Journal of Theoretical and Applied Information Technology
15th May 2018. Vol.96. No 09

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

USING DRIFT INTENSITY AS A BASIS FOR HANDLING CONCEPT DRIFT IN CLASSIFICATION SYSTEMS

[1]HISHAM OGBAH, [2]ABDALLAH ALASHQUR

[1]Faculty of Information Technology,
Applied Science University, Amman, Jordan
[2]Professor, Faculty of Information Technology
Middle East University, Amman, Jordan
Email: hisham.ogbah@gmail.com , aalashqur@meu.edu.jo

ABSTRACT

Concept drift is a known problem that can occur in classifier systems. Detecting and handling concept drift is an active area of research. Once a concept drift is detected, it has to be handled by updating or re-generating the classification model. In this paper, a new approach is introduced for handling concept drift, where a drift intensity measure is used to quantify the intensity of a concept drift. The model generation process uses the drift intensity measure while generating a new model. If the drift intensity is high, the model generation process discards old data (data before the drift occurrence) and builds a new model solely based on the new data after drift. On the other hand, if the drift intensity is low or moderate, the model generation process takes into account both old data and new data but it gives more weight (proportional to the drift intensity) to the new data as compared to old data.

Keywords: Data Mining, *Classification, Concept Drift, Drift Handling, Big Data.*

1. INTRODUCTION

In classification each tuple in a dataset is mapped to one of a limited set of classes. A specific column in the dataset is designated as the *class label*, in which the class name of each tuple is stored [1,2,3,4,5,6]. The primary objective of a classifier is to be able to predict the classification of newly added tuples that have not yet been classified. This is usually accomplished by, first, going through a machine learning phase in which the system learns the classification model from a training dataset (i.e., a pre-classified dataset). After learning the classification model, the system is ready to go through the second phase in which it can predict the classification of new tuples [7,8].

A problem takes place if the distribution of data tuples with respect to classes changes after a period of time. Meaning that data tuples that used to legitimately map to, say, class1, now map to another class, say, class2. However, a system that learned the classification model prior to the change in data distribution will continue to predict class1 for the same piece of data, while in reality it should be class2. Thus a mismatch occurs between the system's predictions and the actual classifications, which is referred to as *concept drift* [9,10,11]. If a concept drift is not detected and dealt with properly, it results in producing inaccurate predictions.

In this paper we introduce a new algorithm for handling a concept drift. It is based on the authors' earlier work on detecting concept drift [6], where a way for quantifying and measuring the drift intesity (DI) was introduced. Further, based on the DI value, the intensity of a drift was categoriezed as either high, medium, or low.

The handling algorithm introduced in this paper generates a new classification model if a drift has been detected. It is different from existing algorithms in that the newly generated model is based on both data before drift as well as data after drift. However the weight given to data after drift by the algorithm is larger than the weight given to data before drift. Therefore data after drift has more influence on the new model generation process than data before drift.

The difference between the weight given to data after drift and data before drift is adjusted by the algorithm in a way proportional to the drift intensity. In other words, the higher the drift intensity is, the more is the influence of data after drift on the model generation process as compared to the influence of data before drift.

In the extreme case when the drift intensity is sever, the algorithm gives a weight of zero to the data before drift and full weight to the data after drift. In this case it generates a new classification model solely based on the data after the drift whereas the data before drift is totally ignored during the model generation process.

The user is given the ability to overwrite the algorithm's default weights. For example, if the drift intensity is low, the algorithm will give some weight to data before drift. The user can overwrite this behavior and decide to give zero weigh to data before drift and full weight to data after drift, thus forcing the generation of a new model solely based on data after drift. This capability makes our new algorithm more generic in the sense that it can mimic the behavior of traditional concept drift algorithms, which totally ignore data before drift when generating a new classification model.

The remainder of this paper is organized as follows. In Section 2 we provide a survey of related work. Section 3 presents the formulas used to compute the Drift Intensity. Section 4 describes the new algorithm used for handling concept drift. Section 5 explains decision tree algorithm and how to apply weight-based approach with classification algorithm. Implementation results are shown in Section 6. Finally, conclusions are given in Section 7.

2. RELATED WORK

Many adaptive algorithms used for learning use the most recent data as a basis for prediction because they assume such data to be most informative. Therefore, data management typically aims at learning from the most recent data. There are two types of training windows size: *fixed* or *variable*. Sliding window of a *fixed size* stores a particular number of the most recent instances. When a new instance arrives, it is saved in the training window and the oldest one is thrown away. The sliding window of *variable size,* on the other hand, varies the number of instances in a the window as time goes on depending on either the indications of a change detector, or after a certain timestamp. This approach uses heuristics to adjust the window size to the current extent of drift [12, 13]. Below is a brief description of some of the techniques and mechanisms that are used to handle the predictive models of evolving data.

The *WINNOW* [14] is a linear classifier that uses an approach in which the trainer responds to each sample according to a certain hypothesis. Based on the current classification of the example, the learner can updates the hypothesis. The main characteristic of *WINNOW* is its resilience to irrelevant features [14]. *WINNOW* does not have explicit forgetting techniques. Therefore, adaptation takes place mainly when the old concepts are diluted, because of the new arriving data. The *WINNOW* is capable of adapting itself to slow drifts over time. Its main limitation is the slow adaption to the occurrence of sudden drifts. A tradeoff between stability and sensitivity is required for setting these parameters [10, 15].

The algorithm family *FLORA* [16] presents a new approach in which the predictive model is kept consistent with a set of very recent instances. FLORA uses a sliding window of fixed size. It stores the most recent models on the basis of a first-in-fist-out (FIFO) approach. At each step the algorithm builds a new model based on instance selection from the training window that moves over recently arrived instances. The model is updated depending on two processes: a learning/selecting process, which updates the model based on the recent data. And a forgetting process that throws away data that is moved out of the window. The primary challenge in this approach is to determine an appropriate window size. In case of using a large window it gives a better performance in stable periods. But it responds to concept drifts in slower fashion. In case of using a short window, it mimics the current distribution more precisely. Therefore, it can guarantee fast adaptation as time goes on with concept drifts. But during stable period a very short window degrades the performance of the system [10, 17].

The *FLORA2* [16] is one of the earliest algorithms, that uses an varying window size. Adaptive window maintains the consistency of the predictive model with current concept. Upgraded versions of the algorithm have been improved to work with recurring concept such as *FLORA3*, and noisy data *FLORA4*. Another study described in [18] presents a theoretically supported technique for recognizing and handling concept drift with Support Vector Machines (SVM). The assumption behind depending on windowing is that the novelty of the data is associated with relevance and significance. But this assumption may not be true in every circumstance. For example, when concepts reoccur or when data is noisy, novelty of data does not mean relevance. Learning adaptive window is presented in [19, 20].

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The author of [21] presents a new approach for gradual forgetting, that is applied to learn concept drift. The approach proposes the introduction of a time-based forgetting function, which makes the last instances observation more significant for the learning algorithms than the old ones. The advantage of time-based forgetting function that there are no instances are completely discarded from the training dataset. And, they are decreased with time. This approach provides each training instance with a weight that reflects their age, according its appearance over time. Instance weighting is based on an the assumption that the significance of an tuple in the training dataset must be decreased with as time goes on. This approach which is linear decrease approach can also be found in [22], and another technique that uses exponential decrease is published in [23].

In general, the existing approaches to cope up with the phenomenon of concept drift can be divided into two categories. First, approaches that adapt a model at regular intervals without considering whether drifts have really occurred or not, such as fixed or variable sliding window and weighted instances. When a sliding window is used, the model is created only from the instances; that are included in the window within a particular number of the most recent instances in fixed sliding window or within a certain time in variable sliding window. Besides forgetting all previous instances. The key point in this approach is how to select the appropriate window size. Whereas the idea of weighted instances is based on that the importance of instances should decrease with time according to their age and appearance. Second, approaches that first detect drifts, and then the model is adapted to these drifts. In order to detect concept drifts, the model is monitored over time. If a concept drift occurred, some procedures to adapt the model to these drift should be taken. For example, when a sliding window of variable size is used these procedures usually lead to decrease the window size according to the extent of concept drift. In both categories, none of them uses weight approach and gives more weight to the recent data, and gives less weight to the older data rather than discard them.

3. DRIFT INTENSITY (DI)

In this section, the *drift intensity (*DI*)* measure, as introduced by the authors of this paper in [6], is summarized. DI is a metric used to measure the intensity of a drift in the underlying data with respect to their classes. The DI value provides a measure of how sever the drift is. Furthermore, in this paper, we use the DI value to guide the concept drift handling process.

A set of mathematical equations were introduced in [6] to measure the DI value. Two sample subsets are taken from the dataset, one from before the drift location (D_{BD}) and the other one from after the drift location (D_{AD}). It is assumed that D_{BD} and D_{AD} are of equal size. Using sample subsets instead of the entire dataset enables us to avoid scanning the whole dataset, which degrades performance when dealing with big data sets. A reasonable size of each of D_{BD} and D_{AD} can be no more that 0.5% of the dataset.

In the following, the error rate in predictions (i.e., the rate at which the predicted classes are inconsistent with the actual classes) in both subsets, D_{BD} and D_{AD}, is used as a way to measure the intensity of a drift.

Let N_{BD} and N_{AD} be the number of data tuples in D_{BD} and in D_{AD}, respectively. Also, let E_{BD} and E_{AD} be the number of inaccurate classifications in D_{BD}, and D_{AD}, respectively. Furthermore, let RE_{BD} and RE_{AD} be the classification error rates in D_{BD} and D_{AD}, respectively. RE_{BD} and RE_{AD} can be calculated as follows.

$$RE_{BD} = {}^{E_{BD}}/_{N_{BD}} \qquad (1)$$

$$RE_{AD} = {}^{E_{AD}}/_{N_{AD}} \qquad (2)$$

The drift intensity can be computed by dividing the error rate before drift by the error rate after drift as shown in below.

$$DI = \frac{RE_{AD}}{RE_{BD}}$$

However, because the resulting value of DI based on the above equation can be a very huge number, the above equation is modified as shown in Equation 3 by applying log_2 in order to attenuate the value of DI.

$$DI = \log_2 \frac{RE_{AD}}{RE_{BD}} \qquad (3)$$

Note that the denominator in Equation 3 (RE_{BD}) becomes zero if E_{BD} is zero. To avoid this zero, Equation 1 is altered by adding "one" to the numerator as shown in Equation 4 below.

$$RE_{BD} = \frac{E_{BD} + 1}{N_{BD}} \qquad (4)$$

In summary, to calculate DI we need to compute first the values of RE_{BD} and RE_{AD} from Equations 4 and 2, respectively. Then we can substitute in Equation 3. An example of how to compute DI is shown below.

Example. Assume a certain dataset contains 3M tuples. Further assume a detection algorithm detects a drift in the third quarter of this dataset.

The sizes of the sample subsets D_{BD} and D_{AD} are 16000 rows each (which is approximately 0.5% of the size of the dataset). Also, assume that the number of inaccurate classifications in D_{BD} is N_{BD} = 600 and the number of inaccurate classifications in D_{AD} is N_{AD} = 4000. This combination of values is summarized in Table 1.

Table 1: Errors Before And After Drift

	No. of elements	No. of inaccurate prediction
Before Drift	$N_{BD} = 16{,}000$	$E_{BD} = 600$
After Drift	$N_{AD} = 16{,}000$	$E_{AD} = 4000$

By Substituting in Equations 2 and 4 we obtain:

$$RE_{BD} = \frac{E_{BD}+1}{N_{BD}} = \frac{601}{16{,}000} = 0.0375$$

$$RE_{AD} = \frac{E_{AD}}{N_{AD}} = \frac{4000}{16{,}000} = 0.25$$

Substituting the above results in Equation 3, we obtain.

$$DI = \log_2 \frac{RE_{AD}}{RE_{BD}} = \log_2 6.66 = \ 2.736$$

3.1 DI Zones

The DI range of values is divided into three zones [6]. The low drift intensity zone Z_L represents values of DI in the range between 0.1 and 3, the medium intensity zone Z_M represents values of DI in the range between 3 and 6, and the high intensity zone Z_H represents any values of DI greater than 6. Table 2 summarizes these zones.

Table 2: DI Zones

Zones	*DI* value range	Intensity of drift
Z_L	0.1 – 3	Low drift intensity
Z_M	3 - 6	Medium drift intensity
Z_H	Greater than 6	High drift intensity

4. CONCEPT DRIFT HANDLING

This chapter provides a description of a new handling algorithm called Weight-based Concept Drift Handling (WCDH) algorithm that we proposed for handling the phenomenon of concept drift in a dataset with class labels. The proposed WCDH algorithm introduces a new approach for handling concept drift depends on weight-based technique.

4.1 Overview of the WCDH Algorithm

The WCDH algorithm depends on the results that are found by the BCDD algorithm [6]. If the BCDD algorithm detects that there is a drift in the underlying data distribution, then the WCDH algorithm is called to handle the classification model depending on the DI value that was obtained from the BCDD algorithm. The following points distinguish WCDH algorithm from existing similar studies:

1) The WCDH algorithm gives the data in the underlying distribution dataset different weights according to their position (i.e., whether before or after the drift). While, the previous studies give all the data in the underlying dataset the same weight except the algorithm in [21], which decreases the weight of tuples gradually according to their age without considering whether a drift occurred or not. Therefore, the WCDH algorithm is different from other algorithms in detecting the drift and distributing the weight between old data and recent data according to the DI value.

2) The WCDH algorithm gives the data before drift less weight than the data after the drift. The change in weight is proportional to the DI value. This means that when building a new classification model it gives more significance to the data after drift and less significance to data before drift. While the previous studies either forget all the data before the drift and use only the data after the drift, or treat the old data the same as the recent data [19.20].

The WCDH algorithm will use the DI as a guidance and give the user advices as to how the concept drift should be handled. When the DI is in Z_H, the system may choose to rebuild a new classification model only based on the data after the drift and totally discard the old data. And, when the DI is in Z_L or Z_M, the system may choose to

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

give more weight to data after drift and, at the same time, take into consideration data before the drift but give it less weight. That means the data after the drift will influence in the newly classification model more than it is influence by data before the drift.

The increase in the weight value after the drift point is proportional to the value of DI. Similarly the reduction of the weight value before the drift is proportional to the value of DI. Therefore, if the DI value is high, the weight of data after the drift can be high and continues to increase until it gives full weight for data after the drift and, in contrast, zero weight for data before the drift when the value of DI is very high. On the other hand, the user can overwrite the default weight generated by the WCDH algorithm and provide another weight value. Also, the user can discard the old data and only consider the recent data on the model generation process when the DI value in the high zone. And so, the WCDH algorithm will have behavior similar to previous algorithms.

This flexibility in determining the weight for data before drift and weight for data after drift gives satisfying results based on the user needs. It must be emphasized that the proposed approach does not consider all the historical data as a training set, but a sufficient subset of the dataset that was selected to compute the DI value.

4.2 Flowchart and Pseudo Code of the Algorithm

Figure 1 shows the flowchart of the WCDH algorithm. At the beginning, the WCDH algorithm receives the following from the system: a sample subset of the dataset before drift (D_{BD}), a sample subset of the dataset after drift (D_{AD}), drift intensity (DI), and the position of drift (P) from the system.

After that, the training set is generated by concatenating the sample datasets D_{BD} and D_{AD}. The algorithm checks the DI value, if the DI value is in Z_H , the algorithm starts by giving the user the option for discarding the old data or computing the weight for D_{BD} and D_{AD} . If the user selects to discard D_{BD} from the generated training set, the algorithm discards D_{BD} and goes to the final step, which is re-generate a new classification model without D_{BD}. On the other hand, if the user selects to compute the weight for D_{BD} and D_{AD} or the DI is not in Z_H, the algorithm invokes Compute_Weight function to compute weight for D_{BD} and D_{AD} based on the formulas introduced in Section 4.3.

After computing the weight for D_{AD} and D_{AD} the algorithm gives the user the option to adjust the default weight value. Accepting the weight generated by the algorithm, the next step is to set weight for the generated training dataset. Otherwise, the algorithm allows the user to overwrite the default weight and then continue with processing weight for each tuple based on their position before or after the drift. The final step is to re-build a new classification model using any classification algorithm based on the weight assigned for tuples in data.

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

4.3 Compute_Weight Function

This section introduces the formulas that are used by WCDH algorithm to compute the weight of data before and after the drift. The weight is computed based on the DI value. The main processing of Compute_Weight function is to calculate the weight to be given to tuples. It gives more weight to data tuples after the drift and less weight to data tuples before the drift. The default if DI is in Z_L or Z_M is to use the weight computed by the system. The user can modify that if he chooses. And, the default if DI is in Z_H is to let the user use the weight values computed by the system or discard the old data and rebuild a new model based on the recent data as shown in Figure 1 and Figure 2.

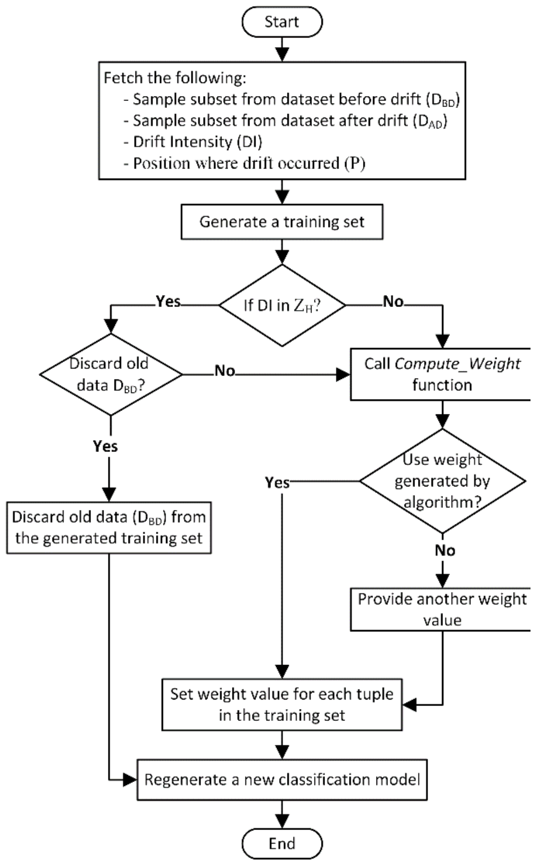

Figure 1: Flowchart of WCDH algorithm

The weight of each tuple in the training set is equal to one. What we want to do is decrease the weight given to data tuples before the drift in a way proportional to DI and, at the same time, increase the weight for data after the drift in a way proportional to DI. The minimum value of the weight before the drift is zero. And, the maximum value of weight after the drift is two. That means the proposed weight that will be added or subtracted from the default weight is in the range of 0 and 1.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
WCDH algorithm
Inputs: - Sample subset from dataset before drift (D_BD);
        - Sample subset from dataset after drift (D_AD);
        - Position where the drift occurred (P);
        - Drift intensity value (DI);
Method:
    W_BD = 1;default weight for each tuple before drift;
    W_AD = 1;default weight for each tuple after drift;
    Generate a training dataset from D_BD and D_AD ;
    if DI in Z_H and user deiced to discard D_BD then
        Discard D_BD from the training set;
    else
        call Compute_Weight(DI) for W_BD and W_AD;
        if user want to overwrite the generated weight then
            provide another weight for W_BD and W_AD;
        end if
        for each row in the generated training dataset
            if row.RID < P then
                row.weight = W_BD;
            else
                row.weight = W_AD;
            end if
        end loop
    end if
    Regenerate a new classification model;
Output: New classification model;
```

Figure 2: Pseudo-code of the WCDH algorithm

In the following, a way to compute the weight based on DI is shown. The weight value is a function of DI. Let DI_{Min} be the minimum value for DI that is equal to 0.1. And, DI_{Max} be either equal to 10 in case of DI < 10 or take the same value as DI in case of DI ≥ 10. We assume that DI_{Max} be as DI value when DI ≥ 10 since the drift in this case is very high and is preferred to give zero weight for data before drift. In other words, DI_{Max} can be expressed as shown in Equation 6.

$$DI_{Max} = \begin{cases} 10, & \text{if DI} < 10 \\ & \text{or} \\ DI, & \text{otherwise} \end{cases} \quad (6)$$

Let W_Δ be the amount of change of the weight that needs to be added or subtracted from the default weight given to tuple. To guarantee that W_Δ is in the range of 0 to 1 we use the following equation.

$$W_\Delta = \frac{DI - DI_{Min}}{DI_{Max} - DI_{Min}} \quad (7)$$

Let W_{BD} be the weight of tuple before the drift, and W_{AD} be the weight of tuple after the drift. We expect the value of W_{BD} to be lower if we subtract W_Δ from the default weight. The formula for calculating the value of W_{BD} can be expressed as shown in Equation 8.

$$W_{BD} = 1 - W_\Delta \quad (8)$$

Also, we expect the value of W_{AD} to be higher if we add W_Δ to the default weight. The formula for calculating the value of W_{AD} can be expressed as shown in Equation 9.

$$W_{AD} = 1 + W_\Delta \quad (9)$$

In conclusion, the value of W_{BD} to be lower by substituting in Equation 8, and the value of W_{AD} to be higher by substituting in Equation 9. The following subsection shows an example.

4.4 Example of Applying *Compute_Weight* Function

In this example we will use the same sample subsets from dataset that we used in the example of applying DI equations. Where DI value is equal to 2.73 and the sample subsets of D_{BD} and D_{AD} that forming the generated training is 16000 rows, each subset is 8000 rows.

For computing W_Δ, first determine the value of DI_{Min} and DI_{Max} which they are equal to 0.1 and 10 respectively. DI_{Max} is 10 since DI value is less than 10. Substituting in Equations 7, W_Δ result is obtained:

$$W_\Delta = \frac{DI - DI_{Min}}{DI_{Max} - DI_{Min}} = \frac{2.736 - 0.1}{10 - 0.1} = 0.266$$

Substituting in Equations 8 and 9, the following results are obtained:

$$W_{BD} = 1 - W_\Delta = 1 - 0.266 = 0.734$$

$$W_{AD} = 1 + W_\Delta = 1 + 0.266 = 1.266$$

The weight for each tuple before the drift point in the training set is equal to 0.734 and the weight for each tuple from the drift point to the end of the training set is 1.266.

5. BUILD A NEW MODEL USING WEIGHT-BASED APPROACH

This section introduces an explanation of building a classification model using classification algorithm with the proposed weight-based approach. And, describes how to integrate weight-based approach within classification algorithms. The weight is distributed on the tuples of the generated training dataset based on their position either before the drift point or after the drift point, then a new classification model is derived based on the analysis of the training dataset. A decision tree classification algorithm is one of the most popular classification algorithms in data mining [24,25] and it is applied in this paper to derive a new classification model. The following subsection describes the decision tree classification and shows how to apply the proposed weight-based approach with the decision tree for building a new classification model.

The decision tree represents an effective classification technique. It aims to split the training set into groups as possible in terms of the variable to be predicted. It takes as input a set of classified data (training set), and produces a tree that resembles a flowchart like tree structure, where each non-leaf node (internal node) represents a test on an attribute and each edge represents a result of the test. If the result of a test has two possible tuples belong to more than one class, further tests are required to complete the classification along that branch through adding more non-leaf nodes to the tree. Alternatively, if the outcome of a test consists of tuples that belong to the same class, then a leaf node is inserted to indicate that the tuples that satisfy the conjunction of the tests along the path from the root-node to the leaf-node are belong to only one class.

Various algorithms are available for constructing decision trees from training datasets such as: ID3 (Iterative Dichotomizer 3), C4.5, and CART (Classification and Regression Tree) [2, 4, 26]. There are many differences between these algorithms such as selecting certain attributes for splitting nodes in the tree.

In order to build a decision tree from training dataset an attribute selection measure is needed. The attribute selection measure is used to determine how the tuples at a given node are to be split. The attribute selection measure provides a ranking for each attribute representing the given training dataset. The attribute that have the highest score of the measure is selected as the splitting attribute for the given tuples.

Several common measures exist in the literature. In this paper a measure called Entropy (also called Information Gain) is used as one of those measures. This measure is used by ID3 for building decision trees [26]. At each node in the decision tree the attribute with the highest Entropy is selected as the splitting attribute. To find the attribute has the highest entropy, a three-step proces is applied. Those steps are represented in the form of three equations 10, 11, and 12, that are explained as the following.

- Step (1). Equation 10 is applied to compute the information needed to classify a tuple in the dataset D, where i is the class label, P_i is the probability a tuple falls in the i^{th} class.

$$Info(D) = -\sum_{i=1}^{m} P_i \log_2 P_i \quad (10)$$

- Step (2). Equation 11 is applied to find the information required to classify the dataset D based on the partitioning by X. Each D_j is a partition of the dataset D, and $|D|$ is the number of the tuples in the partition.

$$Info_X(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times I(D_j) \quad (11)$$

- Step (3). In Equation 12 information gained by branching on attribute X can be obtained by subtracting the result of Equation 11 from the result of Equation 10 as shown below.

$$Gain(X) = Info(D) - Info_X(D) \quad (12)$$

Equation 10 is applied to the dataset only one time at the beginning. Equations 11 and 12 are applied for each attribute. The attribute that has the highest information gain is selected as the splitting attribute. This process is repeated at each level of the tree until leaf nodes are reached, which represent tuples that belong to only one class.

The contribution to improve decision tree is proposed by enabling the decision tree algorithm to support the weight-based approach. Indeed, when the result of a test belongs to different classes and no further splitting is possible, then decision tree selects a leaf node by choosing the most frequent class label. The integrating with proposed weight-based algorithm provides better solution for this issue, where a leaf node is selected by grouping the tuples that have same classes and using SUM function to compute the weight for each class. Then, selects the class that has the highest weight as a leaf node for the selected branch in the tree. The following subsections show how the decision tree with weight-based approach is built.

6. IMPLEMENTATION AND RESULTS

This section discusses the implementation of the WCDH algorithm and results obtained when applying it with decision tree algorithm to build a

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

new classification model. An experiment was conducted to evaluate the effectiveness of the WCDH algorithm with classification algorithm, and how the proposed approach achieves promising results. The model of the experiment is built using a decision tree algorithm. We use the first dataset that was used in evaluation the BCDD algorithm [6], which has one million row. Before conducting the experiment, let's see how the model of the "Deals" dataset was been before it was injected with a drift. Figure 3 shows the decision tree model of Deals dataset.

The following are some notes about the xperiment we conducted:

1- The model was injected with a drift in the right branch in which "Age = 34–72" and goes through the edges identified by conditions "Gender = male", and "Payment Method = credit card" ends up at the leaf node labeled with a "yes". It is assumed that this path is changed to map class "no".
2- The experiment show that the number of tuples before the drift point that map to class "yes" is 1400 and the number of tuples that was injected to map class "no" after the drift is 731.
3- After conducting the proposed approaches for detecting and handling concept drift the decision tree algorithm was called to build a new model.

The dataset was injected to contain a medium DI in Z_M. After running the BCDD algorithm for detecting the drift and measuring its intensity, the BCDD algorithm shows that the DI is 4.6. Then the WCDH algorithm was called to handle the drift by distributing the weight for each tuple in the generated training dataset based on its position before or after the drift. The WCDH algorithm shows that W_{BD} is 0.546 and W_{AD} is 1.454. After that, the decision tree was invoked to build a new classification model.

As shown in Figure 4, the subtrees in which "Age > 34" was pruned since all the leaf nodes for that branch belong to the same class. The leaf node in which "Payment Method = credit card", results to class "no", because the weight of class "no" is higher than the weight of class "yes", where weight of class "no" is 1063.2 and weight of class "yes" is 763.8 therefore this branch is mapped to class "no". The two right branches in Figure 3 which "Age = 34-72" and "Age > 72" have been merged to one branch in which "Age > 34" as shown in Figure 4 since they map to the same class.

From the above experiment it can be concluded that using the weight-based approach for generating a new classification model gives satisfying results. This is the same with what we expected since the weight-based approach will support a classification model to produce more accurate classification.

7. CONCLUSION

The occurrence of a concept drift in data is considered a problem that impacts the results of a classifier. A concept drift inhibits a classifier from generating accurate predictions because it results in making the classification model outdated if not totally obsolete. It is important to be able to detect and handle a concept drift properly. This paper introduces a novel algorithm for handling concept drift. The key distinguishing feature of this algorithm is that it adapts itself based on the drift intensity. If the drift intensity is in the high zone, the algorithms, by default, gives zero weight to data before the drift and full weight to data after the drift whenever it generates a new classification model. The user is given the option to overwrite the default behavior.

However, if the drift is in the low or medium zones, the algorithm computes weights such that the weight given to the data after the drift is more than the weight given to the data before the drift. The higher the drift intensity, the more weight is given to the data after the drift as compared to data before the drift. Again here, the user is given the option to overwrite the default behavior. The user can supply weights different from the default ones to influence how the new model is generated.

This ability to generate a new model based on the new data (after drift) and old data (before drift) can be useful in some applications especially if the drift intensity is small. It keeps the power in the hands of the user, who may decide that the older data has some merit and can be considered in the model generation process but with less weight. The user may also interfere and decide to give older data zero weight, thus, generating the new model solely based on the data after the drift point.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

REFERENCES

[1] 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.

[2] 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.

[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 Computer Science and Information Technology (CSIT), 2014 6th International Conference,* pp. 122-126, March 2014.

[6] H. Ogbah and A. Alashqur, "A New Approach for Detecting Concept Drift and Measuring its Intensity in Large Datasets," *International Journal of Computer Science and Network Security (IJCSNS),* vol. 16, no. 12, pp. 108-115, December 2016.

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

[8] 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.

[9] 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.

[10] I. Zliobaite, "Learning under Concept Drift: an Overview," *arXiv preprint arXiv,* 2010.

[11] I. Zliobaite, M. Pechenizki and J. Gama, "An overview of concept drift applications," *In Big Data Analysis: New Algorithms for a New Society ,Springer International Publishing,* pp. 91-114, 2016.

[12] A. Arasu and G. S. Manku, "Approximate Counts and Quantiles over Sliding Windows," *In Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems,* pp. 286-296, June 2004.

[13] M. Matysiak, "Data Stream Mining: Basic methods and techniques," RWTH Aachen University, 2012.

[14] N. Littlestone, "Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm," *In Foundations of Computer Science,* pp. 68-77, October 1987.

[15] G. A. Carpenter, S. Grossberg and D. B. Rosen, "Fuzzy ART: Fast Stable Learning and Categorization of Analog Patterns by an Adaptive Resonance System," *Neural Network,* vol. 4, pp. 759-771, 1991.

[16] G. Widmer and M. Kubat, "Learning in the Presence of Concept Drift and Hidden Contexts," *Machine Learning,* vol. 23, pp. 69-101, 1996.

[17] A. Tsymbal, "The problem of concept drift: definitions and related work," *Computer Science Department, Trinity College Dublin,,* 29 April 2004.

[18] R. Klinkenberg and T. Joachims, "Detecting concept drift with support vector machines," *In ICML,* pp. 487-494, 2000.

[19] M. A. Maloof and R. S. Michalski, "A method for partial-memory incremental learning and its application to computer intrusion detection," *In Proc. of the 7th IEEE Int. Conf. on Tools with Artif. Intell.,* p. 392–397, 1995.

[20] I. Žliobaite and L. Kuncheva, "Determining the training window for small sample size classification with concept drift," *In Proc. of IEEE Int. Conf. on Data Mining Workshops. ICDMW,* p. 447–452, 2009.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[21] I. Koychev, "Gradual Forgetting for Adaptation to Concept Drift," *In Proc. of ECAI Workshop on Current Issues in Spatio-Temporal Reasoning,* p. 101–106, 2000.

[22] I. Koychev, "Tracking changing user interests through prior-learning of context," *In Proc. of the 2nd int. conf. on Adaptive Hypermedia and Adaptive Web-Based Systems,* p. 223–232, 2002.

[23] R. Klinkenberg, "Learning drifting concepts: Example selection vs. example weighting," *Intelligent Data Analysis,* vol. 8, no. 3, p. 281–300, 2004.

[24] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand and D. Steinberg, "Top 10 algorithms in data mining," *Knowledge and information systems,* pp. 1-37, 2008.

[25] R. C. Barros, M. P. Basgalupp, A. C. P. L. F. d. Carvalho and A. A. Freitas, "A Survey of Evolutionary Algorithms for Decision-Tree Induction," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* pp. 291-312, 2012.

[26] B. Hssina, A. Merbouha, H. Ezzikouri and M. Erritali, "A comparative study of decision tree ID3 and C4.5," *International Journal of Advanced Computer Science and Applications ((IJACSA),* pp. 13-19, 2014.

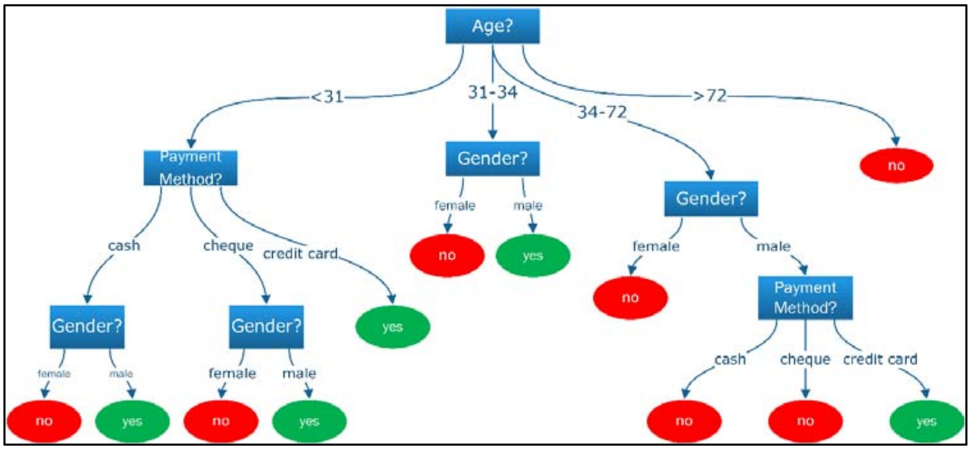

Figure 3: Decision Tree Model of 'Deals' dataset before it was injected with drift

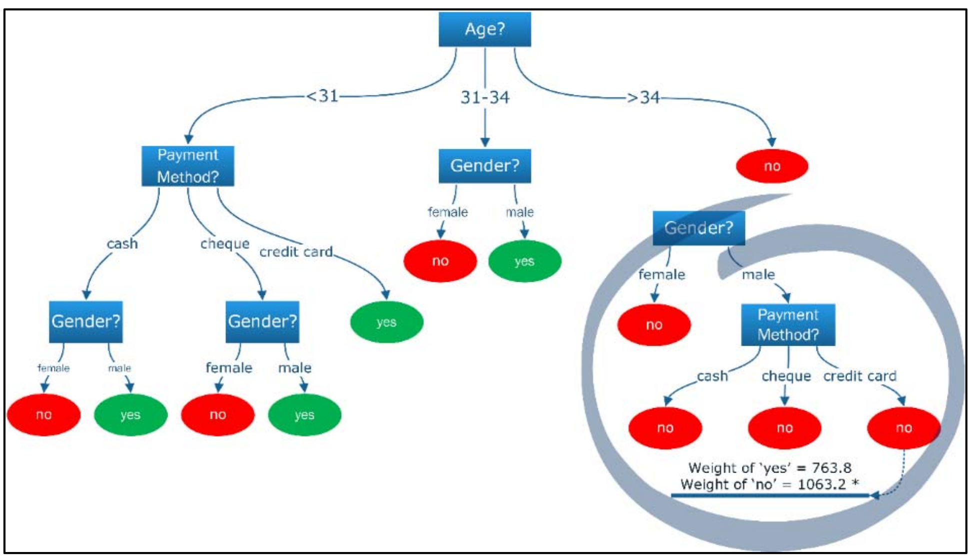

Figure 4: Decision Tree Model of 'Deals' dataset after re-building a new model using Weighting approach