n, ...\}$. If class information is present in the stream, it is removed during segmentation creating an unsupervised change detection context.

The DILCA method described in Section 2.5 is used to compute the distance models of the attributes of each batch. The set of matrices M produced by DILCA is aggregated to numerically summarize the data distribution of each batch in a single statistic using Equation 2.6.1. The resulting statistic, in [0,1], represents both intra- and inter-attribute distributions of a batch.

$$extractSummary(M) = \frac{\Sigma_{M_l \in M} \frac{2 * \sqrt{\Sigma_{i=0}^{|X_l|} \Sigma_{j=i+1}^{|X_l|} M_{X_l}(i,j)^2}}{|X_l| * (|X_l| - 1)}}{|F|} \tag{2.6.1}$$

Algorithm 3 CDCStream

Require: S: stream of instances
Require: k_c: static parameter to detect a change
Require: k_w: static parameter to detect a warning ($k_w < k_c$)

$\sigma_{MAX} = \sigma_{MIN} =$ nil
$L = \emptyset$
while $hasMoreInstances(S)$ **do**
    $S_i = extractNextBatch(S)$
    //the summary statistic is extracted
    $x = extractSummaryStatistic(S_i)$
    $\sigma = \mu = nil$
    if $(L.size > 1)$ **then**
        $\mu = avg(L)$
        $\sigma = stdev(L)$
        if $(\sigma_{MAX} == nil$ or $\sigma > \sigma_{MAX})$ **then**
            $\sigma_{MAX} = \sigma$
        end if
        if $(\sigma_{MIN} == nil$ or $\sigma < \sigma_{MIN})$ **then**
            $\sigma_{MIN} = \sigma$
        end if
    end if
    if $(L.size == 1$ and $\sigma_{MAX} \neq nil$ and $\sigma_{MIN} \neq nil)$ **then**
        $\mu = avg(L)$
        $\sigma = \frac{\sigma_{MAX} + \sigma_{MIN}}{2}$
    end if
    if $(\sigma \neq nil$ and $|x - \mu| \geq k_c\sigma)$ **then**
        //CHANGE HAPPENS
        $L = \emptyset$
    else if $(\sigma \neq nil$ and $|x - \mu| \geq k_w\sigma)$ **then**
        //WARNING HAPPENS
    end if
    $L.add(x)$
end while

Data are managed using a dynamic sliding window method. The historical window L consists of all batch summaries, except the most recent batch which constitutes the current window. The dynamic window grows and shrinks appropriately based on the change status, by forgetting the historical window and/or absorbing the most recent batch. In periods of stability, the historical window continues to grow summary by summary. When a change is detected, abrupt forgetting is employed and the historical window is dropped and replaced with the current window.

In this unsupervised and context-based scenario, a two-tailed statistical test that does generally[3] not assume a data distribution is required for change detection. Chebychev's inequality, meeting both of the requirements, is applied to detect changes in distribution. Chebychev's Inequality states that if X is a random variable with mean μ_X and standard deviation σ_X, for any positive real number k:

$$Pr(|X - \mu_X| \geq k\sigma_X) \leq \frac{1}{k^2} \tag{2.6.2}$$

Namely, $\frac{1}{k^2}$ is the maximum number of the values that may be beyond k standard deviations from the mean. In this study, values of k representing warning and change thresholds were empirically determined as two and three respectively. That is, in order for a warning to be flagged, at least 75% of the values in the distribution must be within two standard deviations of the mean. For a change to be flagged, at least 88.89 % of the values must be within three standard deviations of the mean.

The adaptation method employs global model replacement. When a warning is detected, a new background classifier is created and updated alongside the current working model. Once a change is detected, the current model is entirely replaced with the background model.

CDCStream was compared with two other change detectors: an unsupervised

[3]Some extreme distributions, such as the Lévy distribution, are incompatible

method Concept Drift Detecting Algorithm (CDDA) [80] and a supervised method named Supervised Drift Detection by Classification (SDriftClassif) [24]. CDDA uses a rough-membership approach rooted in rough set theory in conjunction with speed and degree of change parameters to determine if a concept has changed between two time points. The algorithm was adjusted to flag both warnings and changes for the purpose of comparison. It was selected for comparison as it was a recent and successful unsupervised approach. SDriftClassif is a learning approach that monitors the error of a classifier and flags warnings and changes when that error passes the respective thresholds. Comparing CDCStream to SDriftClassif provides an understanding of how CDCStream compares to successful supervised approaches. In all experiments, the Naive Bayes classifier was used as a base learner. The performance measure used for comparison was classification accuracy determined by a prequential evaluator. Four popular real-world data sets were used, Electricity, Forest, Airlines and Knowledge Discovery in Databases, 1999 (KDD99), and their numerical attributes discretized. Test were performed using four different batch sizes: 50, 100, 500 and 1000.

The authors found that, in terms of accuracy, CDCStream generally either outperformed or performed similarly to both SDriftClassif and CDDA. It was never outperformed by more than 1% but it was shown to outperform the other algorithms by up to 16%. In terms of batch size, it was observed that CDCStream is more robust than CDDA (note that batch size was not an available variable for SDriftClassif). CDCStream's overall best performance occurred with batch sizes of 50.

CDCStream tackles an important change detection problem for categorical data. It accomplishes unsupervised, context-based change detection for categorical data, a scarcely pursued, non-trivial task. Its strengths include its ability to analyze data without making prior assumptions, the requirement of very few parameters and the

use of dynamic windowing to appropriately exploit historical data. CDCStream succeeds in yielding competitive results when compared to other state-of-the-art methods, both supervised and unsupervised. However, it is not without limitations, which will be detailed in the following chapter.

2.7 Summary

This chapter presented the various challenges of modern Machine Learning given the great abundance of data collected on a daily basis and the often streaming nature of that data. A major challenge posed by streaming data, concept drift, was detailed and methods of handling it were presented. Further to these challenges, the qualitative nature of categorical data makes change detection in categorical data streams especially arduous. Categorical data analysis techniques, namely similarity measures, were presented and discussed. Finally, a state-of-the-art solution to change detection and adaptation in categorical streams, CDCStream, was detailed. The following chapter presents an extension of the CDCStream algorithm, the so-called FG-CDCStream approach, which addresses the main weaknesses present in the original algorithm. The chapter also lays the foundation for a comparative study between CDCStream and FG-CDCStream methods.

Chapter 3

FG-CDCStream

The previous chapter introduced the CDCStream approach to context-based concept drift detection and adaptation in a categorical setting. The CDCStream algorithm is a novel, cutting-edge, context-based approach to detecting concept drift in categorical data. However, it does have some inherent limitations. This chapter details these limitations and proposes the FG-CDCStream method that addresses these drawbacks.

3.1 Limitations of CDCStream

The CDCStream algorithm utilizes the DILCA framework to produce a set of matrices describing the data distribution of each batch from a stream. It then summarizes the matrices into a single figure representing both inter- and intra-attribute relationships. Chebychev's inequality determines how different the data distributions of the newest batch and that of the previous batches in the window are. If the distributions are approaching a significant difference, but are not different enough to flag a change, a warning period is initiated. If the distributions differ significantly, change is flagged and adaptation is initiated. The adaptation process replaces the current classifier with the backup classifier which was trained from scratch when the warning was flagged. The new classifier is presumably representative of only the current distribution and

thus will be more accurate in its predictions. CDCStream has been shown to perform competitively when its accuracy was compared to that of algorithms with similar goals but different structures [4]. It does, however, present several opportunities for improvement. The limitations of CDCStream are as follows:

- Chebychev's Inequality is too conservative, leading to high detection delay.
- Aggregation into a single summary statistic has a diluting effect.
- Temporal bounds on the post-change replacement classifier training data are unnecessarily broad.

Chebychev's inequality has many attractive features. Its versatility makes it incredibly attractive for a general-use change detector, where the data distribution need not be known, its two-tailed structure is ideal for unsupervised change detection and its incremental nature permits the use of SPC techniques. These assets, however, do come at a price. In order to permit all of these attractive features, the grain of the bound must be quite coarse. This property causes Chebychev's inequality to often be too conservative leading to greater detection delay.

Another obvious limitation of CDCStream is related to the aggregation issues inherent in the batch scenario. The aggregation of a batch into one single summary statistic has a diluting effect that, depending on factors such as change magnitude, duration and location, may result in increased missed detections (MDR) and detection delay (MTD).

A final opportunity for improvement of CDCStream relates to its adaptation strategy. This algorithm assumes that every warning will be rather closely followed by a change whose distribution is similar to the data distribution when the warning was flagged. In reality, this is not always the case. A warning might be caused by noise or simply minor fluctuations in data distribution and may be followed by static periods

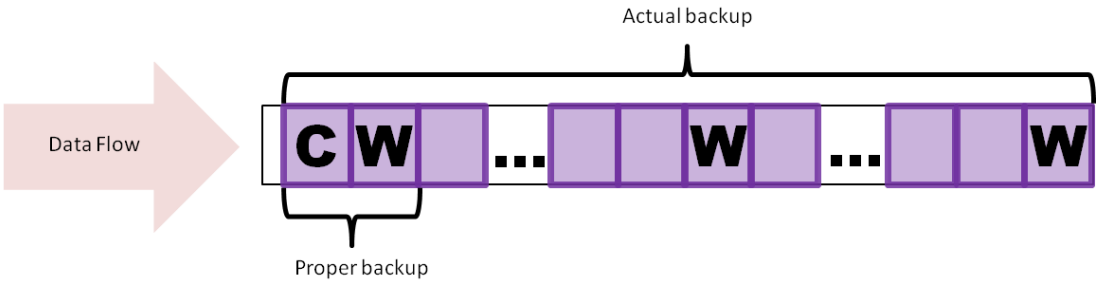


Figure 4: An example scenario where CDCStream's warning system produces a non-representative replacement classifier

rather than an actual change. In CDCStream, when a warning is initiated, the system remains in the warning state until a change is detected. The initiation of a warning period launches the creation of the background classifier that is used to replace the current classifier when change is detected. The issue arises in circumstances where a warning period is initiated, but is not followed by a change rather by another static period. In this case the warning period persists and the background classifier continues to be built until the next, potentially very distant change occurs. When a change does finally occur, the replacement classifier will be outdated, representative of the data distribution from the original warning. Figure 4 depicts a scenario where this issue is particularly detrimental to the learning system. In this scenario the analysis of the oldest batch shown (on the far right) flagged a warning. This warning was not followed by a change, but by a static period. This pattern repeats again (and could potentially repeat any number of times) until finally a warning period is followed by a change. According to the logic of this algorithm, a proper backup classifier would represent the data distribution of only the information following the last warning period. Instead, the actual backup classifier, is representative of the data distribution all the way back to the first warning. This could severely effect any post-change classifications.

Collectively, these limitations have the greatest effect on the detection of abrupt concept drifts. An increased detection delay due to the coarseness of Chebychev's inequality or aggregation dilution would be most detrimental in cases of abrupt drift. Since the distributions change so rapidly, predictions based on the previous distribution would be quite erroneous; more so than if the change was gradual where the stream still contained some instances of the previous distribution. Aggregation dilution would also be more likely to miss changes altogether if the transition period was shorter, providing less of an opportunity to pin the change. Although effectiveness is also reduced in the case of gradual concept drift, the effects are less severe. Finally, a replacement classifier that remains partially representative of the previous distribution is more likely to be effective in classifying the data of a gradual change that is also partially representative of the previous distribution than that of an abrupt change which is not. These limitations, problematic in and of themselves, should theoretically produce a net effect that significantly limits the system's performance upon encountering abrupt concept drift. Unfortunately, the original study proposing CDCStream [4] experimented using data sets with mostly gradual drift expected. However, preliminary tests for this study have shown that CDCStream does, indeed, falter when faced with abrupt concept drift.

3.2 FG-CDCStream: An Extension

This study proposes FG-CDCStream, an extension of CDCStream which offers revisions to remedy the limitations of CDCStream. It is expected that amendments to CDCStream's limitations will improve the system's MTD, MDR and predictive performance and, as an overarching result, its response to abrupt concept drift. Both change detection and adaptation capabilities have been targeted.

It was decided that for this study, Chebychev's Inequality's partnership with

DILCA's is indispensable in order retain the unsupervised and context-based properties of the change detector without making assumptions with regard to data distribution or attribute independence. The use of Chebychev's inequality was retained in FG-CDCStream. An alternative method is certainly of interest for future work.

FG-CDCStream does attempt to reduce detection delay and MDR caused by the batch-based scenario and its associated aggregation. Rather than discarding the batch architecture, since a batch structure is required by DILCA, a major element of this work, a method that could reduce the grain size of the data analysis while retaining the batch framework was sought. Clearly, instance by instance analysis, like that of ADWIN, was not an available solution though something similar in grain size was desirable. Reducing batch sizes was immediately rejected since the sample would be too small for DILCA to effectively summarize the data distribution of each batch which would result in opposite aggregation issues. The solution realized by FG-CDCStream involves overlaying a series of batches, or "tracks", each shifted by one instance from the previous track in order to simulate and instance by instance analysis. A technical explanation follows.

CDCStream uses a dynamic list L of i contiguous batches S of fixed size n to detect a change between current batch and the previous batches remaining in the sliding window. To solve the grain size problem, this thesis proposes the use of overlapped dynamic lists deemed tracks. More formally, FG-CDCStream uses a series of n tracks, each overlapping the previous track's most recent $n-1$ instances. This allows a change test to be performed as each instance is received, as in ADWIN, while incorporating the use of batches. Do note that there is a single initial delay of $2n$ before the first two batches can be compared.

Figure 5 displays the track and batch construction process as instances arrive. For simplicity, this example shows only the structure of batches and tracks and does not include responses to concept drift. A more detailed example including concept

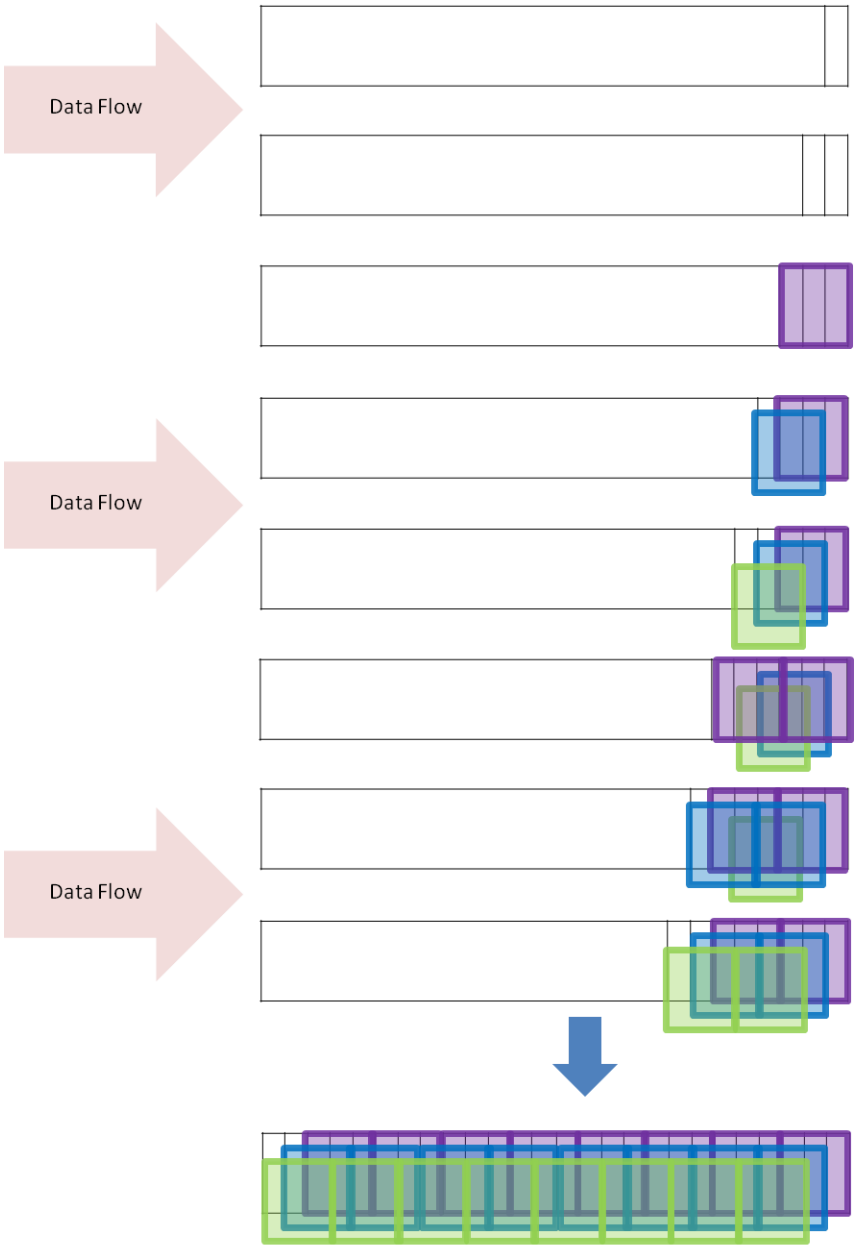


Figure 5: Track and batch building visualization

drift response will follow. For this example, assume a batch size of three instances (a batch size much too small in reality, but sufficient to understand track construction). Until the first three instances are collected, no batches exist. Once the third instance arrives, the first batch, represented by the purple rectangle, is complete. From this point on, a batch is completed upon the arrival of each instance. This is demonstrated with the creation of the blue and then green batches. Upon the arrival of the sixth instance ($2n$), a second batch is added to the original purple track. Note that the track with the newest batch is shown at the front and the least recently updated track in the back. This process continues to the end of the stream or to infinity.

It was noted above that the example in Figure 5 is oversimplified. In reality, instances are stored explicitly only until a batch is complete. A buffer with a container for each track collects instances until n have been collected. When a batch is complete, it is sent to the corresponding track's change detector object which stores only summary statistics to alleviate storage. That track's buffer is then cleared and the main algorithm begins building its next batch upon the arrival of the next instance. The appearance of the next instance will fill a batch belonging to the next track and the process continues as such.

3.2.1 Improving Change Detection

FG-CDCStream was designed with the intention of retaining the appealing qualities of CDCStream while improving change detection elements where opportunities exist. It essentially uses an ensemble of change detectors, each containing slightly shifted information, that vote on whether or not a change has occurred. Only the change detectors in close proximity in the forward direction to the detector that first flags change submit a vote. This is appropriate due to the temporal nature of the data and the desire to detect a change quickly. The voting system decreases the chances of completely losing information due to aggregation. This not only increases the probability

of detecting the change (reducing MDR) and detecting it quickly (reducing MTD), but provides another advantage of decreasing the probability of false detections (increasing MTFA). Since a vote is required to confirm a change, slight variations in distribution that would flag a false change in CDCStream would, in FG-CDCStream, be outvoted and therefore not flag a false change. This further increases the algorithm's robustness. FG-CDCStream overcomes the issues that CDCStream has with regard to the batch scenario and dilution due to aggregation, improving its change detection capabilities in terms of MDR, MTD and MTFA.

3.2.2 Improving Adaptation

A major limitation of CDCStream is its potential for a very unrepresentative replacement classifier during adaptation. This is caused by the termination requirement of a warning period being only a change period regardless of how closely or distantly that change period occurs. This requirement potentially results in a background classifier that is non-representative of the new distribution following a change detection. FG-CDCStream reconciles this by permitting a warning period to terminate if it is followed by either a static period or a change.

Recall that CDCStream uses Chebychev's inequality with a k value of two to trigger a warning ($k_w = 2$) and three to trigger change ($k_c = 3$), as experimentally determined. Instead, FG-CDCStream considers a warning period to begin when the first voter in a series flags change. A warning period concludes when either the change is confirmed by the remaining voters (change period follows) or the change is rejected (static period follows). More precisely, a warning period (warning state) is defined as a period where contiguous change detectors report k values greater than or equal to three (k_c) following a static period (in-control state). The system is in the in-control state during periods of returned k values of less than three. A warning period is terminated by either a change period (as in CDCStream) or another static period

(not considered by CDCStream).

Like CDCStream, initiation of a warning period triggers the creation of a new background classifier which trains on the current batch and all subsequent instances within the current warning period capturing the new data distribution and discarding the old. If change occurs, this background classifier becomes the actual classifier and the previous classifier is discarded. Otherwise, the background classifier is simply replaced when the next warning period begins. In FG-CDCStream, if a warning period is followed by a static period, the background classifier initiated by the warning is ignored rather than continuing to build until the next change occurs. The only background classifier that may be used as an actual replacement classifier in FG-CDCStream is one that is initiated by a warning immediately preceding (ie. there is no static period in between) a detected change. If, by chance, a change occurs without a preceding warning, the replacement classifier is created from the most recent batch only. This is fitting, since a change occurring absent of a warning is likely to be abrupt where a short historical window is appropriate. This alteration to the CDCStream algorithm increases the likelihood of the classifier in use to be representative of the current concept.

3.3 The FG-CDCStream Algorithm

Algorithm 4 describes the technicalities of the FG-CDCStream algorithm. The algorithm requires a data stream and a user-defined window size parameter. An integer variable *trackPointer* keeps an account of which track the current batch (the batch completed by the current instance) belongs to. The k variable represents the k value of Chebychev's inequality which is initiated with a value of zero and is altered by each track's change detector when it is called upon. The *votes* variable stores an integer representing the number of tracks which report a significant deviation in data

Algorithm 4 FG-CDCStream

Require: S: stream of instances, W: window size

1: $trackPointer = 0$, $k = 0$, $k_w = 2$, $k_c = 3$, $votes = 0$, $votes_w = 1$, $votes_c = 15$, $inst = null$
2: **for** $count = 0 \rightarrow W$ **do**
3: $tracks.add$(new change detector)
4: **end for**
5: **while** $hasMoreInstances(S)$ **do**
6: $inst = S.nextInstance$
7: $buffer.add$(new batch container)
8: **for** $count = 0 \rightarrow buffer.size$ **do**
9: $buffer.get(count).add(inst)$
10: **end for**
11: **if** $buffer.size == W$ **then**
12: $k = tracks.get(trackPointer).getKValue(buffer.get(0))$
13: **if** $k \geq k_c$ **then**
14: $votes++$
15: **if** $votes == votes_w$ **then**
16: $warningPeriod = true$
17: $initiateBackgroundClassifier(buffer.get(0))$
18: **end if**
19: **if** $votes == votes_c$ **then**
20: $changePeriod = true$
21: $warningPeriod = false$
22: replace classifier with background classifier
23: nullify background classifier
24: **end if**
25: **else if** $k < k_w$ **then**
26: $votes = 0$
27: **if** $warningPeriod$ **then**
28: $warningPeriod = false$
29: nullify background classifier
30: **end if**
31: **if** $changePeriod$ **then**
32: $changePeriod = false$
33: **end if**
34: **end if**
35: **end if**
36: **end while**
37: $updateClassifier(inst)$
38: $updateBackgroundClassifier(inst)$
39: **if** $buffer.size == W$ **then**
40: $buffer.remove(0)$
41: update track pointer
42: **end if**

distribution: a k value of three.

As each instance arrives, a buffer, which is a list of instance containers (or batches), adds a new batch object to the list. A copy of the current instance is then added to each of the existing batches. Note that upon the arrival of the first instance, only one batch container exists. When the second instance arrives, a new batch is added to the buffer and that instance is added to both batches. At this point, the first batch contains the first and second instances, and the second batch contains only the second instance. This process produces contiguous batches that contain data that are shifted by one instance. This process continues until the buffer contains $batchSize$ batches. At this point the first batch is complete, i.e. it contains $batchSize$ instances. The complete batch is processed, which will be subsequently described. After the batch is processed and summarized, it is removed from the buffer to make room for the next batch.

To process a completed batch, the batch is sent to the change detector associated with the current track. This change detector deviates from that of CDCStream in that it returns a k value of two (k_w) or three (k_c) if either of those occur, otherwise it returns a value of zero. Recall that, for FG-CDCStream, $k = 2$ does not initiate a warning but the notation k_w will be retained for continuity. Even though k_w does not issue warnings, it is still reported since it is permitted during a change period. If $k = k_c$, the current track detected a change. If this is the first change detection in a series, a warning period is initiated. This launches the creation of the background classifier which is built from the current batch. Reported k values of two are permitted within a warning period. If a k value of less than k_w is reported by a track during the warning period, the warning period is terminated, the $votes$ count is reset to zero and the background classifier is removed. This initiates a static period.

If at least $votes_c$ tracks confirm the change that triggered a warning period by returning k values of three (k values of two may be interspersed among these), the

system acknowledges the change. Through experimentation, this study determined $votes_c = 15$ to be appropriate. Confirmed change triggers adaptation by replacing the current classifier with the background classifier. The change period remains, whereby no new warnings or changes may occur, until a k value of less than two is reported, initiating the next static period. The fall of the k value signifies that the current change is fully integrated into the system i.e. the change detectors have forgotten the past distribution and the current model represents the current distribution. Since most change periods tend to taper off as dilution increases, it was determined that a confirmation, similar to that used to confirm change, is not required for the static period.

Whether or not adaptation occurs, a forgetting mechanism is applied to any change detector that produces a k_c. If change is not confirmed, the original change detection was likely incorrect and thus forgetting the corresponding information for that specific track is reasonable. If change is confirmed, it is reasonable to forget the old concept and remember only the new. This effectively resets change detectors that are not performing well, as is often done to ill performing members of learning ensembles. This approach permits outlier information to be discarded and increases the randomness of the ensemble enhancing its effectiveness which was verified empirically during preliminary testing.

Figure 6 depicts an example of the system's forgetting mechanism mechanics. First, the green track's most recent batch detects a change (represented by the exclamation mark in the red triangle). It then forgets all of its past batches retaining only the current one. The next batch to arrive, belonging to the purple track, also detects the change and forgets its past batches. The next batch, belonging to the blue track does not detect the change so it forgets nothing. The next green batch also does not detect change, so track building (or remembering) proceeds as usual. The same is true for the next purple batch. This process occurs regardless of the state of the

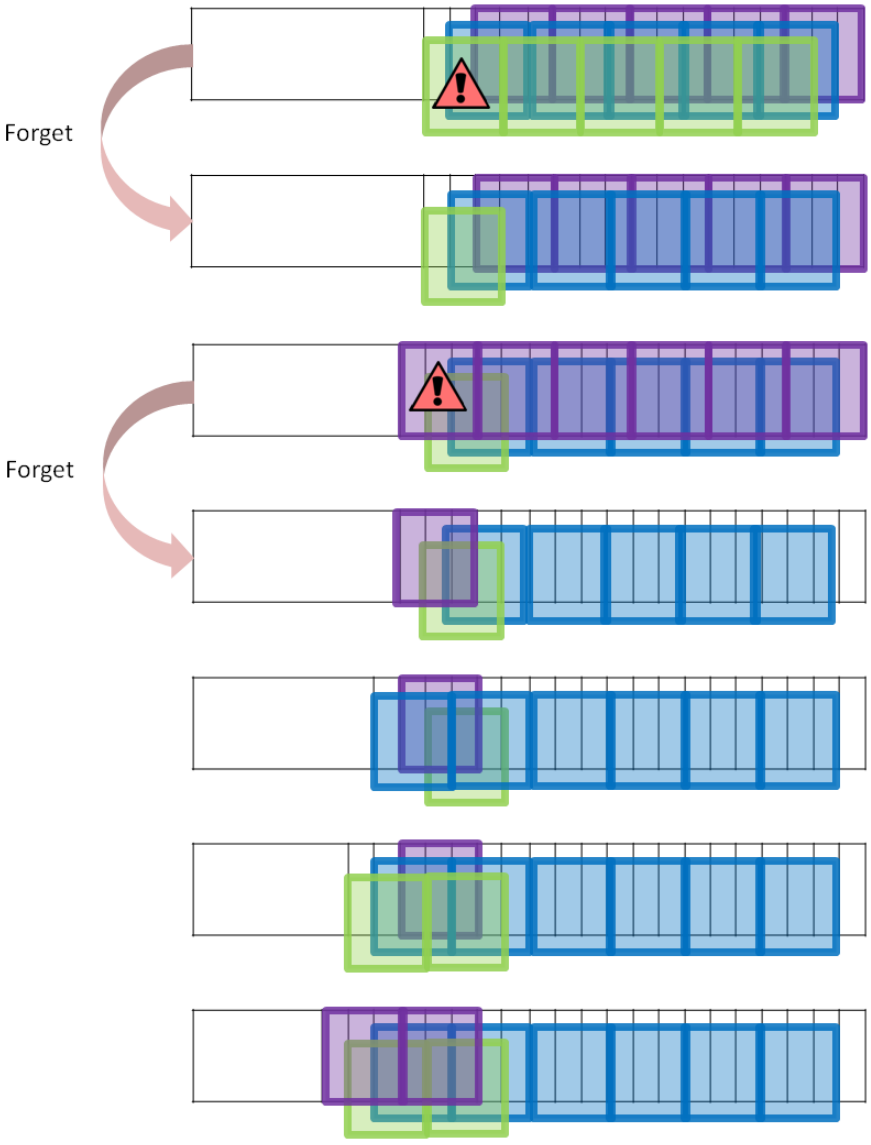


Figure 6: Forgetting mechanism example

system: in-control, warning or out-of-control. In this example, the green and purple tracks may have detected a change due to outlier interference. It is beneficial then, for those tracks to forget this information but for blue, whose summary statistics may not have been as affected by the outlier(s) due the the shifted sample, to retain its information.

3.4 Summary

This chapter discussed the main limitations of CDCStream which collectively cause poor response to abrupt concept drift. It introduces the FG-CDCStream method which addresses the change detection and adaptation limitations of CDCStream by reducing the grain size of the system and employing ensemble type methods of change detection. FG-CDCStream aims to reduce the MDR, MTD and increase MTFA while remaining robust to noise. The FG-CDCStream algorithms pseudocode is presented and described to provide the reader with a solid understanding of the algorithm mechanics. The following chapter provides the experimental framework designed to test the effectiveness of the FG-CDCStream method.

Chapter 4

Experimental Design

This chapter lays the foundation for comparing the CDCStream and FG-CDCStream algorithms with a meticulously selected evaluation strategy and experimental framework. Recall that FG-CDCStream aims to improve CDCStream's response to abrupt concept drift by improving the algorithm's limiting components. The proposed experimental design intends to meaningfully assess the outcome of said modifications.

4.1 Algorithm Evaluation Strategy

Algorithm evaluation techniques are used to appraise the effectiveness of learning systems. An abundance of such techniques has been proposed in the literature, each with varying strengths and weaknesses and differing applicability to specific circumstances. Careful selection of appropriate evaluation procedures that provide meaningful insight is imperative to an informative evaluation. Typical evaluation of a learning system involves four components: performance measures, error estimation, statistical significance testing and test benchmark selection [81].

Typically change detection algorithms are evaluated in conjunction with learning systems and their performance measured only with respect to those learners [7]. Whether or not this approach provides useful information about change detector

performance is debated in the Machine Learning community [7]. On one hand, it is reasonable to assume that a good change detector would improve overall system performance. On the other hand, system complexities and unintended consequences may play a larger roll than previously thought. In fact, various recent studies have provided evidence that change detectors with high false positive rates correlate to high classification performance [81]. This indicates that classification performance may actually be a rather poor indicator of change detector performance [7]. Unfortunately, explicit change detector evaluation is not as common nor as extensively studied as overall learning system evaluation methods and is relatively basic at this point. This study has elected to perform both explicit change detector evaluation to obtain more precise and informative measures of change detector performance as well as the more traditional evaluation methods that observe the performance of the refurbished system.

4.1.1 Performance Measures

Performance measures deal with quantifying properties of interest of an algorithm's operation. They are used to improve researchers' understanding of the strengths and limitations of the examined learning strategies [81] [82]. Several useful performance measures are discussed below.

4.1.1.1 Change Detection Statistics

The experiments in this study aim to assess CDCStream and FG-CDCStream concept drift detection algorithms. A good change detection algorithm maintains a balance between the detection of true changes and false alarms while minimizing detection delay [7]. In order to measure these qualities, change points must be known, and relative to the true change points, false alarms, missed detections and true detections must be defined.

Figure 7: Change detection evaluation concepts [7]

Any change detected previous to the first change is defined as a false alarm. The first change detected after a true change is a true alarm. Any changes detected after the true alarm and before the next change are considered false alarms. Missed detections occur if no change is detected between two true changes. For this study, in consideration of gradual concept changes, no false alarms are counted during the transition period. The first detection after the beginning of the transition period is considered the true detection. Figure 7 [7] displays these concepts which are used to calculate the following measures which can be used to evaluate change detection strategies, namely MTFA, MTD, MDR and MTR.

MTFA Describes the average distance between changes detected by the detector that do not correspond to true changes in the data. A high MTFA is desirable.

MTD Describes how quickly the change detector detects a true change. A low MTD is desirable.

MDR The probability that the change detector will not detect a true change when it occurs. A low MDR is desirable.

MTR Describes the compromise between fast detection and false alarms. A high MTR is desirable.

$$MTR(\theta) = \frac{MTFA}{MTD} * (1 - MDR) \tag{4.1.1}$$

These performance measures allow for a detailed examination of a raw change detector's effectiveness in detecting true changes quickly while remaining robust to noise and issuing few false alarms. The drawbacks of these methods stem from the definitions of the evaluation concepts. For example, if two changes are close together, it is possible that the first true change was detected after the second true change occurred. The lack of an alarm in between the two changes would be considered a missed detection and the true detection of the first change would be considered a true detection for the second change with a smaller than actual time to detection. Unfortunately it is not possible to definitively correlate a particular change alarm to a particular change meaning these measures can be quite rough.

As mentioned above, change detectors that issue many false alarms are associated with high classification performance. So, if the goal is to achieve high classification performance, a low MTFA, though menial, may not be seriously adverse. Also, the inherent relations between these change detection statistics can make interpreting results challenging. For example, a very low MTFA may lead to impressively high MTD and MDR values by increasing the likelihood that true changes will be "detected" by a false alarm that occurs shortly after it. Similarly, a very large MTD could be caused by the combination of a missed detection and a false alarm occurring long after, but still before the next change. Care must thus be taken to remain cognisant of the relationships between these performance measures when interpreting results.

The information that can be obtained by some of these performance measures, namely MTFA, MTD and MTR, is also limited by the fact that they are sometimes incalculable. If fewer than two false alarms occur over the stream, an MTFA will not be calculable and will report as unavailable. Similarly, if no true change is detected, MTD is unavailable. If MTFA and/or MTD are unavailable or if MTD is zero (which is not likely but possible especially if the number of changes is very low), then MTR will also be unavailable.

Despite their limitations, these performance measures provide a means of directly assessing a change detector's performance. They provide insight into the particular behaviour of a change detector under various conditions. A more indirect way of evaluating change detection methods, and the most common in the literature, is the measuring of accuracy-type performance measures.

4.1.1.2 Accuracy-type Statistics

Accuracy-type statistics are arguably the most prominent means of measuring the performance of change detectors in the literature [83]. When using accuracy-type statistics (indirect indicators) to infer the effectiveness of a change detector, it is assumed that superior change detection necessarily implies superior accuracy-type statistics. Since accuracy is affected by may factors (like MTFA as mentioned above), this assumption should be taken with a grain of salt. While accuracy measurements are useful, they do not paint the full picture in terms of evaluation [81]. This section will discuss the advantages and limitations of both traditional classification accuracy as well as agreement statistics.

Classification accuracy, measures the fraction of examples that have been correctly labelled by the classifier. Classification accuracy is given by:

$$accuracy = \frac{number\ of\ correctly\ classified\ instances}{total\ number\ of\ \ instances} \tag{4.1.2}$$

or more formally:

$$accuracy = \frac{1}{N}\sum_{i=1}^{N} I(f(x_i) = y_i) \tag{4.1.3}$$

where N is the total number of instances classified, I is the indicator function that outputs one if the predicate is true and zero otherwise, $f(x_i)$ is the classification label of instance i and y_i is the true label.

The accuracy metric effectively summarizes algorithm performance in consideration of all classes present. However, it does embody some important limitations and can be misleading when applied to class-imbalanced data or data where certain classes are more important than others. For example, a classifier that always labels instances with the most prevalent class in the data set would achieve accuracy proportional to the prevalence of that class. This could potentially yield very high accuracy and thus mislead an interpreter to think that this classification system is indeed effective in discerning one class from another [81].

Agreement statistics aim to reconcile the potential coincidental concordance of classification with true labels that is present in evaluations using classification accuracy. Three main agreement statistics include the S coefficient [84], Scott's π (pi) statistic [85] and Cohen's κ (kappa) statistic [86]. Each accounts for chance agreements but in different ways. The S coefficient adjusts the proportion of agreement based on the number of classes present. The more informative Scott's π and Cohen's κ consider the ratio of the difference between observed and chance agreements and the maximum possible agreement. Unlike Cohen's κ, Scott's π considers the probability that a label is assigned irrespective of the label assigning process. Also, while the π statistic assumes classes are balanced, the κ statistic is able to incorporate changes in class distribution. For these reasons, the κ statistic is the preferred agreement statistic in the evaluation of learning algorithms. [87] [81]

The kappa statistic is defined as:

$$\kappa = \frac{p_0 - p_c}{1 - p_c} \tag{4.1.4}$$

where p_0 is the accuracy of the classifier and p_c is the probability that a change classifier, giving the same class distribution as the tested classifier, classifies correctly. A chance classifier is one that does not consider input data and classifies using only

past label information. For example, a majority class classifier predicts every instance is labelled with the majority class of either the stream or the most recent window. If κ=0, the classifier makes correct predictions no more often than the change classifier. If κ=1 the classifier is always correct.

The kappa plus (κ^+) statistic [87] is an extension of the kappa statistic that considers the temporal dependence often present in data streams. The κ^+ statistic is given by:

$$\kappa^+ = \frac{p_0 - p'_e}{1 - p'_e} \tag{4.1.5}$$

where p'_e is the prequential accuracy (which is detailed in the following section) of the No-Change classifier. The No-Change classifier utilizes temporal dependence information by predicting that an instance is labelled with the same class as the true label of the previous instance. If $\kappa^+ < 0$, the tested classifier performed worse than the No-Change classifier. If $0 < \kappa^+ \leq 1$, the tested classifier performed better. In cases where there is no temporal dependency in a data set, the kappa plus statistic produces the same result as the Kappa Statistic. It is recommended that both κ and κ^+ be examined in evaluating performance in the data stream setting [81] [87].

This study returns results for classification accuracy, κ and κ^+ accuracy-type measures, though κ^+ results will be discussed in the greatest detail due to their comprehensiveness. This study also considers the progression of accuracy-type statistics throughout the stream, not only the final values, in order to gain more information about change detector performance. For example, the steepness of the drop in accuracy-type performance at a change point, and the swiftness of recovery provides more information than a single value representative of overall accuracy.

4.1.1.3 Cost Measures

Although cost efficiency was not a major goal of this thesis, the time and memory costs of CDCStream and FG-CDCStream should be analyzed and compared. In this study, cost measures considered include model serialized size (bytes), time (CPU seconds) and the memory-time ratio (RAM-hours) where every Gigabyte (GB) of Random-Access Memory (RAM) deployed for one hour is a RAM-hour [88]. Given that the FG-CDCStream algorithm employs $batchSize$ tracks and processes each of them much like CDCStream processes a single track, it is expected that FG-CDCStream will be more time and memory intensive than CDCStream.

4.1.2 Error Estimation Methods

Following the selection of appropriate performance measures, an informative evaluation applies error estimation methods which produce the least biased estimate of said performance measures. Traditional error estimation techniques like k-fold cross validation, leave-one-out and bootstrapping are not well suited to a data stream setting since they all involve re-sampling of the data. Two distinguished error estimation methods have taken hold in the data stream mining community: holdout and prequential (or interleaved test-then-train) [81] [89].

4.1.2.1 Holdout

The holdout method involves an independent holdout or test set from the training set and applying the current model to it periodically [90]. An advantage of using the holdout method is that it is a more accurate and unbiased estimator. However, issues arise when concept drift is present since the test set may not be representative of the current concept [91].

4.1.2.2 Prequential Error Estimation

The potential issue of using a test set belonging to an outdated concept is not present in prequential error estimation [92] which tests the model on each instance before using it for training. This allows the model to be tested on instances it has not yet examined as well as update the accuracy estimate incrementally [89]. Prequential error is given by:

$$S = \sum_{i=1}^{n} f(x_i, y_i) \tag{4.1.6}$$

where f is a loss function (for example, zero-one loss) between the label assigned by the model and the actual observed label. Prequential evaluation tends to be pessimistic, reporting higher error than holdout under the same conditions [90]. Error estimation can be strongly influenced by the first section of the stream before the classifier has trained on many instances. To mitigate this undesirable result, prequential error may be calculated using a forgetting mechanism like sliding window or fading factor [90]. It has been demonstrated that using such forgetting mechanisms permits prequential error estimates to converge to holdout estimates [90].

For this study, in which temporal concepts are critical, prequential error estimation was selected so as not to risk improper pairing of test and training concepts. The sliding window forgetting mechanism was applied in order to achieve improved accuracy of estimation. The window size parameter was fixed to the size of a batch with the intention remaining consistent with the current concept from the perspective of the change detector, even in periods of abrupt change.

4.1.3 Statistical Significance

Applying performance measures and error estimation methods to a series of experiments provides useful information about the learning strategy's behaviour in each test

domain. While comparing such results of various learning strategies is informative, it may be unclear whether the differences in performance observed are due to the actual characteristics of the learning systems or simply coincidental. Statistical significance testing is used to improve understanding of the significance of the results of experiments [81] in order to make generalizable inferences beyond a single study [93]. It has, however, been argued by many that the issues inherent in statistical testing, or Null-hypothesis Statistical Testing (NHST), diminish its merits, even to the point where discontinuation has been suggested [81]. This has been debated by scientists across fields for decades [93].

Different null-hypothesis tests are available for different tasks. This study aims to compare the performance of exactly two algorithms, FG-CDCStream and its predecessor CDCStream, across multiple matched domains (on the same data sets). Two well renowned tests that are appropriate for this setting [81], the Signed Test [94] and the Wilcoxon Signed-Rank Test [95] were considered for this study. Other popular competitors were immediately rejected due to being ill fit to this setting. For example, the Friedman Test [96] performs better when more than two algorithms are compared and the McNemar test [97] which is better suited to comparing two algorithms over a single domain [81].

The **Signed Test** is a computationally simple NHST. The number of datasets on which a classifier A outperforms[1] another classifier B (n_A) is compared with the number of times classifier B outperforms A (n_b). The information provided by this test, however, is limited in that it does not consider the quantity of the difference between classifiers. If one classifier marginally outperformed another on a particular data set the victory would be treated the same as if the second classifier was grossly outperformed [81] [94].

The **Wilcoxon Signed-Rank Test**, a similar NHST, does consider the extent

[1] with respect to a performance measure

to which the performance of two classifiers differ, as well as the the direction of that difference, making it considerably more powerful than the Signed Test. This test operates under the assumption that, for the difference in performances to be statistically significant, the outperforming classifier performs better most of the time, and when it does not, it under-performs by only a small degree [81] [95].

It is clear that the Wilcoxon Signed-Rank Test is more informative than the Signed Test as it provides qualitative and quantitative information in regards to the performance of the competing algorithms. However, due to the nature of the performance measures used in this study, unfortunately neither test is appropriate. The Signed Test requires the values of the performance measures to be ranked. Due to the high degree of association between the change detection performance statistics, ranking each performance statistic on its own would be uninformative and misleading. For example, an MDR might rank higher than another, but that victory may be the result of an unreasonably low MTFA causing changes to be "detected" by what are truly false alarms. The change detection statistics must be considered together in order to infer the underlying behaviours of the change detector. Ranking the measures as some sort of collective, though, would be complex and imprecise. The Wilcoxon Singed-Rank Test suffers the same issues and furthermore requires a quantitative value per performance measure per test which, as discussed above, is not always available for the change detector statistics. An alternative would be to perform statistical significance testing on accuracy-type performance measures alone. However, as discussed above, considering only accuracy-type statistics can be equivocal and potentially misleading, as would be performing statistical significance tests on accuracy-type measures alone.

It is the responsibility of any researcher to provide meaningful information by selecting the methods most appropriate for their work [93]. With its undoubted ill fitness to this particular study as well as its debated merits in the scientific community, this study elected to forgo statistical significance testing after careful consideration.

It decidedly would not fulfil its purpose in providing meaningful insight or sound grounds on which to project generalized inferences.

4.1.4 Test Benchmark Selection

A well substantiated architecture of an experimental framework is critical to an informative study. Algorithm application, results and analysis strongly depend on the domains on which they are assessed [81]. This study intended to explore the performance of the FG-CDCStream algorithm under a broad spectrum of circumstances with respect to evolving concepts, compared to its predecessor CDCStream. The final sections of this chapter describe the design elements of this experimental framework and how they are finally consolidated.

4.2 Software Tools

Experimentation was conducted using the Massive Online Analysis (MOA) framework for data stream mining [98]. MOA is an open source software closely related to its well known offline counterpart Waikato Environment for Knowledge Analysis (WEKA) [99]. Its open source nature allows developers to extend MOA with ease. MOA contains a collection of popular data stream mining algorithms and simple experimentation options. It includes popular algorithms for classification, clustering, concept drift detection, algorithm evaluation and data stream generation. It also allows for the importing of pre-existing datasets which it uses to simulate a data stream.

This study focused on the task of classification of data streams with concept changes. In MOA, classification is performed by applying a specified classification algorithm to a particular stream of data using a named error estimation method (or evaluation method). Depending on the evaluation method, the user may specify

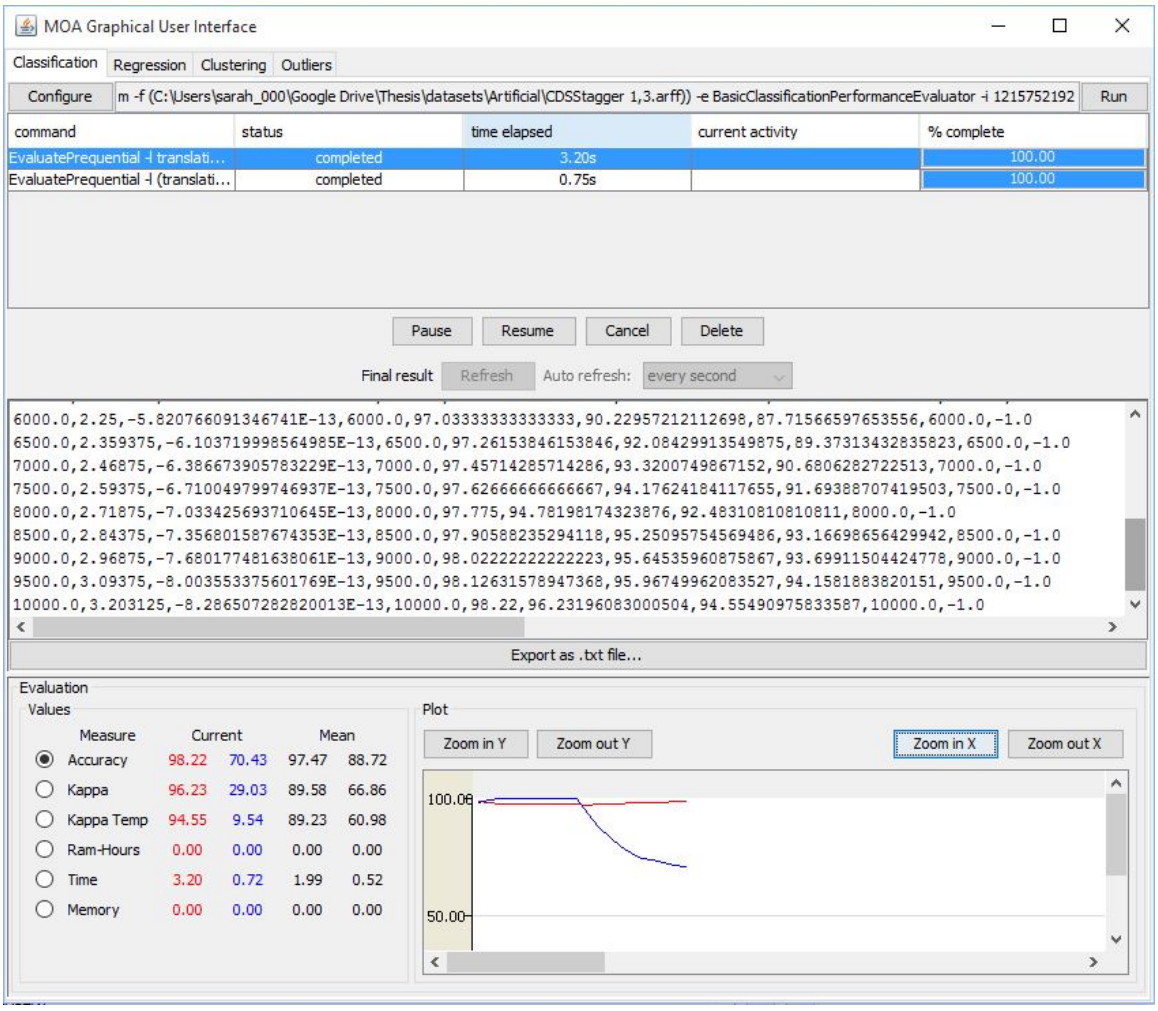
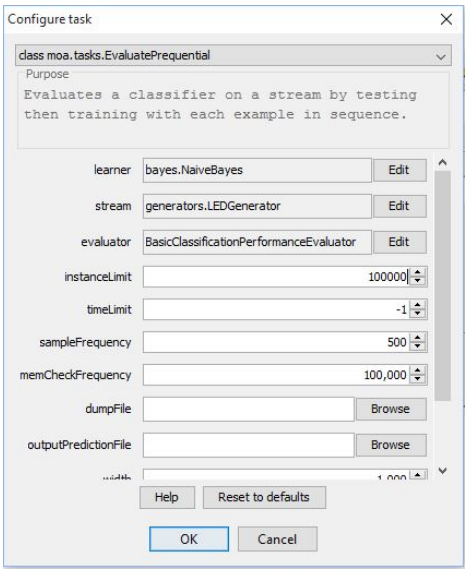


Figure 8: Comparing algorithm classification performance in MOA

parameters such as instance limit, time limit, window size, etc.

Note that the latest versions of MOA do contain change detector evaluation tools. However these tools, which provide only three data streams for testing and no import option, would not be useful for the extensive testing required by this study. For this reason, the built-in change detection evaluation tool was not used in this study. Rather, MOA's data stream generators were used to create data with known change injection locations. Unfortunately, the data generation tools do not include full gradual concept transitions, rather change blips of various lengths, so some manual effort was required.



(a) MOA GUI

```
java -cp moa.jar;weka.jar -javaagent:sizeofag.jar
     moa.DoTask EvaluatePrequential -l bayes.NaiveBayes
     -s generators.LEDGenerator -i 100000
```

(b) Command line code

Figure 9: Methods of configuring a classification task in MOA

In terms of experiment results, MOA outputs performance measures at regular intervals throughout the stream. This provides information about the algorithm's behaviour through time, rather than just a final overall value, which is particularly useful in examining points of interest like change points. The evaluation section of MOA's Graphical User Interface (GUI), as depicted in Figure 8, automatically displays a side-by-side comparison of various performance measures for each algorithm as well as a graph to visually compare those values over time. In this example, it is immediately clear that the blue algorithm faltered at the change point without having to search through the numerical results.

In MOA, experimentation can be done through the GUI, or command line code. The GUI is quite intuitive and useful for quick visualization of results. Using command line code is particularly advantageous when automating testing. Figure 9 displays a simple example of a configuration of a classification task using each of the available methods. Due to the large number of tests with different configurations, this study used command line code embedded in a Visual Basic Scripting Edition (VBS)

script for the experimentation.

To compare algorithm performance, a copy of the DILCA and CDCStream code used in [4] was obtained directly from the author via email communication. This code utilized MOA libraries but was not fully integrated. For experimental continuity, the code was adapted and fully integrated into MOA. The results from the experiments were output from MOA to Comma Separated Values (CSV) files where they were later analyzed using Microsoft Excel and VBS scripts.

4.3 Data Streams

Data set selection is a non-trivial and particularly critical element of an experimental framework. It is imperative to consider a large sample of problems under a diverse set of circumstances in order to avoid bias as best as possible and identify specific areas of strength and weakness [81]. This study selected both artificial and real data sets comprised of diverse characteristics. Various different circumstances were also induced on the synthetic data specifically. The single characteristic that did remain constant across all tests was the strictly categorical data type.

Numerical attributes that were present were discretized into five bins via unsupervised equi-width discretization, as done in the original CDCStream study [4]. Other discretization methods were tested prior to formal experimentation. These tests showed that even manual context-based discretization made only a negligible difference in results so the original discretization scheme was retained.

Both synthetic and real-world data were used for experimentation in this study. Synthetic data permits testing in highly structured scenarios while real-world data experimentation allows researches to test algorithms in a practical setting. Since these types of data have rather complimentary strengths and limitations, when used

in tandem they are able to capture the behaviour of the change detector more comprehensively. Experimentation using synthetic data, however, was of primary interest in this study since it provides known parameters that are fuzzy at best in real data. Synthetic data provides explicit knowledge of change point location, type of change (abrupt or gradual) and change duration. This information is critical to this study since FG-CDCStream aims to perform similarly to CDCStream in periods of gradual change and superior to CDCStream in periods of abrupt change. It aims to detect changes more quickly than CDCStream with its reduced grain size. In order to test FG-CDCStream's effectiveness with respect to its aims, it is imperative to have this knowledge that only synthetic data can provide. The real data sets were less crucial to this study, but were included for continuity and synergy.

The data sets used each contain both binary and multi-class classification tasks. The selected data sets also contain size and dimensionality diversity with varying attribute cardinality. Change injection sites and durations (abrupt to gradual) as well as noise levels were varied.

4.3.1 Synthetic Data Streams

Synthetic data provides research insights that real data might not contribute precisely. This approach also promotes the study of variabilities that may not be sufficiently present or obvious in the real data studied, but conceivably may be in other real data [81]. The use of artificial data provides significant user control for studying specific aspects of change detector performance since change characteristics (like location, width and magnitude) are known, allowing change detector specific performance measures (like MTFA, MTD, MDR and MTR) to be evaluated. However do note that the use of artificial data may be limited in its ability to simulate practical settings [81].

Varying degrees of noise were tested using synthetic data in order to test and compare the algorithms' robustness to noise. Each synthetic data set was injected

Dataset	Classes	Features	Categorical	Numerical
LED	10	24	24	0
Stagger	2	3	3	0
Mixed	2	4	2	2
Agrawal	2	9	3	6
ConceptDriftStream	varies	varies	varies	varies

Table 1: Synthetic Data (Basic Characteristics)

with 0, 1, 2, 3, 4, 5, 10, 15, 20 and 25% noise using the WEKA "addNoise" filter. This noise was applied to every attribute but the class attribute, since CDCStream and FG-CDCStream are unsupervised change detectors.

The synthetic datasets selected for this study provide user control over the frequency of changes, change injection location, duration and magnitude. This control permits testing that demonstrates the behaviour of each drift detection method under structured circumstances allowing the researcher to obtain precise indications of change detector behaviour. Five of MOA's (or MOA extensions) synthetic data set generators, summarized in Table 1, were used in various configurations to produce the synthetic data for this study.

The **Light Emitting Diode (LED)** data set generator [100] is made up of 24 attributes representing 24 LEDs on a screen. Each LED is in either the on or off state. The classification task is to determine the number that is displayed on the screen. In this data set concept drift is controlled by the user. The user may allow from zero to seven of the seven relevant attributes to drift. The difference between the number of attributes that drift from one concept to another is indicative of the change magnitude. For example, if a stream with zero drifting attributes changes to seven drifting attributes, a difference of seven, the change magnitude is greater than a stream with four drifting attributes changing to five, a difference of one. These

properties make this data set generator an attractive one in studying concept drift detection.

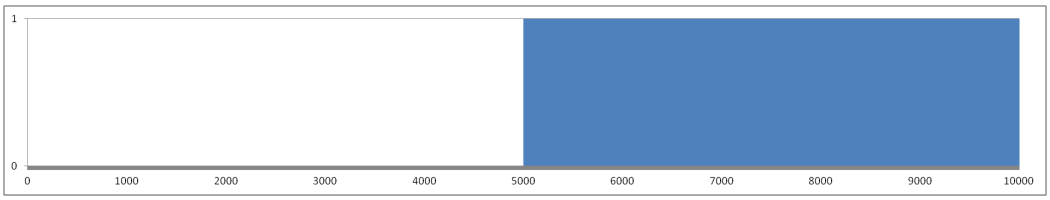
The **Stagger** data set generator, as described in [101], consists of size, colour and shape properties of simple objects. Three exclusive Stagger functions dictate which simple objects are classified as true while all other configurations are classified as false:

1. size = small and colour = red
2. colour = green or shape = circular
3. size = (medium or large)

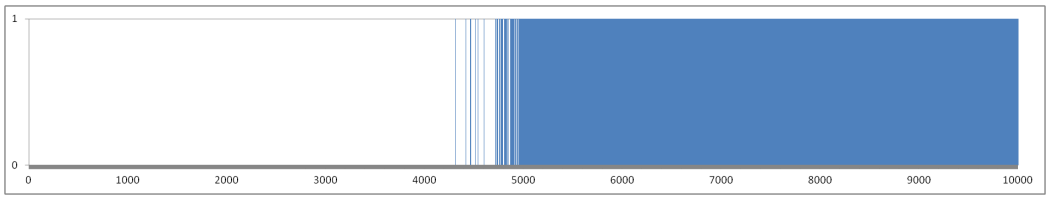
Concept drifts occur when one function is replaced by another, either gradually (dispersed) or abruptly. This generator is ideal for producing datasets that contain very abrupt concept changes that are naturally categorical. However, its extremely low cardinality may impede proper context selection by the change detectors.

The **Mixed** data set generator, introduced in [24], contains four variables v, w, x and y. If exactly two of v, w and y are greater than $0.5 + 0.3 * sin(3\pi x)$, the instance is classified as positive and otherwise, negative. Context drift occurs when this classification is reversed. Since this function can be switched to its opposite, when concept change occurs it will be considered a high magnitude change. Several characteristics of this data set may prove trying for CDCStream and FG-CDCStream. First, this data set is naturally numerical. Discretization contributes to a loss of information and the addition of some degree of noise. Second, like the Stagger data set, this data set has very low cardinality, a potential issue for context selection.

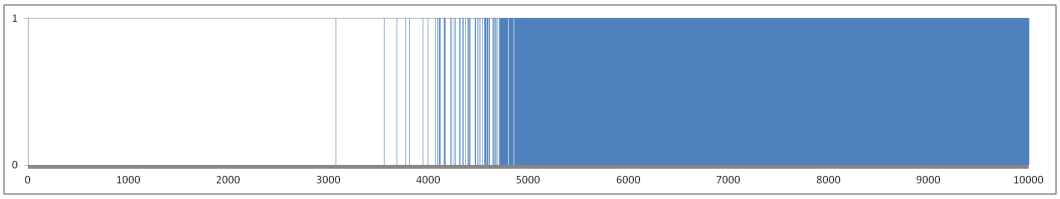
In the **Argarwal** data set each example represents a person. This generator uses ten functions of various complexities which determine whether a person is in "GroupA" or "GroupB". Concept drift is induced by switching functions throughout the stream. Note that some of the defining functions are very similar contributing to



(a) Drift width of 0 instances (abrupt change)



(b) Drift width of 500 instances



(c) Drift width of 1000 instances

Figure 10: Change injection for stream samples of size 10000, central drift location of 5000 and varying drift widths. Zero indicates the original concept and 1 indicates the new concept.

overlapping class boundaries.

Finally, **ConceptDriftStream** allows the user to inject a data stream with another at a specified position over a specified duration. It uses the sigmoid function to inject the drift based on its position and width. This property permits the injection of concept drift. A stream may also be injected with another ConceptDriftStream resulting in the potential for many drift injections of varying durations.

ConceptDriftStream's sigmoid function produces only a drift blip which slowly returns to the original concept. Some manual effort was required to achieve a full concept transition. The data sets were cut where the injected concept became fully realized and a stream of that new concept was attached to follow. Care was taken to ensure the addition did not include any of the same instances that occurred during the transition by using the same random seed and removing the first instances (the

ones which already occurred during the transition). In other words, if the transition used x instances from the second concept, the following stream began at instance x+1.

The point at which the change becomes fully realized, depends on the change duration. The duration determines how abrupt or gradual the drift is. A duration of zero produces a very abrupt drift while longer durations produce more gradual drifts. For this study, abrupt concept drift windows had size zero and two durations of gradual drift were selected: short (500 instances) and long (1000 instances). A visualization of the concept drifts using the sigmoid function is shown in Figure 10 where the concept changes from white to blue.

The magnitude of the concept drift depends directly on the nature of the data stream. For example, if the first stream is an LED stream with zero drifting attributes and the injected stream is the same with seven drifting attributes, the magnitude of the change will be greater than if the second stream had only one drifting attribute. For the remaining data sets, change magnitude cannot be reliably measured.

Table 2 displays the change configurations studied, not including change magnitude or the ordering of changes since both are generator dependant. Abrupt and gradual changes were injected. Streams of one, four and seven changes were studied in various orders, including streams of heterogeneous (both abrupt and gradual) drifts. Different stream sample sizes, change widths, patterns and distances between changes (as well as magnitudes and orderings) were studied on account of comprehensiveness. Streams including no changes were also created and tested as a baseline.

For single change scenarios, changes were injected half way through the stream in order to observe system behaviour well before and well after the change. For multiple abrupt changes, four different distances between changes were studied: 500, 1000, 2500 and 5000 instances. This was done in order to assess the change detectors'

Description	**Changes**	**Instances**	**Width(s)**	**Location(s)**
Abrupt	1	10000	0	5000
	4	2500 5000 12500 25000	0, 0, 0, 0	500, 1000, 1500, 2000 1000, 2000, 3000, 4000 2500, 5000, 7500, 10000 5000, 10000, 15000, 20000
	7	4000 8000 20000 40000	0, 0, 0, 0, 0, 0, 0	500, 1000, 1500, 2000, 2500, 3000, 3500 1000, 2000, 3000, 4000, 5000, 6000, 7000 2500, 5000, 7500, 10000, 12500, 15000, 17500 5000, 1000, 1500, 20000, 25000, 30000, 35000
Gradual	1	10000	500 1000	5000
	4	21000	500, 500, 500, 500 1000, 1000, 1000, 1000	500, 1000, 1500, 2000
	7	36000	500, 500, 500, 500, 500, 500, 500 1000, 1000, 1000, 1000, 1000, 1000, 1000	5000, 10000, 15000, 20000, 25000, 30000, 35000
Heterogeneous	4	21000	0, 500, 1000, 0 1000, 500, 0, 500 500, 1000, 0, 1000	5000, 10000, 15000, 20000
	7	36000	0, 500, 1000, 0, 500, 1000, 0 1000, 500, 0, 500, 1000, 0, 500 500, 1000, 0, 500, 0, 0, 1000	5000, 10000, 15000, 20000

Table 2: Synthetic Data Drift Characteristics

abilities to detect changes in succession and to compare recovery times. For multiple gradual changes, a slightly different approach was taken. In this case, the use of the sigmoid function causes varying separation between change injection points, compressing or elongating the change period. So, rather than varying the change injection location, the change widths were varied. Finally, the streams containing multiple heterogeneous changes were designed to include all change widths in differing (but altogether balanced) quantities and positions.

A great number of synthetic data sets were created for this study. Bi- and multiclass data sets with varying cardinalities were created. Changes of varying types, widths, magnitudes, multiplicities, positions and distances from other changes were induced and various noise levels were applied. These synthetic data sets were intentionally designed to represent the vast potential of real world data and test the effectiveness of FG-CDCStream relative to CDCStream in this multitude of scenarios.

4.3.2 Real Data Streams

While synthetic data provides rigid structure in modelling situations that may arise in real world data, real data offers its own unique and complimentary advantages. It provides researchers with insight on the behaviour of algorithms in actual, practical settings, which synthetic data simply cannot achieve. However, it is important to be cognisant of the limitations of experimentation using real data sets. For example, since details like the precise change point locations, durations, distributions or noise levels are virtually unknown in real data, conclusions drawn from these experiments may be vague [81].

The real data sets used in this study, as summarized in Table 6, include all those used in the original study presenting CDCStream [4]. The data sets selected represent frequently used benchmarks within the Machine Learning community thus easing comprehensiveness and simplifying potential comparisons to other works. These data

Dataset	Instances	Classes	Features	Categorical	Numerical
Electricity	45,312	2	8	0	8
Forest Covertype	581,012	7	54	44	10
KDD99	148,517	2	41	7	34
Airlines	539,383	2	7	4	3

Table 3: Real Data Characteristics

sets may be found in the University of California, Irvine (UCI) Machine Learning Repository[2] and the Kaggle dataset repository[3].

The **Electricity** data set [39] represents changing electricity prices in New South Wales from 1996 to 1998. Every half an hour price changes were recorded to have risen or fallen based on supply (the cheapest combination of electricity from all power stations) and demand for electricity. The task with this data set is to classify the price change as an increase or decrease. Concept drift occurs gradually in this data set due to seasonality, changing user consumption patterns and unexpected events [89] and is therefore expected to be predominantly gradual. This data set is often used as a benchmark data set in data stream classification, though it has been noted that high auto-correlation decreases its value as a tool in concept drift detector evaluation [89]. In this study, a normalized version of this data set was used.

The **Forest Coverytype** data set consists of cartographic variables (for example, soil types present) from Colorado wilderness areas with minimal human management alterations [102]. The task is to classify the forest covertype as one of seven types (for example, Aspen, Douglas Fir etc.) for each 30 meter by 30 meter cell of forest. In this data set, concept drift is expected to be principally gradual, occurring due to ecological processes and changing elevations. An adaptation of this data set, sorted based on elevation was also used, as in [4], to introduce natural gradual drift.

[2]http://archive.ics.uci.edu/ml/
[3]https://www.kaggle.com/datasets

The **KDD99** data set [103] originated from the Knowledge Discovery in Databases Cup data mining competition in 1999. It contains a nine week Transmission Control Protocol (TCP) dump from a simulated US Air Force local-area network which was later organized into higher level features. The training data was then infused with 24 types of network attacks. The test data received an additional 14 types of attacks to better simulate reality. This study used an adapted version of the data set [4] which combines the training and test sets. The task for this data set is to determine whether the network activity is normal or not. Concept drift occurs when normal network traffic patterns change over time [104] and is most often gradual.

The **Airlines** data set [105] is the largest in terms of number of instances of all of the real datasets used in this study. It contains departure, duration, and arrival information about flights within the USA from October 1987 to April 2008. This study, similar to [4], used a smaller adaptation of the original data set. The task with this data set is to determine whether or not the flight will be delayed. Concept drift occurs due to weather, unexpected events and changes in airline procedures over time and is more likely to be gradual than abrupt.

As in the synthetic data sets, bi- and multi-class real data sets with varying cardinalities were selected for this study. The data selection element of this experimental framework design was carefully constructed to obtain as much information about the change detectors' behaviours as possible. Care was taken to be as comprehensive as was achievable and avoid bias where possible. Real and synthetic datasets were used in tandem to capitalize on their complimentary strengths and weaknesses.

4.4 Classifiers

The original CDCStream study [4] tested its strategy using only the Naïve Bayes classifier. For a more comprehensive understanding of the behaviour of both CDCStream

and FG-CDCStream, the most basic incremental classifiers of various types (Bayesian, decision tree and K-Nearest Neighbour (KNN)) were selected: Naïve Bayes, the basic Hoeffding Tree and Instance-based K-Nearest Neighbour (IbK). The Naïve Bayes classifier uses probabilistic methods based on Bayes' theorem to make predictions. Bayes theorem describes the probability of a hypothesis (like an instance belonging to a certain class) being true, given a set of conditions (like the attribute description of an instance) [13]. A decision tree is an structure similar to a flow chart where internal nodes represent attribute tests, branches represent test outcomes and terminal nodes hold class labels [13]. The Hoeffding tree utilizes the Hoeffding bound, which measures the deviation of the measured mean from the true mean, to split nodes on the fly [27]. KNN classifiers use the nearest neighbours, as determined by a distance function, to determine the class of a target instance [13]. For example, the target instance might be assigned the mode class of its nearest neighbours. IbK is a storage-saving KNN algorithm that uses only specific instances for learning, automatically determining the appropriate value for k [106]. Each of the classifiers used for experimentation are built-in to MOA.

4.5 Batch Sizes

Different batch sizes have different effects on the components of the change detection strategies studied, especially on DILCA and summary statistic calculations. Batch size also affects remembering, forgetting and adaptation strategies. The experiments in this study tested batch sizes of 30, 50 and 100. Preliminary testing showed that smaller and larger data sets were not as effective for both change detectors, which is corroborated by the original study which found a batch size of fifty to be ideal for CDCStream [4].

4.6 Hardware and Software Specifications

Experimentation was performed on a machine with the following hardware and software specifications:

Processor Intel i7-4770

Memory 16GB

Motherboard Gigabyte Z87-D3HP

Operating System Windows 10 Pro x64

MOA 2013.11

WEKA 3.7.8

JDK 1.7.0_45

4.7 Summary

The objective of FG-CDCStream is to reduce the grain of the batch-based CDCStream in order to improve its response to abrupt concept drift while remaining robust to noise. With these intentions in mind, an evaluation strategy was proposed with justified performance measures and error estimation methods. Change detector specific performance measures as well as predictive performance measures were selected to quantify the differences between the two algorithms. The experimental design is reasonably comprehensive with a wide variety of test parameters in order to avoid bias and obtain a clear picture the relative strengths and weaknesses of the change detection strategies. It also drills down to focus on particular qualities of interest in this study. The following chapter presents the results obtained from the application of this methodology.

Chapter 5

Experimental Results and Analysis

This chapter presents and analyzes the results of the proposed experiments discussed in the previous chapter. It begins by introducing properties that had a great effect on results, trimming the breadth of the results subsequently discussed in greater detail. It continues to describe experimental results on synthetic data by change type (abrupt, gradual, both and a no-change baseline) and discusses how the change detectors perform in these scenarios in the presence of noise. This is followed by results produced from the real data sets. Finally, a discussion of the general trends in cost is presented.

5.1 Data Set Cardinality

To begin, the apparent effects of data set cardinality will be discussed. The results of the tests performed using low cardinality data sets Stagger (cardinality of three), Mixed (cardinality of four) and Argarwal (cardinality of five) show that both algorithms, FG-CDCStream and CDCStream, were unsuccessful in detecting change in every scenario tested. The tests on these data sets resulted in either unacceptably high alarm rates or extremely low alarm rates that missed detecting the true change

by several thousand instances. These results are likely caused by the algorithms' inability to select an adequate context from so few attributes. However, the algorithms were successful in locating change points in the LED data set which has a cardinality of twenty-four with seven of those attributes being relevant.

The results of this study suggest that context-based change detection algorithms using the DILCA framework for context selection should only be applied to data sets with a minimum cardinality that is somewhere in the range of 7 to twenty-four. Since the algorithms are non-functional on data sets with cardinalities less than seven, any further discussion of results omits the tests from the following data sets: Stagger, Mixed, and Argarwal. Results from the real data sets Electricity and Airlines must be analyzed with caution as their cardinalities are 8 and 7, respectively.

5.2 Batch Sizes

The effects of batch sizes on the experimental results must be discussed in order to provide context for the remaining elements of the results analysis. Figure 11 displays the averaged test results for all data sets studied, save those removed due to low cardinality. Do note that these results are a generalization particular to this study, since optimal batch size is data set dependent. As mentioned in Section 4.1.1.1, sometimes change detector statistics can be unavailable for various reasons. For example, if no change was detected, no MTD would be available. Unavailable figures, of course, could not be displayed on the graphs. Thus, little information on CDCStream in cases of abrupt drift is present in Figure 11. Note also that MTFA, MTD and MTR values represent their respective average values when values were available. It is not a true average, but is displayed with the intention of providing the reader with a general sense of what the value is when it is available. The statistics displayed are not directly comparable by change type since the data sets, number of data sets and

(a) FG-CDCStream (b) CDCStream

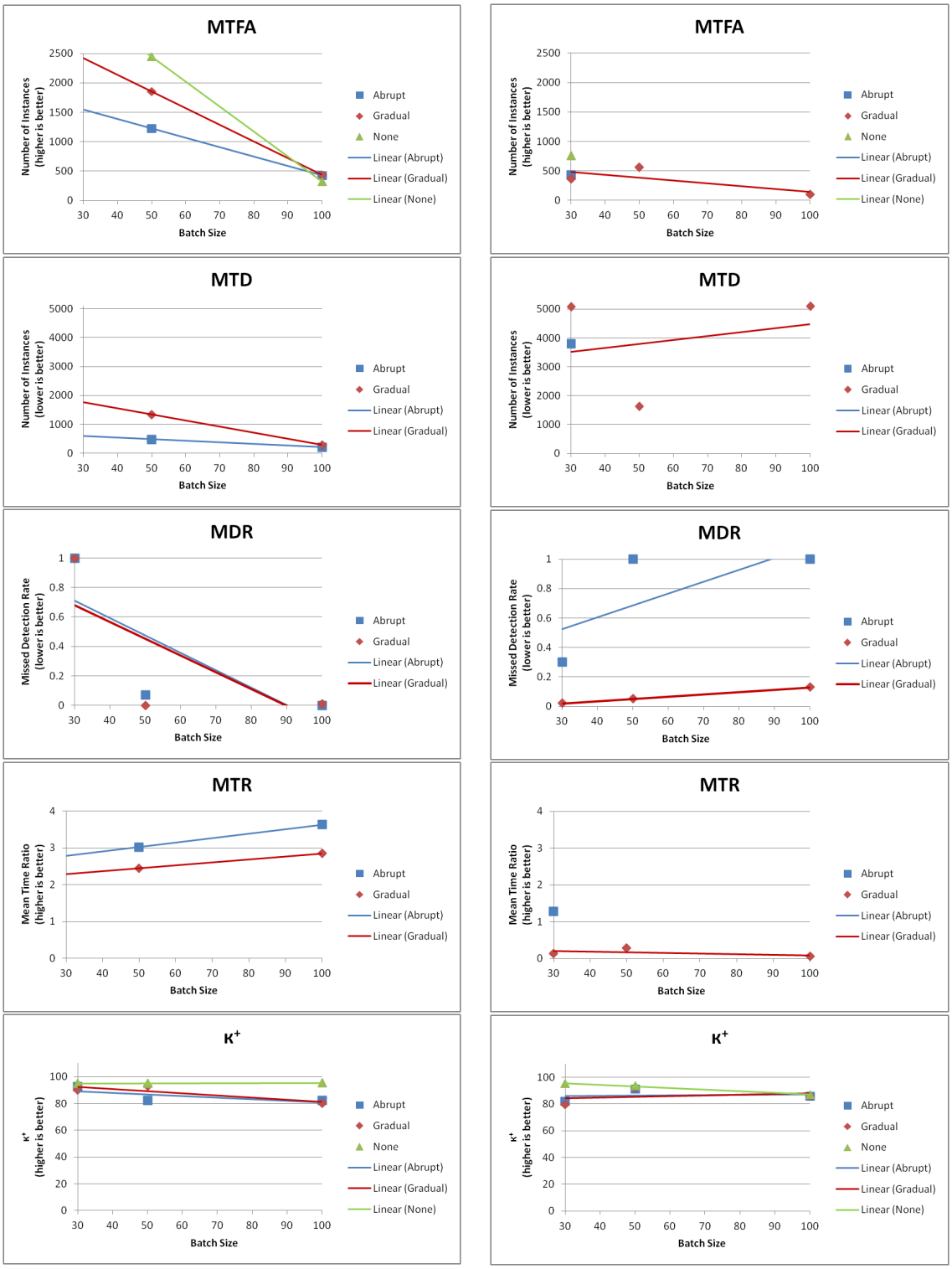


Figure 11: Trends in performance measures by batch size

available statistics belonging to each change type vary. More directly comparable results will be presented in the following sections.

First consider batch sizes of thirty. It is immediately clear from Figure 11 that this batch size is too small for meaningful aggregation. FG-CDCStream is very clearly non-functional with this batch size with an average MDR of one in both change type scenarios. It does not even produce false alarms in this case. Since FG-CDCStream employs overlapped batches (which is essentially re-sampling) and uses a voting system to detect change, it is less likely to flag small local changes in distribution as a change. In contrast, CDCStream's analysis using contiguous batches makes small local distribution differences more likely to flag change or be missed entirely, depending highly on their locations in the stream. Although in a different manner, CDCStream is also essentially non-functional in this case. Its MDR values in this case are lower than those associated with the other batch sizes but the excessively high MTD values indicate that the true changes are in fact not being detected. Rather, a false alarm is going off at some point late in the stream well after the true change actually occurred and is being counted as a successful alarm since no other changes occurred in between. It is curious that the MTFA values would be so low with such a high MTD. If many false alarms were occurring, surely one would occur closer to the true change point. Upon further examination a pattern was observed where, typically, a surge of false alarms occurs near the beginning of the stream and once again near the change detection point whilst completely missing the actual change. Interestingly, despite the dysfunction of both change detectors in this case, the κ^+ statistic remains fairly high. This case further demonstrates the importance of examining change detection statistics, not only accuracy-type statistics, when evaluating the effectiveness of change detectors.

Now consider batch sizes of 100. This batch size tends to produce relatively low MTFA values for both change detection algorithms. For CDCStream this batch

size produces unacceptably excessive MTD values, high MDR values and low MTR values. This batch size is clearly inappropriate for CDCStream. This is likely due to the high level of aggregation clouding true distribution patterns. This effect is much less prominent in FG-CDCStream presumably due to the re-sampling and voting techniques applied. However, a great drop in MTFA values is observed which making the cause of the observed low MTD and MDR values more questionable than they would be in the presence of a higher MTFA.

Finally, observe the performance statistics for batch sizes of 50. CDCStream very clearly performs best in this scenario, with a higher MTFA, MTR and κ^+ values, low MDR and significantly lower MTD values. FG-CDCStream also performs well with this batch size having high MTFA, MTR and $\kappa+$, and low MDR and MTD. With a higher MTFA, the other change detection statistics are also more reliable in this case than the case having batch sizes of 100. Given these findings, it is most reasonable to compare the performance of FG-CDCStream and CDCStream using a mid-range batch size of fifty while analysis of the results of the other batch sizes tested would provide little information of value. Moving forward, the results will be discussed in this context of batch sizes of 50.

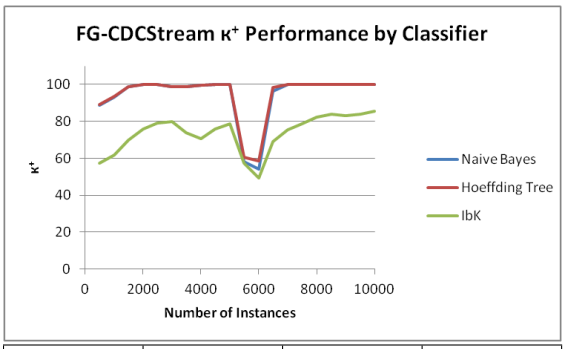
5.3 Classifiers

Recall the three different classifiers used in these experiments as discussed in Section 4.4, namely Naïve Bayes, Hoeffding Tree and IbK. Table 4 shows the average performance of FG-CDCStream and CDCStream by classifier by change type. First note that with change detection and classification being separate modules for both change detectors, the classifier used has no effect on the change detection performance statistics which are therefore left out of this visualization.

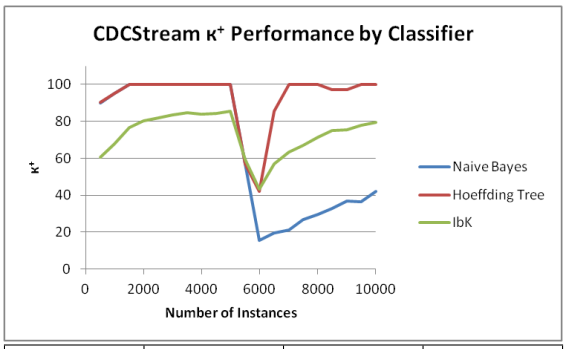
In Table 4, it is clear that regardless of change type, the Hoeffding tree tends to

Change Type	**Algorithm**	**Classifier**	$\boldsymbol{\kappa^+}$	$\boldsymbol{\kappa}$	**Accuracy (%)**
Abrupt	FG	HT	93.74	93.67	95.72
		NB	91.42	91.45	94.55
		IbK	79.93	79.92	84.55
	CDC	HT	71.70	76.28	83.81
		NB	57.58	60.05	71.65
		IbK	60.33	69.34	80.72
Gradual	FG	HT	90.36	91.38	95.19
		NB	87.92	88.90	94.23
		IbK	76.05	77.47	83.23
	CDC	HT	86.72	88.44	92.89
		NB	81.93	83.25	90.62
		IbK	68.05	71.06	78.66
None	FG	HT	80.71	78.25	91.55
		NB	80.48	78.69	91.41
		IbK	66.50	64.79	82.35
	CDC	HT	91.04	90.56	96.31
		NB	84.74	84.52	93.02
		IbK	75.92	74.96	86.69

Table 4: Average algorithm performance comparison by classifier



MTFA	MTD	MDR	MTR
1352.00	529.00	0.00	2.56



MTFA	MTD	MDR	MTR
N/A	N/A	1.00	N/A

Figure 12: Algorithm performance by classifier for the LED 0,6 data set

perform best in terms of accuracy-type statistics. Again, note that these statistics are not directly comparable by change type. Although interesting, these results are due to the inner workings of the classifiers themselves and do not reflect any information about the FG-CDCStream or CDCStream algorithms.

Of greater interest is the differences in the classifiers' performances when used in conjunction with FG-CDCStream and CDCStream. For example, the Hoeffding Tree and Naïve Bayes classifiers produce more similar results when FG-CDCStream is applied to them than when CDCStream is applied. Observe that, in most cases, the performance statistics rank more favourably for FG-CDCStream. In many cases, especially in abrupt drift scenarios, CDCStream's low ranking can be attributed to its failure to detect any change. As change detection improves from abrupt to gradual, the gap between the performances lessens. The greater gap between the performance in Hoeffding Tree and Naïve Bayes change detectors in CDCStream compared to FG-CDCStream, then, is likely a consequence of the Hoeffding Tree's superior ability to conform to streaming data naturally through data acquisition and without explicit concept drift detection. This phenomenon is demonstrated in Figure 12 which shows a typical case where CDCStream does not detect any change and its Hoeffding Tree version adapts significantly faster than its Naïve Bayes (with IbK in between). In this

scenario, FG-CDCStream does detect the true change and both its Hoeffding Tree and Naïve Bayes classifiers adapt relatively quickly.

These results provide evidence that performance measures such as accuracy and κ statistics may be heavily influenced by the ability of the incremental classifier used to naturally adapt through data acquisition, regardless of explicit change detection. Researchers must be cognisant of this when using such performance statistics to assess change detection methods. Otherwise, unfair comparisons between change detectors might be made. For example, if, from Figure 12, the Hoeffding tree classifier alone was used to compare CDCStream and FG-CDCStream, it would appear that the two algorithms perform similarly. However, it is known that CDCStream does not detect the change at all. Thus, utilizing classifiers with advanced implicit adaptation to compare explicit change detection algorithms is misleading and inappropriate. Of the classifiers studied, the least misleading accuracy-type results are produced by tests using the Naïve Bayes classifier. Even though the Hoeffding tree classifier achieves the highest accuracy-type results, its superior ability to adapt on its own makes it ill-fit for comparing change detectors. To reduce redundancy and improve the meaningfulness of the analysis of results, and for continuity with the CDCStream study [4], only results of tests using the Naïve Bayes classifier will henceforth be discussed.

5.4 Synthetic Data Streams

This section explores the behaviour of the FG-CDCStream and CDCStream algorithms when applied to data streams containing known abrupt and gradual concept drift with varying parameters. Streams containing both single and multiple (homo- and heterogeneous) drifts are explored. To ensure fair comparison, the algorithms are also applied to streams that do not contain concept drift. The effect of noise on all of the aforementioned configurations is analyzed.

5.4.1 Abrupt Drift Detection

Various scenarios within the abrupt change type were tested including single and multiple change data streams. Figure 13 displays the results for tests involving a single abrupt drift. It is immediately clear that information for MTFA, MTD and MTR performance measures are not available for CDCStream. This is because CDCStream was, unexpectedly, entirely unsuccessful in detecting change in the case of the abrupt drifting data sets tested. This is evident in the graph displaying MDR results. CDCStream's accuracy-type statistics are the same as those that would result from a Naïve Bayes classifier without any explicit change detection. The result is more detrimental in cases of greater degree of difference between the old concept and the new (the change magnitude). Observe the κ^+ graph. It so happens that the data sets resulting in the lowest final κ^+ vales are those with higher change magnitude values.

In contrast, FG-CDCStream has very reasonable change detection statistics in the single abrupt change scenario. These measures remain extremely consistent across all of the data sets studied regardless of varying parameters like change magnitude. It issues false alarms at a low rate with MTFA values around 1400. FG-CDCStream is successful at detecting the true change in every case with a detection delay of around 500. A detection delay of around 500 instances is comparable to other, less specialized, mainstream change detectors [3]. See Figure 14 for visualization of the concept change point (from the white to the blue concept), and the point at which that change is detected on average. FG-CDCStream's final κ^+ statistic is consistently 100% meaning that after a change, the system recovers completely. This, however, does not provide information about the effect on the κ^+ performance measure surrounding a change point, which is arguably more informational.

Figure 15 illustrates the κ^+ response surrounding the change points of data sets

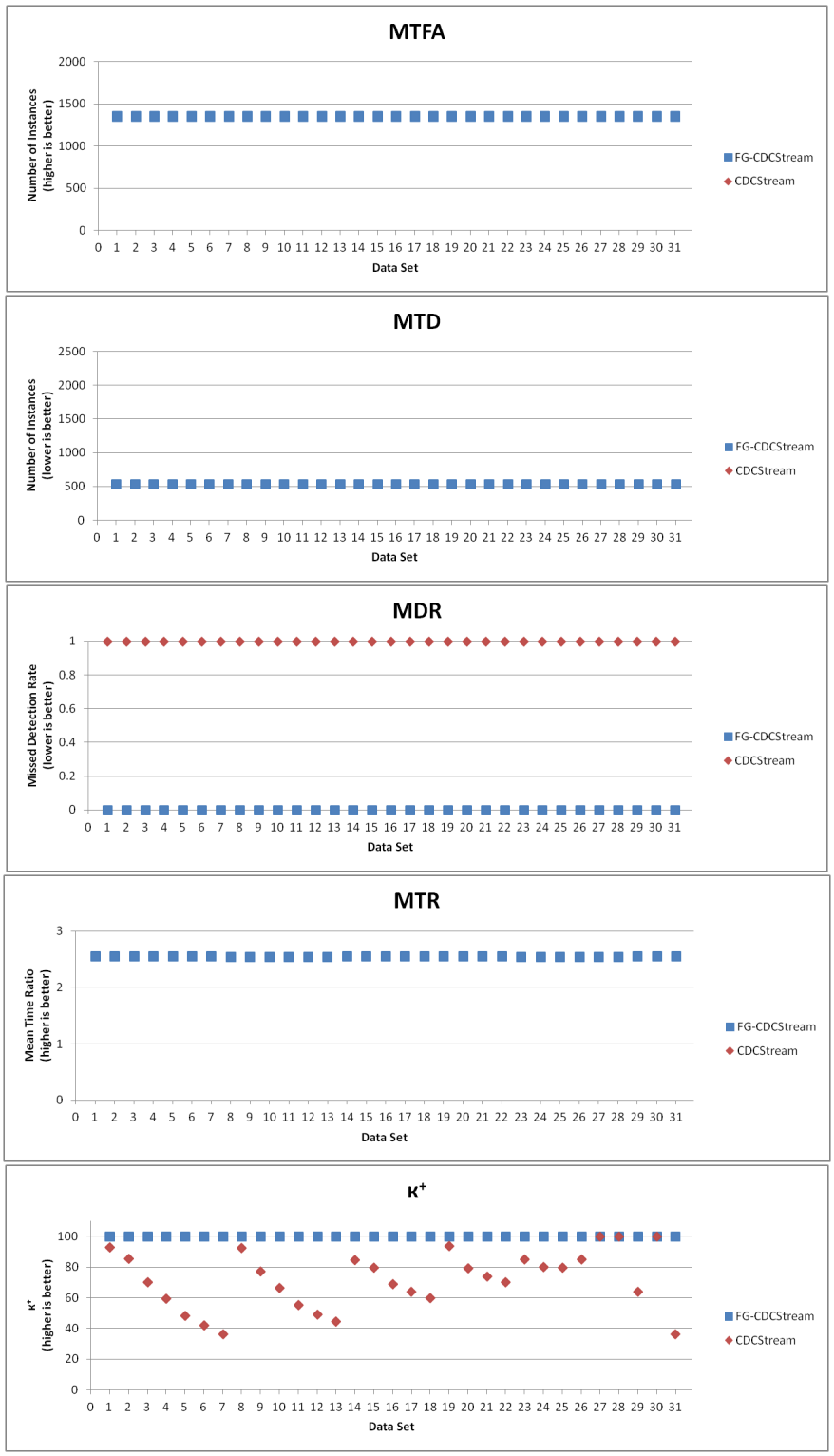


Figure 13: A graphical comparison of performance statistics of FG-CDCStream and CDCStream on 31 different data sets containing a single abrupt change each. Note that there were no calculable MTFA, MTD or MTR statistics for CDCStream in the abrupt drift scenario and therefore they are not represented in those graphs.

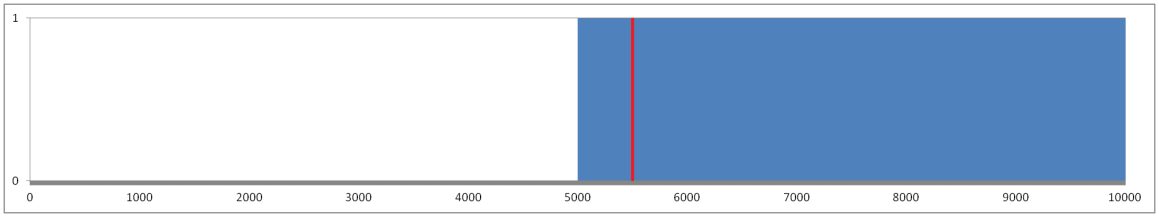


Figure 14: Average change detection location of FG-CDCStream (red) in streams containing a single abrupt change

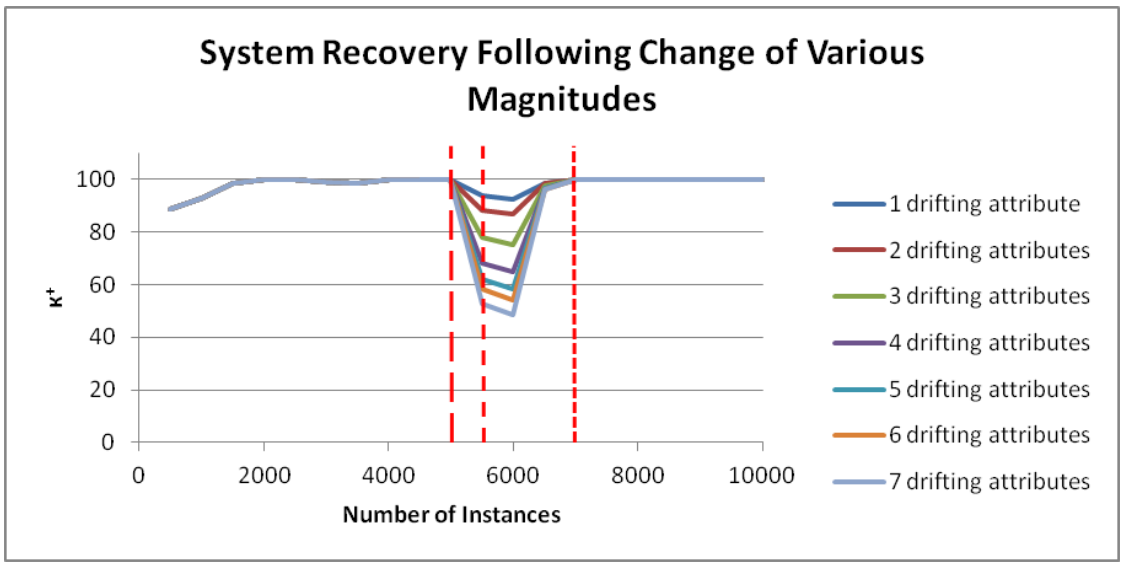


Figure 15: System recovery following a single abrupt change. The long dashed line indicates the location of the true change, the medium dashed line indicates the approximate change detection and the short dashed line indicates full classifier recovery.

containing changes of varying magnitudes. Generally, as magnitude increases, the slope between the true change and the detection point increases accordingly. Immediately following the change detection, the slope begins to even out until finally reversing and returning κ^+ to its pre-change state. Interestingly, even though κ^+ values decline to very different degrees across the seven magnitudes of change, recovery times for all magnitudes are strikingly similar; each are about 2000 instances. Requiring this amount of time to recover from a change is much less of an issue in cases of low change magnitude where the fluctuation in κ^+ is minor than in cases of high change magnitude where it is more consequential.

Consider, now, the multiple change scenario. Recall that multiple changes were injected into streams at varying distances from one another: 500, 1000, 2500 and 5000 instances. The averaged results and corresponding trends may be observed in Figure 16. Similar to the single abrupt change scenario, CDCStream failed to detect concept drift in every multiple abrupt change case. Its accuracy-type statistics then, once again, are equivalent to those of a regular incremental Naïve Bayes classifier with no explicit change detection functionality. The κ^+ graph in Figure 16 shows that as distance between changes increases, κ^+ increases only slightly for CDCStream. This slight increase is due entirely to the natural adaptation of the incremental classifier over time.

For FG-CDCStream change detector performance and distance between changes generally correlate positively, as one would expect. The greater the distance between the changes the more time FG-CDCStream has to detect and recover from change, evolving a more streamlined representation of the current concept by growing its more accurate tracks' windows of the current concept and pruning the less accurate ones. This is corroborated by the trends observed in the performance measures.

A clear increase in MTFA occurs with increasing distance between changes. This trend corresponds to the systems ability to forget more of previous concepts (discard

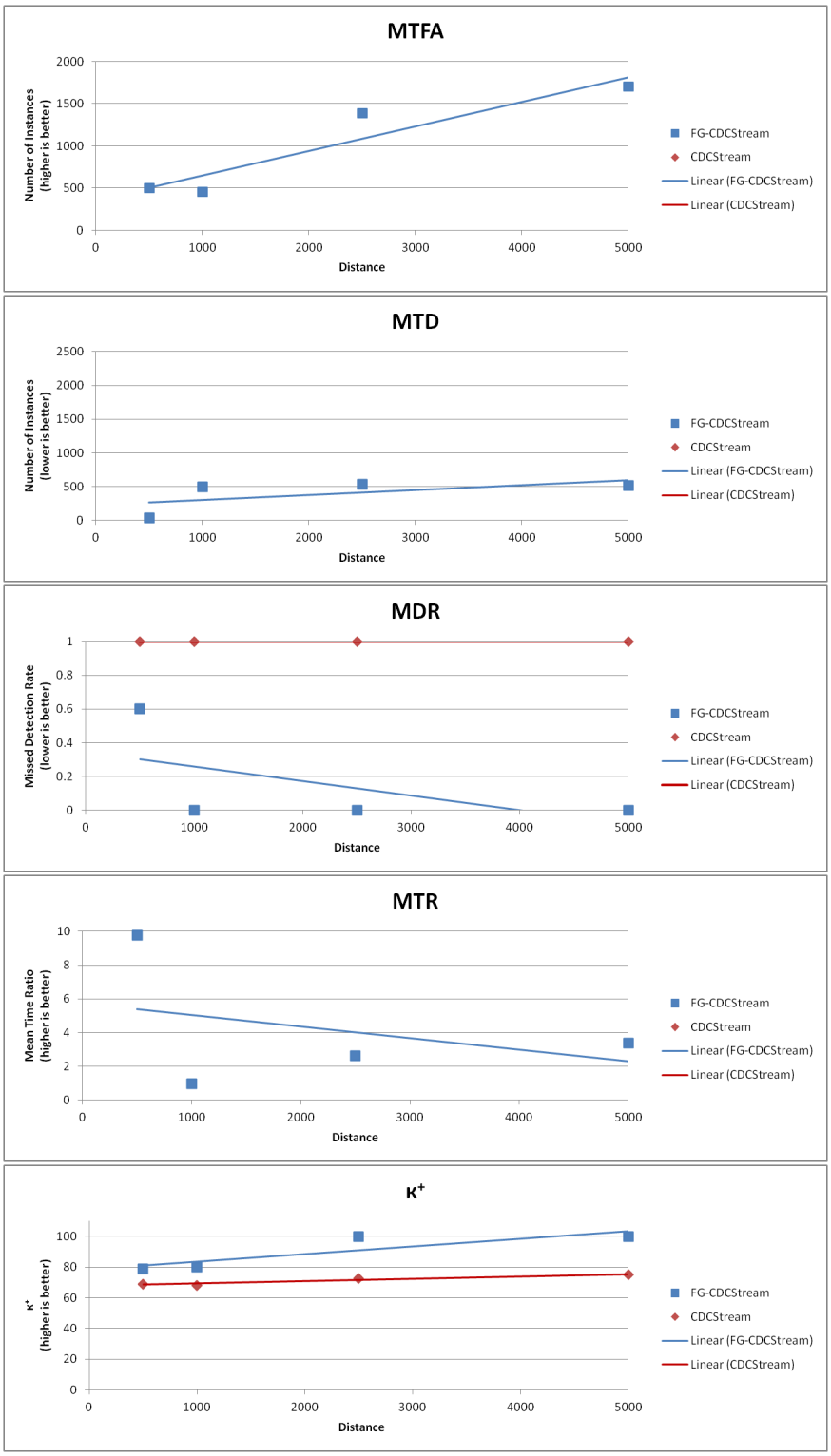


Figure 16: A graphical comparison of averaged performance statistics of FG-CDCStream and CDCStream and the associated trends as distance between the changes increases. Note that there were no calculable MTFA, MTD or MTR statistics for CDCStream in the abrupt drift scenario and therefore they are not represented in those graphs.

more of the less accurate tracks) when more time is available between changes. When less time is available, it is likely that more tracks still contain the information of previous concepts.

The average MTD values increase from a distance of 500 to 1000 and then remain fairly constant at 500 instances (the same MTD observed in the single change case). The considerably low MTD values at distances of 500 has several causal factors. First, at this distance the MTFA is very low, around 500. If a false alarm is triggered about every 500 instances, it is more likely that true changes at distances of 500 instances might be considered detected by what was in reality a false alarm occurring very shortly after the true change. Second, the values displayed are the averages of the available values and with such a high MDR (0.6) there are very few values available. Although the average MTD at distances of 500 is wonderfully low, it is actually the result of an unreliable system at this distance interval.

The MDR follows a similar pattern to that of the MTD where the value at distances of 500 is somewhat of an outlier while MDR remains quite constant for the subsequent distance intervals. At a distance of 500 about 60% of all true changes go undetected. This figure drops to 0% at distances of at most 1000 and remains 0% for all remaining distances tested.

The average MTR value at distances of 500 is very high which is largely due to the considerably low MTD values at distances of 500. Transitively, in this case, this high MTR is not indicative of a well functioning system. Although the trend line shows a negative slope, with the omission of the MTR value for distances of 500, the trend is actually positive over distance. Since in these cases MTD and MDR remain basically constant, the increase can be attributed to the higher MTFA values.

Similarly, the positive slope observed in the case of the κ^+ statistic can also be attributed to the increasing MTFA for distances of 1000 or greater. Since in these cases all changes are successfully detected and detected at around the same times,

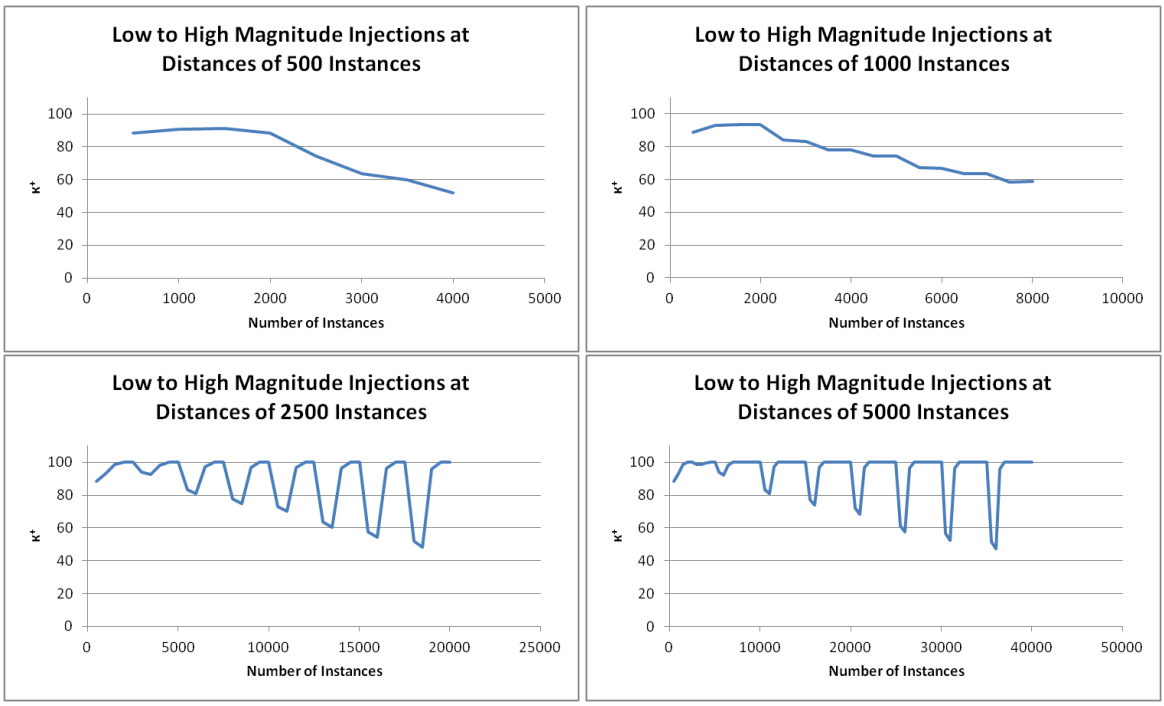


Figure 17: FG-CDCStream κ^+ performance on the LED 43526170 data set with varying distances between change injection points.

the increases in κ^+ are due to there being fewer unnecessary refreshes to the classifier. The average final κ^+ values for FG-CDCStream are consistently higher than those of CDCStream and increase significantly more as distance between changes increases.

Visualizations of how distance effects the κ^+ performance measure throughout the stream is shown in Figure 17. The particular stream represented was injected with seven changes from low to high change magnitudes. At shorter distances, change points are less defined and the system has a more difficult time recovering, especially from higher magnitude change. This is due to less reliable change detection as discussed above and therefore less representative models.

The general observation to make with regard to FG-CDCStream's behaviour in the face of multiple abrupt changes is that it is certainly not suited to handle multiple changes at distances of 500 or less. Given that a typical MTD in the single change scenario is just over 500, it is reasonable to expect poor performance when

multiple changes separated by only 500 instances. With the tested distance intervals greater than 500, for all performance measures, performance either improves or remains constant as distance between changes increases, as expected.

FG-CDCStream's superior performance to CDCStream is very dramatic in the case of abrupt drift. For every data set in the abrupt change scenario CDCStream failed to report the true change. Its accuracy-type statistics are the same that would result in the application of a regular incremental Naïve Bayes classifier on a batch size of fifty without any explicit change detection at all. Therefore, it can be confidently stated that CDCStream is not suitable for use in abrupt drift scenarios. FG-CDCStream is certainly suitable for use on streams containing a single abrupt concept drift or multiple abrupt drifts separated by at least 1000 instances.

5.4.2 Gradual Drift Detection

In the gradual concept drift circumstance where CDCStream is indeed functional. Like the abrupt change tests, streams containing single and multiple changes were tested with small (500) and large (1000) change widths. Note that the small change width has a greater static period before the full transition and vice versa for the large change width. As discussed in Section 4.3.1, distance between change points is not varied in this scenario (as was done in the abrupt drift case) since varying separation between change injection points only compresses or elongates the change period increasing or decreasing static instances between changes respectively.

Recall that change detection statistics in the gradual scenario have slightly different implications than their abrupt scenario counterparts. For example, larger MTDs are not as detrimental as in the abrupt drift scenario. Observe Figure 10c. With a drift width of 1000 instances, the transition period begins around instance 3500, but the change doesn't really start to establish until about instance 4200. Longer MTDs in this case are, to a degree, more acceptable. Also, since no false alarms may occur

during the transition period, in cases of large widths where fewer false alarms occur overall, MTFA can be expected to be slightly higher than its abrupt drift counterpart regardless of change detector functionality. MDR measurements are likely to be lower in this scenario, since there is a transition period giving the change detector more time to flag change. This effect, of course, increases as drift width increases. Overall, the gradual drift context and its implications must be steadfastly considered when interpreting change detection statistics in this scenario.

Figure 18 shows the results for the single gradual change tests with drift widths of 500. In this case, FG-CDCStream and CDCStream perform quite similarly with the main difference occurring in the MTFA measure. In terms of MTFA, FG-CDCStream produces values comparable to those which it produced in the abrupt drift scenario, though at times slightly higher as expected. CDCStream, although, impressively produces the ideal MTFA (no MTFA) while still detecting every true change presented. This difference in MTFA values was further examined to determine its true magnitude. It was found that CDCStream throws typically only one false alarm (and so has no measurable MTFA) where FG-CDCStream generally throws about three (producing a measurable MTFA). This difference is clearly a small one, as is corroborated by the striking similarities in the other performance measures (excluding MTR which is unavailable for CDCStream due to its reliance on MTFA values).

Figure 19 shows that both FG-CDCStream and CDCStream detect the true change about 400 instances after it is completely established. The performance of FG-CDCStream and CDCStream remains remarkably similar in terms of accuracy-type statistics surrounding change points of all magnitudes studied, as shown in Figure 20. From these results it may be extrapolated that in gradual drift scenarios with only a single change with a short width, the performance of FG-CDCStream and CDCStream is, for all intents and purposes, equivalent.

The performance differences between FG-CDCStream and CDCStream do become

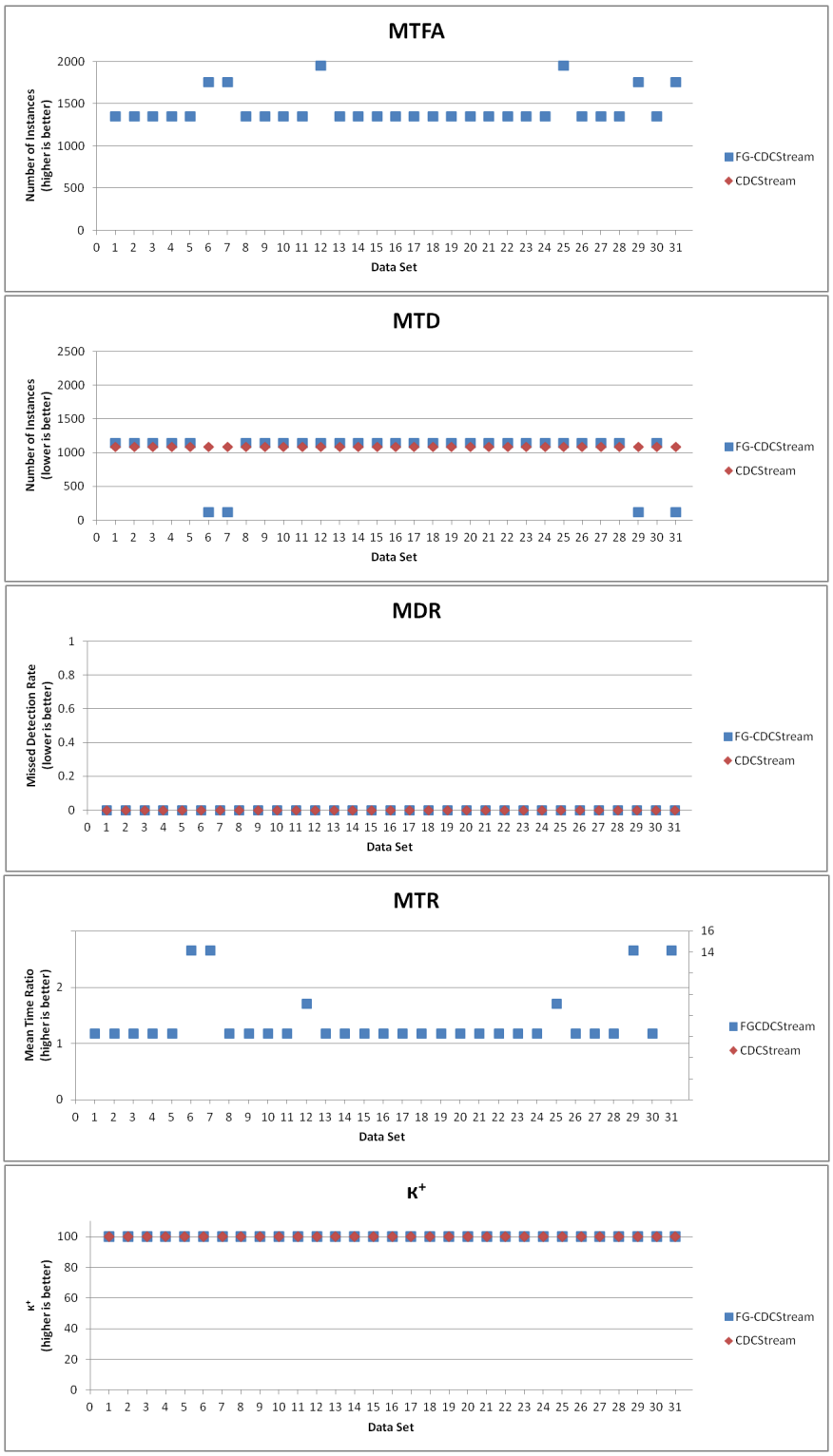


Figure 18: A graphical comparison of performance statistics of FG-CDCStream and CDCStream on 31 different data sets containing a single gradual change of width 500. Note that there were no calculable MTFA or MTR statistics for CDCStream in this scenario and therefore they are not represented in those graphs.

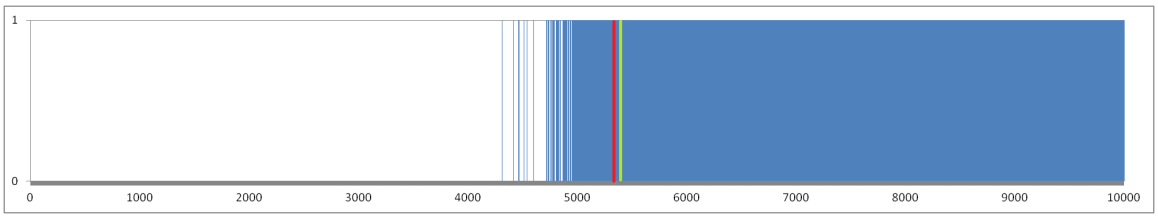


Figure 19: Average change detection location of FG-CDCStream (red) and CDCStream (green) in streams with a single gradual change of width 500

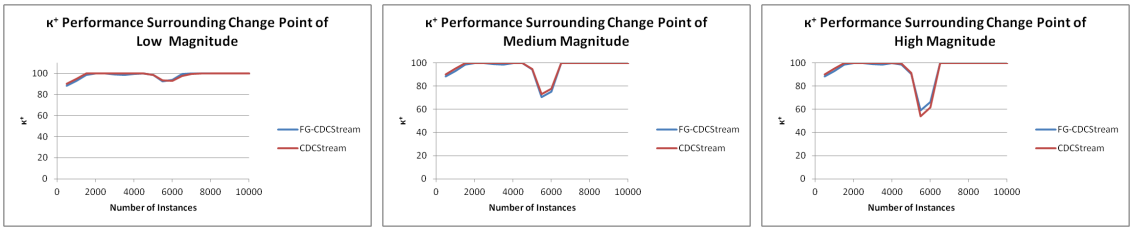


Figure 20: κ^+ performance for streams with a single gradual drift of width 500 with low (1), medium (4) and high (7) change magnitudes of datasets LED 3,4 g w500, LED 2,6 g w500 and LED 0,7 g w500 respectively.

apparent, however, in gradual drift circumstances with larger drift widths. This can be observed in Figure 21. When drift width is large, FG-CDCStream unquestionably outperforms CDCStream. For every performance measure, FG-CDCStream either meets or exceeds that of CDCStream. In this case, FG-CDCStream produces a MTFA fairly consistent with those produced in the abrupt drift and small gradual drift scenarios. CDCStream's MTFA is unacceptably low leading the classifier to update very frequently (about every 500 instances) and thus resulting slower detection of true changes.

Both algorithms are successful at detecting the true change in all cases, although CDCStream tends to produce a substantially higher MTD than that of FG-CDCStream. Figure 22 shows that, on average, CDCStream detects change about 400 instances after the full establishment of the change (as it does in the small width case) while FG-CDCStream does manage to detect the change during the end of the transition. FG-CDCStream's MTDs in this scenario somewhat higher than those

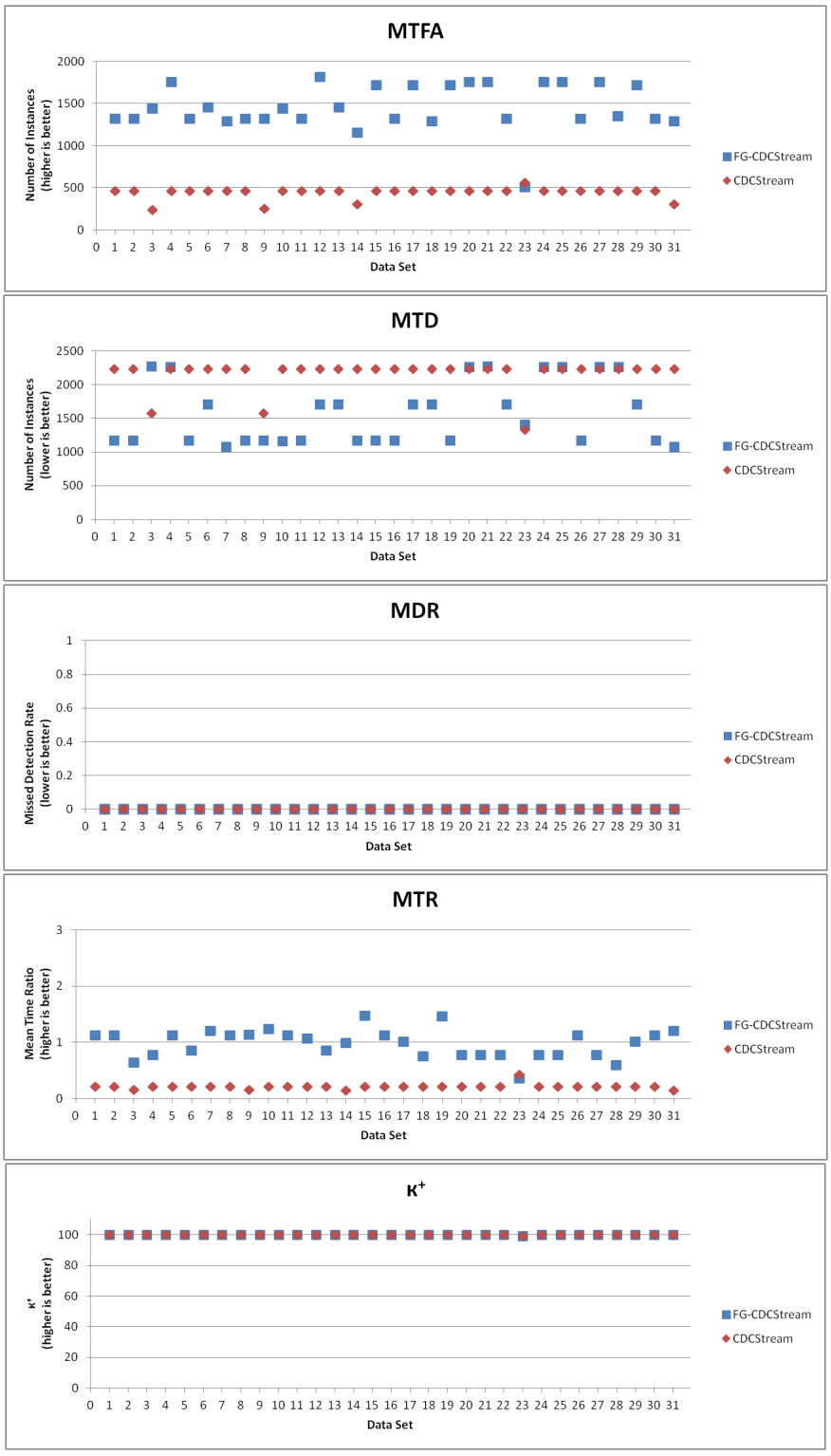


Figure 21: A graphical comparison of performance statistics of FG-CDCStream and CDCStream on 31 different data sets containing a single gradual change of width 1000

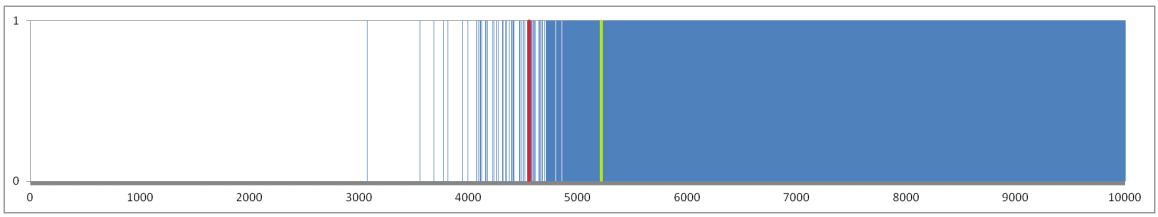


Figure 22: Change detection location of FG-CDCStream (red) and CDCStream (green) in streams with a single gradual change of width 1000

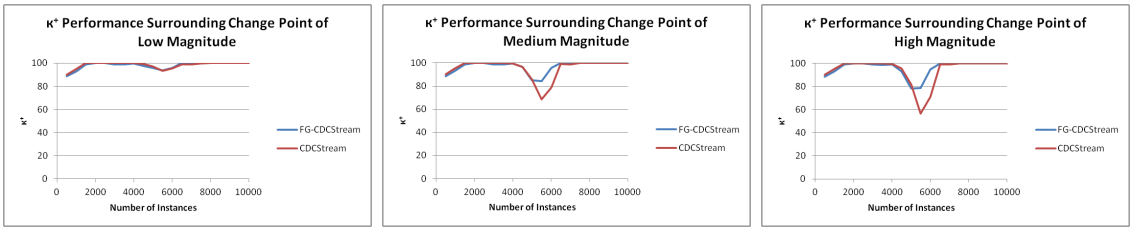


Figure 23: κ^+ performance for streams with a single gradual drift of width 1000 with low (1), medium (4) and high (7) change magnitudes of datasets LED 3,4 g w500, LED 2,6 g w500 and LED 0,7 g w500 respectively.

produced in the gradual drift, with short width scenario and even more so than those produced in the abrupt drift case. As discussed above, this is expected due to the slight change in how performance statistics are measured and it is not necessarily indicative of poorer change detector performance.

In terms of accuracy-type performance statistics, both algorithms produce essentially the same averaged final values, although their post-change recoveries do differ, as demonstrated in Figure 23. The general trend that presents itself is that, in cases of low magnitude change, FG-CDCStream and CDCStream perform similarly, i.e. they experience similar drops in performance with similar recoveries. However, in cases of drift of medium and high magnitude CDCStream tends to take a greater performance hit and recover more slowly in comparison to FG-CDCStream. Since CDCStream's recovery patterns in both low and high drift width scenarios are in fact very similar to one another, this result speaks more to the improved behaviour of FG-CDCStream.

FG-CDCStream performs reasonably well in single gradual drift cases of all drift

widths studied. It performs similarly to CDCStream in the short width case and either meets or exceeds CDCStream's performance in all performance measures in the long width case. In the long width case, CDCStream's incredibly low MTFA and very high MTD make it quite undesirable for use in this scenario.

Streams with multiple gradual drifts of widths 500 and 1000 were also studied. Figure 24 displays the results of tests for FG-CDCStream and CDCStream and the trends produces based on change width. Observe that both algorithms remain fairly consistent in terms of MTFA values regardless of change width. FG-CDCStream does produce a substantially higher, and thus more desirable, MTFA in both cases. Interestingly, both algorithms produce higher MTFA values in the multiple gradual drift scenario than the single. This may be related to there being far fewer instances in the stream that are not involved with a transition creating less opportunity to flag a false alarm.

FG-CDCStream falters considerably, though, in terms of MTD when drift widths are 500, differing substantially from the single change case. This issue is likely due to a large number of tracks resetting their change detectors after unconfirmed change and therefore prolonging a confirmed detection. Since the change period is short, the change detectors end up in varying states and taking longer to agree on the correct concept. MTD values are so high here that FG-CDCStream is not recommended for use on multiple drift streams where the drift width is 500 or less. Conversely, when there are several changes of width 1000, the longer change period allows more detectors to reach the same state faster. Fortunately, in this case MTD values do improve extensively, returning to an useful range. For CDCStream, the opposite trend occurs, though less dramatically. This is very similar to the trend observed in the single change scenario for CDCStream.

Unlike the single change scenario, however, CDCStream fails to detect many

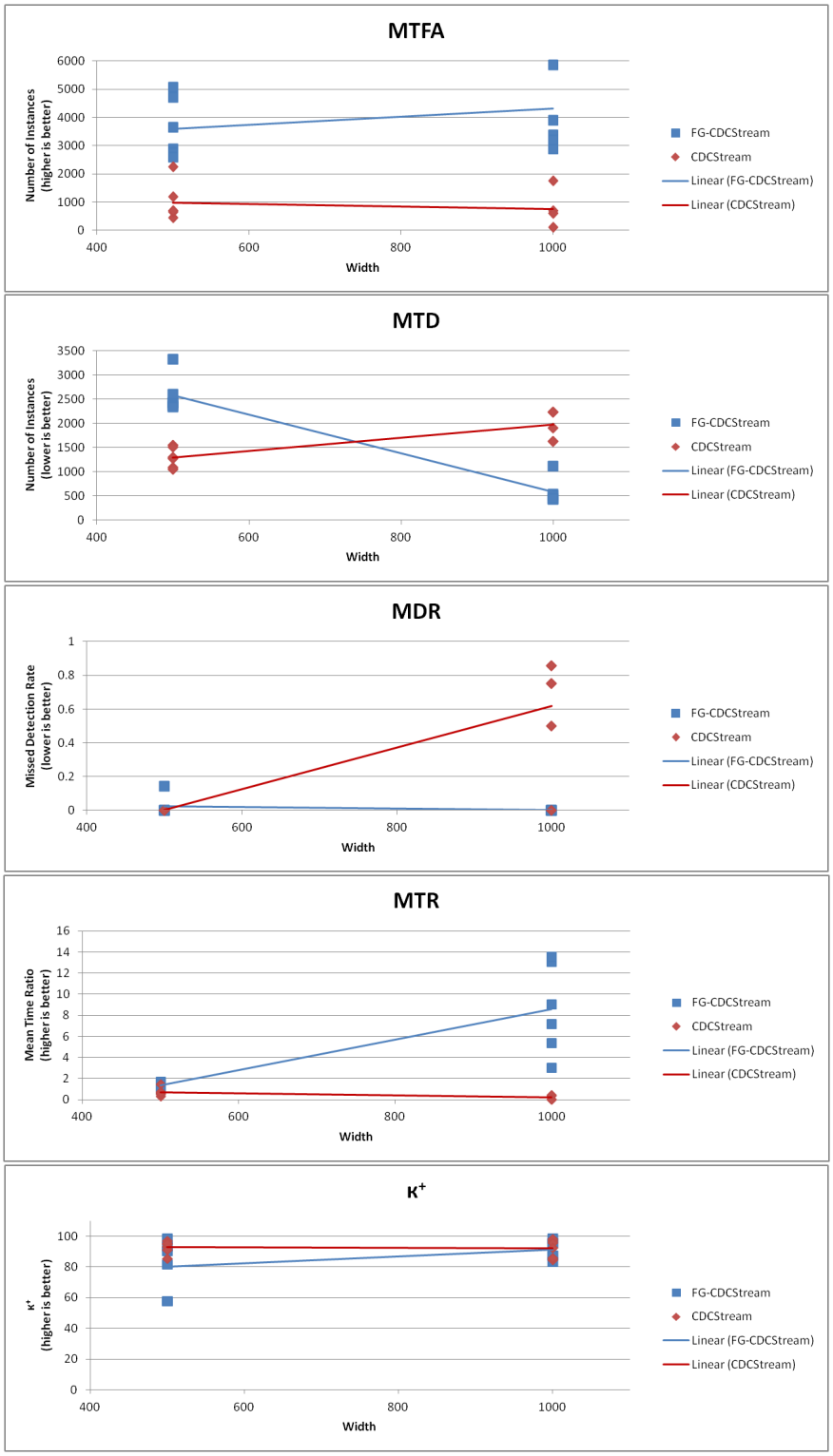


Figure 24: A graphical comparison of averaged performance statistics of FG-CDCStream and CDCStream and the associated trends as change width increases.

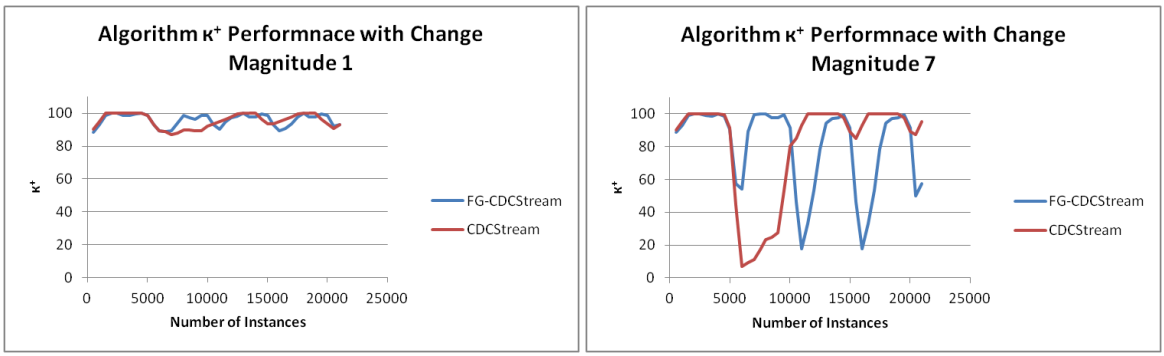


Figure 25: The κ^+ performance of FG-CDCStream and CDCStream on streams with multiple gradual drifts of drift width 500 with high (LED 07070 g) and low (LED 01010 g) magnitude.

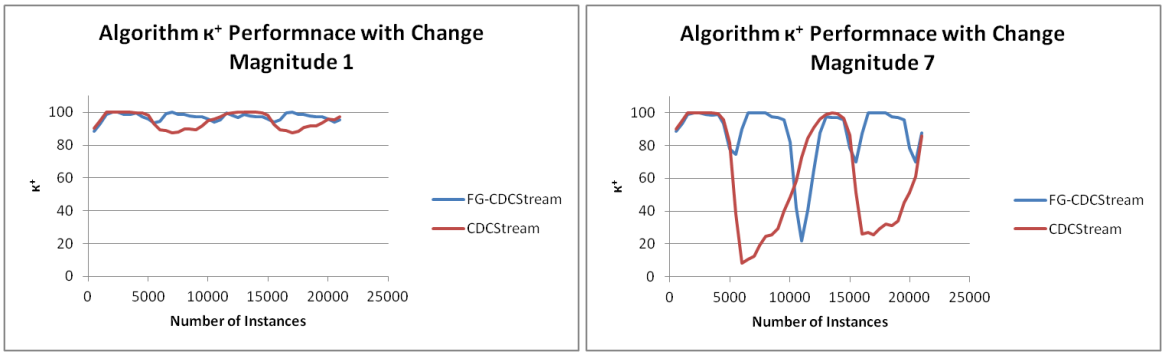


Figure 26: The κ^+ performance of FG-CDCStream and CDCStream on streams with multiple gradual drifts of drift width 1000 with high (LED 07070 g) and low (LED 01010 g) magnitude.

changes of drift with 1000 rendering it very unreliable in this case. It does successfully detect all changes of width 500. For one of the tests with drift widths of 500, FG-CDCStream did produce and MDR slightly higher than 0, but successfully detected every change in all other tests (though, as discussed above,unacceptably slow for widths of 500).

Since, for FG-CDCStream, MTD improved considerably while MTFA and MDR remained fairly constant, a significant improvement in MTR is observed from widths of 500 to 1000. CDCStream's MTR remains consistently low across the change widths studied.

In terms of accuracy-type statistics, FG-CDCStream improves from widths of 500

to 1000 while CDCStream remains consistent such that they converge at widths of 1000. For FG-CDCStream in the 500 drift width scenario, the impact of too many tracks resetting their change detectors before a gradual change is confirmed is less detrimental when the concepts are similar, as shown in Figure 25. In this case, when change magnitude is lower, FG-CDCStream produces similar accuracy-type results to CDCStream. When change magnitude is greater, FG-CDCStream has quite a difficult time recovering since its MTD is so high. When change width grows to 1000, FG-CDCStream recovers comparably well to even high magnitude drift while CDCStream, unable to detect most changes, struggles considerably to recover.

Overall, in the case of gradual concept drift, FG-CDCStream succeeds in enhancing CDCStream's change detection capabilities, generally improving MDR, MTD and MTFA measures. However, in the case of a single change of width 500 FG-CDCStream only meets the performance of CDCStream and when multiple changes of width 500 occur it tends to produce higher MTD values. In every gradual drift case studied, FG-CDCStream's robustness to change (its recovery surrounding the change point(s)) meets or exceeds that of CDCStream, as was intended in its design.

5.4.3 Multiple Heterogeneous Change Detection

Thus far, stream samples containing multiple concept changes have been homogeneous only. These streams contained only abrupt drifts or only gradual drifts of equal length. In this section, the results from streams containing multiple heterogeneous (abrupt and gradual of different lengths) concept changes are discussed. Recall from Section 4.3.1 that three different change patterns were used containing both abrupt and gradual changes in different width combinations and permutations. Let the width pattern 0, 500, 1000, 0, 500, 1000, 0 be Pattern A, 1000, 500, 0, 500, 1000, 0, 500 be Pattern B and 500, 1000, 0, 500, 0, 0, 1000 be Pattern C. Note that patterns A and C contain three abrupt changes and four gradual changes while Pattern B includes

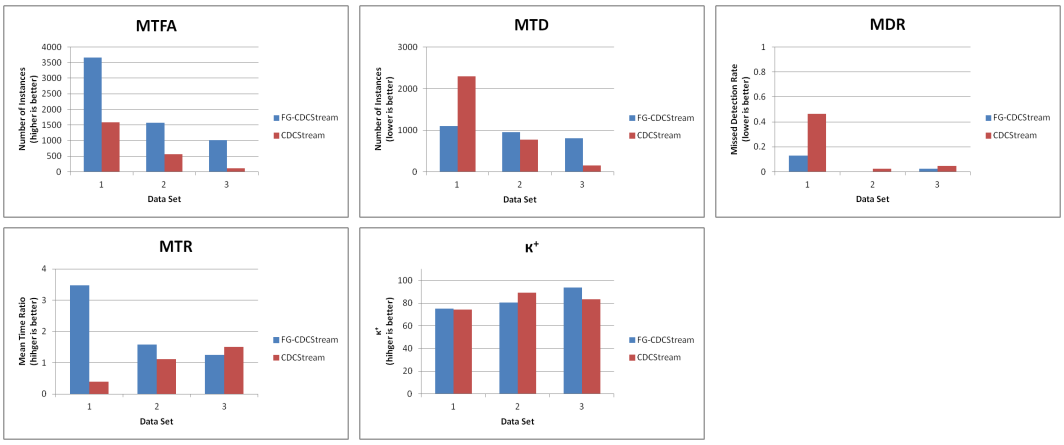


Figure 27: A graphical comparison of averaged performance statistics of FG-CDCStream and CDCStream on 6 different data sets containing a both abrupt and gradual changes arranged in three different patterns: A(1), B(2) and C(3).

only two abrupt changes and five gradual. Pattern C is the only pattern containing side-by-side abrupt changes. In streams tested with only four changes rather than the full seven, patterns are the same minus the last three changes listed.

Figure 27 shows the generalized results of the tests performed on streams containing heterogeneous changes in the patterns described. Note that, with fewer data points, a bar chart was used in place of a scatter plot to ease comparison. It is immediately clear that CDCStream performs very poorly on pattern A in terms of MDR and MTD. The system detected only about half of the changes present in the streams and with a significant delay of about 2300 instances. Its low MTR is reflective of this. Interestingly, its final κ^+ value is typically not particularly low. In fact, it is quite similar to that of FG-CDCStream which performs significantly better in all other performance measures. Observing the κ^+ behaviour over the course of the stream (Figure 28 for pattern A) shows that although CDCStream's κ^+ statistic is typically below that of FG-CDCStream over the stream (especially at higher magnitudes of change), they tend to converge after the final change point.

FG-CDCStream and CDCStream appear to perform more similarly on Pattern

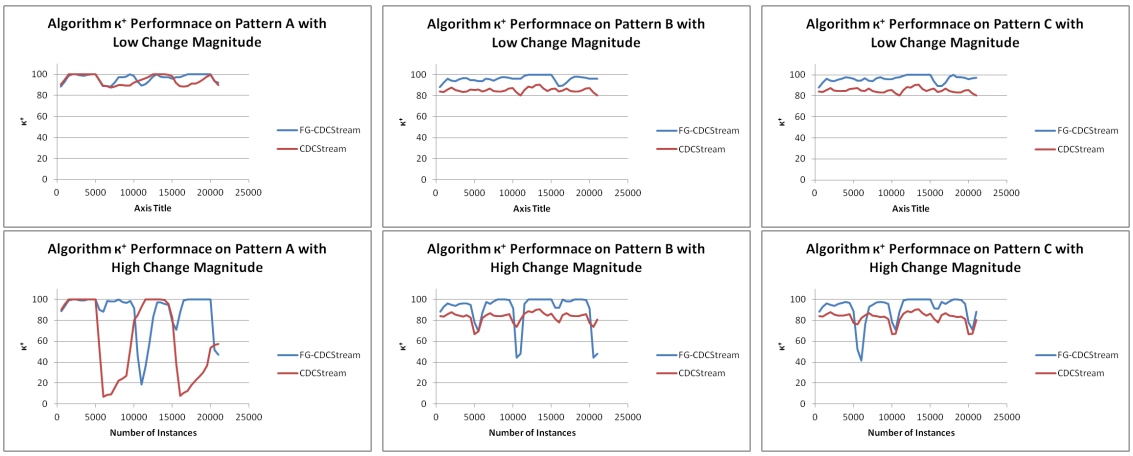


Figure 28: Algorithm κ^+ performance on the LED 01010 g (low magnitude), LED 03725 g (medium magnitude) and LED 07070 g (high magnitude) data sets using different change patterns.

B, with only two abrupt changes and four gradual, a lower ratio of abrupt to gradual change. Although FG-CDCStream does produce a superior MTFA, the other measures do not differ significantly. Figure 28 shows that FG-CDCStream typically maintains a higher κ^+ throughout the stream, but often takes a greater plunge in the face of higher magnitude changes. This is likely due to CDCStream's incredibly low MTFA causing it to update very frequently.

CDCStream updates even more frequently (significantly impeding its usefulness) when used on Pattern C. In this case, its MTFA is so low that it is effectively useless as a change detector. With updates occurring about every fifty instances (the size of a batch) it is no marvel that most changes are detected, and quite quickly. FG-CDCStream produces a much higher MTFA making the measurements of its other change detector statistics more reliable and less misleading. This case also produces FG-CDCStream's best final κ^+" value of all the patterns, likely due to the two contiguous abrupt changes near the end of the stream. Figure 28 shows that the κ^+ trends for this pattern are quite similar to those of Pattern B, but with a significantly less prominent performance drop near the end of the stream for FG-CDCStream and

a slightly greater drop for CDCStream, as expected upon encountering abrupt drifts.

CDCStream's best performance in heterogeneous streams is without question Pattern B. This is expected since this pattern contains the highest number of gradual drifts, the majority of which are of width 500. With its excessively high MDR and MTD for Pattern A and unreasonably low MTFA for Pattern C, CDCStream would not be recommended for use in these scenarios or similar ones. FG-CDCStream produced consistently more reliable results in the case of multiple heterogeneous changes. This result is consistent with what should be expected given FG-CDCStream's typically superior performance in both abrupt drift and the larger gradual drift scenarios and comparable performance in the smaller gradual drift scenario.

5.4.4 No Change Detection

Algorithm	**MTFA***	κ^+	κ	**Accuracy (%)**
FG	2448.67	100.00	100.00	100.00
CDC	N/A	100.00	100.00	100.00

* Average value when values are available

Table 5: Algorithm performance on the LED 0 data set which contains no concept changes.

Algorithm performance on data streams containing no changes was studied in order to achieve an understanding of behaviour during static periods. Unfortunately results became very limited with the removal of low cardinality data sets leaving only the LED data set with zero drifting attributes. The results of this test, though certainly not conclusive, are shown in Table 5.

In the no change scenario, CDCStream performs ideally, flagging zero false alarms. FG-CDCStream does flag a small number of false alarms resulting in a high MTFA. In Figure 29 the effect of these false alarms can be observed in the slight performance blips of FG-CDCStream around instances 3000 and 7500. Although both algorithms

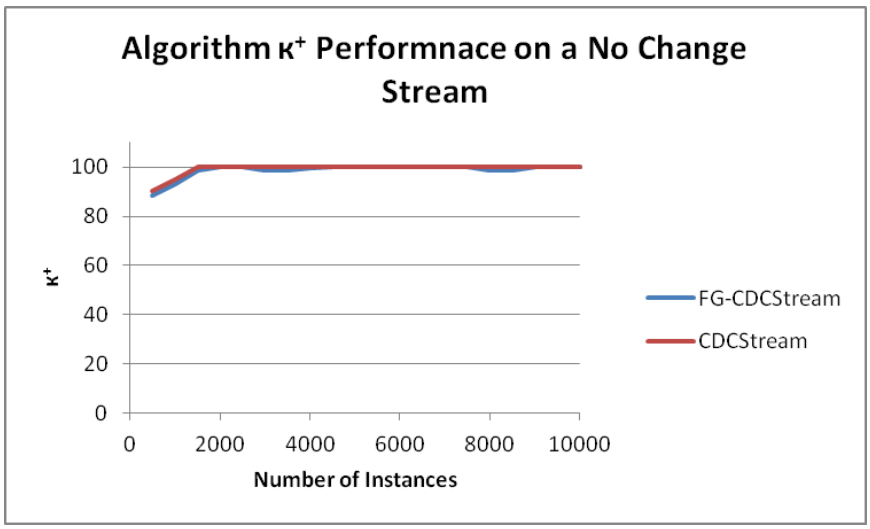


Figure 29: Algorithm κ^+ performance on the LED 0 data set which contains no concept changes.

do perform well, CDCStream performs optimally in this case. In future work, a greater number of no-change data sets should be studied in order to obtain more telling results regarding algorithm performance without any changes present.

5.4.5 Handling Noise

Given that most real data does contain some amount of noise, it is important to understand how a change detector will behave when faced with noisy data. The design of FG-CDCStream intended to improve upon CDCStream's robustness to noise. Various degrees of noise were injected into the same data sets tested thus far for abrupt, gradual, multiple heterogeneous and no changes. Recall that noise ratios were injected into each attribute of each instance, with the exception of the class attribute since change detection in this case in unsupervised. The noise percentages used are as follows: 1, 2, 3, 4, 5, 10, 15, 20 and 25. As more noise is added, the overall performance is expected to decline. That is, MTFA, MTR and κ^+ are expected to decline to some extent while MTD and MDR are expected to increase. This section discusses the results of the tests, the trends that emerge and the fitness of FG-CDCStream

compared to CDCStream in the face of noisy data.

5.4.5.1 Abrupt Drift Detection with Noise

Recall that in the abrupt drift scenario, for the data sets tested in this study, CDCStream was entirely non-functional when 0% noise was present. Given this information, it seemed of little value to test CDCStream's performance on those same data streams with increasing noise since any detections would be a result of distribution changes caused by the injected noise. In the interest of completeness, though, CDCStream was tested on the noise injected streams. As expected, any measurable results were incredibly sparse making measuring any meaningful trends impossible. It is most likely that the sparse and sporadic changes in performance measures as the noise parameter varied were a result of distribution changes caused by noise. Since CDCStream was non-functional on the noise-free data steams and since the CDCStream's results on the noise injected streams provided no usable information, CDCStream will be left out of this subsection and analysis of its performance in the face of noise will resume with the presentation of gradual drift streams.

FG-CDCStream did consistently and reliably produce meaningful results when tested on noisy data streams containing abrupt drift. Figure 30 shows the averaged results of 31 tests on streams containing a single abrupt change at each of the tested noise percentages as well as the 0% noise results for comparison. Observe that each of the performance measures does become less desirable as noise increases.

MTFA has perhaps the most drastic performance drop as it declines sharply after each attribute contains about 2% noise. It is likely that the distribution disruptions caused by the noise causes the system to fire more false alarms. The low MTFA is likely a cause of the increased MTD as noise increases. Since the system is updating more frequently and many of the tracks represent the past distribution, it takes longer to establish a true change in that past distribution. MDR also shares a positive

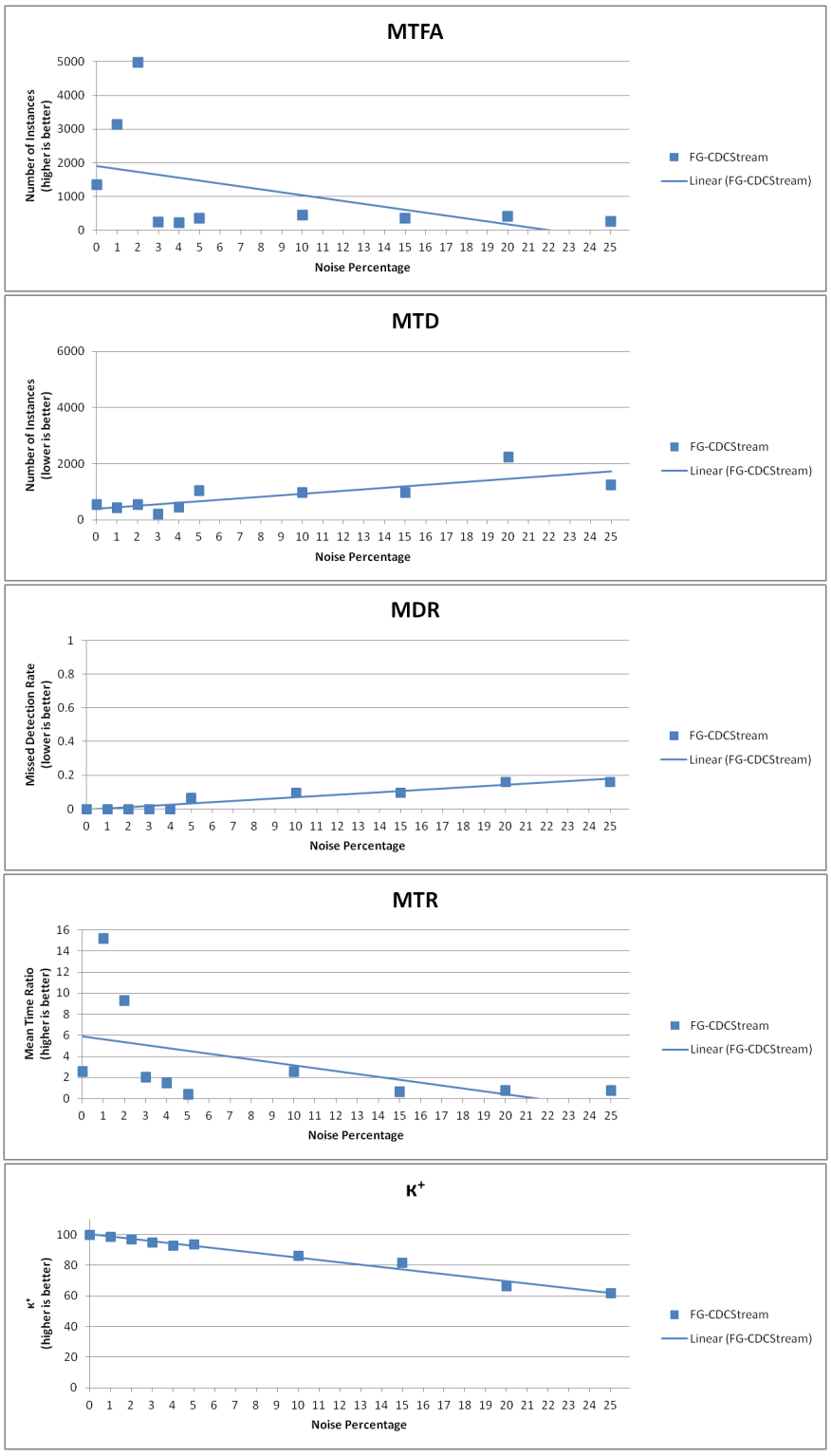


Figure 30: A graphical comparison of averaged performance statistics of FG-CDCStream on data sets containing a single abrupt change over increasing noise levels.

relationship with noise percentage. The increase that MDR experiences is from zero to about 15% missed detections, which is impressively low for having 25% noise injected in every attribute. It must be noted that luck may play a role in this low MDR given the substantial decline in MTFA observed; some of the true changes in high noise may be "detected" by a false alarm going off near a true change. However, since the rise to a higher MDR progresses slowly and evenly rather than drastically as in MTFA, it is not likely that the majority of true detections are caused by the low MTFA. FG-CDCStream's MTR declines as expected given the trends of the MTFA, MTD and MDR. FG-CDCStream's κ^+ values decline very linearly from 100% at 0% noise to finally about 60% at 25% noise.

Figure 31 displays the results for the performance of FG-CDCStream on streams containing multiple abrupt changes at varying distances from one another. In this figure, linear trend lines are shown in the absence of data markers to avoid clutter while clearly communicating the observed trends. Recall from Section 5.4.1 that FG-CDCStream performed undesirably for changes at distances closer than 1000 instances to one another and correlated positively to distance between changes. Similar trends are observed in the presence of noise.

As in the noiseless case, the highest MTFAs are produced at distances of 5000 down to the lowest at 500. In this case MTFA tends to either remain fairly constant or reduce slightly. The decline is far less overt than its single change counterpart.

In terms of MTD, FG-CDCStream tends to essentially retain its noiseless values, except in cases where distance is 5000 instances. In this case, MTD increases similarly to the single change case. Even with the incline, the average MTD value for streams with 25% change, about 1470, is high but within the range that has been observed without noise.

MDR values tend to remain fairly similar to their noiseless counterparts at distances of 5000. This trend likely occurs since there is so much space between changes

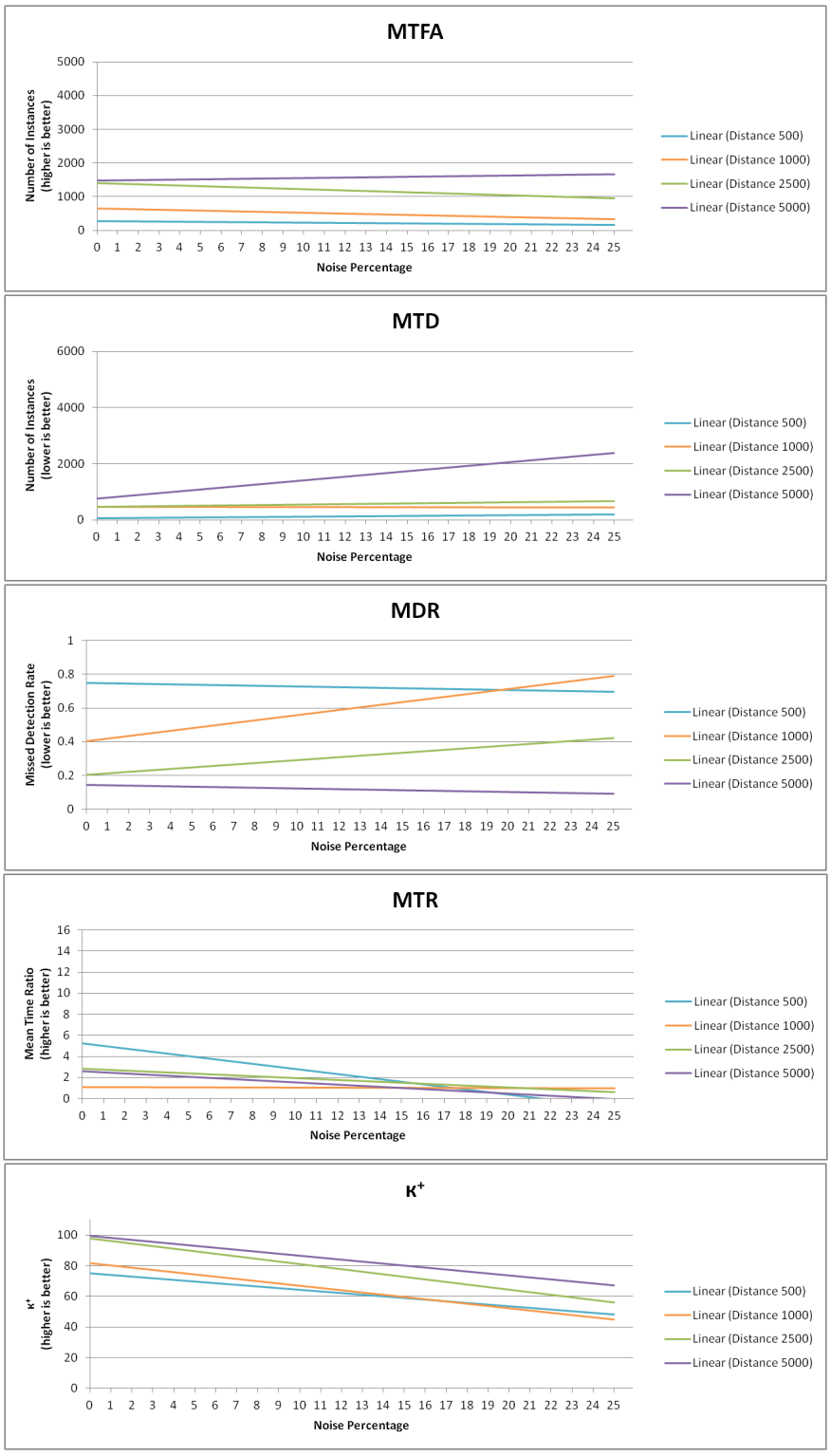


Figure 31: A graphical comparison of linear trends of the averaged performance statistics of FG-CDCStream on six data sets containing a multiple abrupt changes at varying distances from one another over increasing noise levels.

it is much easier for the system to detect them, but at a price of slower detection that comes with the noise increase. It follows that the positive slope is higher at distances of 2500 and greater still at distances of 1000. It is likely that this trend would continue for distances of 500, if not for the ineffectively low MTFA in this scenario. The retention of an incredibly low MTFA increases the chance of accidental detection due to a false alarm. This is corroborated by the typically very low MTD values observed at this distance.

At all distances, MTR and κ^+ display the expected negative correlation to noise percentage. In terms of κ^+, the slopes are slightly lower than that of the single drift scenario. This suggests that the small gains of robustness in MTFA, MTD and in some cases MDR do have their effect on accuracy-type measures.

Given the results for abrupt changes, single or multiple, FG-CDCStream would be recommended for use on streams with 0-15% change in every non-class attribute generally, although this recommendation would vary based on the application. Also, distances between changes should not average any less than 1000 and would preferably be 2500 or greater for best performance. FG-CDCStream does seem to be fairly robust to noise, especially when multiple changes are present and the distance between those changes is large. Once again, what is satisfactory would depend on the particular application. In general, though, since greater than 15% noise in every attribute is a rather high noise ratio to be occurring in any real data set, it is likely that FG-CDCStream would perform reasonably satisfactorily on most data sets containing abrupt drift of any noise level containing abrupt change. Do recall, though, that since this is an unsupervised change detector, increased noise in the class attribute would produce significant decline in accuracy-type statistics while the other performance statistics would remain the same.

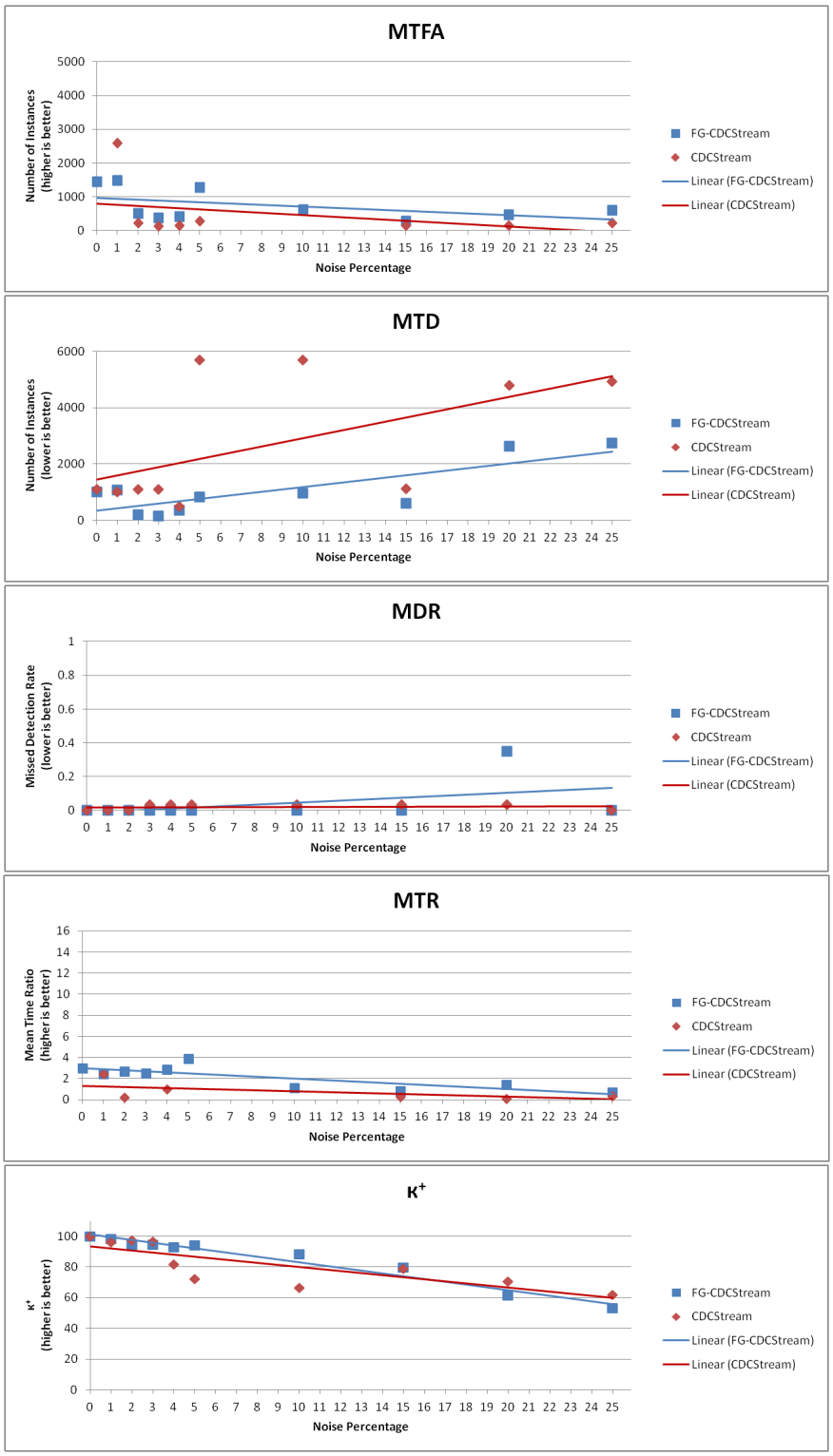


Figure 32: A graphical comparison of averaged performance statistics of FG-CDCStream and CDCStream on data sets containing a single gradual change, of change width 500, over increasing noise levels.

5.4.5.2 Gradual Drift Detection with Noise

Fortunately, as in the noiseless case, CDCStream is effective on gradual changes and so is available for comparison to FG-CDCStream in this case. Figure 32 shows how the algorithms' performance measures compare upon streams containing a single change of short width. As in the noiseless case, the two algorithms perform very similarly at this change width. A notable difference from the noiseless case is that when noise is present CDCStream does produce measurable MTFA values, and these values are very similar to those of FG-CDCStream. Also, in the presence of noise, CDCStream tends to have rather high MTD values that increase considerably after about 4% noise is introduced. These values, with the exception of the outlier at 15% noise, are far too large for an acceptable change detector. FG-CDCStream's MTD falters at a slower rate, remaining in a useful range for much longer. The sudden spike in CDCStream's MTD at 5% noise indicates quite a lack of robustness in this measure while FG-CDCStream's MTD increases much more gradually. Quizzically, this difference in MTD performance does not produce any significant difference in κ^+ in this case.

Although the two algorithms do perform very similarly in terms of averaged final κ^+ values, FG-CDCStream's superior MTD values permit it to recover more quickly from changes of higher magnitude. Examples of κ^+ performance observed throughout entire stream samples are shown in Figure 33.

In the case of single gradual change of width 500, FG-CDCStream does succeed in improving upon CDCStream's MTD as well as on robustness. For all other measures, performance was essentially equivalent for the two algorithms.

Somewhat different trends are observed in the case of single gradual change with large width. A major difference occurs in CDCStream's MTD results. Not only is CDCStream's MTD significantly lower in this case, the trend is also in the opposite

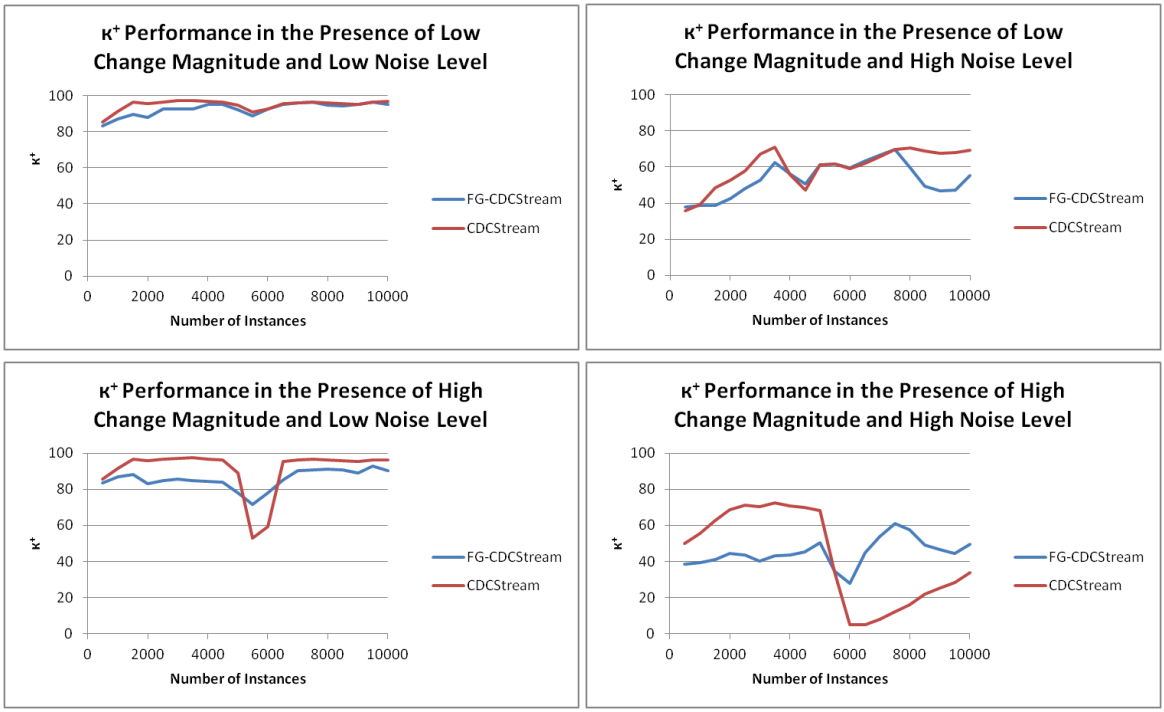


Figure 33: The κ^+ performance of FG-CDCStream and CDCStream on streams with a single gradual drifts of drift width 500 from low (LED 3,4 g) to high (LED 0, 7 g) change magnitude (top to bottom) and low (3) to high (25) noise percentage.

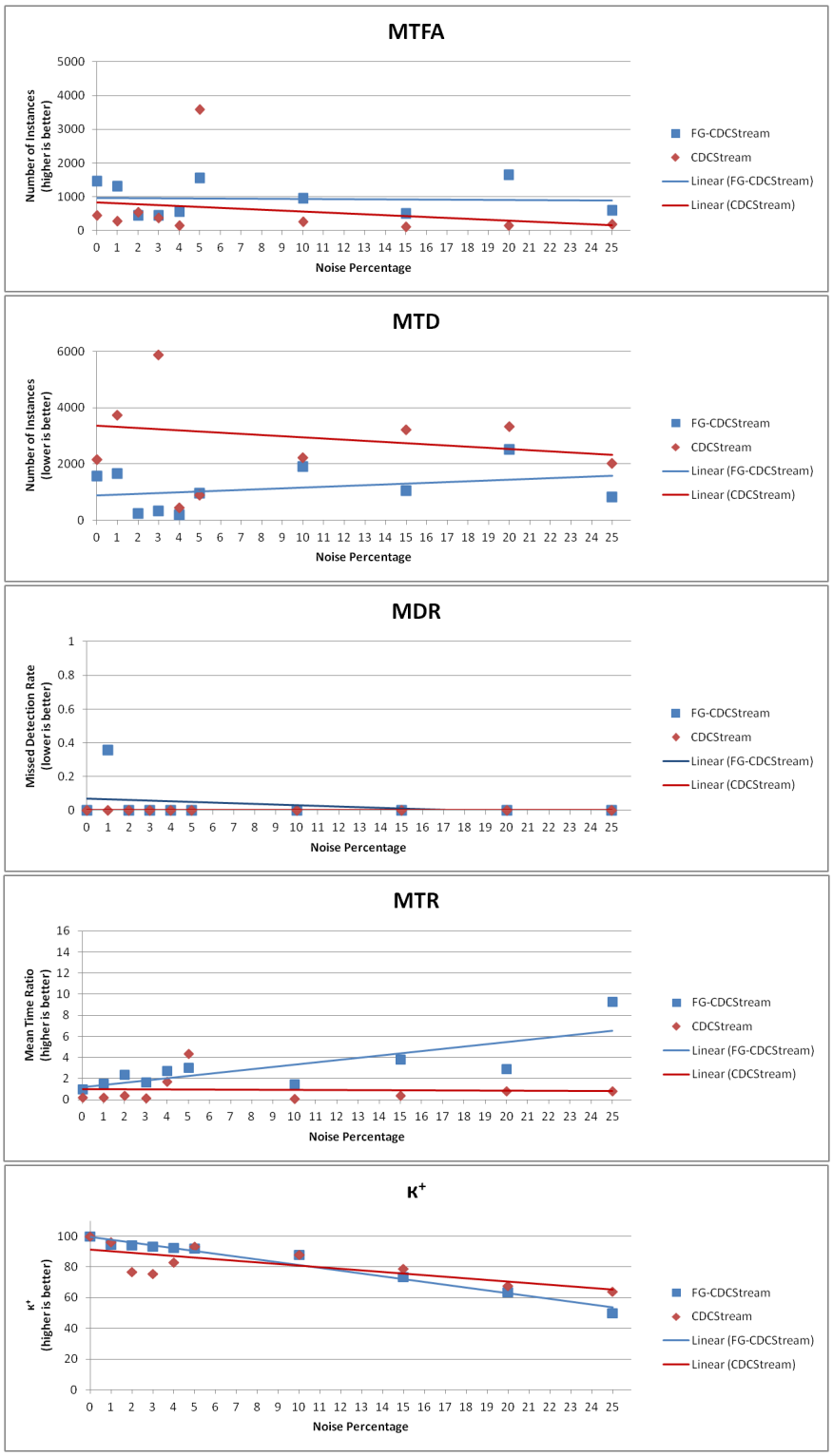


Figure 34: A graphical comparison of averaged performance statistics of FG-CDCStream and CDCStream on data sets containing a single gradual change, of change width 1000, over increasing noise levels.

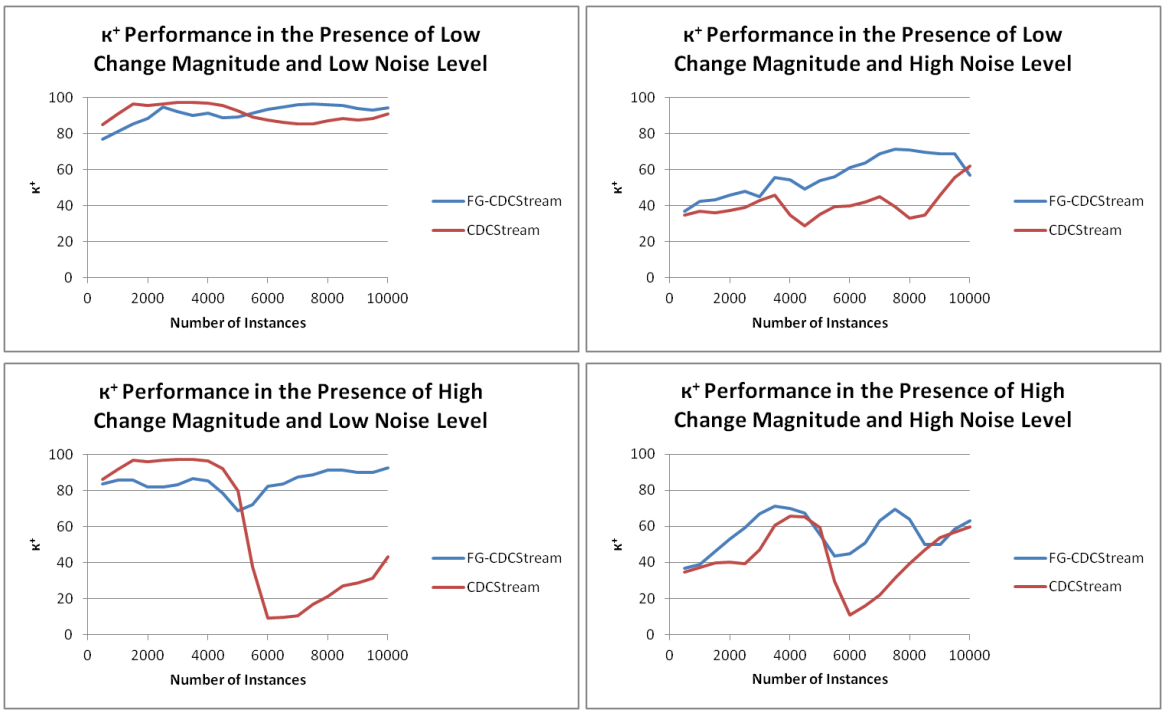


Figure 35: The κ^+ performance of FG-CDCStream and CDCStream on streams with a single gradual drifts of drift width 1000 from low (LED 3,4 g) to high (LED 0, 7 g) change magnitude (top to bottom) and low (3) to high (25) noise percentage.

direction. Given that every other performance measure for CDCStream is quite similar to those of short width, it is likely that the high MTD at 3% noise is in fact an outlier and the true trend is more neutral to positive. The difference in MTD may also be caused in part by the slightly higher MTFAs produced at this change width.

FG-CDCStream outperforms CDCStream more significantly with longer drift widths as it did in the noiseless case. In terms of MTFA, FG-CDCStream is very robust, essentially retaining its noiseless MTFA values across the increasing noise percentages while CDCStream declines to a unusably low MTFAs. FG-CDCStream's MTD values typically remain well below those of CDCStream and its MTR well above. MDR averages remain similar, save for a clear outlier where FG-CDCStream produces an MDR of about 0.35 at noise percentage of one.

As in the case where change width was 500, averaged final κ^+ for both algorithms

are very similar as noise increases, but, with change widths of 1000, tend to be even more divergent throughout the stream. Figure 35 shows some examples of how κ^+ typically transforms over the course of a stream for each algorithm. In this case, even when change magnitude is low, CDCStream tends to perform more poorly than FG-CDCStream, especially following the concept change.

In terms of single gradual drift with noise, FG-CDCStream generally meets or exceeds the performance of CDCStream, as in the noiseless case. It tends to be more robust surrounding change points and able to recover more quickly. It is also slightly more robust to noise with its performance statistics remaining in more desirable ranges for longer as noise percentage is increased.

There are some notable differences from the single gradual change to the multiple gradual change scenario. For change widths of 500, CDCStream displays a drastic increase in MTFA as noise increases. This unexpected trend can be explained by the sparse and greatly varying MTFA values produced at higher noise levels in these cases. In other words, averaging MTFA values for noise levels of 10% or higher in this case is indeed not very informative. As such, this positive trend is not particularly reliable. FG-CDCStream, on the other hand, produces results that are fairly consistent eliciting a more reliable trend line. For FG-CDCStream, MTFA decreases as expected and at about the same rate as it does in the single gradual change of width 500 case. MTFA values are greater than those in the single drift with noise scenario, consistent with the results observed in the noiseless cases.

CDCStream's MTD values increase slightly as expected while FG-CDCStream's MTD remains fairly consistent with a slight decrease showing in the trend line, likely due to the specifics of this particular sample. MDR increases as expected for both algorithms, although FG-CDCStream proves more robust in performance for this measure increasing to about 0.5 while CDCStream reaches about 0.75 at 25% noise. Both algorithms perform similarly in both MTR and κ^+. It should be noted, though,

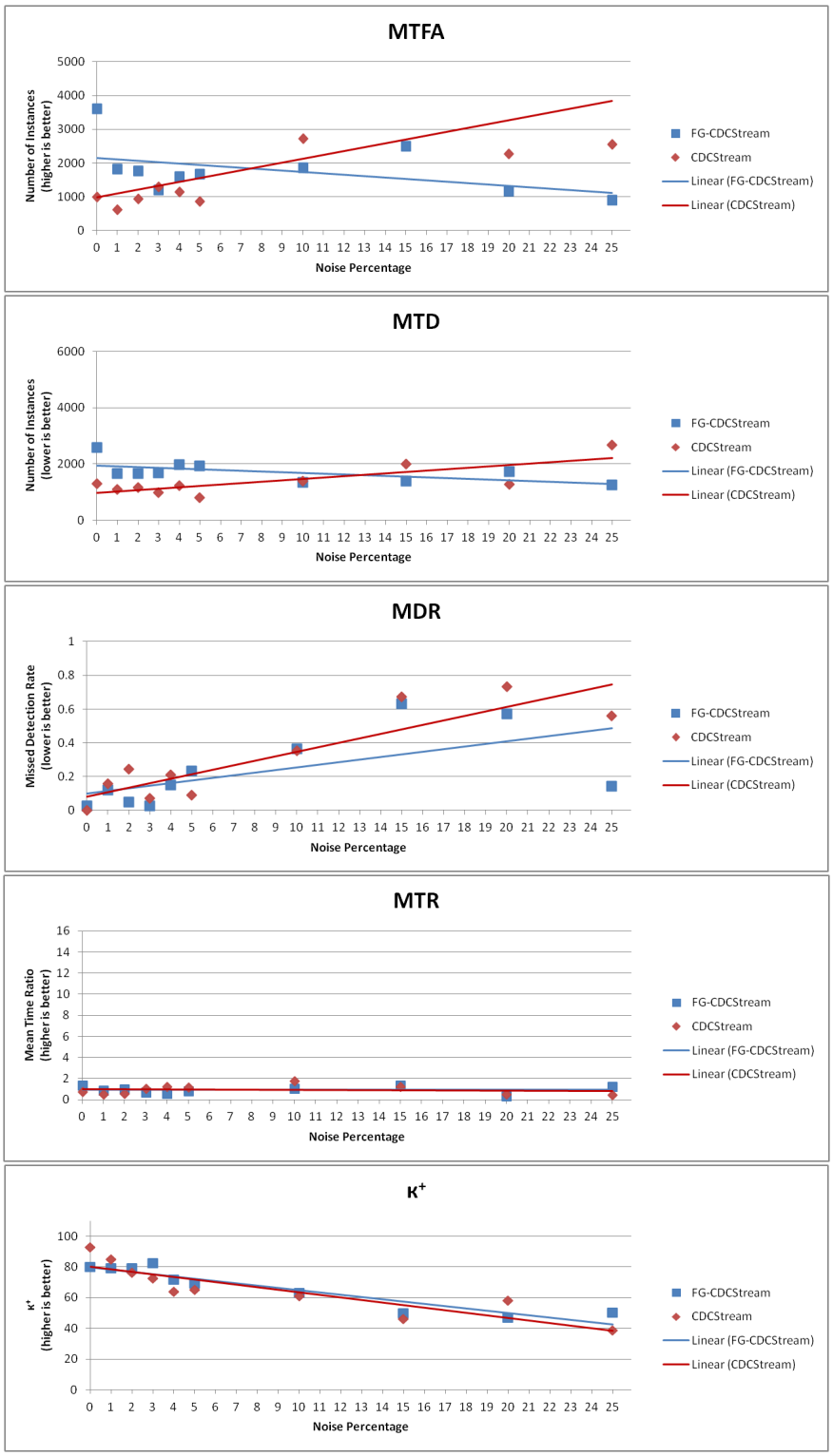


Figure 36: A graphical comparison of averaged performance statistics of FG-CDCStream and CDCStream on data sets containing a multiple gradual changes, of change width 500, over increasing noise levels.

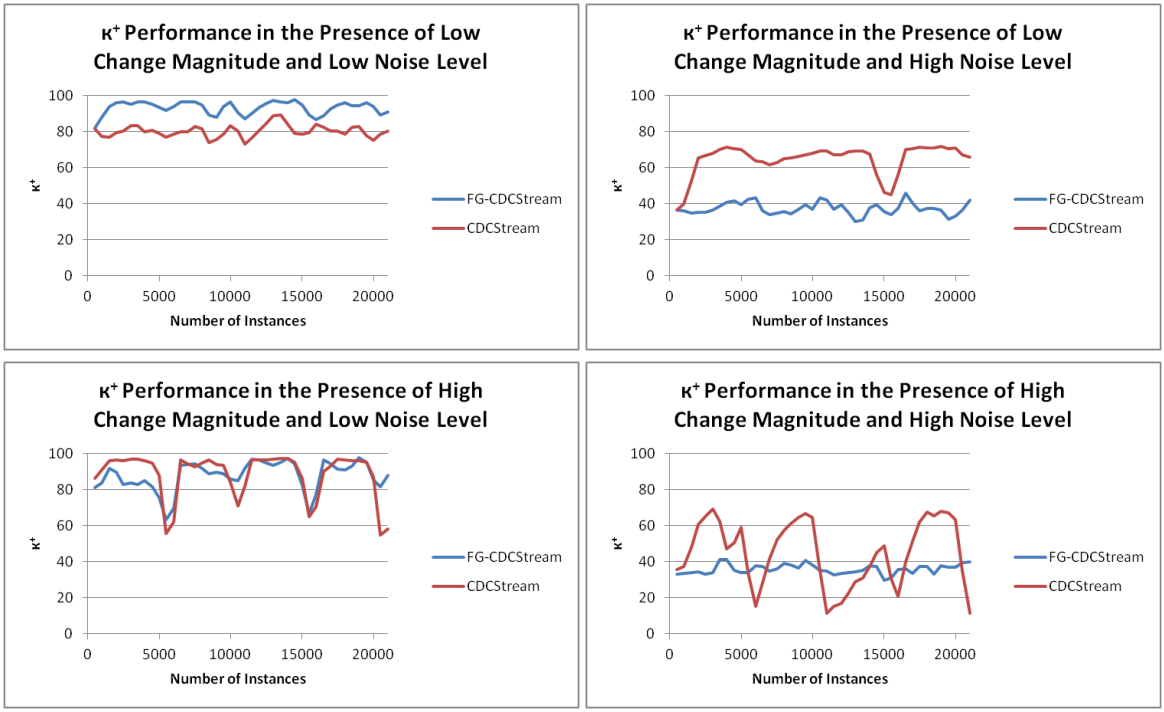


Figure 37: The κ^+ performance of FG-CDCStream and CDCStream on streams with a multiple gradual drifts of drift width 500 from low (LED 01010 g) to high (LED 07070 g) change magnitude (top to bottom) and low (3) to high (25) noise percentage.

that MTR values are quite sparse for CDCStream at higher noise levels. However, κ^+ values are abundantly available. As in the single change case, even though averaged final κ^+ values are very similar, they do tend to vary as a stream progresses, as shown in Figure 37. In this case, neither algorithm tends to consistently outperform the other, although at higher noise levels FG-CDCStream does tend to remain more robust and vary less.

At large widths, as shown in Figure 34, CDCStream's MTFA increases only slightly, which, as in the short width case, is likely related to the sparse and highly varied MTFA values produced at noise levels of 10% and higher. FG-CDCStream's MTFA decreases at a greater rate than in the short width scenario, but it still does not reduce beyond about 1000 instances.

While CDCStream remains very robust in MTD, its values are typically higher than those of FG-CDCStream, especially in the presence of lower noise levels. Note also that although MTD remains fairly consistent for CDCStream, MDR is quite high: around 0.6. In other words, by the time noise levels reach about 15% only about 40% of the changes present are being detected, but those that are are detected are so in about the same amount of time as those detected in 0% noise. FG-CDCStream, consistently through noise level increases, detects about 20% more true changes than CDCStream, and most often detects them faster. These trends lead to FG-CDCStreams higher MTR values in lower noise values, dropping off in higher noise levels due to the drop in MTFA.

Despite these triumphs, the final κ^+ values of the two algorithms remain very similar. Figure 39 shows examples of how the κ^+ measure changes over the course of a stream. Once again, FG-CDCStream tends to be slightly more robust with slightly higher κ^+ throughout the stream, especially in cases of higher change magnitude. This is likely due to the higher MDR of CDCStream.

In general, FG-CDCStream successfully outperforms CDCStream on noisy data

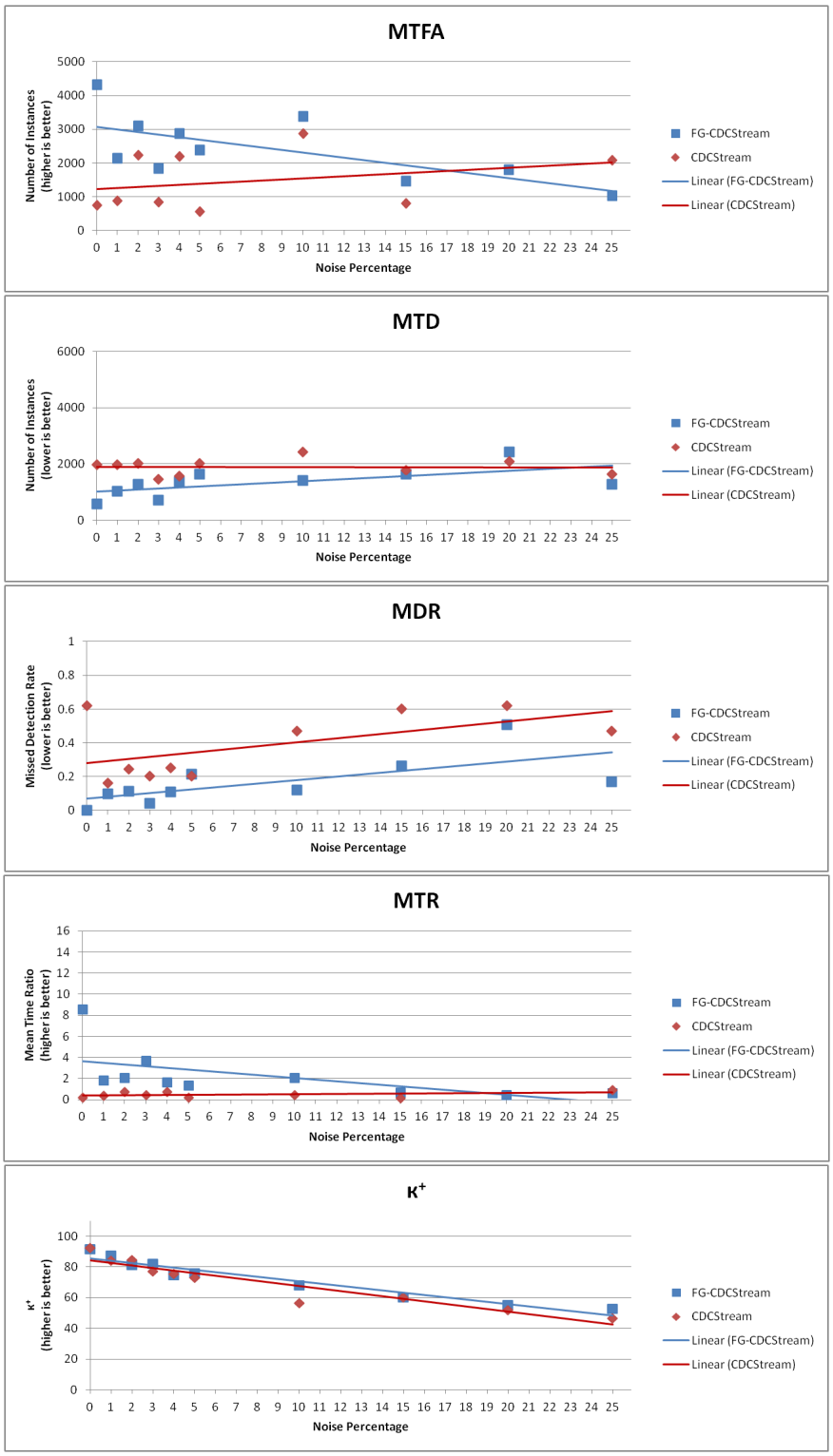


Figure 38: A graphical comparison of averaged performance statistics of FG-CDCStream and CDCStream on data sets containing a multiple gradual changes, of change width 1000, over increasing noise levels.

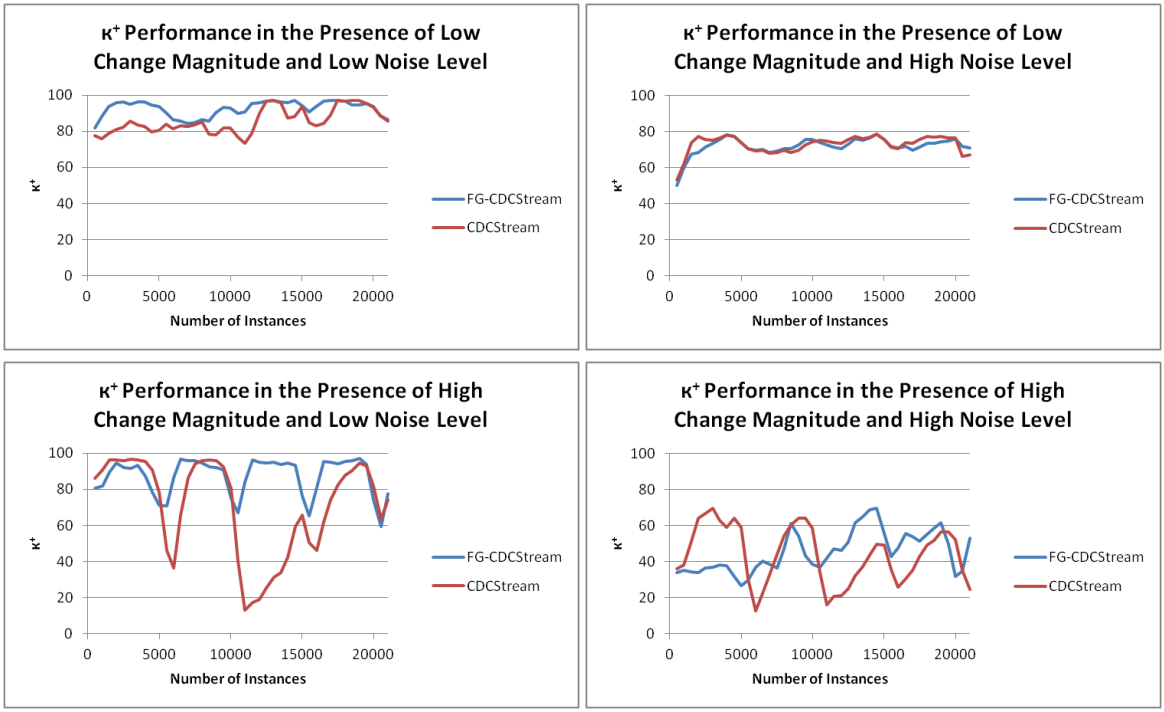


Figure 39: The κ^+ performance of FG-CDCStream and CDCStream on streams with a multiple gradual drifts of drift width 1000 from low (LED 01010 g) to high (LED 07070 g) change magnitude (top to bottom) and low (3) to high (25) noise percentage.

streams. This is most prominent in MTD in the single drift scenario and MDR in the multiple drift scenario. FG-CDCStream's robustness to noise is typically superior, especially in cases of gradual drift with large width.

5.4.5.3 Multiple Heterogeneous Change Detection with Noise

It has been observed that in the presence of multiple homogeneous changes and increasing noise, FG-CDCStream outperforms CDCStream when changes are abrupt and more subtly when changes are gradual. These results are congruous with the result observed in the noiseless scenario. Assuming that this consistency persists into the case of multiple heterogeneous changes, it could be extrapolated, in this case, that CDCStream would be moderately outperformed by FG-CDCStream. As shown in Figures 40, 41 and 42, this is indeed the case.

For Pattern A, the trends as noise increases for both algorithms are quite similar, with the exception that CDCStream produces a significantly and consistently higher MDR than FG-CDCStream. For Patterns B and C, CDCStream does produce more reasonable MDRs, similar to those of FG-CDCStream, but its MTFA values are consistently unreasonably low and well outperformed by FG-CDCStream especially in lower noise percentages. For Pattern C, FG-CDCStream also rather significantly (by about 15% by 25% noise) outperforms FG-CDCStream as noise percentage increases. This notable trend is likely due to there being two contiguous abrupt changes in Pattern C. It is probable that CDCStream's MDR values are more reasonable for Patterns B and C since its MTFA is so incredibly low. With such a low MTFA it is likely that several true changes were considered detected by a false alarm occurring shortly after the true change. The frequent updates caused by a low MTFA would also likely inadvertently keep MTD values low.

In the case of multiple heterogeneous changes, FG-CDCStream's performance measures responded to noise by decreasing in performance as noise increased. The

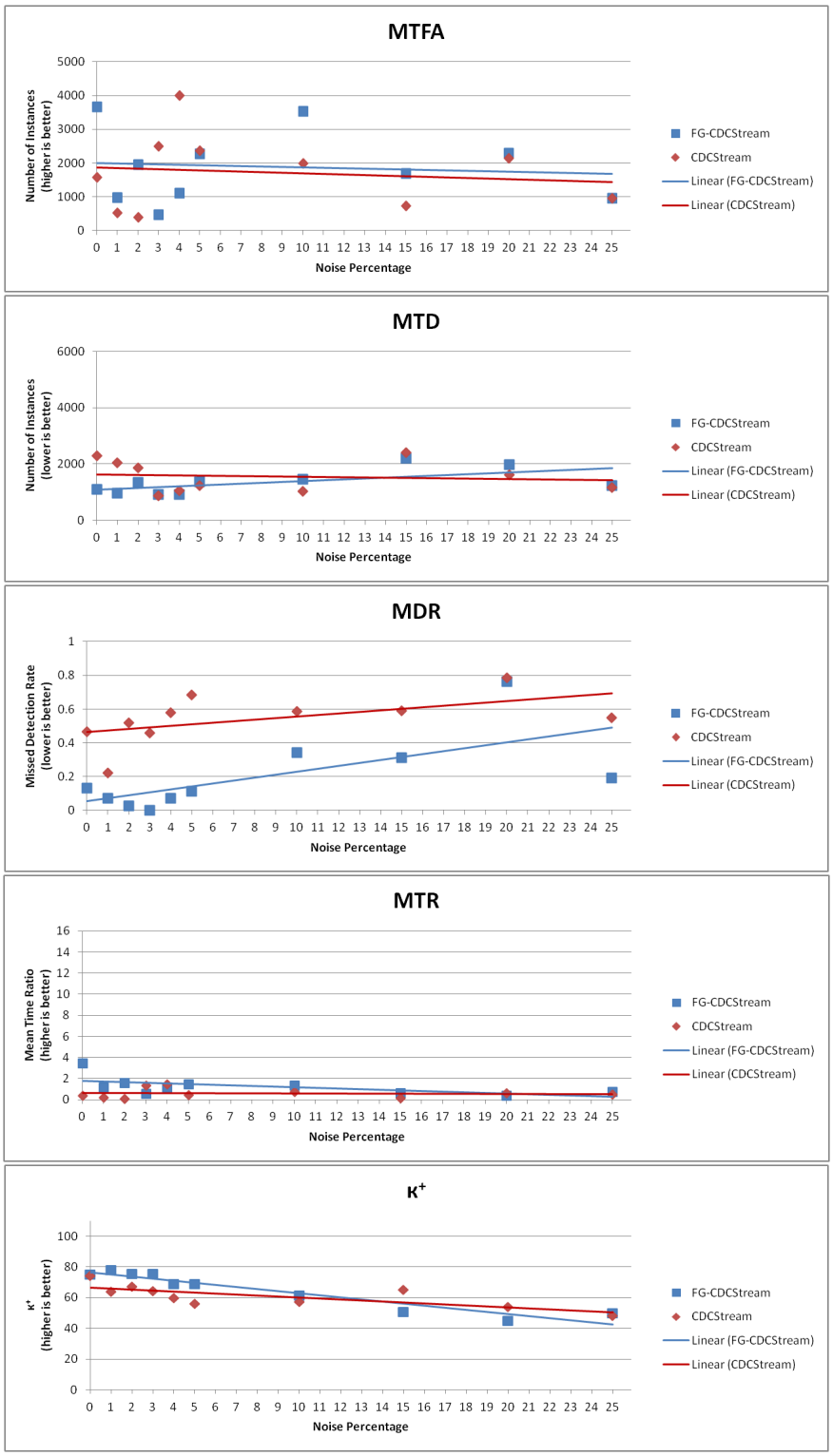


Figure 40: A graphical comparison of averaged performance statistics of FG-CDCStream and CDCStream on data sets containing a multiple heterogeneous changes, in Pattern A, over increasing noise levels.

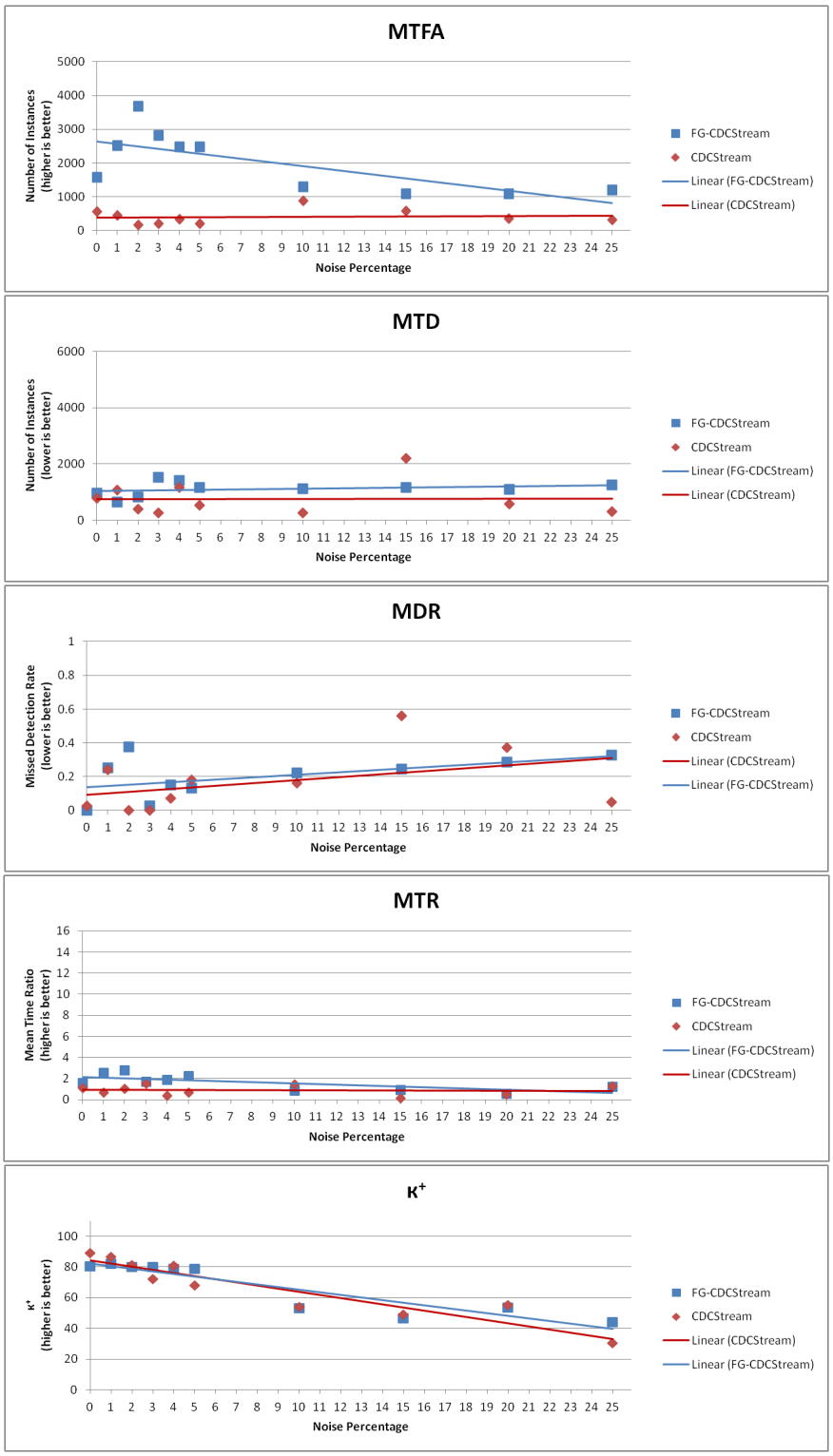


Figure 41: A graphical comparison of averaged performance statistics of FG-CDCStream and CDCStream on data sets containing a multiple heterogeneous changes, in Pattern B, over increasing noise levels.

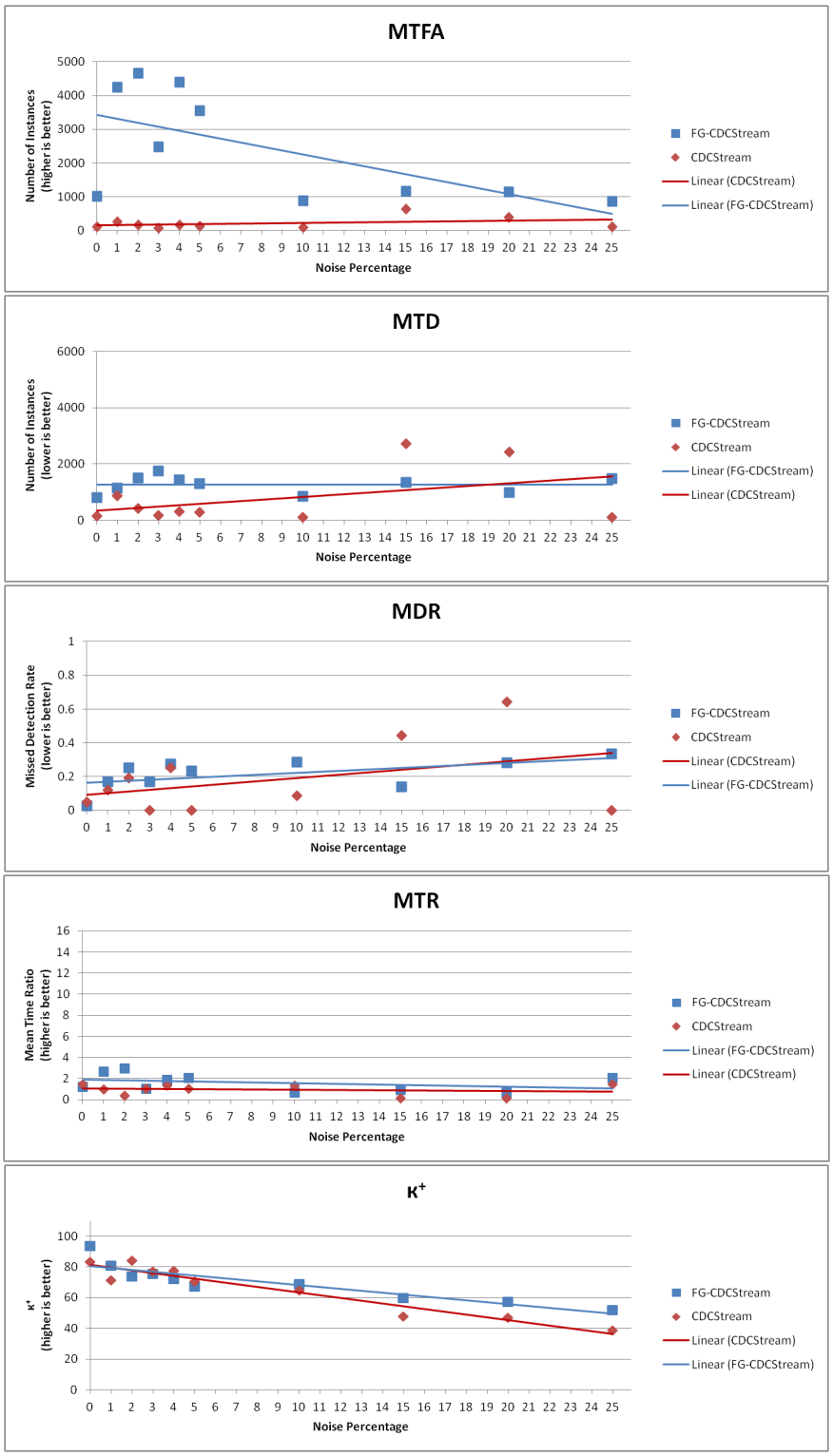


Figure 42: A graphical comparison of averaged performance statistics of FG-CDCStream and CDCStream on data sets containing a multiple heterogeneous changes, in Pattern C, over increasing noise levels.

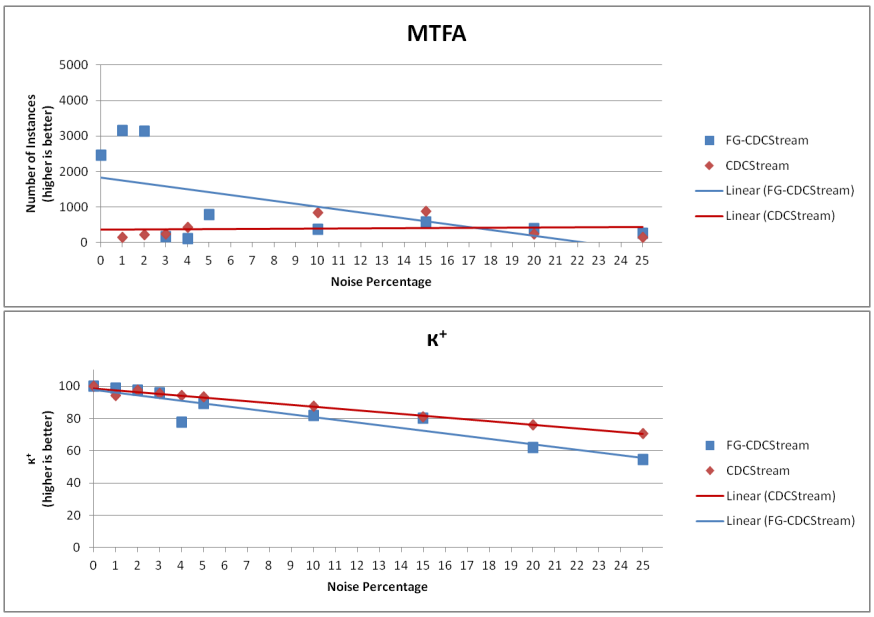


Figure 43: A graphical comparison of averaged performance statistics of FG-CDCStream and CDCStream on data sets containing no changes, over increasing noise levels. Note that since no changes occur, the only performance statistics available are MTFA and κ^+.

declines were typically not drastic and were also typically smooth. CDCStream is generally outperformed by FG-CDCStream in terms of change detection performance statistics as well as robustness.

5.4.5.4 No Change Detection with Noise

Recall that in the noiseless case, CDCStream produced very few false alarms when presented with a data stream segment containing no change. Curiously, when even as little as 1% noise is added, CDCStream produces very frequent false alarms, as shown in Figure 43. FG-CDCStream, however, does tend to retain a higher MTFA until about 10% noise at which point the two algorithms behave quite similarly. For data streams with lower noise levels, FG-CDCStream seems to outperform CDCStream with its higher MTFA and similar κ^+ values while in streams of greater noise levels

Data Set	**Algorithm**	**MTA**	$\boldsymbol{\kappa^+}$	$\boldsymbol{\kappa}$	**Accuracy (%)**
Electricity	FG	49.75	-91.55	45.45	49.75
	CDC	51.78	-95.07	44.33	51.78
Covertype	FG	250.19	-495.00	63.11	88.10
	CDC	51.92	-5.00	91.20	97.90
Covertype Drift	FG	689.44	98.74	99.33	99.70
	CDC	394.35	99.16	99.55	99.80
KDD99	FG	1650.95	67.29	68.58	84.20
	CDC	539.57	67.49	68.77	84.30
Airlines	FG	11879.89	21.83	29.49	64.90
	CDC	339.98	21.38	29.12	64.70

Table 6: Average algorithm performance in real data streams

CDCStream seems to outperform FG-CDCStream with its similar MTFA and superior κ^+ values. Recall that only one data set was used to acquire these trends and as such they are not to be taken as definitive, but rather as an example.

5.5 Real Data Streams

The performance of FG-CDCStream and CDCStream on real data streams was studied in order to shed light on their behaviour under real world circumstances. The same real data sets as used in the CDCStream study [4] were considered. Since real world data typically lack change information, the information gained from the results fo the tests on these data sets is limited.

The results from the real data sets in this study are presented in Table 6. Since the locations of true changes are unknown, Mean Time between Alarms (MTA) will be used in place of MTFA. Note that the accuracy results listed for the CDCStream algorithm vary from those reported in [4] due to the different evaluation methods

used. This study used a prequential windowed classification performance evaluator while [4] used an interleaved test then train basic classification performance evaluator. When tested using the same evaluation method as [4] CDCStream does yield precisely the same results as that study.

From Table 6, it is clear that FG-CDCStream consistently produces fewer alarms. In the case of the Covertype Drift, KDD99 and Airlines data sets it appears that FG-CDCStream's higher MTA is more appropriate than the lower MTA of CDCStream since they produce similar accuracy-type statistics, although this inference cannot be truly verified.

The electricity data set produces quite poor results for both classifiers. Observe that the κ^+ statistic is negative. This indicates that the learning system performs significantly worse than the No-Change classifier when accounting for temporal distribution. The success of the No-Change classifier on this data set is likely due to the short time intervals (half of an hour) in between data points. It is likely that, for the most part, electricity prices would continue to increase or decrease as they did in the previous half hour. In κ^+ as well as the other accuracy-type statistics, both algorithms perform similarly for this data set.

It is unclear which MTA is more appropriate for the Covertype data set, however. Whether the κ^+ statistic can be relied upon in this case is questionable since the minimum value should be -100.00 and the result of FG-CDCStream is much lower. In light of this issue, the κ^+ statistic will not be discussed for this data set. The extremely low MTA produced by CDCStream in this case does yield high κ and accuracy statistics. This may be a result of good change detection or just a result of updating often. It seems unlikely, however, that true changes would occur about every 50 instances for this data set. FG-CDCStream does produce a higher MTA value, but significantly lower κ and accuracy statistics. This may be due to poorer change detection or to the slower update rate or some combination of both.

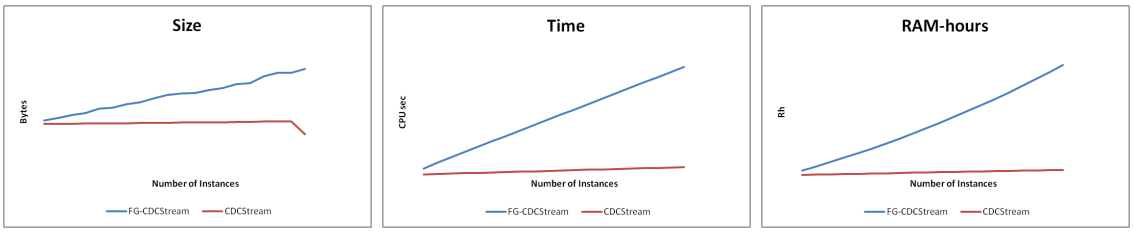


Figure 44: General trends in size, time and model costs

From the results obtained it seems that FG-CDCStream and CDCStream perform similarly on real data sets where the majority of the changes are most likely gradual. Although certainly not conclusive due to the small sample of data sets, these results do corroborate the results produced from the synthetic data sets injected with gradual change. A more comprehensive comparative study of algorithm performance on real data sets is recommended for future work.

5.6 Cost Considerations

The algorithm costs will be discussed very generally since they were not a main concern for this study, although they are of interest. The general trends do tend to remain fairly constant regardless of change type, number of changes in a stream or noise level. The costs measured include model serialized size (bytes), time (CPU seconds) and the memory-time ratio (RAM-hours) and the trends are displayed in Figure 44.

In terms of size, time and Rh measures, the measures produced by FG-CDCStream tend to increase significantly more than those of CDCStream over time. These results were expected, given the extra processing steps and memory requirements involved with storing and processing the multiple tracks of FG-CDCStream. These results imply that FG-CDCStream is less suitable than CDCStream to tasks that are very time and/or memory sensitive than CDCStream. Both the time and memory efficiency of

FG-CDCStream should be improved in future work.

5.7 Summary

Through this experimentation, it was determined that certain parameters are better suited to the change detectors studied than others. This reduced the scope of the study, eliminating inappropriate parameter ranges. It was found that DILCA-based context selection is not well suited data sets with low cardinalities (between two and six) and the scope of the study was reduced to contain only data sets with cardinalities of seven or greater. It was found that batch sizes do have a large effect on results, and the ideal batch size for both algorithms is about 50, though this could change in varying other parameters. Although the classifiers have no effect on the change detection modules, it was noted that a classifier's ability to adapt independently of a change detector is important to consider when comparing change detectors. Classifiers with greater natural adaptation were removed from the study to more fairly compare change detection capabilities. Results were discussed in detail within this more appropriate and informative scope.

It was shown that FG-CDCStream did generally succeed in achieving its objective: improving CDCStream's change detection performance as well as its adaptation strategy. In the cases studied, FG-CDCStream was able to meet or exceed CDCStream's change detection performance statistics and recover from change more quickly (improved adaptation). As expected, these improvements were most notable in abrupt change detection in which case, unexpectedly, CDCStream did not function at all. FG-CDCStream did also demonstrate a performance improvement with respect of CDCStream in the presence of gradual drift(s), multiple heterogeneous drifts and noise. Both algorithms performed similarly on real data in terms of accuracy-type

statistics, although FG-CDCStream tended to produce fewer change alarms. No further information on change detector behaviour could be gathered from the real data tests due to lack of information on the changes.

Finally, it was observed that FG-CDCStream is more time and memory intensive than CDCStream, especially as more instances are examined. This result comes at no surprise since the storage and analysis of multiple tracks certainly commands greater time and memory resources than a single track. The improvement of FG-CDCStream's cost efficiency will be an objective of future work.

Overall, FG-CDCStream succeeded in improving the change detection capabilities of CDCStream, increasing the breadth of change scenarios that can be detected by this unsupervised, context-based change detector for categorical data. FG-CDCStream provides the capability of detecting abrupt concept drift, a significant advancement, and, although less substantially, improves performance during gradual change detection and increases robustness to noise. In light of its achievements, FG-CDCStream comes at the price of greater time and memory requirements. Efficiency, as well as other possible advancements and further experimentation will be discussed in detail in the following chapter.

Chapter 6

Conclusions

This thesis focused on improving the change detection and adaptation capabilities of the most recent, state-of-the-art unsupervised change detection method for categorical data that is strictly context-based: CDCStream [4]. The goal of this study was to produce an extension of this algorithm, FG-CDCStream, that generates improved change detection statistics and is more effective in adaptation to change. It was intended that these improvements would enhance system performance when exposed to concept drift, especially that which is abrupt. Experiments were performed in order to compare the performance of the two algorithms in terms of change detection statistics as well as accuracy-type statistics on streams containing gradual and abrupt changes in a variety of scenarios. This chapter discusses the contributions of this thesis and presents opportunities for future work.

6.1 Contributions

The current state of change detection for categorical data is largely learning-based whereby changes are detected via error monitoring [4]. This is not surprising, since typical statistical methods require numerical input and determining numerical similarity measures for categorical attributes has proven non-trivial [10] [11] [12]. Since

fluctuations in error measures can not be definitively attributed to concept drift alone, a change detector based on data distributions is more informational. In the case of categorical data, due to its inherent challenges, the study of this type of change detection has been very limited [4].

The design of CDCStream [4] intended to aid in closing this research gap. While reasonably successful, CDCStream has several limitations that provide opportunity for improvement. First, as determined in this study, CDCStream fails to detect abrupt changes. This is due to its coarse grain size and high level of aggregation. These qualities affect its performance when presented with gradual drift as well, although not as dramatically. Second, CDCStream's adaptation strategy lacks rigidity, allowing for potentially unrepresentative learners to take effect after a change, effecting the system's accuracy and its recovery after a change.

This thesis presented an adaption of CDCStream, FG-CDCStream, designed to improve change detection and adaptation capabilities especially in the abrupt change scenario. As a solution to the coarse grain and aggregation issues of CDCStream, FG-CDCStream utilizes a series of overlapping data "tracks" which take slightly different samples of data and vote to determine change, similar to ensemble learning. As an improvement to CDCStream's adaptation strategy, FG-CDCStream ensures more strictly that the classifying learner present after a change occurs is representative of the current concept. It was expected that improving upon the limitations of CDCStream would improve change detection and adaptation capabilities of the learning system, especially in the presence of abrupt concept drift.

The results of experimentation indicate that FG-CDCStream successfully resolves the limitations of CDCStream. FG-CDCStream does possess abrupt change capabilities where CDCStream does not. It typically produced more desirable change detection statistics (namely MTFA, MTD, MDR and MTR) and generally recovered faster from change, indicating a superior adaptation strategy. This was achieved at

the expense of greater time and memory requirements.

This thesis presents FG-CDCStream a fine-grained, unsupervised and context-based change detection algorithm for streaming categorical data. This algorithm has the ability to provide users with more precise information than that of learning-based methods about the changing distributions, abrupt or gradual, in categorical data streams. It provides context-based change detection capabilities for categorical data previously undocumented in Machine Learning research and contributes to narrowing this research gap.

6.2 Future Work

This study presents plenty of opportunities to further narrow the research gap in future work. The most obvious opportunity for improvement lies in algorithm efficiency. A more time- and memory-efficient system would certainly expand the breadth of applications that could benefit from utilizing FG-CDCStream. This might be done by reducing sample sizes, the number of tracks or the number of operations on tracks. Developing an incremental adaptation of DILCA, if possible, would certainly be profitable. Perhaps a more advanced adaptation strategy could be adopted whereby only one learner need be developed at a given time, rather than two.

A limitation to FG-CDCStream is that it does retain the use of Chebychev's inequality. As discussed, this is associated with higher detection delays than less conservative concentration inequalities. So, even though FG-CDCStream improves the MTD of CDCStream, its MTD values may still be high. This limitation should be considered in future studies.

Automatic parameter adjustments would likely benefit this system as well. For example, the forgetting mechanism could become more rigid ensuring all tracks are more representative of the actual concepts in the data. For example, a greater magnitude

change might trigger more abrupt forgetting or require fewer voters for confirmation. A more stringent solution to the adaptive sliding windows might be employed.

This study would benefit from being supplemented by further exploration of the effects of different types of data streams on algorithm performance. For example, a more comprehensive study on real data might provide further insight into algorithm behaviour. Additional research on no-change data streams would be beneficial. Further, studying how attribute cardinality effects the algorithm would be useful as well.

Further study of CDCStream will consider the functionality of this algorithm with respect to asynchronous and distributed streams, and implement the required adaptations.

Finally, a less directly related suggestion for future work would be to investigate effective means of assessing change detector performance. Changes in accuracy-type performance measures cannot be definitively attributed to change detector performance alone. It might be more useful to have accuracy-type statistics that summarize system behaviour surrounding a change, rather than simply analyzing graphs, as done in this study, or taking only a final or averaged value. For instance, a statistic relating slopes in performance drops and recoveries or the areas between pre-change and post-recovery values might be useful. Improved change detection statistics would also be advantageous. The current change detection statistics available are not always measurable and some have no pre-determined range, making both individual assessment as well as comparison difficult.

References

References are listed in order of appearance in the thesis, as specified by the Institute of Electrical and Electronics Engineers (IEEE) referencing style guide.

[1] A. Bifet and R. Kirkby, "Data Stream Mining: A Practical Approach," 2009.

[2] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A Survey on Concept Drift Adaptation," *ACM Computing Surveys*, vol. 46, no. 4, 2013.

[3] R. Pears, S. Sakthithasan, and Y. S. Koh, "Detecting Concept Change in Dynamic Data Streams," *Machine Learning*, vol. 97, no. 3, pp. 259–293, 2014.

[4] D. Ienco, A. Bifet, B. Pfahringer, and P. Poncelet, "Change Detection in Categorical Evolving Data Streams," in *Symposium on Applied Computing*, (Gyeongju, Korea), 2014.

[5] J. Howard, "Getting in Shape for the Sport of Data Science," 2001.

[6] T. R. Hoens, R. Polikar, and N. V. Chawla, "Learning from Streaming Data with Concept Drift and Imbalance : An Overview," *Progress in Artificial Intelligence*, vol. 1, no. 1, pp. 89–101, 2012.

[7] A. Bifet, J. Read, and B. Pfahringer, "CD-MOA : Change Detection Framework for Massive Online Analysis," *Advances in Intelligent Data Analysis XII*, vol. 8207, pp. 92–103, 2013.

[8] J. Gama, P. P. Rodrigues, and R. Sebastiao, "Evaluating Algorithms that Learn from Data Streams," in *ACM Symposium on Applied Computing*, ACM, 2009.

[9] F. Cao, J. Z. Huang, and J. Liang, "Trend analysis of categorical data streams with a concept change method," *Information Sciences*, feb 2014.

[10] F. Cao, J. Liang, D. Li, L. Bai, and C. Dang, "Knowledge-Based Systems A dissimilarity measure for the k -Modes clustering algorithm," *Knowledge-Based Systems*, vol. 26, pp. 120–127, 2012.

[11] S. Boriah, V. Chandola, and V. Kumar, "Similarity Measures for Categorical Data : A Comparative Evaluation," in *Proceedings of the 8th SIAM International Conference on Data Mining*, pp. 243–254, SIAM, 2008.

[12] M. Alamuri, B. R. Surampudi, and A. Negi, "A Survey of Distance / Similarity Measures for Categorical Data," in *2014 International Joint Conference on Neural Networks (IJCNN)*, pp. 1907–1914, IEEE, 2014.

[13] J. Han, M. Kamber, and J. Pei, *Noisy Data*. Elsevier, 3 ed., 2012.

[14] C. C. Aggarwal and P. S. Yu, "Data Mining Techniques for Associations , Clustering and Classification," in *Methodologies for Knowledge Discovery and Data Mining*, pp. 13–23, Springer Berlin Heidelberg, 1999.

[15] R. Jin and G. Agarwal, "Frequent Pattern Mining In Data Streams," in *Data Streams: Models and Agorithms*, ch. 4, pp. 61–84, Springer Science & Business Media, 2007.

[16] R. Agrawal, S. Member, T. Imielinski, and A. Swami, "Database Mining : A Performance Perspective," *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 6, pp. 914–925, 1993.

[17] C. C. Aggarwal, "An Introduction to Data Streams," in *Data Streams: Models and Agorithms*, ch. 1, pp. 1–8, Springer Science & Business Media, 2007.

[18] S. Kamburugamuve, "Survey of Streaming Data Algorithms," tech. rep., Indiana University, 2013.

[19] L. O'Callaghan, A. Meyerson, R. Motwani, N. Mishra, and S. Guha, "Streaming-data Algorithms for High-quality Clustering," in *ICDE*, p. 0685, IEEE, 2002.

[20] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "On Clustering Massive Data Streams: A Summarization Paradigm," in *Data Streams: Models and Agorithms*, ch. 2, pp. 9–38, Springer Science & Business Media, 2007.

[21] M. G. Kelly, D. J. Hand, and N. M. Adams, "The Impact of Changing Populations on Classifier Performance," in *Proceedings of the fifth ACM SIGKDD*

international conference on Knowledge discovery and data mining, vol. 32, pp. 367–371, 1999.

[22] L. I. Kuncheva, "Classifier Ensembles for Changing Environments," *Multiple Classifier Systems, Lecture Notes in Computer Science*, vol. 3077, pp. 1–15, 2004.

[23] J. Vinagre and A. M. Jorge, "Forgetting mechanisms for incremental collaborative filtering," in *Proc of 2010 WTI Workshop*, pp. 1–11, 2010.

[24] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with Drift Detection," in *Advances in Artificial Intelligence - SBIA 2004, Volume 3171 of Lecture Notes in Computer Science*, pp. 286–295, Springer Berlin Heidelberg, 2004.

[25] A. Bifet and R. Gavald, "Learning from Time-Changing Data with Adaptive Windowing," in *SIAM International Conference on Data Mining*, 2006.

[26] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[27] H. Abdulsalam, D. B. Skillicorn, and P. Martin, "Classification Using Streaming Random Forests," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 1, pp. 22–36, 2011.

[28] A. Wald, *Sequential Analysis*. John Wiley and Sons, Inc., 1947.

[29] A. S. Polunchenko and A. G. Tartakovsky, "State-of-the-Art in Sequential Change-Point Detection," *Methodology and Computing in Applied Probability*, vol. 14, no. 3, pp. 649–684, 2012.

[30] E. S. Page, "Continuous Inspection Schemes," *Biometrica*, vol. 41, no. 1, pp. 100–115, 1954.

[31] M. Baena-Garcia, J. del Campp-Avila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-bueno, "Early Drift Detection Method," in *Fourth international workshop on knowledge discovery from data streams*, pp. 77–86, 2006.

[32] S. Bach and M. Maloof, "Paired Learners for Concept Drift," in *Proceedings of the Eigth IEEE International Conference on Data Mining*, (Los Alamitos, CA), pp. 23–32, IEEE Press, 2008.

[33] K. Nishida and K. Yamauchi, "Detecting Concept Drift Using Statistical Testing," in *Discovery Science*, pp. 264–269, Springer Berlin Heidelberg, 2007.

[34] P. Vorburger and A. Bernstein, "Entropy-based Concept Shift Detection," in *ICDM'06. Sixth International Conference on Data Mining*, pp. 1113–1118, IEEE, 2006.

[35] D. Kifer, S. Ben-david, and J. Gehrke, "Detecting Change in Data Streams," in *Proceedings of the Thirteenth International Conference on Very Large Databases - Volume 30*, pp. 180–191, VLDB Endowment, 2004.

[36] G. Lugosi, "Concentration Inequalities in Machine Learning," 2013.

[37] C. C. Aggarwal and P. S. Yu, "A Survey of Synopsis Construction in Data Streams," in *Data Streams: Models and Agorithms*, ch. 9, pp. 169–207, Springer Science & Business Media, 2007.

[38] F. Chung and L. Lu, "Old and New Concentration Inequalities," in *Complex Graphs and Networks*, ch. 2, pp. 23–56, AMS, 2006.

[39] M. Harries, "Splice-2 Comparative Evaluation: Electricity Pricing," tech. rep., University of New South Wales, 1999.

[40] A. Patil and M. S. Santhanam, "Random matrix approach to categorical data analysis," *Physical Review E*, vol. 92, no. 3, p. 032130, 2015.

[41] A. Agresti, *Categorical Data Analysis*. John Wiley & Sons, 3 ed., 2013.

[42] P. Jaccard, "The distribution of the flora in the alpine zone. 1," *New Phytologist*, vol. 11, no. 2, pp. 37–50, 1912.

[43] C. Stanfill and D. Waltz, "Toward Memory-Based Reasoning," *Communications of the ACM*, vol. 29, no. 12, pp. 1213–1228, 1986.

[44] S. Cost and S. Salzberg, "A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features," *Machine Learning*, vol. 10, no. 1, pp. 57–78, 1993.

[45] D. R. Wilson and T. R. Martinez, "Improved Heterogeneous Distance Functions," *Journal of Artificial Intelligence Research*, vol. 6, pp. 1–34, 1997.

[46] Z. Huang, "Clustering large data sets with mixed numeric and categorical values," in *Proceedings of the 1st Pacific-Asia Conference on Knowledge Discovery and Data Mining,(PAKDD)*, (Singapore), pp. 21–34, 1997.

[47] Z. Huang and M. Ng, "A fuzzy k-modes algorithm for clustering categorical data," *IEEE Transactions on Fuzzy Systems*, vol. 7, no. 4, pp. 446–452, 1999.

[48] D. Kim, K. Lee, and D. Lee, "Fuzzy clustering of categorical data using fuzzy centroids," *Pattern Recognition Letters*, vol. 25, no. 11, pp. 1263–1271, 2004.

[49] Z. Huang, "Extensions to the k-means algorithm for clustering large data sets with categorical values," *Data Mining and Knowledge Discovery*, vol. 2, no. 3, pp. 283–304, 1998.

[50] D. Kim, K. H. Lee, K. Lee, and D. Lee, "A k-populations algorithm for clustering categorical data," *Pattern Recognition*, vol. 38, no. 7, pp. 1131–1134, 2005.

[51] O. San, V. Huynh, and Y. Nakamori, "An alternative extension of the k-means algorithm for clustering categorical data," *International Journal of Applied Mathematics and Computer Science*, vol. 14, pp. 241–247, 2004.

[52] Z. He, S. Xiaofei, and B. Dong, "K-histograms: An efficient clustering algorithm for categorical dataset," *arXiv preprint cs/0509033*, 2005.

[53] D. Ienco, R. G. Pensa, and R. Meo, "From Context to Distance: Learning Dissimilarity of Categorical Clustering," *ACM Transactions on Knowledge Discovery from Data*, vol. 6, pp. 1–25, mar 2012.

[54] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, "A geometric framework for unsupervised anomaly detection," in *Applications of data mining in computer security*, pp. 77–101, Springer, 2002.

[55] K. Jones, "A statistical interpretation of term specificity and its application in retrieval,," *Journal of Documentation*, vol. 28, no. 1, pp. 11–21, 1972.

[56] V. Cheng, C.-h. Li, J. T. Kwok, and C.-k. Li, "Dissimilarity learning for nominal data," *Pattern Recognition*, vol. 37, no. 7, pp. 1471–1477, 2004.

[57] J. Xie and M. J. Zaki, "Learning Dissimilarities for Categorical Symbols," *Journal of Machine Learning Research-Proceedings Track*, vol. 10, pp. 97–106, 2010.

[58] G. Sudipto, R. Rastogi, and K. Shim, "Rock: A robust clustering algorithm for categorical attributes," in *Data Engineering, 1999. Proceedings., 15th International Conference on*, pp. 512–521, IEEE, 1999.

[59] V. Ganti, J. Gehrke, and R. Ramakrishnan, "CACTUSclustering categorical data using summaries," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 73–83, ACM, 1999.

[60] D. Gibson, J. Kleinberg, and P. Raghavan, "Clustering categorical data: An approach based on dynamical systems," *Databases*, vol. 1, 1998.

[61] M. Zaki and M. Peters, "Clicks: Mining subspace clusters in categorical data via k-partite maximal cliques," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pp. 355–356, 2005.

[62] D. Goodall, "A new similarity index based on probability," *Biometrics*, vol. 22, no. 4, pp. 882–907, 1966.

[63] E. Smirnov, "On exact methods in systematics," *Systematic Zoology*, vol. 17, no. 1, pp. 1–13, 1968.

[64] M. Anderberg, *Cluster Analysis for Applications*. New York: Academic Press, 1973.

[65] S. Q. Le and T. B. Ho, "An association-based dissimilarity measure for categorical data," *Pattern Recognition Letters*, vol. 26, pp. 2549–2557, 2005.

[66] D. H. Fisher, "Knowledge acquisition via incremental conceptual clustering," *Machine Learning*, vol. 2, no. 2, pp. 139–172, 1987.

[67] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman, "Autoclass: a bayesian classification system," in *Readings in knowledge acquisition and learning.*, pp. 431–441, 1993.

[68] P. Gambaryan, "A mathematical model of taxonomy," *Izvest. Akad. Nauk Armen. SSR*, vol. 17, no. 12, pp. 47–53, 1964.

[69] D. Lin, "An Information-Theoretic Definition of Similarity," in *ICML '98: Proceedings of the 15th International Conference on Machine Learning*, (San Fransisco, CA, USA), pp. 296–304, Morgan Kaufmann Publishers Inc., 1998.

[70] T. P. Burnaby, "On a method for character weighting a similarity coefficient, employing the concept of information," *Mathematical Geology*, vol. 2, no. I, pp. 25–38, 1970.

[71] L. Yu and H. Liu, "Feature Selection for High-Dimensional Data : A Fast Correlation-Based Filter Solution," in *Proceedings of the 20th International*

Conference on Machine Learning (ICML), (Washington, DC, USA), pp. 856–863, 2003.

[72] Z. Khorshidpour, S. Hashemi, and A. Hamzeh, "Distance Learning for Categorical Attribute Based on Context Information," in *2nd International Conference on Software Technology and Engineering (ICSTE), vol. 2*, pp. V2–296, IEEE, 2010.

[73] D. Barbara, Y. Li, and J. Couto, "Coolcat: an entropy-based algorithm for categorical clustering," in *Proceedings of the eleventh international conference on Information and knowledge management*, pp. 582–589, ACM, 2002.

[74] T. Li, S. Ma, and M. Ogihara, "Entropy-based criterion in categorical clustering," in *Proceedings of the 21st International Conference on Machine Learning (ICML)*, pp. 536–543, ACM, 2004.

[75] P. Andritsos, P. Tsaparas, R. Miller, and K. Sevick, "Limbo: Scalable clustering of categorical data," in *Advances in Database Technology-EDBT 2004*, pp. 123–146, Springer, 2004.

[76] S. Rissino and G. Lambert-Torres, "Rough set theory–fundamental concepts, principals, data extraction, and applications," *Data mining and knowledge discovery in real life applications*, p. 438, 2009.

[77] D. Chen, D. Cui, C. Wang, and Z. Wang, "A rough set-based hierarchical clustering algorithm for categorical data," *International Journal of Information Technology,*, vol. 12, no. 3, pp. 149–159, 2006.

[78] F. Cao, J. Liang, L. Bai, X. Zhao, and C. Dang, "A framework for clustering categorical time-evolving data," *IEEE Transactions on Fuzzy Systems*, vol. 18, no. 5, pp. 872–882, 2010.

[79] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[80] F. Cao and J. Z. Huang, "A Concept-Drifting Detection Algorithm for Categorical Evolving Data," in *Advances in Knowledge Discovery and Data Mining*, pp. 485–496, Springer, 2013.

[81] N. Japkowicz and M. Shah, *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, 2014.

[82] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009.

[83] I. Zliobaite, M. Budka, and F. Stahl, "Towards cost-sensitive adaptation : When is it worth updating your predictive model ?," *Neurocomputing*, vol. 150, pp. 240–249, 2015.

[84] E. Bennett, R. Alpert, and A. Goldstein, "Communications through limited response questioning," *Public Opinion Q*, vol. 18, pp. 303–308, 1954.

[85] W. Scott, "Reliability of content analysis: The case of nominal scalecoding," *Public Opinion Q*, vol. 19, pp. 321–325, 1955.

[86] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[87] A. Bifet, J. Read, B. Pfahringer, and G. Holmes, "Pitfalls in Benchmarking Data Stream Classification and How to Avoid Them," *Machine Learning and Knowledge Discovery*, vol. 8188, pp. 465–479, 2013.

[88] A. Bifet, G. Holmes, B. Pfahringer, and E. Frank, "Fast Perceptron Decision Tree Learning from Evolving Data Streams," in *Proceedings of the 14th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, PAKDD*, pp. 299–310, 2010.

[89] I. Zliobaite, "How good is the Electricity benchmark for evaluating concept drift adaptation," *CoRR*, vol. 1301, no. 3524, pp. 1–6, 2013.

[90] J. Gama, R. Sebasti, and P. P. Rodrigues, "Issues in Evaluation of Stream Learning Algorithms Categories and Subject Descriptors," in *Proceedings of the 15th annual SIGKDD international conference on knowledge discovery and data mining*, (New York, USA), pp. 329–338, ACM, 2009.

[91] V. Lemaire, C. Salperwyck, and A. Bondu, "A Survey on Supervised Classification on Data Streams," in *Business Intelligence: Lecture Notes in Business Information Processing*, pp. 88–125, Springer International Publishing, 2015.

[92] A. David, "Statistical Theory: The Prequential Approach," *Journal of the Royal Society-A*, vol. 147, no. 2, pp. 278–292, 1984.

[93] L. L. Harlow, S. A. Mulaik, and J. H. Steiger, *What if there were No Significance Tests?* New York, New York, USA: Taylor & Francis, classic ed., 2016.

[94] W. J. Dixon and A. M. Mood, "The statistical sign test," *Journal of the American Statistical Association*, vol. 41, no. 236, pp. 557–566, 1946.

[95] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[96] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the American Statistical Association*, vol. 32, no. 200, pp. 675–701, 1937.

[97] Q. McNemar, "Note on the sampling error of the difference between correlated proportions or percentages," *Psychometrika*, vol. 12, no. 2, pp. 153–157, 1947.

[98] A. Bifet, H. Kremer, and T. Seidl, "MOA : Massive Online Analysis , a Framework for Stream Classification and Clustering . Learning Examples," in *Workshoo on Applications of Pattern Analysis*, vol. 11, pp. 44–50, 2010.

[99] G. Holmes, A. Donkin, and I. H. Witten, "WEKA : A Machine Learning Workbench," in *Proc Second Australia and New Zealand Conference on Intelligent Information Systems*, 1994.

[100] L. Breiman, J. Friedman, C. J. Stone, and R. Olshen, *Classification and Regression Trees*. Taylor & Francis, 1984.

[101] J. Schlimmer and R. Granger, "Incremental Learning from Noisy Data," *Machine Learning*, vol. 1, no. 3, pp. 317–354, 1986.

[102] J. A. Blackard and D. J. Dean, "Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables," *Computers and Electronics in Agriculture*, vol. 24, no. 3, pp. 131–151, 1999.

[103] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, W. Street, N. York, and P. K. Chan, "Cost-based Modeling and Evaluation for Data Mining With Application to Fraud and Intrusion Detection : Results from the JAM Project ," in *DARPA Information Survivability Conference and Exposition*, pp. 130–144, 2000.

[104] S. K. Mukkavilli and S. Shetty, "Mining Concept drifting Network traffic in cloud computing environments," in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 721–722, 2012.

[105] E. Ikonomovska, J. Gama, and S. Džeroski, "Learning model trees from evolving data streams," in *Data Mining and Knowledge Discovery*, no. 23, pp. 128–168, 2010.

[106] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-Based Learning Algorithms," *Machine Learning*, vol. 6, no. 1, pp. 37–66, 1991.

Appendix A

Implementation

A.1 FG-CDCStream

As mentioned in Section 4.2, a copy of the CDCStream code used in [4] was obtained via email communication with the author, Dr. Dino Ienco. The following algorithm utilizes CDC objects adapted from said code. The mechanics of CDC objects is described in Section 3.3.

```
1  import java.util.ArrayList;
2  import java.io.*;
3
4  import moa.classifiers.translations.CDC;
5  import moa.classifiers.trees.HoeffdingTree;
6  import moa.classifiers.AbstractClassifier;
7  import moa.core.Measurement;
8  import moa.options.IntOption;
9  import moa.options.MultiChoiceOption;
10
11 import weka.classifiers.bayes.NaiveBayes;
12 import weka.classifiers.lazy.IBk;
13 import weka.core.Attribute;
14 import weka.core.Instance;
15 import weka.core.Instances;
16
17 public class FG-CDCStream extends AbstractClassifier
18 {
```

```
19    private static final long serialVersionUID = 1L;
20
21    public IntOption windowSizeOption = new IntOption("
          windowSize", 'w',
22        "Size of window", 50, 10, 1000);
23
24    public MultiChoiceOption learnerOption = new
          MultiChoiceOption(
25        "learner", 'l', "Learner to train", new String[]{
26            "Naive Bayes", "IBk", "Hoeffding Tree"},
27            new String[]{"Train on incremental NB classifier",
28            "Train on IBk KNN",
29            "Train on Hoeffding Tree",
30        }, 0);
31
32    int windowSize;
33    int totInst ;
34    int nchange ;
35    boolean backInit ;
36    double k = 0.0;
37    ArrayList<Instances> buffer;
38    ArrayList<CDC> tracks;
39    int trackPointer;
40    NaiveBayes nb , nbBack ;
41    IBk ibk, ibkBack;
42    HoeffdingTree tree, treeBack;
43    boolean backupIsNull;
44    int votes;
45    String learnerType;
46    ArrayList<Attribute> attributes ;
47    boolean warningPeriod;
48    boolean changePeriod;
49      FileOutputStream f;
50
51    @Override
52    public String getPurposeString() {
53      return "CatStream_NB.";
54    }
55
56    @Override
57    public void resetLearningImpl()
58    {
59      windowSize = windowSizeOption.getValue();
```

```
60      totInst = 0;
61      nchange = 0;
62      backInit = false;
63      k = 0.0;
64      buffer = new ArrayList<Instances>(windowSize);
65      tracks = new ArrayList<CDC>(windowSize);
66      trackPointer = 0;
67
68      switch (this.learnerOption.getChosenIndex())
69      {
70      case 0: //NB
71        nb = null;
72        nbBack = null;
73        learnerType = "NaiveBayes";
74        break;
75      case 1: //IBk
76        ibk = null;
77        ibkBack = null;
78        learnerType = "IBK";
79        break;
80      case 2: //HT
81        tree = null;
82        treeBack = null;
83        learnerType = "HoeffdingTree";
84        break;
85      }
86
87      backupIsNull = true;
88      votes = -1;
89
90      for(int i=0; i<windowSize; i++)
91      {
92        tracks.add(new CDC());
93      }
94      warningPeriod = false;
95      changePeriod = false;
96    }
97
98    @Override
99    public void trainOnInstanceImpl(Instance inst)
100   {
101     totInst ++;
102
```

```
103       if(totInst == 1)
104       {
105         attributes = new ArrayList<Attribute>();
106         for(int i=0; i<inst.numAttributes();++i)
107         {
108           attributes.add( inst.attribute(i));
109         }
110       }
111       buffer.add(new Instances("temp", attributes ,0));
112       buffer.get(0).setClassIndex(inst.numAttributes()-1);
113       for(int j=0; j<buffer.size(); j++)
114       {
115         buffer.get(j).add(inst);
116       }
117
118       if(buffer.get(0).size() == windowSize)
119       {
120         k = tracks.get(trackPointer).getKValue(buffer.get(0));
121
122         if(k >= 3)
123         {
124           votes++;
125
126           //WARNING
127           if(votes == 1)
128           {
129             warningPeriod = true;
130             switch (this.learnerOption.getChosenIndex())
131             {
132             case 0: //NB
133               nbBack = new NaiveBayes();
134               break;
135
136             case 1: //IBk
137               ibkBack = new IBk();
138               break;
139
140             case 2: //HT
141               treeBack = new HoeffdingTree();
142               treeBack.prepareForUse();
143               treeBack.resetLearning();
144               break;
145             }
```

```
146
147             backupIsNull = false;
148             backInit=true;
149           }
150           //CHNAGE
151           if(votes == 15)
152           {
153             System.out.println(totInst);
154
155             nchange++;
156             changePeriod = true;
157             warningPeriod = false;
158
159             switch (this.learnerOption.getChosenIndex())
160             {
161             case 0: //NB
162               nb = nbBack;
163               nbBack = null;
164               break;
165
166             case 1: //IBk
167               ibk = ibkBack;
168               ibkBack = null;
169               break;
170             case 2: //HT
171               tree = treeBack;
172               treeBack = null;
173               break;
174             }
175             backupIsNull = true;
176           }
177         }
178         //STATIC
179         else if(k < 2)
180         {
181           votes = 0;
182           if(changePeriod)
183           {
184             changePeriod = false;
185           }
186
187           if(warningPeriod)
188           {
```

```
189             warningPeriod = false;
190
191             switch (this.learnerOption.getChosenIndex())
192             {
193             case 0: //NB
194               nbBack = null;
195               break;
196
197             case 1: //IBk
198               ibkBack = null;
199               break;
200             case 2: //HT
201               treeBack = null;
202               break;
203             }
204
205             backupIsNull = true;
206           }
207         }
208       }
209       try
210       {
211         if (backInit)
212         {
213           switch (this.learnerOption.getChosenIndex())
214           {
215           case 0: //NB
216             nbBack.buildClassifier(buffer.get(0));
217             break;
218
219           case 1: //IBk
220             ibkBack.buildClassifier(buffer.get(0));
221             break;
222           case 2: //HT
223             treeBack.resetLearning();
224
225             for(int i=0; i<buffer.get(0).size(); i++)
226             {
227               treeBack.trainOnInstanceImpl(buffer.get(0).get(i)
                      );
228             }
229             break;
230           }
```

```
231           backInit = false;
232         }
233         else
234         {
235           if(!backupIsNull)
236           {
237             switch (this.learnerOption.getChosenIndex())
238             {
239             case 0: //NB
240               nbBack.updateClassifier(inst);
241               break;
242
243             case 1: //IBk
244               ibkBack.updateClassifier(inst);
245               break;
246
247             case 2: //HT
248               treeBack.trainOnInstanceImpl(inst);
249               break;
250             }
251           }
252         }
253       }catch(Exception e){System.out.println(e.getMessage());}
254
255       try {
256
257         Instances subset = new Instances("temp", attributes ,0)
              ;
258         subset.setClassIndex(buffer.get(0).numAttributes()-1);
259         subset.add(inst);
260
261         switch (this.learnerOption.getChosenIndex())
262         {
263         case 0: //NB
264           if (nb == null)
265           {
266             nb = new NaiveBayes();
267             nb.buildClassifier(subset);
268           }
269           else
270           {
271             nb.updateClassifier(inst);
272           }
```

```
273           break;
274
275         case 1: //IBk
276           if (ibk == null)
277           {
278             ibk = new IBk();
279             ibk.buildClassifier(subset);
280           }
281           else
282           {
283             ibk.updateClassifier(inst);
284           }
285           break;
286         case 2: //HT
287           if(tree == null)
288           {
289
290             tree = new HoeffdingTree();
291             tree.prepareForUse();
292             tree.resetLearning();
293
294             tree.trainOnInstanceImpl(inst);
295
296           }
297           else
298           {
299             tree.trainOnInstanceImpl(inst);
300           }
301           break;
302         }
303
304       } catch (Exception ex) {
305         System.out.println("Xke: "+ex.getMessage());
306       }
307
308       if(totInst >= windowSize)
309       {
310         //delete the block that was just processed and add a
              new blank block to the end of the buffer
311         buffer.remove(0);
312
313         if(trackPointer == windowSize - 1)
314         {
```

```
315           trackPointer = 0;
316         }
317         else
318         {
319           trackPointer++;
320         }
321       }
322     }
323
324     @Override
325     public double[] getVotesForInstance(Instance inst)
326     {
327       double [] result = new double [inst.numClasses()];
328
329       for(int i = 0; i<result.length; i++)
330       {
331         result[i]=0.0;
332       }
333
334       try{
335
336         switch (this.learnerOption.getChosenIndex())
337         {
338         case 0: //NB
339
340           for(int i=0; i<inst.numClasses(); i++)
341           {
342             if(i == (int)nb.classifyInstance(inst))
343             {
344               result[i] = 1.0;
345             }
346           }
347           break;
348
349         case 1: //IBk
350           for(int i=0; i<inst.numClasses(); i++)
351           {
352             if(i == (int)ibk.classifyInstance(inst))
353             {
354               result[i] = 1.0;
355             }
356           }
357           break;
```

```
358         case 2: //HT
359           result = tree.getVotesForInstance(inst);
360           break;
361         }
362       }catch(Exception e)
363       {
364         System.out.println("CLassification Fail");
365       }
366       return result;
367     }
368
369     @Override
370     protected Measurement[] getModelMeasurementsImpl() {
371       return null;
372     }
373
374     @Override
375     public void getModelDescription(StringBuilder out, int
          indent) {
376     }
377
378     @Override
379     public boolean isRandomizable() {
380       return false;
381     }
382 }
```