Preprint of:
Zhao, D., Koh, Y. S., Fournier-Viger, P. Measuring Drift Severity by Tree Structure Classifiers (2022). International Joint Conference on Neural Networks (IJCNN 2022),

Measuring Drift Severity by Tree Structure Classifiers

Di Zhao
School of Computer Science
The University of Auckland
Auckland, New Zealand
dzha866@aucklanduni.ac.nz

Yun Sing Koh
School of Computer Science
The University of Auckland
Auckland, New Zealand
y.koh@auckland.ac.nz

Philippe Fournier-Viger
College of Computer Science and Software Engineering
Shenzhen University
Shenzhen, China
philfv@szu.edu.cn

***Abstract*—Streaming data has become more common as our ability to collect data in real-time increases. A primary concern in dealing with data streams is concept drift, which describes changes in the underlying distribution of streaming data. Measuring drift severity is crucial for model adaptation. Drift severity can be a proxy in choosing concept drift adaptation strategies. Current methods measure drift severity by monitoring the changes in the learner performance or measuring the difference between data distributions. However, these methods cannot measure the drift severity if the ground truth labels are unavailable. Specifically, performance-based methods cannot measure marginal drift, and distribution-based methods cannot measure conditional drift. We propose a novel framework named Tree-based Drift Measurement (TDM) that measures both marginal and conditional drift without revisiting historical data. TDM measures the difference between tree classifiers by transforming them into sets of binary vectors. An experiment shows that TDM achieves similar performance to the state-of-the-art methods and provides the best trade-off between runtime and memory usage. A case study shows that the online learner performance can be improved by adapting different drift adaptation strategies based on the drift severity.**

***Index Terms*—Data Stream, Concept Drift, Drift Severity**

I. Introduction

Concept drift is recognized as the root cause of decreased effectiveness in many data-driven information systems [1], [2]. A concept drift renders an old classifier irrelevant for new data; thus, the classifier's performance deteriorates. Increasing attention has been paid to approaches that perform concept drift detection in an unsupervised manner, *i.e.*, without access to the ground truth labels [3]. To tackle this, current research on concept drifts measures drift severity [2] as a proxy to indicate whether a drift needs to be adapted.

To solve this problem, we present a post-hoc drift severity measurement framework called *Tree-based Drift Measurement* (TDM). TDM measures a drift's severity by calculating the difference between the trees built before and after a concept drift. TDM assumes that regardless of the type of concept drift, it will cause changes in the model structure. TDM is based on the Drift Detection Method (DDM) framework [4]. It measures the drift severity when the DDM detects a concept drift. Since our framework works in an online environment, we choose the Hoeffding Tree (HT) [5] as online classifier.

Figure 1 shows the four components of TDM: *Input*, *Modelling*, *Vectorization* and *Measurement*. The *Input* component receives the data stream and stores the data in two windows, the reference window and the current window. The reference window stores the data from the past concept, and the current window stores the data from the most recent concept. If a concept drift is detected, the *Modelling* component creates two Hoeffding Trees, that is, for the reference window and the current window. The *Vectorization* component then transforms the two Hoeffding Trees into two sets of vectors, and the *Measurement* component measures the difference between the two sets of vectors.

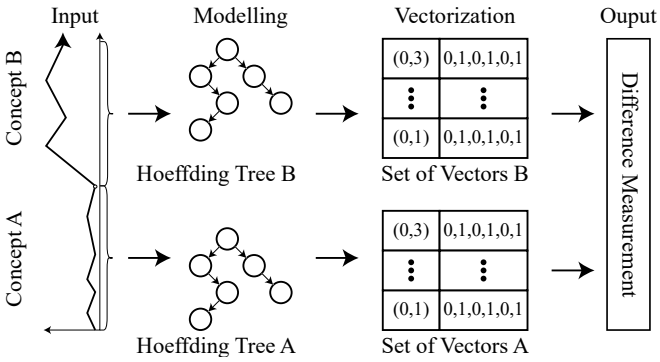

Fig. 1. Framework of TDM

The major contribution of this paper is three-fold:

- The TDM framework and implementation. TDM measures the drift severity of each concept drift identified in a stream.
- We propose a feature dictionary to transform any tree-based classifier into a set of vectors and indicate the difference between trees' structures by measuring the similarity between two sets of vectors. We use Operational Taxonomic Units Expression to reduce the cost of calculating similarities.
- Our experiment shows that TDM is able to measure the drift severity without access to the data and provides the best trade-off between runtime and memory usage. A case study demonstrates how the online learner performance

improves by adapting different drift adaptation strategies according to the drift severities.

The rest of the paper is organized as follows. Section II reviews related work. Section III introduces preliminaries about concept drift and drift severity measurement. Section IV describes implementation details of the TDM and discusses its complexity. Section V presents the datasets evaluation results of TDM and describes a case study. Lastly, Section VII concludes the paper, discusses the limitations of TDM and plans for future work.

II. Related Work

Performance-based drift detection algorithms form the largest category of drift detection algorithms. Performance-based algorithms focus on tracking changes in the online error rate of base classifiers. If an increase or decrease of the error rate is proven to be statistically significant, a drift alarm and upgrade process will be triggered [2]. For example, the CUMulative SUM (CUSUM) [6] method is a sequential analysis technique that detects changes by using the residual of the learner, and there are different variations of that methods [7]. The FLOating Rough Approximation (FLORA) family of algorithms use the overall accuracy and coverage of the learner as a measure of change [8], [9]. Drift Detection Method (DDM) [4] is the first algorithm that defines the warning level and drift level for concept drift detection. Currently, most of the state-of-the-art drift detection methods are based on DDM [10]–[14].

Distribution-based drift detection algorithms use a distance function to quantify the dissimilarity between historical and new data distribution. If the dissimilarity is statistically significant, the system will trigger a learning model up-gradation process. They are able to provide location information about the drift but also have higher computational costs than the performance-based algorithms. The first formal treatment of distribution-based drift detection in data streams proposed by Kifer et al. [15] is based on a two-window paradigm. Statistical Change Detection [16], PCA-based Change Detection Framework [17], and Least Squares Density Difference Change Detection Test [18] were proposed to detect changes in multidimensional data streams. Feature Drift Detection [19] was proposed to detect changes in the feature distribution individually. BDDM [20] uses Bhattacharyya distance to identify gradual or abrupt concept drifts in the data distribution. CURIE [21] detects concept drift by representing the distribution of the data stream in the grid of cellular automata. Approaches such as pattern-based change detection work on an abstract form of the data, preventing the search for changes on the raw data [22]. ERICS [23] shows that the concept drift corresponds to a change in the distribution of optimal model parameters.

In general, performance-based drift detection techniques cannot directly measure the drift severity because they mainly focus on monitoring the performance of the learning system, not the changes in the concept itself. The degree of decrease in learning performance can be used as an indirect measurement to assess the severity of concept drift. Still, the meaning of these decreases are not discussed in prior studies [2]. Distribution-based drift detection methods are able to directly quantify the severity of a concept drift since the measurement used to compare two data samples already reflects the difference.

III. Preliminaries

We consider a data stream classification problem where a sequence of observations (X, y) is received over time, where X is a d-dimensional vector that represents the feature space, and y represents a single discrete class label. A concept drift is a period over which the joint distribution of observations changes, $i.e.$, a concept drift from time t_1 and t_n implies $p_{t_1}(X, y) \neq p_{t_n}(X, y)$ [24]. Since the joint probability $P_t(X, y)$ can be decomposed into $P_t(X) \times P_t(y|X)$, concept drift can be categorized into three types: (1) The drift in marginal distribution while conditional distribution remains unchanged. This drift is called marginal drift or virtual drift; it will not affect the real decision boundary. (2) The drift in conditional distribution while marginal distribution remains unchanged. This drift will change the real decision boundary and decrease learning performance; it is called conditional drift or real drift. (3) The drift in both marginal and conditional distribution. It is a mixture of the previous two types of drifts. Table I shows the formal definition of each type of concept drift.

TABLE I
Different Types of Concept Drift

Drift Type	Marginal Distribution	Conditional Distribution
Marginal Drift	$P_t(X) \neq P_{t+1}(X)$	$P_t(y\|X) = P_{t+1}(y\|X)$
Conditional Drift	$P_t(X) = P_{t+1}(X)$	$P_t(y\|X) \neq P_{t+1}(y\|X)$
Mixed Drift	$P_t(X) \neq P_{t+1}(X)$	$P_t(y\|X) \neq P_{t+1}(y\|X)$

Unlike the drift regions, which focus on the conflict regions between a new concept and the previous concept, $i.e.$, local drift and global drift, drift severity refers to the magnitude of difference between the new concept and the previous concept [2]. Formally, the drift severity can be represented as $\psi = \Psi(P_t(X, y), P_{t+1}(X, y))$, where $\Psi(\cdot, \cdot)$ is a function to measure the difference between two concepts and the range of ψ depends on the measurement functions. Current methods measure the drift severity by monitoring changes in the learner performance or the difference between data distributions. However, it is hard to measure marginal drift using performance-based methods since a marginal drift does not change the actual decision boundary of the data. On the other hand, it is hard to measure conditional drift with distribution-based approaches because most of them are unsupervised. Another limitation is that distribution-based methods need knowledge about the data, increasing storage consumption and the risk of data leaks. To overcome the limitations of the current methods, we propose TDM, a Tree-based Drift Severity Measurement framework.

IV. Tree Based Drift Measurement (TDM)

This section introduces the proposed framework, TDM, which is able to measure drift severity without access to the data. Instead, TDM measures the difference between the tree-based classifiers' structures. TDM is a post-hoc framework based on the DDM framework and is compatible with any tree-based classifier. Our experiments use the Hoeffding tree as a baseline implementation, and it can be extended to other tree-based models such as VFDT. TDM is applied when DDM detects a concept drift. The assumption is that regardless of the type of concept drift, it will cause changes in the tree structure. TDM builds two Hoeffding Trees (HT) for the new concept and the previous concept [5]. Then it transforms the two HTs into two sets of vectors by *Feature Dictionary* (FD) and uses *Operational Taxonomic Units* (OTUs) to measure the difference between the two sets of vectors. This section first briefly explains why we choose the Hoeffding Tree as the base classifier. Then we demonstrate how TDM transforms HT into a set of vectors by FD. After that, we discuss why we use the binary similarity measures instead of tree-edit distance to measure the difference between trees. Lastly, we discuss the time and memory complexity of TDM.

Hoeffding Tree When the generation process of tree classifiers is deterministic, the structural difference between them provides additional information on the severity of changes in the data distribution. We choose Hoeffding Tree as the classifier because it is a deterministic tree-based classifier, which always chooses the feature with the highest information gain and is designed for the online environment.

Feature Dictionary We proposed an approach called the *Feature Dictionary* (FD) to transform each path of a tree into a vector. A tree path consists of a set of internal nodes and a leaf node, where each internal node represents a feature value test for a feature f_i and the leaf node represents a classification result. We one-hot encoded categorical features and binned the numeric features; therefore, we can use binary similarity measures to measure their difference. The feature dictionary maps feature f_i into a two-digit binary vector with three possible values $[0,1], [1,0]$ and $[0,0]$, where $[0,1]$ represents that f_i appears in the path and satisfies the feature value test, $[1,0]$ indicates that f_i appears in the path but does not satisfy the feature value test, and $[0,0]$ represents that f_i is absent in the path. Figure 2 shows how FD transforms each path of a tree into binary vectors. As shown in Figure 2, the vector of a path consists of two parts: a sequence of binary vectors to represent the feature value tests and a pair of values to represent the classification result.

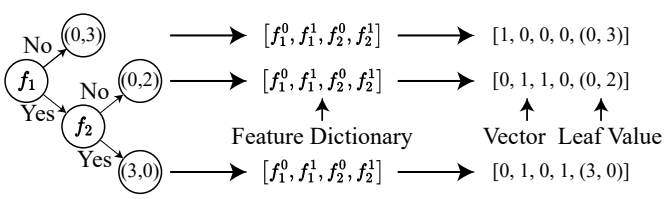

Fig. 2. Example of Feature Dictionary

Binary Similarity and Distance Measures Most approaches measure the difference between the structure of trees by *Tree Edit Distance* (TED). However, the TED problems are generally NP-hard. The high computational complexity makes the tree edit distance infeasible for data stream mining. By transforming trees into a set of vectors and incorporating with *Operational Taxonomic Units* (OTUs) expression, TDM is able to use the binary similarity measures to measure the difference between trees efficiently.

TABLE II
OTUs Expression of Binary Examples v_i and v_j

v_j \ v_i	1 (Presence)	0 (Absence)	Sum
1 (Presence)	$a = v_i \cdot v_j$	$b = \bar{v}_i \cdot v_j$	$a+b$
0 (Absence)	$c = v_i \cdot \bar{v}_j$	$d = \bar{v}_i \cdot \bar{v}_j$	$c+d$
Sum	$a+c$	$b+d$	$v_n = a+b+c+d$

Operational Taxonomic Units Expression Table II shows how two binary vectors v_i and v_j are represented by OTUs expression. In our cases, v_i and v_j are the vectors transformed from paths of trees. By transforming binary vectors into a, b, c, d, v_n, we are able to compute different similarity measures efficiently. As shown in Table II, the "positive matches", a, is the number of features present in both v_i and v_j; the "v_i absence mismatches", b, is the number of features present in v_j but absent from v_i; the "v_j absence mismatches", c is the number of features present in v_i but absent in v_j; and the "negative matches", d, is the number of features absent in both v_i and v_j. The total number of matches between v_i and v_j is the diagonal sum, $a+d$; and the total number of mismatches between v_i and v_j is another diagonal sum $b+c$. The total sum of contingency table, $a+b+c+d$, is always equal to v_n [25]. Table III shows an example of how different similarity and distance measures are computed by the OTUs.

TABLE III
Example of How OTUs Represent Similarity and Distance Functions

Measurement	OTU Expression	Measurement	OTU Expression
Innerproduct	$a+d$	Hamming	$b+c$
Jaccard	$\frac{a}{a+b+c}$	Cosine	$\frac{a}{\sqrt{(a+b)(a+c)}}$

Algorithm 1 shows how TDM measures the difference between two trees. As shown in Lines 4-10, TDM measures the similarity between the vectors that exist in both Tree_A and Tree_B. Then Lines 16-18 and 22-25 show that TDM assigns similarity 0 to the vectors that only exist in one of the two trees. The vector with similarity 0 indicates no vector similar to it. Lastly, TDM normalizes and summarises these similarities to get the similarity between the two trees. The only difference using distance measurement is that TDM assigns distance 1 to the vectors that only exist in one of the two trees, representing no vector similar to these vectors.

Time and Memory Complexity The time complexity for constructing a feature dictionary is $\mathcal{O}(T_n)$, where T_n is the number of nodes of two trees since creating a feature dictionary requires traversing both trees. After constructing the feature dictionary, TDM transforms each path of trees into binary vectors. In the worst case, the time complexity for transforming a tree into a set of vectors is $\mathcal{O}(T_p P_n)$, where T_p is the number of paths and P_n is the number of nodes in the longest path. By using OTUs, the time complexity for calculating the similarity between each pair of paths is $\mathcal{O}(1)$; thus, the time complexity for calculating the similarity between two sets of vectors are $\mathcal{O}(TA_p TB_p)$ in the worst case, where TA_p and TB_p are the number of paths for Tree$_A$ and Tree$_B$, respectively. The overall time complexity of TDM is $\mathcal{O}(T_n + TA_p PA_n + TB_p PB_n + TA_p TB_p)$.

TDM stores a feature dictionary and two sets of vectors. The storage complexity of a feature dictionary is $\mathcal{O}(n_f)$ where n_f is the number of features used to build the Tree$_A$ and Tree$_B$. Each vector requires $\mathcal{O}(n_d)$ memory, where n_d is the length of the vector; thus, in the worst case, we need $\mathcal{O}(n_d(TA_p + TB_p))$ memory to store all vectors. The overall memory complexity of TDM is $\mathcal{O}(2n_f + n_d(TA_p + TB_p)$.

V. Experiments

This section evaluates the performance of TDM in three sets of experiments. The first experiment investigates how TDM performs on different similarity measures by comparing the performance of each similarity measure against the ground truth. The second experiment compares the performance of TDM against three state-of-the-art baselines. The last experiment compares the time and memory usage of TDM against the baselines. We begin by reviewing the benchmark datasets, baselines, evaluation metrics and experiment setting. We then report and analyze the results to validate our approach. Finally, we present a case study to validate whether the online learner performance can be improved by adapting different drift adaptation strategies according to the drift severity.

Datasets To evaluate our approach, we used eight well-known datasets from UCI Machine Learning Repository [26], Massive Online Analysis (MOA) [27], and Kaggle[1]. We inject three types of concept drift: conditional drift, marginal drift for the majority class, and marginal drift for the minority class. We inject conditional drift by reversing the labels of examples and inject marginal drift by swapping the values of important features [19]. We one-hot encoded categorical features and binned the numeric features using equal-quantiles binning.

Experimental Setting For all datasets, we create five variants with drift severity $0\%, 12.5\%, 25\%, 50\%, 100\%$. All experiments are carried out on Intel Xeon E3-12xx vs (Ivy Bridge, IBRS) 2.70 GHz (16 processors), 64 GB RAM, Windows Server 2012 R2 Standard. Due to the page limitation, we only show part of the results here; the complete results and code are available in here. All experiments are averaged over 50 runs using different random seeds.

[1]https://www.kaggle.com/sulianova/cardiovascular-disease-dataset

Algorithm 1: Measure Difference Between Two Trees

Input: TV_i: A set of vectors of Tree$_i$, $i \in \{A, B\}$; G_i: A group of vectors with same key in TV_i, $i \in \{A, B\}$; $M(\cdot, \cdot)$: Difference measurement function, either Similarity or Distance; D_G Difference measurement results of each vector pair in a specific group; D_T Difference measurement results of each group;
Output: ψ: Difference between TV_A and TV_B
foreach G_A *in* TV_A **do**
  if TV_B.hasGroupWithSameKey(G_A) **then**
    $G_B \leftarrow TV_B$.findGroupWithSameKey(G_A);
    foreach *vector* v_A *in* G_A **do**
      foreach *vector* v_B *in* G_B **do**
        $D_v \leftarrow M(v_A, v_B)$;
        D_G.append(D_v);
    if M *is Similarity Measurement Function* **then**
      D_T.append(Max(D_G) / (G_A.Size + G_B.Size));
    else
      D_T.append(Min(D_G) / (G_A.Size + G_B.Size));
  else
    if M *is Similarity Measurement Function* **then**
      D_T.append($0/G_A$.Size);
    else
      D_T.append($1/G_A$.Size);
foreach *Group* G_B *in* TV_B **do**
  if TV_A.NoGroupWithSameKey(G_B) **then**
    if M *is Similarity Measurement Function* **then**
      D_T.append(0 / G_B.Size);
    else
      D_T.append(1 / G_B.Size);
$\psi \leftarrow$ Sum(D_T);

Baselines We compare the designed approach with different state-of-the-art baselines. To investigate whether TDM is able to measure the drift severity of conditional drift, we compare TDM against the **Classification Accuracy**, which is the most straightforward way to monitor the model performance. To investigate whether TDM is able to measure the drift severity of marginal drift, we compare TDM against the **Wasserstein Distance** [19], which is one of the state-of-the-art distribution-based drift detection methods. To investigate how TDM performs against the machine learning interpretation techniques, we compare TDM against **Permutation Importance** [28], which is a popular machine learning interpretation technique that measures the decrease in model performance when a single feature value is randomly shuffled.

Evaluation Metrics We proposed a metric called *quantification score* (QS) to evaluate how TDM and baselines perform against the ground truth. Equations 1 2 present the QS score, where Ψ is the difference measure that returns the difference ψ_i between M_0 and M_{δ_i}, M_0 is the dataset without concept drift and M_{δ_i} is the dataset with concept drift and the drift severity is δ_i. We normalized the results of each difference measure and transformed the results of distance measures ψ_D to the corresponding similarity measures ψ_S by the function $\psi_S = 1 - \psi_D$, which allows us to evaluate the distance measures without changing the function of QS. After calculating the QS score, we use the Friedman-Nemenyi test to compare the performance of TDM and baselines.

$$QS = \sum_{i=1}^{n} \mathbb{I}(\Psi(M_0, M_{\delta_i})) \tag{1}$$

$$\mathbb{I}(\Psi(M_0, M_{\delta_i})) := \begin{cases} 0 & \exists j(\psi_i > \psi_j, \delta_i < \delta_j) \\ 1 & \text{Otherwise} \end{cases} \tag{2}$$

A. Similarity Measures Comparison using QS

This section presents experiments that use quantification scores to evaluate how TDM performs with different similarity measures. We compared 76 similarity measures discussed by Choi et al. [29]. Figure 3 shows the comparison of some popular similarity measures: Cosine Similarity, Fager&McGowan Similarity, Dispersion, Inner Product, Euclidean Distance, Chord Distance, Hamming Distance and Hellinger Distance. The x axis shows the experimented datasets with different drift severities, and the y axis shows the similarity measures. The value in each cell represents the similarity between M_0 and M_{δ_i} by the similarity measure shown in the y axis. Table IV shows the QS score for each similarity measure.

From the results in Figure 3, we observe the following (1) The dataset with 100% conditional drift is hard to measure. (2) The performance of TDM depends on the choice of similarity measures. For example, the Fager&McGowan Similarity performs poorly for most of the datasets. The Fager&McGowan Similarity does not always increase when the drift severity decreases. On the other hand, after transforming into similarity, the Chord Distance always increases as the drift severity decreases. From Table IV, we observe that Chord Distance gets the highest QS score among all similarity measures.

For Observation 1, we reason that reversing all dataset labels will not change the inner nodes of tree paths but only the classification result of leaf nodes. As the size of the dataset increases, it deteriorates the performance of TDM because the increase in the variety of classification results leads to the increase in the overlap between two trees, which confuses TDM. For Observation 2, we reason that different similarity measures focus on different points. For example, some similarity measures may focus on the features present in both v_i and v_j, but others may focus on the features present in v_i but absent in v_j. Thus, the choice of similarity measures should be considered task by task.

B. Quantification Score Comparison against Baselines

This section presents experiments to compare the quantification score of TDM against the baselines. The similarity measure used here is Chord Distance since the results of Section V-A show it has the highest quantification score among the 76 similarity measures. From the results in Figure 4, we have the following observations: (1) Overall, TDM performs slightly better than Classification Accuracy and Wasserstein Distance but not significant enough. (2) TDM outperforms Permutation Importance.

For Observation 1, we reason that regardless of the concept drift type, it will change the structure of the tree, and the magnitude of change can be reflected by measuring the difference between trees. For Observation 2, we reason that the Permutation Importance is designed to interpret the feature importance changes during the concept drift. This information may not be able to reveal the magnitude of drift severity. Overall, we conclude that TDM is able to measure the drift severity by measuring the difference between tree classifiers and outperforms the baselines.

C. Time and Memory Usage Comparison

Lastly, we present experiments to compare the runtime and memory usage against baselines. Table V shows the runtime and memory usage for each algorithm, where NaN represents that the Wasserstein Distance cannot measure the conditional drift. Table V shows the following observations: (1) TDM takes up the second-least storage space and runs in an acceptable time. (2) The runtime of Permutation Importance is much higher than the others, which makes Permutation Importance infeasible for the online environment.

Overall, by evaluating TDM on eight datasets and comparing it to the state-of-the-art methods, we have shown that TDM is able to measure the drift severity of different types of concept drifts and provides the best trade-off in runtime and memory usage.

VI. Case Study

This section demonstrates a case study to show how the online leaner performance improves when adapting different drift adaptation strategies according to the drift severities.

Datasets We used three famous synthetic concept drift data generator: AGRAWAL [30], RandomRBF, and SEA [31]. AGRAWAL generator produces a stream containing nine features, six numeric and three categorical. AGRAWAL defines ten functions to generate binary class labels from the features. The features and functions are provided in the original paper [30]. RandomRBF generator produces a radial basis function stream. It generates several centroids with a random central position, a standard deviation, a class label and weight. RandomRBF generator creates a new sample by choosing one of the centroids at random, taking into account their weights, and offsetting the features at a random direction from the centroid's centre. The SEA generator generates three numerical features that vary from 0 to 10, where only two of them are relevant to the classification task. SEA provides four

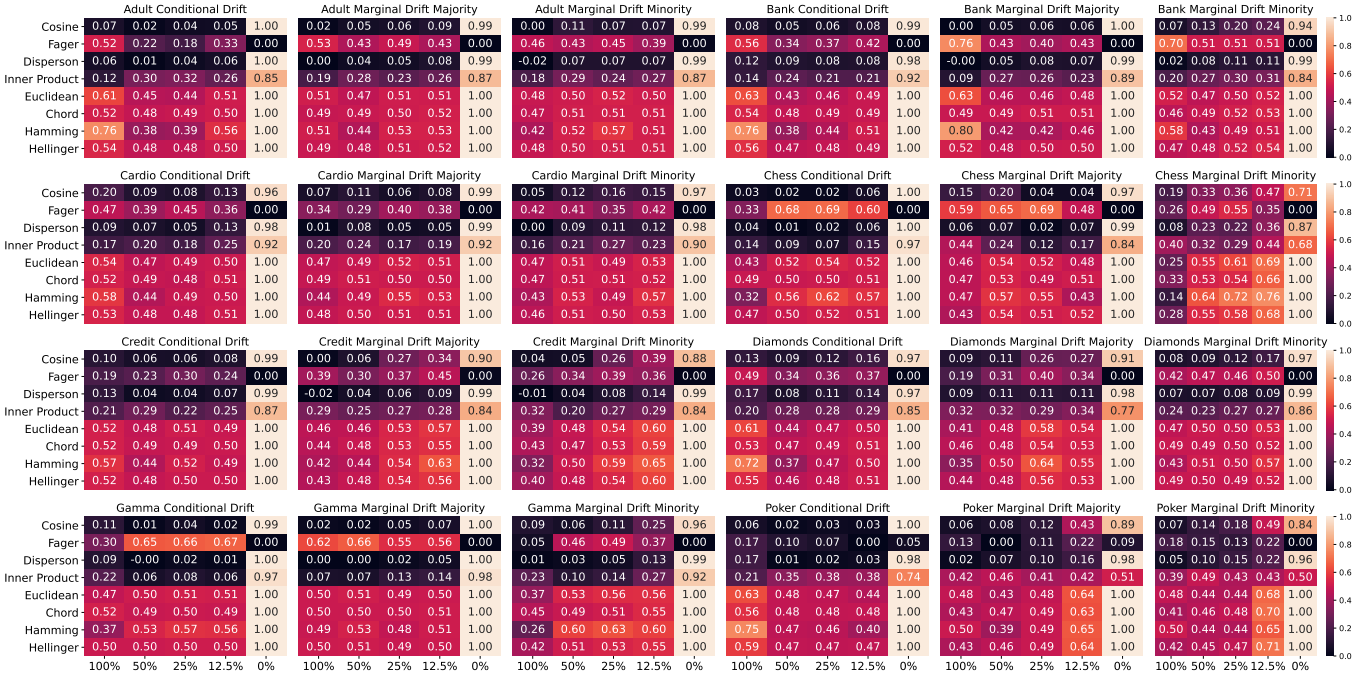

Fig. 3. Similarity Measures Comparison on Eight Datasets

TABLE IV
EVALUATION OF 59 SIMILARITY AND 17 DISTANCE MEASUREMENTS

Similarity	Score	Similarity	Score	Similarity	Score	Distance	Score
Jaccard	0.825	Mounford	0.883	Sokal&Sneath-V	0.817	Hamming	0.775
Dice	0.817	Otsuka	0.817	Cole	0.867	Euclid	0.767
Czekanowski	0.817	Mc Connaughey	0.883	Stlies	0.775	Squared-Euclid	0.775
3w-Jaccard	0.817	Tarwid	0.833	Ochiai-Ii	0.817	Canberra	0.775
Nei&Li	0.817	Kulczynski-Ii	0.817	Yuleq	0.825	Manhattan	0.775
Sokal&Sneath-I	0.825	Driver&Kroeber	0.817	Yulew	0.842	Mean-Manhattan	0.758
Sokal&Michener	0.758	Johson	0.817	Kulczynski-I	0.858	Cityblock	0.775
Sokal&Sneath-Ii	0.758	Dennis	0.850	Tanimoto	0.825	Minkowski	0.775
Roger&Tanmoto	0.767	Simpson	0.817	Disperson	0.842	Vari	0.767
Faith	0.767	Braun&Banquet	0.817	Hamann	0.800	Sizedifference	0.767
Gower&Legendre	0.758	Fager&Mcgowan	0.517	Michael	0.825	Shape difference	0.758
Intersection	0.792	Forbes-Ii	0.858	Goodman&Kruskal	0.858	Patterndifference	0.758
Innerproduct	0.742	Sokal&Sneath-Iv	0.792	Anderberg	0.858	Lance&Williams	0.817
Russell&Rao	0.792	Gower	0.758	Baroni-Urbani&Buser-I	0.817	Bray&Curtis	0.817
Cosine	0.825	Pearson-I	0.850	Baroni-Urbani&Buser-Ii	0.858	Hellinger	0.817
Gilbert&Wells	0.792	Pearson-II	0.817	Peirce	0.708	**Chord**	**0.892**
Ochai-I	0.817	Pearson-III	0.808	Eyraud	0.850	Yule Q	0.825
Forbesi	0.817	Pearson&Heron-I	0.842	Tarantule	0.842		
Fossum	0.842	Pearson&Heron	0.833	Ample	0.842		
Sorgenfrei	0.850	Sokal&Sneath-III	0.800				

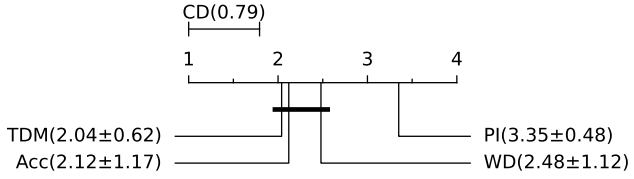

Fig. 4. Evaluation of All Datasets

Note: Acc for Classification Accuracy, WD for Wasserstein Distance, PI for Permutation Importance.

classification functions to generate the data. These functions compare the sum of the two relevant features with a threshold value, unique for each classification function. The classification functions are provided in the original paper [31].

Experimental Setting We generate a large data stream with 5000 examples and a small data stream with 1000 examples in each concept for each generator. We generate two types of concept drift for each data stream, abrupt and gradual drift. The drift width of abrupt and gradual drift is set to 1 and 10% of the examples in the single concept, respectively. Each

TABLE V
RUNTIME AND MEMORY USAGE ANALYSIS (ACCURACY FOR CLASSIFICATION ACCURACY, WD FOR WASSERSTEIN DISTANCE, PI FOR PERMUTATION IMPORTANCE, NAN REPRESENTS THAT THE WASSERSTEIN DISTANCE CANNOT MEASURE THE CONDITIONAL DRIFT)

	Dataset	Drift Type	TDM	Accuracy	WD	PI
Runtime (s)	Adult	Conditional	286.69 ± 0.31	84.01 ± 0.20	NaN	≥ 8 hours
	Bank	Conditional	193.26 ± 0.88	23.78 ± 0.08	NaN	≥ 8 hours
	Chess	Conditional	116.44 ± 0.43	28.26 ± 0.12	NaN	≥ 8 hours
	Diamonds	Conditional	116.20 ± 0.42	37.56 ± 0.19	NaN	≥ 8 hours
	Adult	Majority	266.40 ± 0.49	86.50 ± 0.21	1.22 ± 0.02	≥ 8 hours
	Bank	Majority	169.30 ± 0.84	30.28 ± 0.30	0.79 ± 0.01	≥ 8 hours
	Chess	Majority	111.98 ± 0.06	23.89 ± 0.23	0.45 ± 0.01	≥ 8 hours
	Diamonds	Majority	114.34 ± 1.07	36.27 ± 0.26	0.45 ± 0.02	≥ 8 hours
	Adult	Minority	262.40 ± 0.64	77.78 ± 0.35	1.12 ± 0.04	≥ 8 hours
	Bank	Minority	173.23 ± 1.38	23.93 ± 0.03	0.75 ± 0.02	≥ 8 hours
	Chess	Minority	108.50 ± 0.45	28.66 ± 0.07	0.43 ± 0.01	≥ 8 hours
	Diamonds	Minority	114.95 ± 0.56	36.34 ± 0.15	0.45 ± 0.01	≥ 8 hours
Memory Usage (MB)	Adult	Conditional	4.10 ± 0.00	2.04 ± 0.00	NaN	93.89 ± 0.00
	Bank	Conditional	2.54 ± 0.00	1.23 ± 0.00	NaN	60.71 ± 0.00
	Chess	Conditional	1.61 ± 0.00	0.80 ± 0.00	NaN	36.91 ± 0.00
	Diamonds	Conditional	1.12 ± 0.00	0.55 ± 0.00	NaN	35.79 ± 0.00
	Adult	Majority	3.23 ± 0.00	2.04 ± 0.00	134.34 ± 0.00	93.92 ± 0.00
	Bank	Majority	1.93 ± 0.00	1.23 ± 0.00	87.05 ± 0.00	60.07 ± 0.00
	Chess	Majority	1.54 ± 0.00	0.80 ± 0.00	52.79 ± 0.00	36.82 ± 0.00
	Diamonds	Majority	1.36 ± 0.00	0.55 ± 0.00	51.92 ± 0.00	36.04 ± 0.00
	Adult	Minority	3.29 ± 0.00	2.04 ± 0.00	134.34 ± 0.00	93.03 ± 0.00
	Bank	Minority	1.99 ± 0.00	1.23 ± 0.00	87.05 ± 0.00	60.14 ± 0.00
	Chess	Minority	1.57 ± 0.00	0.80 ± 0.00	52.79 ± 0.00	36.85 ± 0.00
	Diamonds	Minority	1.32 ± 0.00	0.55 ± 0.00	51.92 ± 0.00	36.00 ± 0.00

data stream consists of three concepts where the drift severity between concept$_A$ and concept$_B$ is less than the drift severity between concept$_B$ and concept$_C$. Table VI shows the functions used to generate the case study data. The labels of datasets are balanced.

TABLE VI
CASE STUDY DATA GENERATION FUNCTIONS

	Concept$_A$	Concept$_B$	Concept$_C$
AGRAWAL	Function 1	Function 3	Function 4
RandomRBF	4 Drift Centroids	5 Drift Centroids	10 Drift Centroids
SEA	Function 1	Function 3	Function 2

Baselines and Evaluation Metrics We choose the Hoeffding Tree as our base classifier and the remove-and-retrain framework as the baseline. Each time drift is detected, the baseline retrains a Hoeffding Tree. We use Classification Accuracy as the evaluation metric since the labels of datasets are balanced.

Evaluation From the results in Figure 5, we make two observations: (1) In concept$_B$, TDM gets higher accuracy than the baseline because it does not retrain the model when the drift occurs between concept$_A$ and concept$_B$. (2) Both TDM and baseline improve the online learner performance by retraining the model when the drift occurs between concept$_B$ and concept$_C$.

For Observation 1, we reason that a small drift severity indicates that the previously learned knowledge is helpful in the new concept. Thus, transferring the previous model to the current concept may increase performance instead of retraining a new model. For Observation 2, we reason that a large drift severity indicates that previously learned knowledge cannot benefit the new concept. Thus, we need to retrain a new model for the new concept.

Overall, we conclude that the online learner performance can be improved by adapting different drift adaptation strategies based on the drift severity.

VII. CONCLUSION

This paper presents a post-hoc drift severity measurement framework called *Tree-based Drift Measurement*. The highlight of TDM is that it measures the drift severity by measuring the difference between tree classifiers, which reduces the time and memory complexity of measuring drift severity and preserves data privacy. The evaluations show that TDM is time and memory efficient and outperforms the baselines. The case study shows that the online learner performance can be improved by adapting different drift adaptation strategies based on the drift severity. One future work for TDM is finding a better way to deal with the numeric features since the splitting strategy of numeric features may violate the robustness of TDM.

REFERENCES

[1] Ł. Korycki and B. Krawczyk, "Concept drift detection from multi-class imbalanced data streams," in *International Conference on Data Engineering*. IEEE, 2021.

[2] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[3] V. Cerqueira, H. M. Gomes, and A. Bifet, "Unsupervised concept drift detection using a student–teacher approach," in *International Conference on Discovery Science*. Springer, 2020.

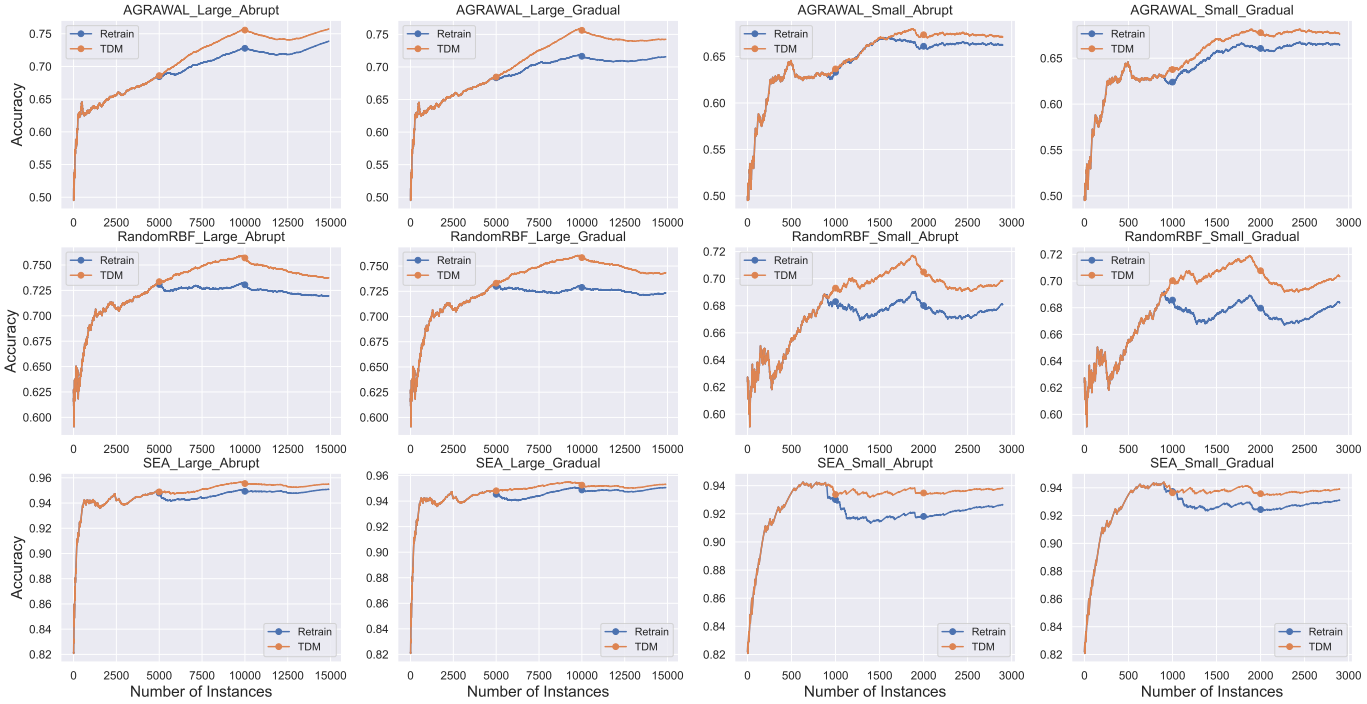

Fig. 5. Case Study for the Performance of TDM against the Baseline Framework

[4] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence - SBIA*. Springer, 2004.

[5] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *International Conference on Knowledge Discovery and Data Mining*. ACM, 2001.

[6] E. S. Page, "Continuous inspection schemes," *Biometrika*, 1954.

[7] A. Bifet, J. Read, B. Pfahringer, G. Holmes, and I. Žliobaitė, "Cd-moa: Change detection framework for massive online analysis," in *Internatonal Symposium on Intelligent Data Analysis*. Springer, 2013.

[8] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine learning*, 1996.

[9] G. Widmer, "Combining robustness and flexibility in learning drifting concepts," in *European Conference on Artificial Intelligence*. Citeseer, 1994.

[10] J. Gama and G. Castillo, "Learning with local drift detection," in *International Conference on Advanced Data Mining and Applications*. Springer, 2006.

[11] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, "Early drift detection method," in *Workshop on Knowledge Discovery from Data Streams*, 2006.

[12] I. Frias-Blanco, J. del Campo-Ávila, G. Ramos-Jimenez, R. Morales-Bueno, A. Ortiz-Diaz, and Y. Caballero-Mota, "Online and non-parametric drift detection methods based on hoeffding's bounds," *IEEE Transactions on Knowledge and Data Engineering*, 2014.

[13] A. Liu, G. Zhang, and J. Lu, "Fuzzy time windowing for gradual concept drift adaptation," in *International Conference on Fuzzy Systems*. IEEE, 2017.

[14] S. Xu and J. Wang, "Dynamic extreme learning machine for data stream classification," *Neurocomputing*, 2017.

[15] D. Kifer, S. Ben-David, and J. Gehrke, "Detecting change in data streams," in *International Conference on Very Large Data Bases*. Toronto, Canada, 2004.

[16] X. Song, M. Wu, C. Jermaine, and S. Ranka, "Statistical change detection for multi-dimensional data," in *International Conference on Knowledge Discovery and Data Mining*. ACM, 2007.

[17] A. A. Qahtan, B. Alharbi, S. Wang, and X. Zhang, "A pca-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams," in *International Conference on Knowledge Discovery and Data Mining*. ACM, 2015.

[18] L. Bu, C. Alippi, and D. Zhao, "A pdf-free change detection test based on density difference estimation," *IEEE Transactions on Neural Networks and Learning Systems*, 2016.

[19] D. Zhao and Y. S. Koh, "Feature drift detection in evolving data streams," in *International Conference on Database and Expert Systems Applications*. Springer, 2020.

[20] I. Baidari and N. Honnikoll, "Bhattacharyya distance based concept drift detection method for evolving data stream," *Expert Systems with Applications*, 2021.

[21] J. L. Lobo, J. Del Ser, E. Osaba, A. Bifet, and F. Herrera, "Curie: a cellular automaton for concept drift detection," *Data Mining and Knowledge Discovery*, 2021.

[22] A. Impedovo, C. Loglisci, M. Ceci, and D. Malerba, "jkarma: A highly-modular framework for pattern-based change detection on evolving data," *Knowledge-Based Systems*, 2020.

[23] J. Haug and G. Kasneci, "Learning parameter distributions to detect concept drift in data streams," in *International Conference on Pattern Recognition*. IEEE, 2021.

[24] B. Halstead, Y. S. Koh, P. Riddle, M. Pechenizkiy, A. Bifet, and R. Pears, "Fingerprinting concepts in data streams with supervised and unsupervised meta-information," in *International Conference on Data Engineering*. IEEE, 2021.

[25] P. Bille, "A survey on tree edit distance and related problems," *Theoretical Computer Science*, 2005.

[26] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[27] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl, "Moa: Massive online analysis, a framework for stream classification and clustering," in *Workshop on Applications of Pattern Analysis*. PMLR, 2010.

[28] L. Breiman, "Random forests," *Machine Learning*, 2001.

[29] S.-S. Choi, S.-H. Cha, and C. C. Tappert, "A survey of binary similarity and distance measures," *Journal of Systemics, Cybernetics and Informatics*, 2010.

[30] R. Agrawal, T. Imielinski, and A. Swami, "Database mining: A performance perspective," *IEEE Transactions on Knowledge and Data Engineering*, 1993.

[31] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," in *International Conference on Knowledge Discovery and Data Mining*. ACM, 2001.