A New Approach for Detecting Concept Drift and Measuring its Intensity in Large Datasets

Hisham Ogbah* **Abdallah Alashqur***
Computer Science Department Software Engineering Department
*Faculty of Information Technology
Applied Science Private University, Amman, Jordan

Abstract
The importance of data mining in general and classification in particular has increased in recent years due to the overwhelming amount of digital data that is produced world-wide on a daily basis. In classification, data tuples are mapped to a limited number of classes. The classifier learns (or derives) a classification model from a pre-classified dataset. The learned classification model can be represented in different forms such as a decision tree, set of rules, or support vector machines, to name a few. After the classifier completes the learning phase, it can predict the class of newly added data based on the model that it learned. Quite often a concept drift may occur due to changes in the environment, style, trend, or for many other reasons. Data that used to map to, say, class_a before the drift, now maps to class_b. But based on the knowledge embodied in the model, the system will still wrongfully predict class_a for the same data. This difference between what the model would predict and the actual classification is a sign that a concept drift has occurred and the classification model has become obsolete. In this case, a new model needs to be generated. In this paper we introduce a new efficient algorithm for detecting the occurrence of a concept drift and introduce a way of measuring the intensity of the drift. Measuring the intensity of the drift is important because it impacts how we may choose to deal with it going forward.

Key words:
Classification, Concept Drift, Drift detection, Big Data

1. Introduction

Classification maps each data tuple in a dataset to the most appropriate class selected from among a small set of classes [1, 2, 3, 4, 5]. A column in the dataset, usually referred to as class label, is used to store the class name of each tuple. The main goal of classification and prediction is to be able to predict the classes of new tuples that have not been classified yet. This is usually performed by passing through a learning phase in which the system learns the classification criteria from a pre-classified dataset (i.e., a training set). The learned criteria is referred to as the classification model. Once the system learns the classification model from the training set, it can use it as a basis to predict the classes of newly inserted tuples [6, 7].

A problem occurs if the distribution of data with respect to the classes changes over time. Meaning that real world data that used to map to class_a, for example, now maps to class_b. But a system that learned the classification model before the change in data distribution will continue to predict class_a for the same piece of data. Thus a mismatch occurs between the classification that the system predicts and the actual classification. This is what is called concept drift [8, 9, 10]. Concept drift, if not detected and handled properly, is considered a major problem in classification systems because it results in producing erroneous predictions.

To further demonstrate the effect of concept drift on a specific application, consider the sample data of Table 1. Suppose that this data is for a JobFinder classifier that is used in a recruiting agency. We assume that the classifier has already learned the model from a pre-classified training set. What JobFinder does is that it uses the job applicant's information to predict whether this applicant is going to find a job fairly quickly or he/she will take a long time. The classifier's predictions help the recruiting agency plan its priorities by knowing what to expect ahead of time. The information based on which the system makes its prediction are Age, Gender (G), Major, and Years_of_Experience (YE). The column named PCL (Predicted Class Label) shows the system's prediction of how long it expects the applicant to take before finding a job. Three different classes exist in PCL, namely, short (less than two weeks), medium (two to six weeks), and long (over six weeks). Later, when the job seeker finds a job, the actual classification becomes known and is recorded in the column named ACL (Actual Class Label).

We notice that nine out of the first ten tuples in Table 1 have a PCL class that matches the ACL class. The only exception is row number 4 where the predicted class was short whereas the actual class was medium. This means that the classification model was 90% accurate in its predictions (i.e., the erroneous predictions where 10%) for the first ten applicants. On the contrary, for the last ten applicants in the table (rows 11 through 20), the accuracy was 40% (i.e., the error rate was 60%). This indicates that a drift has occurred in the data distribution relative to the classes in the last 10 tuples in Table 1 because the accuracy is very low. This concept drift can be due to many factors. For example certain jobs that were very hot in the past have become saturated recently. Another reason could be that universities have recently started to incorporate extra special training in their curriculum in order to better prepare their graduates,

Manuscript received December 5, 2016
Manuscript revised December 20, 2016

which results in new graduates being able to find jobs much faster than used to be.

Table 1: Sample data for a JobFinder application

RID	Age	G	Major	YE	PCL	ACL
1	middle	f	marketing	3 - 6	medium	medium
2	youth	f	IT	3 - 6	medium	medium
3	youth	m	engineering	< 3	long	long
4	senior	m	IT	> 6	short	medium
5	youth	f	nursing	3 - 6	medium	medium
6	senior	m	marketing	> 6	short	short
7	youth	m	accounting	< 3	long	long
8	youth	f	marketing	< 3	long	long
9	middle	f	accounting	> 6	short	short
10	youth	f	nursing	3 - 6	medium	medium
11	youth	m	IT	< 3	long	long
12	youth	m	engineering	< 3	long	medium
13	youth	m	marketing	< 3	long	medium
14	middle	f	accounting	3 - 6	medium	medium
15	middle	f	accounting	> 6	short	short
16	youth	m	IT	< 3	long	medium
17	youth	m	IT	< 3	long	medium
18	middle	f	engineering	3 - 6	medium	medium
19	youth	m	marketing	< 3	long	medium
20	senior	m	engineering	> 6	short	medium

Once a concept drift is detected, the system has to be retrained. For example JobFinder classifier can be given the last 10 tuples in Table 1 along with the actual classification in the ACL column (i.e. without the PCL column) as a training set. From this training set the system can re-learn the classification model in order to be more accurate in its future predictions. In real-world data, the number of tuples in a table like Table 1 can be by the thousands or even millions depending on the type of application it is used for. Several algorithms and approaches have been reported in the literature for detecting the existence of concept drift in data [11, 12, 13]. The contribution of this paper is twofold. First, we introduce a new concept drift detection algorithm that is based on the idea of binary search to be able to find the drift location in a dataset. This is particularly useful for huge datasets since binary search has better performance than linear search methods. The second contribution of the paper is that we introduce a way to quantify and measure the drift intensity. The drift intensity can be large if the error rate after the point where the drift occurred is high. Also the drift intensity can be low or medium. To our knowledge, none of the existing drift detection techniques provides a way to measure the intensity of the drift.

Knowing the drift intensity can be helpful in that it provides some guidance as to how the concept drift should be handled. For example if the drift intensity is high, the system may choose to totally discard the old data (data before the drift point) when it regenerates a new classification model. On the other hand, if the drift intensity is low, the system may choose to give more weight to the data after the drift point and, at the same time, take into consideration data before the drift but give it less weight. Therefore the newly built classification model is influenced by data after the drift position more than it is influence by data before the drift position. This is useful when a lot of knowledge is embodied in the classifications performed before the drift and the user does not want to lose such knowledge.

The remainder of this paper is organized as follows. In Section 2 we provide a survey of related work. Section 3 describes the new algorithm used for detecting concept drift. Section 4 presents the formulas used to compute the Drift Intensity. Implementation results are shown in Section 5. Finally, conclusions are given in Section 6.

2. Related Work

In recent years, several studies have tried to come up with ways to detect the phenomenon of concept drift. The drift detection approach refers to the techniques used for explicit drift detection. The purpose of a drift detection technique is to identify the location in the dataset where a drift has occurred. Below is a brief description of some of the existing concept drift detection techniques.

The approach used in [11] is based on *Statistical Process Control* (*SPC*), which is standard statistical mechanisms to control and monitor the quality of a product through a continuous manufacturing sector. The *SPC* considers learning model as a process, and observers the evolution of this task. The *SPC* can be implemented to measure the change rate as interval between warning and uncontrolled, where short intervals indicate fast drifts, and longer intervals indicate slower drifts. The change rate can also be measured as the rate errors to the number of instances through warning. The *SPC* depends on the estimates of the error variance to assign the action bounds, which shrink as the trust of the error estimates raises. Other drift detection methods based on *SPC* are proposed in [14, 15].

The authors in [12] use an *Exponentially Weighted Moving Average* (*EWMA*) for detecting concept drift to monitor the misclassification rate of a classifier. *EWMA* calculates a recent estimate of the error rate, μ_t, by gradually down-weighting older data: $Z_0=\mu_0$, $Z_t = (1 - \lambda)Z_{t-1} + \lambda N_t$, $t > 0$, where N_t is the error at the current instance. It can be displayed that, independently of the distribution of the X_t variables, the mean and standard deviation of Z_t equal to: $\mu_{Zt} = \mu_t$, $\sigma_{Zt} = \sqrt{\frac{\lambda}{\lambda-2}\,(1-(1-\lambda)^{2t})}\,\sigma_x$, where σ_x is the standard deviation. Suppose that before the change point that $\mu_t = \mu_0$, and the *EWMA* estimate Z_t will fluctuate around this value. When a change occurs, the value of μ_t changes to μ_1 , and Z_t will react to this by diverging away from μ_0 and towards μ_1. This can be used for drift detection by flagging that a drift has occurred when: $Z_t > \mu_0 + L\sigma_{Zt}$, where L, the control limit, determines how far Z_t must diverge from μ_0 before the change alarm is flagged.

In [13], the authors present another approach for detecting drift by monitoring distribution on two different time-windows. This method typically uses a fixed bookmark window that summarizes the past information with a sliding detection window over the most recent instances. These two windows are compared over distributions with statistical tests based on the *Chernoff* bound to decide whether the two distributions are not equal. The window can monitor single variable or multivariate raw data (independently for each class). The *VFDTc* [16] came in the same line. It has the capacity to transact with drift by constantly monitoring differences between two distribution classes.

In [17], the authors present an entropy-based weight to measure the distribution difference between two sliding windows including respectively older and most recent instances. If the distributions are similar, the result of the entropy measure will point in a value of 1, and if they are completely different the result of the entropy measure will point in value of 0. The entropy measure is constantly monitored and observed the drift over time, and drift is marked when the entropy measure decreases under a given fixed user defined threshold. Another examples include drift detection methods proposed by [18, 19]. They use the *Kullback-Leibler* (*KL*) difference to measure the distance between the likelihood distributions of two windows (old & recent) to detect potential drifts.

The *AD*aptive sliding *WIN*dow (*ADWIN*) [20, 21] present another approach for detecting drift using a sliding window. The inputs of the algorithm are a confidence value $\delta \in (0,1)$, and let $x_1, x_2, x_3, \ldots, x_t$ be a sequence of real values. Each x_t is generated according to some distribution D_t, independently for every *t*. Denote as u_t the expected value for x_t when it is drawn according to D_t. *ADWIN* assumes that x_t is always bounded in [0,1], by an easy re-scaling, which can handle any value in which the interval is known $[a, b]$ such that $a \leq x_t \leq b$ with probability 1. Nothing else is known about the sequence of distribution D_t; in particular, μ_t is unknown for all *t*.

The concept behind *ADWIN* can be formatted like the following: whenever two big enough sub windows of *W* show distinct enough averages, one can conclude that the corresponding expected values are different, and the older (sub) window is dropped. Generally, large enough and distinct enough are translated into the computation of a cut value ϵ_c (which rely on δ, the length of the sub windows, and the averages of their contents). In another words, *W* is kept as long as possible while the null hypothesis μt has remained constant in *W* is sustainable up to confidence δ.

3. Concept Drift Detection

This section provides a description of a new detection algorithm called Binary Concept Drift Detection (BCDD) algorithm that we propose for detecting the existence of a concept drift. The BCDD algorithm is meant to be especially useful for large datasets in Big Data applications because it adopts the binary search approach.

3.1 Overview of the BCDD Algorithm

This algorithm finds the beginning of a drift and then measures the drift intensity (*DI)* value. The following are the advantages of BCDD algorithms existing algorithms:

1) The BCDD algorithm uses the binary search technique for detecting the phenomenon of concept drift, which is unlike other existing algorithms [11, 12, 21]. The binary search technique is fast and has a time complexity of O (log n). This gives a performance advantage when the size of data is huge. The BCDD is capable of using binary search because data in the dataset is ordered based on the insert timestamp.
2) Not only that the BCDD algorithm detects the existence of a drift, but it also measures its intensity. The DI value can be used later at the time of handling the drift and re-learning the model. If DI is high, much more weight is given to the data after the drift as compared to the weight given to the data before the drift. This means that data after the drift has a much larger influence on the newly generated classification model than data before the drift. On the other hand, if DI is low then data after the drift is given a moderately higher influence than data before the drift. Handling the concept drift by using DI is beyond the scope of this paper and is part of a future research.

Before describing the BCDD algorithm, we define some terms that are used by the algorithm.

1) E_R (Error Rate): represents the error rate found in the predicted classifications (PCL) as compared to the actual classifications.
2) T_{ER} (Tolerated Error Rate): represents the acceptable tolerated rate of inaccuracy. A rate above T_{ER} is considered a drift whereas a rate below T_{ER} is treated as just temporary noise.
3) We use a *Begin Window* (W_{Begin}) and an *End Window* (W_{End}) as sample windows that will be examined during the detection process. W_{Begin} is a set of rows taken at the beginning of a dataset, whereas W_{End} is a set of rows taken at the end of the dataset.
4) The size (W_{Size}) of W_{Begin} or W_{End} is defend based on the following criteria. First, if the dataset contains more than 50,000 tuples, W_{Size} is selected to be 0.2% of the size of the dataset. Second, if the dataset contains less than or equal to 50,000 tuples, W_{Size} is fixed at 100 tuples. We think that 0.2% for a window size is sufficient in large datasets. A dataset of 1M rows can have a window size of 2000 rows, which is considered a

descent sample sufficient for assessing the classification accuracy. However a size other than 0.2% can be selected if needed.

5) As long as E_R is less than T_{ER}, the algorithm assumes there is no concept drift. If E_R is larger then T_{ER}, this indicates that the inaccuracy rate is higher than what is permissible, which in turn indicates the existence of a concept drift. The value of T_{ER} is supplied to the system by the user and depends on the type of application. Some applications may tolerate higher rate of inaccuracy than others.

Before explaining the details of the flowchart of the BCDD algorithm as shown in Figure 1, we first give a brief overview of how it works. At the beginning, the BCDD algorithm divides the input dataset into two halves, and identifies W_{End} at the end of the first half. It then examines the E_R value within W_{End}. If the value of E_R is greater than T_{ER}, then the algorithm concludes that a concept drift has happened somewhere within in the first half. Therefore the algorithm continues searching the first half by further dividing it into two halves and repeating the process. On the other hand, if E_R is less than T_{ER}, this means that the first half is drift-free and the algorithm moves to examine the second half of the dataset. It identifies W_{Begin} at the beginning of the second half. If the data within W_{Begin} shows a drift (by examining E_R and comparing it with T_{ER}), then the algorithm concludes that a concept drift has started from W_{Begin} of the second half. However if data within the W_{Begin} is drift-free, the second half is divided into two halves and the process is repeated.

3.2 Flowchart and Pseudo Code of the Algorithm

Figure 1 shows the flowchart of the BCDD algorithm. At the very beginning of the algorithm, the size of W_{Begin} and W_{End} windows is determined based on the number of rows in the entire dataset. The algorithm receives the dataset form the system and declares new variables named as *Input_DS* and *P*. *Input_DS* is a variable that is assigned the dataset that will be used in the detection process. Input_Ds is initially set to equal the entire dataset, but later it will be assigned the appropriate half when the algorithm starts to divided the dataset. The other input, *P*, is the position where the drift has occurred, which is initially set to zero. The algorithm keeps updating P until it finishes. The final value of P is where the drift has occurred. If P stays zero till the end, then the entire dataset is drift-free.

The algorithm then checks if the size of the *Input_DS* is large enough to be divided into two halves. If true, the *Input_DS* is divided into two halves. Following that, the algorithm selects W_{End} of the 1st half and examines E_R within this window. If the value of E_R is greater than T_{ER}, the algorithm concludes that the concept drift has happened somewhere within the 1st half. Then, the next step is to set *P* to a new value, which is the row at the beginning of the window W_{End}. If the data at the 1st half is large enough for further division, it will be divided and the process is repeated. Otherwise, the algorithm has identified the location of the drift.

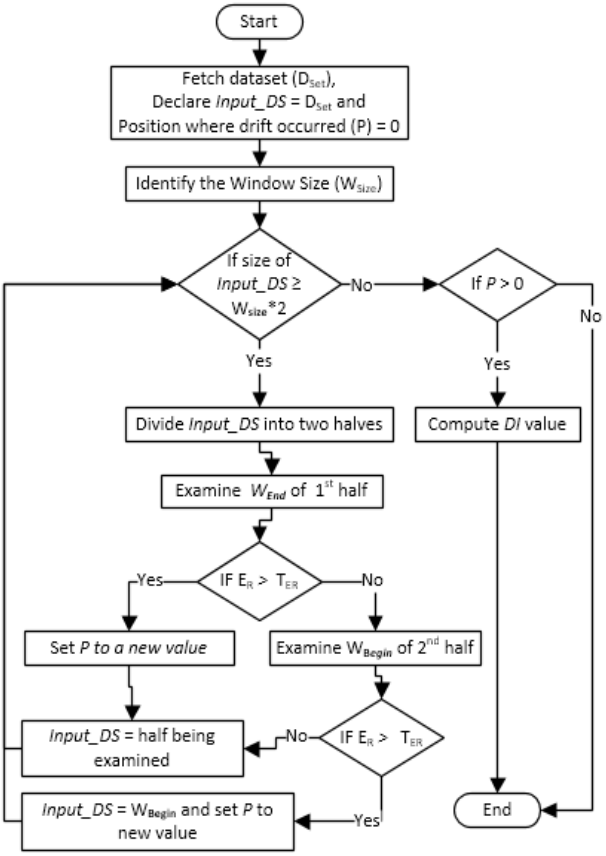

Figure 1: Flowchart of the BCDD algorithm

If the data within W_{End} of the 1st half, in any one of the iterations, shows no concept drift has occurred (since $E_R \leq T_{ER}$), then the algorithm moves to the 2nd half and selects W_{Begin} of the 2nd half. After that, the algorithm examines E_R inside W_{Begin}. If the data within W_{Begin} shows no drift has occurred (since $E_R \leq T_{ER}$), then the algorithm resets the *Input_DS* with the data of the 2nd half and re-sends the *Input_DS* to the decision diamond at the top and the process is repeated. If in any of these iteration we reach a point where the size of Input_DS < 2 * W_{Size}, looping ends and the algorithm proceeds to exist. Before it exists, it checks to see if P > 0 (meaning a drift position has been identified), and in this case it calls the function Comput_DI, which computes the drift intensity based on the formulas introduced in Section 4. Otherwise the algorithm exists with a value of P = 0. A value of "P = 0" is used as a flag that indicates that no drift has occurred.

Figure 2 shows the pseudo code of the BCDD algorithm. It works similar to the logic explained for the flowchart.

BCDD algorithm

Inputs:
- D_{Set} , (Dataset) List of data $\{x_1, \ldots, x_n\}$ with Predict Class Labels (*PCL*) $\{y_1, \ldots, y_n\}$ and Actual Class Labels (*ACL*) $\{z_1, \ldots, z_n\}$;
- T_{ER} , Tolerated Error Rate;
- P = 0; Initially to be Zero.
- DI = 0; Initially to be Zero.

Method:

(1) *Input_DS* = D_{Set};
(2) W_{Size} = Set window size based on *Input_DS*;
(3) W_{End}, W_{Begin}; Declare windows.
(4) **While** *Input_DS*.Count > W_{Size} * 2 **do**
(5) Divide *Input_DS* into 2 halves and use *Ceiling* function for 1st half to round up to the next number;
(6) W_{End} = Select W_{End} of 1st half;
(7) **if** E_R of W_{End} > T_{ER} **then**
(8) P = identify the beginning of drift in W_{End};
(9) *Input_DS* = 1st half;
(10) **else**
(11) W_{Begin} = Select W_{Begin} of the 2nd half;
(12) **if** E_R of W_{Begin} > T_{ER} **then**
(13) *Input_DS* = W_{Begin};
(14) P = identify the beginning of drift in W_{Begin};
(15) **else**
(16) *Input_DS* = 2nd half;
(17) **end if**
(18) **end if**
(19) **end while**
(20) **if** P > 0 **then**
(21) $DI = Compute\_DI(P)$;
(22) **end if**

Output : ***P*** identifies the beginning of drift; ***DI*** value;

Figure 2: Pseudo-code of the BCDD algorithm

4. Drift Intensity (DI)

This section introduces the formulas that can be used to measure the *drift intensity* (*DI*). *DI* measures the intensity of a drift of the underlying data relative to their classes. The *DI* value has special significance because, first, it gives an indication of how sever the drift is, and, second, it can be used to guide the drift handling process. A very sever drift as measured by *DI* may be handled differently from a mild Drift.

4.1 Measuring DI

To measure *DI*, we have developed a set of mathematical equations that can be used for that purpose. What we want to do is take a sample subset of the dataset from before the drift location and another sample subset from the portion that is after the drift location. We will use the error rate (i.e. the rate of discrepancies between the predicted classes and the actual classes) in both subsets as a way to measure the drift intensity.

Let D_{BD} be a sample subset from the dataset before drift location and let D_{AD} be a sample subset from the dataset after drift location. The size that we use for each of these sample subsets can be around 0.5% of the entire dataset, but other sizes can be used. Let n_{BD} be the number of data tuples in D_{BD}, and n_{AD} be the number of data tuples in D_{AD}. Let e_{BD} be the number of inaccurate classifications in D_{BD} , and e_{AD} be the number of inaccurate classifications in D_{AD}.

We assume that the sizes of the samples D_{BD} and D_{AD} are sufficient to be a good representative sample of the rows before the drift and those after the drift, respectively. The reason for choosing samples instead of the entire dataset is to avoid scanning the entire dataset and thus improve the performance. Let the rate of errors in D_{BD} and in D_{AD} be RE_{BD} and RE_{AD}, respectively. RE_{BD} and RE_{AD} can be computed as shown in Equation 1 and Equation 2, respectively.

$$RE_{BD} = \frac{e_{BD}}{n_{BD}} \quad \ldots. (1)$$

$$RE_{AD} = \frac{e_{AD}}{n_{AD}} \quad \ldots. (2)$$

We expect the value of *DI* to be higher if RE_{AD} is higher. Also we expect *DI* to be higher of RE_{BD} is lower. In other words, *DI* is a directly proportional to RE_{AD} and inversely proportional to RE_{BD}. Therefore *DI* can be expressed as shown in Equation 3.

$$DI = \frac{RE_{AD}}{RE_{BD}} \quad \ldots. (3)$$

However, since the number resulting from dividing RE_{AD} over RE_{BD} can be a very large number, we modify Equation 3 by using $\log_2$ to attenuate the resulting value. The resulting formula is shown in Equation 4.

$$DI = \log_2 \frac{RE_{AD}}{RE_{BD}} = \log_2(RE_{AD}) - \log_2(RE_{BD}) \quad (4)$$

We observe that in Equation 4 we need to avoid the occurrence of a zero in the denominator when e_{BD} is zero. Therefore we modify Equation 1 by adding one fictitious erroneous classification to the numerator. Hence, we replace Equation 1 with Equation 5 shown below.

$$RE_{BD} = \frac{e_{BD} + 1}{n_{BD}} \quad \ldots.(5)$$

In conclusion, to compute *DI* we need to find the values of RE_{BD} and RE_{AD} from Equations 5 and 2, then substitute in Equation 4. The following subsection shows an example.

4.2 Example of Applying DI Equations

Assume a dataset contains 1.5M data rows. The dataset has been sent to the BCDD algorithm for detecting if there is a drift. The BCDD algorithm returned that there is a drift in the dataset somewhere in the third quarter of the dataset. Assume that the size of the sample subsets D_{BD} and D_{AD}

contain 8000 rows each (i.e., around 0.5% or the dataset). Also, assume that the number of inaccurate classifications in D_{BD} is 300 and the number of inaccurate classifications in D_{AD} is 2000. This data is summarized in Table 2 below.

Table 2: Errors before and after a drift

	No. of elements	No. of inacurate prediction
Before Drift	n_{BD} = 8,000	e_{BD} = 300
After Drift	n_{AD} = 8,000	e_{AD} = 2000

Substituting in Equations 2 and 5 to obtain:

$$RE_{BD} = \frac{e_{BD}+1}{n_{BD}} = \frac{301}{8{,}000} = 0.0376$$

$$RE_{AD} = \frac{e_{AD}}{n_{AD}} = \frac{2000}{8{,}000} = 0.25$$

And substituting these results in Equation 4, we obtain.

$$DI = \log_2 \frac{RE_{AD}}{RE_{BD}} = \log_2 6.65 = 2.73$$

4.3 Zones of the DI

Here we divide the *DI* range of values into three zones. If *DI* is from 0.1 to 3 then we assume that *DI* falls in the low drift intensity zone ($Zone_L$). If *DI* is from 3 to 6 then *DI* falls in the medium drift intensity zone ($Zone_M$). And finally, any *DI* value above 6 is considered to be in the zone of high *DI* ($Zone_H$). Table 3 summarizes these zones.

Table 3: Zones of the DI

Zones	*DI* range	Intensity of drift
$\boldsymbol{Zone_L}$	0.1 – 3	Low drift intensity
$\boldsymbol{Zone_M}$	3 - 6	Medium drift intensity
$\boldsymbol{Zone_H}$	Above 6	High drift intensity

5. Implementation and Results

This section discusses the implementation of the BCDD algorithm and the performance results obtained when comparing it with another popular detection algorithm. Our goal is to demonstrate that a detection algorithm based on the binary search technique such as BCDD achieves better performance than other algorithms.

For this purpose, a classification system was developed for implementing and evaluating the BCDD algorithm. The classification system was developed on a machine with the specifications shown in Table 4.

Table 4: Specifications of experimental environment

System Model	HP ProBook 450 G1
Operating System	Microsoft Windows 10 Pro 64-bit
Processor	Intel(R) Core(TM) i7-4702MQ CPU @ 2.20GHz, 2201 Mhz, 4 Core(s), 8 Logical Processor(s)
RAM	Installed Physical Memory 8.00 GB

In our implementation, the tools and programs that were used are as follows,

- Microsoft .Net Framework Version 4.6.
- Microsoft Visual Studio Enterprise 2015.
- C# Programming Language.
- SQL Server 2016.

Many experiments were conducted to evaluation the performance of the BCDD algorithm and compare it with another algorithm, ADWIN algorithm [20, 21]. The following reasons justify the selection of ADWIN algorithm for comparison with the BCDD algorithm.

- The ADWIN algorithm proved its robustness in detecting concept drift [22, 23].
- The ADWIN algorithm is incorporated into several predictive and clustering methods, and is integrated with statistical approaches such as Kernel Density Estimation (KDE) [24, 25].

5.1 Datasets Used in Evaluation

The dataset that we used to conduct our experiments is based on a dataset that we imported from RapidMiner [26]. RapidMiner is a public software platform that is widely used to provide data mining tools for research and educational purposes. The dataset that was imported is called "Deals" and it contains about 1,000 tuples. The data in the “Deals” dataset is about customers’ predictions. It is basically used to predict whether a current customer is likely to continue to be a future customer. The prediction depends on a set of customer’s attributes such as: age, gender, and payment method. The attribute "future customer" is the class label. Table 5 shows sample data from the “Deals” dataset.

Table 5: Sample data of "Deals" dataset

Row No.	Age	Gender	Payment Method	Future Customer
1	64	male	credit card	yes
2	35	male	cheque	yes
3	25	female	credit card	yes
4	39	female	credit card	no
5	39	male	credit card	yes
6	28	female	cheque	no
7	21	female	credit card	yes
8	48	male	credit card	yes
9	70	female	credit card	no
10	36	male	credit card	yes

The number of tuples in "Deals" dataset is small. A set of procedures were performed to increase the size of the dataset and create a set of versions of the “Deals” dataset that are huge in size. So we created ten datasets where the smallest one has one million row. These datasets are summarized in Table 6.

For testing purposes, a drift was injected in each of these datasets somewhere in the fourth quarter of the dataset.

Table 6: Characteristics of the used dataset

Dataset	No. of Tuples	Size (MB)
DS_1	1 Million	65
DS_2	2 Million	130
DS_3	3 Million	195
DS_4	4 Million	259
DS_5	5 Million	324
DS_6	6 Million	389
DS_7	7 Million	454
DS_8	8 Million	519
DS_9	9 Million	584
DS_{10}	10 Million	648

5.2 Experiment Procedure and Results

The goal of the experiments is to evaluate the performance of the BCDD algorithm and compare it with that of the ADWIN algorithm [20, 21]. Before describing the results, the following are some notes about the experiments we conducted.

1. T_{ER} was selected to be up to 3%. If the algorithm detects that that E_R is greater than 3%, then this indicates the existence of a drift.
2. Before running any of the algorithms, a timer was set for measuring the execution time for each run.
3. Each of the two algorithms, BCDD and ADWIN, was run against each of the datasets shown in Table 6 and the time it took to detect the drift in each case was recorded.

The charts shown in Figure 3 depict the performance of each algorithm as the size of the dataset increases. We notice that as the size of the dataset increases, the performance of BCDD algorithm progressively outperforms that of ADWIN. In other words, the performance of ADWIN degrades faster than BCDD as the size of the dataset is increased. This is in line with what we expected since binary search of ordered data outperforms linear search and the performance becomes more obvious as the size of the data increases.

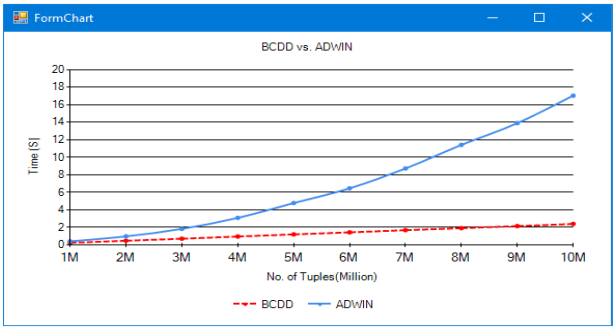

Figure 3: Results of comparison BCDD algorithm with ADWIN algorithm performance

6. Conclusion

Concept drift is a major problem for classification systems. A concept drift prevents a classifier from producing accurate classifications because it makes the classification model either outdated or totally obsolete. Before we can handle the problem of concept drift, we need to be able to detect its existence and measure its intensity. In this paper, we have introduced a novel algorithm for detecting concept drift. It is different from existing algorithms in that it is based on the idea of binary search, by progressively dividing the dataset into halves until a drift is found or the dataset is declared to be drift-free. Consequently, the performance of our algorithm is better that of other algorithms especially for huge datasets as demonstrated in Section 5.

Further, we introduced a set of formulas that can be used for measuring the drift intensity. Knowing the drift intensity can help us determine how we want to handle the drift. If the drift intensity is high, for example, then we may choose to recreate the classification model solely based on the data after the drift and totally ignore the data before the drift. If the drift intensity is low, then we may generate the new classification model based on both data before drift and data after drift, but with data after the drift having more influence (by giving it more weight) on the model generation process. In a future research, we will examine the details of how a new classification model can be generated by taking the drift intensity into consideration.

References

[1] A. Alashqur , "Representation Schemes Used by Various Classification Techniques–A Comparative Assessment," International Journal of Computer Science Issues (IJCSI), vol. 12, no. 6, pp. 55-63, November 2015.

[2] A. Alashqur, "A Novel Methodology for Constructing Rule-Based Naïve Bayesian Classifiers," International Journal of Computer Science & Information Technology (IJCSIT), vol. 7, no. 1, pp. 139-151, February 2015.

[3] H. Ogbah, A. Alashqur and H. Qattous, "Predicting Heart Disease by Means of Associative Classification," International Journal of Computer Science and Network Security (IJCSNS), vol. 16, pp. 24-32, September 2016.

[4] J. B. Gray and G. Fan , "Classification tree analysis using TARGET," Computational Statistics & Data Analysis, vol. 52, no. 3, pp. 1362-1372, 2008.

[5] D. AL-Dlaeen and A. Alashqur , "Using Decision Tree Classification to Assist in the Prediction of Alzheimer's Disease," in In Computer Science and Information Technology (CSIT), 2014 6th International Conference on (pp. 122-126). IEEE., March 2014.

[6] S. B. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques," Informatica, vol. 31, pp. 249-268, 2007.

[7] M. Goudbeek and D. Swingley, "Supervised and Unsupervised Learning of Multidimensional Acoustic Categories," Journal of Experimental Psychology: Human

Perception and Performance, vol. 35, no. 6, p. 1913–1933, 2009.
[8] R. Elwell and R. Polikar, "Incremental Learning of Concept Drift in Nonstationary Environments," IEEE TRANSACTIONS ON NEURAL NETWORKS, vol. 22, no. 10, pp. 1517-1531, OCTOBER 2011.
[9] I. Zliobaite, M. Pechenizki and J. Gama, "An overview of concept drift applications," Springer International Publishing, vol. 16, no. 978-3-319-26989-4, pp. 91-114, 2016.
[10] I. Zliobaite, "Learning under Concept Drift: an Overview," arXiv, 2010.
[11] C. Lanquillon, "Enhancing text classification to improve information filtering," Kunstliche Intelligenz, vol. 16, no. 2, pp. 37-38, 2002.
[12] G. J. Ross, N. M. Adams, D. K. Tasoulis and D. J. Hand, "Exponentially weighted moving average charts for detecting concept drift," Pattern Recognition Letters, vol. 33, no. 2, p. 191–198, 15 January 2012.
[13] D. Kifer, S. Ben-David and J. Gehrke, "Detecting change in data streams," Proceeding VLDB '04 Proceedings of the Thirtieth international conference on Very large data bases, pp. 180-191 , 2004.
[14] J. Gama, P. Medas, G. Castillo and P. Rodrigues, "Learning with Drift Detection," In Proc. of the 17th Brazilian symp. on Artif. Intell. SBIA, p. 286–295, 2004.
[15] E. Ikonomovska, J. Gama and S. Džeroski, "Learning model trees from evolving data streams," Data Min Knowl Disc, vol. 23, no. 1, p. 128–168, 2011.
[16] J. Gama, R. Fernandes and R. Rocha, "Decision trees for mining data streams," Intelligent Data Analysis , vol. 10, no. 1, pp. 23-45 , 2006.
[17] P. Vorburger and A. Bernstein, "Entropy-based Concept Shift Detection," Sixth International Conference on Data Mining (ICDM'06), pp. 1113 - 1118, 2006.
[18] T. Dasu, S. Krishnan, S. Venkatasubramanian and K. Yi, "An Information-Theoretic Approach to Detecting Changes in Multi-Dimensional Data Streams," In Proc. Symp. on the Interface of Statistics, Computing Science, and Applications, 2006.
[19] R. Sebastião and J. Gama, "Change Detection in Learning Histograms from Data Streams," Progress in Artificial Intelligence: 13th Portuguese Conference on Aritficial Intelligence, pp. 112-123, 2007.
[20] A. Bifet and R. Gavaldà, "Kalman Filters and Adaptive Windows for Learning in Data Streams," In Proc. of the 9th International Conference on Discovery Science, pp. 29-40, 2006.
[21] A. Bifet and R. Gavalda, "Learning from Time-Changing Data with Adaptive Windowing," In Proc. of SIAM international conference on Data Mining, p. 443–448, 2007.
[22] I. ˇZliobait˙, J. Bakker and M. Pechenizkiy, "OMFP: An Approach for Online Mass Flow Prediction in CFB Boilers," 12th International Conference, DS, p. 272–286, 2009.
[23] J. Bakker , M. Pechenizkiy, I. Žliobaitė, A. Ivannikov and T. Kärkkäinen, "Handling outliers and concept drift in online mass flow prediction in CFB boilers," In Proceedings of the Third International Workshop on Knowledge Discovery from Sensor Data, pp. 13-22, 2009.
[24] A. Bifet, G. Holmes, B. Pfahringer and R. Gavalda, "Improving Adaptive Bagging Methods for Evolving Data Streams," In Asian Conference on Machine Learning, pp. 23-37, 2009.
[25] A. Al-Mamun, A. Kolokolova and D. Brake, "Detecting Contextual Anomalies from Time-Changing Sensor Data Streams," Proceedings of the ECMLPKDD Doctoral Consortium, 2015 .
[26] "rapidminer," 19 Nov. 2016. [Online]. Available: https://rapidminer.com/.

Hisham Ogbah is currently working towards the MSc degree in Computer Science at the Applied Science University (ASU) in Amman, Jordan. He received his B.Sc. degree in Computer Science from Sikkim Manipal University, India, in 2008. Between 2009 and 2013 he worked as a software developer in Yemen. His research interests include classification techniques and concept drift.

Abdallah Alashqur is an associate professor in the Software Engineering Department at the Faculty of IT, Applied Science University (ASU), Amman, Jordan. Dr. Alashqur holds a Master's and a Ph.D. degrees from the University of Florida, Gainesville. After obtaining his Ph.D. degree in 1989, he worked for around seventeen years in industry (in the USA). He joined ASU in 2006. His research interests include data mining and database systems.