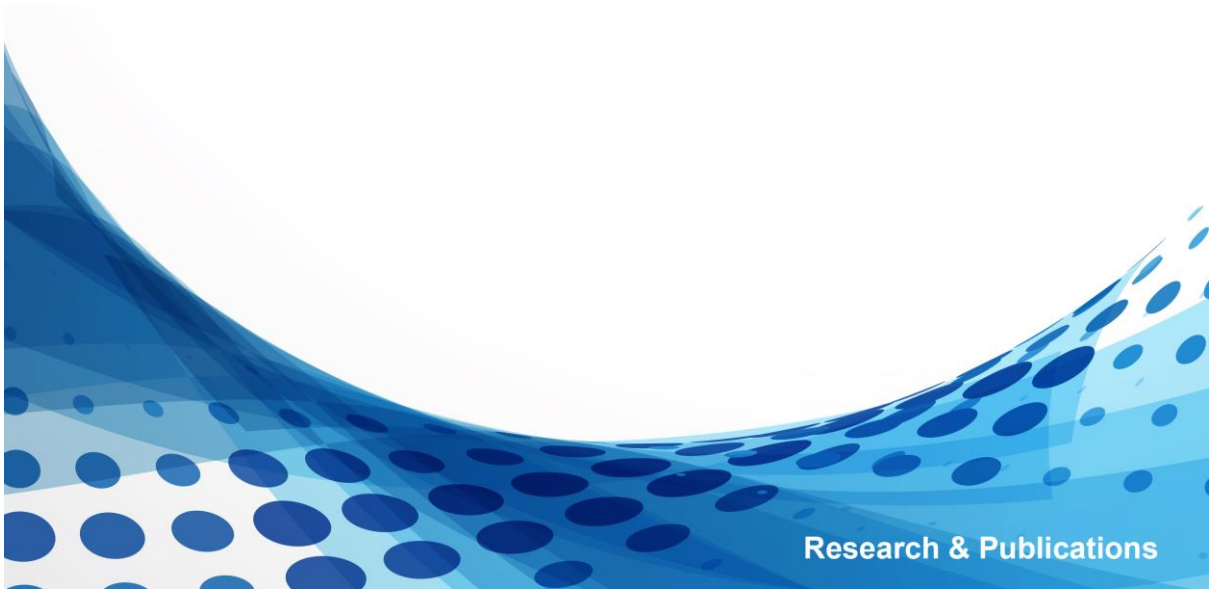
INDIAN INSTITUTE OF MANAGEMENT AHMEDABAD

IIMA
Working Paper

Optimal Transport based Drift Detection for Sensor Streams: Method and Applications in Transportation

Arnab Kumar Laha
Shikha Verma



Research &
Publications
W. P. No. 2021-09-01

Optimal Transport based Drift Detection for Sensor Streams: Method and Applications in Transportation

Arnab Kumar Laha
Shikha Verma

September 2021

INDIAN INSTITUTE OF MANAGEMENT AHMEDABAD

Optimal Transport based Drift Detection for Sensor Streams: Method and Applications in Transportation

Arnab Kumar Laha[a] (arnab@iima.ac.in), Shikha Verma[b] (phd16shikhav@iima.ac.in),

[a] Faculty Wing 15, Indian Institute of Management, Ahmedabad,Heritage Campus, Ahmedabad,India-380015

[b] D29R36, Indian Institute of Management, Ahmedabad,New Campus, Ahmedabad,India-380015

Corresponding Author:

Arnab Kumar Laha

Room 37, KLMDC, Indian Institute of Management, Ahmedabad,Heritage Campus, Ahmedabad,India-380015

Tel: +91-90990-25301

Email: arnab@iima.ac.in

Optimal Transport based Drift Detection for Sensor Streams: Method and Applications in Transportation

Arnab Kumar Laha[a], Shikha Verma[a]

[a]*Indian Institute of Management, Ahmedabad*

Abstract

With increasing adoption of Internet of Things (IoT) across the transportation sector, there is a growing need for developing algorithms for analyzing data streams. Due to dynamic operating environment conditions in the transportation domain, the nature of the data streams frequently change and static predictive models are often not successful when dealing with, non-stationary data streams. Further, labelled data is often unavailable or is costly to acquire in real time. Thus, effective algorithms for such problems would aim to maximize accuracy while minimizing the labelled data requirements. In this paper, we propose a new algorithm namely, the Optimal Transport based Drift Detection (OTDD) algorithm, that aims to address the accuracy-labeling requirement trade-off. Experiments on artificial and real-life data sets from the transportation domain demonstrate that the OTDD algorithm performs better than some of the widely used competing algorithms in addressing the accuracy-labeling requirement trade-off.

Keywords: Concept drift, Data streams, Intelligent transportation systems, Kullback-Leibler Divergence, Wasserstein barycentre

1. Introduction

We live in the age of big data where vast troves of data from social media, sensing devices, mobility traces, internet traffic, healthcare systems are being generated every second. It is common to describe 'Big Data' by a number of V's such as Volume, Velocity, Variety, Veracity, Value etc. The reader may refer to Marz & Warren (2013) for details on Big Data terminology. In this paper, we are primarily concerned with Velocity which refers to continuous generation of data from sources such as sensors, video and audio streams, search queries, social media posts etc. The high frequency of data generation create a need for processing and analysing at much higher frequencies than traditional warehoused data (O'Leary (2013)). In many cases, the speed at which the data is generated is very fast and as a result of which the total volume of the generated data is very huge. This makes iterative computation with data streams often infeasible and special 'single-pass' algorithms need to be developed to analyse these data streams efficiently (Marz & Warren (2013)).

*Corresponding author.
Email addresses: arnab@iima.ac.in (Arnab Kumar Laha), phd16shikhav@iima.ac.in (Shikha Verma)

Availability of Big Data has accelerated the pace of machine learning research where the dominant stream of literature has been to develop predictive models that leverage big data to aid data-driven decision making. This is facilitated by a rapid decrease in cost of storing and analyzing data that has enabled practitioners and researchers to mine the patterns to improve decision making (Ghofrani et al. (2018); Wang et al. (2019)).

However, the new forms of data are posing a challenge to the assumptions of model building paradigms for predictive tasks. Typically, models are built in batch mode i.e historical instances are used to train models which are later used to predict outputs on unseen data. Model performance on unseen data, defined by metrics such as accuracy, is key to building effective models as it denotes how well the model has learned the patterns from the data. However, in data streams the focus is on rapid adaptation of the predictive models to the changing behaviour of the data streams. The predictive models built using the batch paradigm are not well suited for use with data streams. Thus, new predictive methods that adapt to the streaming paradigm are required (see Section 3.2).

Since data streams are continuous and potentially infinite in nature it is often difficult and costly to store the entire data for a long time. Thus, there is a need to form compact representations of the data so that these summarised information can be efficiently stored, while the rest of the data can be discarded to reduce storage and computation costs. Building good summarization techniques for data streams is a challenging task as it involves a trade-off of two conflicting objectives: (a) maximize capture/coverage of information present in the original data stream and (b) minimize memory/disk space requirements and storage costs. Summarising data streams not only minimizes storage requirement and cost but also ensures fast retrieval and lookup of values for concept drift detection algorithms as compared to using the data in its expanded original form (Aggarwal & Philip (2007)).

Some of the challenges faced while analyzing data streams emanates from the constraint of learning from a 'single-pass' i.e. each observation can be analyzed only once. This creates a need for a forgetting/discarding mechanism for older observations of the data stream. The widely used sliding window model is a simple example of a forgetting mechanism.

The present paper uses classifier outputs from successive windows, summarised using optimal transport theory to monitor drift over the progression of data stream. The main contribution of this paper is a new predictive method for class labels that demands much less label information than the alternative algorithms that have been proposed in the literature. It uses a novel idea of detecting drift using the Optimal Transport distance between Wasserstein barycenters of collections of histograms to rapidly adapt the predictive model in the presence of concept drift (see Section 4.1).

The encouraging performance of the Optimal Transport based Drift Detection Algorithm (OTDD) vis-a-vis three alternative algorithms is documented by its application on two artificial data-sets and two real-life datasets from transportation domain. Through this, the paper also significantly contributes to the need of developing accurate, context-specific, drift aware algorithms that have limited label requirements (Žliobaitė et al. (2016)). The rest of the paper is organized as follows. Section 2 gives an overview of challenges, application areas, and previous research in drift detection and stream mining in transportation. Section 3 gives a detailed explanation of our proposed data summa-

rization technique using optimal transport theory and concept drift detection algorithm using classifier information and limited data labels. Section 4 contains the pseudocode and details of the OTDD and the Data-driven Threshold Estimation(DDTE) Algorithm. Section 5 introduces the datasets used for evaluation of proposed algorithms and the evaluation metric used for performance comparison.Section 6 contains results of the experimental evaluation of our proposed method on artificial and real-life datasets. Section 7 contains implications of our research for practitioners and researchers and highlights directions for future research.

2. Literature Review

Data stream is defined as a temporally ordered sequence of observations pertaining to a given phenomenon. The velocity of the data stream is the number of observations generated by the data source within a period of time (Aggarwal (2007)). In many applications, such as data coming from sensors, the velocity of the data stream is very high. The velocity and volume of data stream pose several challenges that stream mining algorithms need to adhere to: single possible pass at data, disk storage of a long (potentially infinite) stream in its original form, and handling time evolving characteristics of data, a phenomenon known as concept drift.

Drift detection methods help in identifying the presence of concept drift in a data stream. Methods for adapting to the concept drift can be classified into two major categories: (a) Passive and (b) Active. Passive drift adaptation involves forgetting previous data and using only the most recent data irrespective of whether a drift has occurred or not. This can be achieved by creating forgetting mechanisms that discards the old data partly or wholly, as soon as the new observations arrive. The sliding window technique is a widely used Passive drift adaptation method. Active drift adaptation methods involve an explicit drift detection mechanism that involves monitoring data characteristics over time and raising an alarm when a concept drift is detected. It is followed by executing an associated action like updating the model parameters (Kuncheva (2009)).

In a supervised learning setting, one of the simplest approaches to concept drift detection is tracking of the accuracy of the predictive model (a.k.a. learner) for incoming data instances. An alarm is raised if the performance of the model deteriorates and drops below a specified threshold. It is important to note that these methods rely on the complete availability of labels which inform the classifier whether its prediction was correct.

However, the availability of labels in many real-life situations is not immediate and may involve substantial cost (Bahri et al. (2021), Khamassi et al. (2018)). For example-in digital advertising, ad-click spam detection models are used to classify whether a click originated from a human user or not. Fraud patterns of click bots keep evolving, hence creating a need for drift detection algorithms. However, the availability of 'ground truth' of knowing whether it was a human click or not poses a challenge to streaming data algorithms that require constant access to new labels (Oentaryo et al. (2014)).

Similarly in the transportation domain, GPS and sensor data are used to detect the mode of travel being used (car/bus/train/walking). Machine learning models trained to discriminate among modes can classify a particular sensor

reading as ‘car’ but the classifier is not ‘informed’ whether the prediction was correct or not, unless supplemented by self-report of the user about mode being used. Hence, the task of obtaining labels is often costly and not instantaneous (Krempl et al. (2014), Žliobaite (2010)).

Clearly, passive approaches that update blindly are unsuitable for environments where access to labels is restricted or drift signaling has to be followed by an associated action. Therefore, we need active approaches that constantly monitor data in the stream to check for concept drift and only raise a need for label acquisition and retraining, when a drift is detected based on available data.

One of the early approaches in change detection was the Page-Hinkley test (Sebastiao et al. (2011),Page (1954),Hinkley (1971)). This test is based on CUSUM, which is the cumulative sum of the difference between the current observed value and mean of observations till the current moment. To ascertain drift, the test monitors the cumulative sum of the differences and if it exceeds a certain threshold, a drift is signalled.

Drift Detection Method (DDM) proposed by Gama et al. (2004) exploits the PAC learning model (Mitchell (1997)) which states that the error rate of an algorithm will decrease with increase in number of training instances provided the data remains stationary. An increase in error rate of learner is suggestive of concept drift which renders the current model obsolete for predictions on newly arriving data.

Baena-Garcıa et al. (2006) propose the Early Drift Detection Method (EDDM) that uses a distance measure which is defined as the number of observations between two successive classification errors. If the data is stationary, the distance between successive errors is likely to be large. If the distance between two consecutive errors becomes smaller, it is likely that a drift has occurred and the data used to train the predictive model no longer represents the current reality. The EDDM defines two thresholds- (i) warning level and (ii) drift level. If the incoming examples cross the warning threshold, they will be stored in memory in anticipation of drift but the same model continues to be used for prediction. If the drift threshold is crossed, the model is reset on most recent data and threshold values are updated. It demonstrates a better performance than the DDM in detecting gradual drifts.

The Hellinger Distance based Drift Detection Method (HDDDM) proposed by Ditzler & Polikar (2011) divides the stream of predictor variables into windows of fixed size and summarizes each predictor variable using a histogram. The initial data window acts as the ’reference’ against which all subsequent windows would be compared. For each predictor variable, Hellinger distance is used to measure how ‘different’ the reference window and the current window are. A drift tracking metric is defined as the difference between the average Hellinger distance at the current and the previous window. An adaptive threshold is computed at every iteration and an alert for drift is raised if the observed drift tracking metric exceeds the threshold. If no drift is detected, the observations of the current window (predictors and labels) are added to the training set for the classifier, which is then retrained. If a drift is detected, all old observations are discarded from the training set and the classifier is only trained on the current window. In essence, HDDDM is an incremental learning algorithm where the rules for updating the training set for the learner are dependent on presence/ absence of drift.

All the above mentioned algorithms rely on the (near) instantaneous availability of labels which is not only a restrictive assumption while building models for data streams, but it also influences the choice of evaluation metric while choosing among various drift detection algorithms (Gama et al. (2004),Baena-Garcıa et al. (2006)). ,In presence of full access to ground truth labels, measures of algorithm performance such as accuracy, precision, recall etc. are used as metrics to choose among different classification algorithms. It is expected that algorithms that use labels from the entire data stream to be more accurate than algorithms that have partial access to labels. This is so because, with full information on the labels the relationship between the predictors and the labels can be better estimated by the classifier algorithm. When limited/ no labels are available classifiers are often not up-to date with the current concept in a streaming setup, leading to them being less accurate. However, while deploying concept drift detection algorithms in most real-life situations, we encounter the constrained availability of labels.

An Intelligent Transport System (ITS) typically combines advanced technology that includes sensors, high speed computing and advanced data modeling techniques to address operational and strategic challenges in transportation. The use of sensors, smart cards, and GPS traces for transport data collection across different modes of transportation coupled with rapid decrease in data storage cost and pervasive availability of cloud based computing technology has provided unprecedented opportunities for transport authorities and researchers to leverage "big data" for monitoring, planning and operational purposes (Sussman (2008), Ghofrani et al. (2018); Wang et al. (2019)).

Across the last two decades, transportation research has seen several attempts to exploit sensor data for modelling transportation activity. In railways, Nair et al. (2019); Nabian et al. (2019) use historical running times, schedule and rolling stock data to predict train delays. For surface modes, spatio-temporal traffic patterns, road topology and internet query data has been used to predict traffic volumes and demand in sharing systems (Markou et al. (2019); Liao et al. (2018)). Hou et al. (2015) use ensemble machine learning models like AdaBoost and random forest on vehicle trajectory data to develop a lane change assistive sytem to minimise unsafe manoeuvres and increase road safety. Chen et al. (2009) use data collected using loop detectors along freeways build support vector classifiers and their ensembles to predict traffic incidents.

In real-life transportation problems the mobility patterns and mode choices are dynamic and affected by multiple factors such as availability, service quality and pricing. This contributes to evolving nature of data ('concept drift'), especially when mobility data is analysed as a data stream. In the last decade, there has been growing interest in devising new methods for analysing data streams. These have exhibited better performance than methods that fail to account for concept drift for transportation problems. Moreira-Matias et al. (2016) use online learning methods to prevent bus bunching and consequently, reduce passenger waiting time. Laha & Putatunda (2018) use incremental learning algorithms and their ensembles to predict destination location in GPS taxi sharing systems. Moreira-Matias et al. (2013) use sliding window time series models to predict spatio-temporal demand for taxi services.

An important example of concept drift in the transportation context is the introduction of a new transport mode in a city. A classifier trained on sensor data that was collected prior to the introduction of the new transport mode would

be unable to detect it since the classifier would not be 'aware' of this 'change in data', and as a consequence would incorrectly classify the sensor readings of this new mode into one of the existing modes.

Concept drift can be classified in two ways: (a) Virtual and (b) Real (Khamassi et al. (2018)). Virtual drift occurs when the distribution of the predictor(X), denoted as p(x), changes but the distribution of the response(Y) given the predictor, denoted as p(y|x), remains unchanged. On the other hand, Real concept drift occurs when the distribution p(y|x) changes. Real concept drift is responsible for the degradation of the predictive performance of models built in batch mode as they have learnt from a different joint distribution of Y and X. In this paper, we will focus on detecting Real concept drift.

3. Background

This section gives a brief overview of concepts used in later sections of the paper. Section 3.1 gives an overview of the classification task, logistic regression and decision tree classifiers used in the OTDD Algorithm. Section 3.2 gives a brief introduction to model building paradigms for data streams. Section 3.3 presents the need for summarization schemes for data streams. Section 3.4 and 3.5 develops the foundation in optimal transport and Kullback-Leibler Divergence for use in the OTDD algorithm.

3.1. Classification

Classification is one of the prediction tasks prevalent in supervised learning literature where the learner's output is a categorical variable. Hastie et al. (2009) define classification as a function approximation task. The goal of classification is to make a good prediction $\hat{Y}$ for the dependent categorical variable Y (which takes only finitely many values), given the value of the predictors X.

The logistic regression model is widely used for classification as it gives the probability of an observation belonging to each of the possible classes based on the observed values of the predictors. Based on this, a rule can be devised which can predict the class of this new observation. An often used rule is to predict the class for the which the estimated probability is maximum. For a binary classification problem, the logistic regression model is given by equation 1:

$$\begin{aligned} P(Y=1) &= \frac{e^Z}{1+e^Z} \\ \text{where } Z &= \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_m X_m \end{aligned} \tag{1}$$

For a multi-class classification problem for K classes, the logistic regression model is given by equation 2:

$$\begin{aligned} \log \frac{Pr(G=1|\boldsymbol{X}=\boldsymbol{x})}{Pr(G=K|\boldsymbol{X}=\boldsymbol{x})} &= \beta_{10} + \boldsymbol{\beta_1}^T \boldsymbol{x} \\ \log \frac{Pr(G=2|\boldsymbol{X}=\boldsymbol{x})}{Pr(G=K|\boldsymbol{X}=\boldsymbol{x})} &= \beta_{20} + \boldsymbol{\beta_2}^T \boldsymbol{x} \\ &\vdots \\ \log \frac{Pr(G=K-1|\boldsymbol{X}=\boldsymbol{x})}{Pr(G=K|\boldsymbol{X}=\boldsymbol{x})} &= \beta_{(K-1)0} + \boldsymbol{\beta_{K-1}}^T \boldsymbol{x} \end{aligned} \tag{2}$$

where β_{i0} are constants, $\boldsymbol{\beta_i}^T = (\beta_{1i}, \ldots, \beta_{mi})$ for each i and $\boldsymbol{x} = (x_1, \ldots, x_m)$. For more details about the logistic regression model, see Hastie et al. (2009).

Another commonly used model for classification is the decision tree. It uses the 'divide and conquer' strategy to divide the dataset into groups such that observations within a group are homogeneous (Larose (2015)). The tree building process starts with the root node where all the observations are present. Then the root node is split into two/more internal nodes based on a certain predictor from the group of predictors based on a splitting criteria. There are several criteria used for splitting. One of the criteria used for selection of variable for splitting is Gini impurity (a.k.a. Gini index). The formula for Gini impurity at a given node for a variable with J classes is given by equation 3:

$$I_G(\boldsymbol{p}) = \sum_{n=1}^{J} p_i(1 - p_i) \tag{3}$$

where p_i is the proportion of observations with class label i, where $i \in \{1, 2, \ldots, J\}$ (James et al. (2013))

Intuitively, the Gini impurity measures the total variance across the J classes. Gini impurity is lowest, equals 0, when all observations in a given node are from the same class. Thus, lower values of Gini impurity measure are desirable . Hence, the variable that yields the minimum value of Gini impurity is selected for each split. The splitting process continues till the termination criteria is met and the nodes with vertex degree 1 are termed as leaf nodes. For each leaf node, the dominant class of the observations present in the leaf node (i.e. the class that has the maximum number of observations in that leaf node) gets assigned as the class label for the leaf node. Any new observation whose predictor values fall in the leaf node will be classified as this class label. The advantage of using decision trees is its ability to provide interpretable classification rules which makes it a desirable choice for prediction over black-box models such as neural networks (Kumar (2017)). Interested readers are referred to Han et al. (2011) for more details.

3.2. Model Building paradigms

The conventional and widely used paradigm of predictive model building involves building a model on historical data and using it for predictions on unseen data. The paradigm assumes that the data generating process remains unchanged throughout i.e. there is no concept drift present. This approach is referred to as *batch* approach henceforth.

As mentioned earlier, data streams are often subject to concept drift. Thus, the current data may be generated from a model which is different from the historical data. Batch approach fails to factor in this change in the data generating process and a different approach is required. One of the possible approaches is to discard old data and only include recent data for analysis. Windowing is a commonly used technique for analysing data streams which is built on this premise. In the Sliding window model, incoming data is partitioned into chunks and the recent data is used for training the predictive models. The underlying assumption is that when data streams exhibit concept drift, more recent data contains the latest 'concept', hence it should be used to train the model. In this paper, by *stream* approach we refer to the process where we divide the incoming data stream into windows of fixed size and use them to train the predictive model at fixed intervals. As we are using more recent data, we expect the predictive performance metrics to be higher

for the *stream* approach as compared to the *batch* approach. However, it is important to note that *stream* approach requires (near) instantaneous availability of labels which in practice may not be possible.

Some of the existing methods in the literature focus on drift detection in unlabeled data stream environments utilizing only the predictor information to detect concept drift. The *HDDDM* algorithm takes this approach whereas the *OTDD* algorithm takes a different route where the relationship between the predictors and the labels is utilized (see Algorithm 1).

3.3. Threshold setting

In previous literature, drift detection algorithms have used a notion of threshold or cutoff value which is compared with the drift tracking metric at every iteration. If the metric exceeds this threshold, the current data window is declared to be 'different' from the current concept. The threshold value is an important input to drift detection algorithms as it determines its sensitivity.

It is often seen that the threshold is static i.e. it does not change over time (Gama et al. (2004); Baena-Garcıa et al. (2006)). However, when concept drift is present the use of static threshold is not recommended as that can lead to an increase in false alarms and/or false negatives both of which impacts the performance of the drift detection methods adversely. Thus,this is a serious limitation.

Another approach for threshold setting involves experimenting with various possible values and choosing a value with the highest resulting accuracy on the test data (Ditzler & Polikar (2011)). This approach involves re-running multiple experiments every time a concept drift is detected to accommodate the varying nature of concept drifts. This approach involves considerable computational costs and may be challenging to deploy in real-time applications.

A third approach is to set the threshold via the domain knowledge of a committee of domain experts. This is a subjective approach and makes the algorithm less autonomous in terms of human involvement (Khamassi et al. (2018)). A data-driven approach to setting the threshold would be objective, reliable, modular and agnostic to drift detection logic. It would also minimise parameter dependence, making it more usable (Krempl et al. (2014)). For the OTDD Algorithm, we propose a data-driven approach to dynamically estimate the threshold each time a drift is detected (see Algorithm 2).

3.4. Optimal Transport

In optimal transport based drift detection (OTDD), we train the classifier (logistic regression or decision tree) on the reference window for which labels are available. After the model is built for each observation in the reference window, we obtain the 'fitted value', which in this case is a probability distribution (a.k.a. probability histogram) on the possible values taken by the response (i.e. the labels). Let $\mathscr{C}$ denote the set of all these fitted probability distributions. As can be seen from Algorithm 1, we intend to compare sets of probability distributions and for this we need a measure to summarise a set of probability distributions. We use Wasserstein barycentre as the summary measure.

Next, we provide a short background of the Optimal Transport problem and the associated distance measure which we refer to as Optimal Transport distance. Figure 1 (a) shows the height(h) of a pile of sand at different locations(l) along the x-axis. Figure 1)(b) gives the desired distribution of this pile of sand. The task at hand is to transform pile (a) to pile (b). In order to do that, some portion of pile (a) has to be shifted from one place to another until the shape of the pile resembles pile (b). A weighted sum of distances by which the pile has been shifted, where the weights are the amount of sand shifted gives the Earth mover's distance between piles (a) and (b). The optimal transport problem seeks to minimize the the Earth mover's distance between piles (a) and (b). More formally, suppose I_0 and I_1 are two mass distributions over their respective domains Ω_0 and Ω_1. Let $c(x, y)$ is the cost function that represents the cost of moving one unit of mass from point x to point y. Further let, $\gamma \in$ MP be a measure-preserving map (which means the entire mass from I_0 is moved to I_1) which provides the transportation plan. $\gamma(A, B)$ represents the mass to be moved from set A to set B. Then, the mathematical formulation of the optimal transport problem is given by:

$$K(I_0, I_1) = \min_{\gamma \in MP} \int_{\Omega_0 \times \Omega_1} c(x, y) d\gamma(x, y) \tag{4}$$

where the minimum is taken over all measure preserving transportation plans γ (Kolouri et al. (2017)). A special case of Wasserstein Distance is the 1^{st} Wasserstein distance where the area under both the original and target distribution sums to 1. In this paper, we deal with probability distributions (which sum to 1 by definition) and hence all further references to Wasserstein Distance would imply 1^{st} Wasserstein Distance,denoted by $W(I_0, I_1)$.

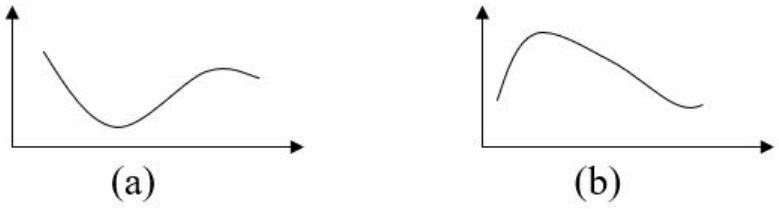


Figure 1: Sand pile example to illustrate the Optimal Transport problem.

The Wasserstein barycentre is a summary measure for a set of probability distributions. Suppose $\mathscr{C} = \{P_1, \ldots, P_N\}$. The Wasserstein barycentre P_B minimises the sum of Wasserstein distances from each of the individual probability distributions. Formally,

$$P_B = \arg\min_{P} \sum_{j=1}^{N} W(P, P_j) \tag{5}$$

where the minimum is taken over the set of all probability distributions (P). In this paper, we use the terms 'Wasserstein barycentre' and 'barycentre' interchangeably.

It might be argued that a simple average could be taken of all distributions for summarization. Consider the distributions shown in Figure 2. Figure 3 shows the simple average of the probability distributions (to be referred to

as Euclidean average henceforth) and the barycentre distribution. It is seen that if we take a Euclidean average of the distributions, the resultant distribution does not look 'representative' of the constituent distributions whereas the barycentre distribution is much more representative. This motivates the use of barycentre distribution to summarise the set of probability distributions.

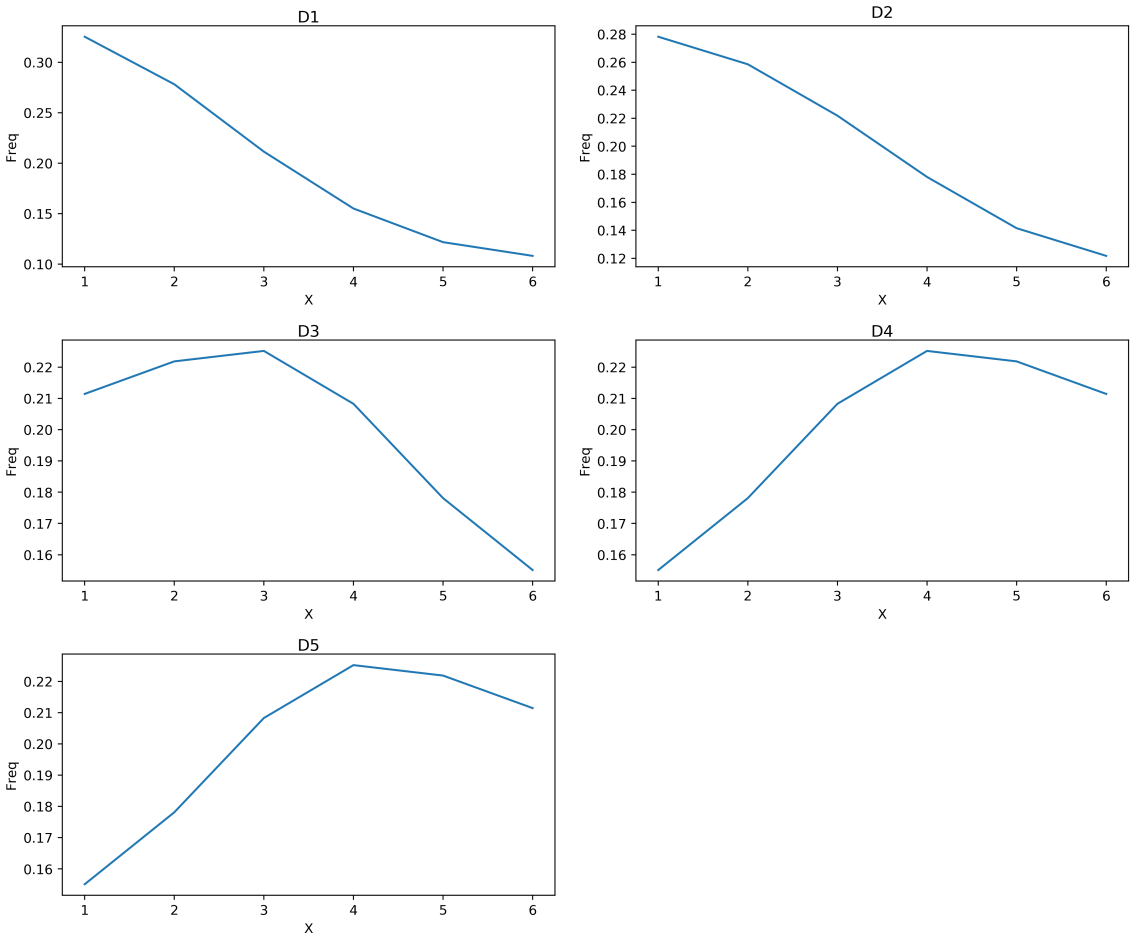


Figure 2: Five example Distributions

After summarising outputs of classifiers in each window via the barycentre, the OTDD Algorithm (see Algorithm 1) requires a way to measure how the barycentre of the current window is different from barycentre of the reference window. For this, we use the Kullback- Leibler Divergence, discussed in section 3.5.

3.5. Kullback-Leibler Divergence

To measure the distance between barycentres of the reference window and current window, we use Kullback-Leibler (KL) divergence. KL Divergence gives a measure of similarity of two probability distributions, hence is suitable for comparing barycentres in the OTDD algorithm (see Algorithm 1).

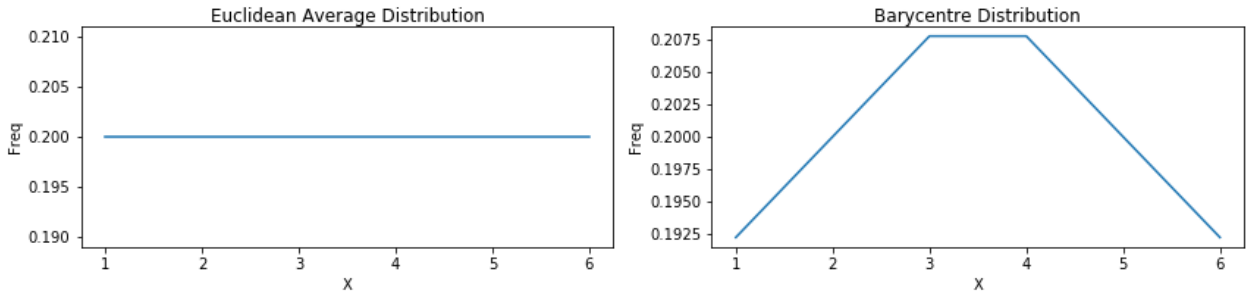


Figure 3: Summarised Distributions

The KL-Divergence between two discrete probability distributions P and Q having a common support χ is defined as

$$KL(P,Q) = \sum_{x \in \chi} p(x) \log \left(\frac{p(x)}{q(x)} \right) \tag{6}$$

where $p(x)$ and $q(x)$ are the probability mass functions of the probability distributions P and Q respectively.

It is easy to see that if the distributions P and Q are identical then $KL(P,Q) = 0$. In general, smaller values of KL divergence indicate that the two distributions P and Q are similar while larger values of KL-divergence are indicative of dissimilarity between the two distributions. It is worthwhile to note that KL-Divergence is a quasi-distance metric and not a distance metric as it doesn't satisfy two conditions to be a distance metric (Deza & Deza (2006)). It is not symmetric as $KL(P,Q) \neq KL(Q,P)$. Further, it also does not satisfy the triangle inequality. For more details on the KL-divergence, the reader may refer to (Abou-Moustafa & Ferrie (2012))

4. Method

This section provides an intuitive explanation of the proposed Optimal Transport drift detection Algorithm (OTDD) and Data-Driven Threshold Estimation Algorithm (DDTE) alongwith the pseudocodes.

4.1. Optimal Transport based Drift Detection Algorithm

In this subsection, we introduce the Optimal Transport based Drift Detection (OTDD) algorithm which uses a probabilistic classifier (such as multinomial logistic regression classifier, decision trees etc.) to model the relationship between the outcome variable (class) and the predictors in a multi-class classification problem in a streaming data set-up. The stream is divided into windows of a fixed size (*win_size*) given by the user. For the initial window (*ref_win*), the classifier (*ref_model*) is trained. The trained classifier is then applied to each observation in this window. It may be noted that, the output of the classifier when applied to an observation is a probability distribution on the outcome classes that can be visually represented as a probability histogram. Since there are *win_size* number of observations

in the initial window, we obtain that many probability histograms. We call this collection of probability histograms *ref_prob*. In what follows, we use the terms 'histograms' and 'probability histograms' interchangeably.

Applying this scheme of obtaining outputs from the trained classifier (*ref_model*) on the next data window (win_1), we obtain another collection of histograms ($prob_1$). The collection *ref_prob* serves as the reference against which the characteristics of the next window,captured in $prob_1$ is compared. The complexity in this exercise arises from the fact that we need to compare two collection of histograms- $prob_1$ with *ref_prob*. Towards this task, we summarise each collection of histograms using its Wasserstein barycentre (see Section 3.4) which gives a 'representative' histogram. Then, we compare the barycentres of the two collections of histograms. Let the barycentre of the *ref_prob* be *ref_bary* and that of the $prob_1$ be $bary_1$. We compare *ref_bary* and $bary_1$ using KL-divergence (see Section 3.5). If the value of K-L divergence exceeds the *threshold* c, a drift is signalled. Otherwise, it is assumed that there is no drift and the algorithm proceeds to consider the next window. The determination of the value of *threshold* c using the initial window as the reference is discussed in section 4.2.

The signalling of drift is accompanied by updating the reference entities and measures (i.e *ref_win, ref_model, ref_prob, ref_bary and threshold*) against which all successive windows, until the next drift signalling, are compared. It involves retraining the classifier on the window where the algorithm signals a drift. This is done in order to capture the latest characteristics of the data i.e 'concept'. It is followed by obtaining the updated *ref_prob* and representing it via it's *ref_bary*. The threshold c is also updated using the algorithm 2 as it is dependent on the *ref_win*. In what follows, we will refer to this entire process of updating the reference entities and measures as the *reference updation* procedure.

In general, for the t-th window win_t, we obtain a collection of histograms ($prob_t$) from the output of the current probabilistic classifier (*ref_model*) which is summarised using it's Wasserstein barycentre ($bary_t$). The $bary_t$ of the t-th window is compared to the current *ref_bary* using K-L Divergence. If the value of K-L Divergence exceeds the current threshold c, the algorithm signals a drift and the *reference updation* procedure is followed which updates all the reference entities and measures. Otherwise, the algorithm proceeds to the next window.

The pseudocode of the proposed OTDD algorithm is shown in Algorithm 1.The description of the variables used in the algorithm 1 is given below:

Datastream$_x$= vector of predictors
DataStream$_y$=y is class label

win_size = number of observations included in initial and successive windows
ref_win$_x$=initial data window of Datastream$_x$
ref_win$_y$=initial data window of Datastream$_y$
ref_model = probabilistic classifier model trained on ref_win$_x$ and ref_win$_y$
ProbClassifier=function that returns the classifier model

Class probability=function that returns collection of probability histograms by ProbClassifier

ref_prob=collection of probability histograms obtained as output from ref_model

ref_bary=Wasserstein barycentre corresponding to ref_prob

threshold=threshold value corresponding to ref_win$_x$ and ref_win$_y$

Every time the OTDD Algorithm detects a drift, the *threshold* gets updated using the Data-Driven Threshold Estimation (DDTE) Algorithm which provides a dynamic way to obtain the threshold value. Please refer to section 4.2 for a description of the DDTE algorithm.

It is important to note the fact that the OTDD algorithm only requests for labels when a drift is detected to retrain the classifier. When there is no drift, the algorithm does not require ground truth labels. In this manner, the demand for labeled information is significantly reduced over the entire data stream. The OTDD Algorithm offers the flexibility to choose any probabilistic classifier as long as it provides probability histograms as the output. In this paper, we have considered logistic regression and decision tree classifiers as the $ProbClassifier$ in the OTDD Algorithm(section 3.1).

4.2. Data Driven Threshold Estimation (DDTE) Algorithm

In this section we discuss the procedure for determining the threshold which plays a critical part in the execution of the OTDD algorithm. The Data-Driven Threshold Estimation (DDTE) algorithm (also referred to as Algorithm 2) would provide an effective method of determining the threshold dynamically over time. The OTDD algorithm requires the threshold to be updated every time a drift is detected, which makes a static threshold approach unusable.To address this,the DDTE Algorithm provides for a dynamic approach for threshold determination every time a drift is signalled by the OTDD algorithm.

The DDTE-algorithm takes in the data window regarded as the current 'reference' (ref_win$_x$, ref_win$_y$) and its associated barycentre (ref_bary) as the input. Here Y is a categorical variable with K classes. Let p_i denote the proportion of the observations belonging to class i in the ref_win$_y$, $i = 1, \ldots, K$. The DDTE algorithm assumes that the distribution of the predictors in each class is a multivariate normal (MVN) distribution. The predictors of the i-th class are assumed to come from MVN distribution with mean μ_i and variance-covariance matrix $\mathbf{\Sigma}$. It may be noted that the variance-covariance matrix is assumed to be the same for all the classes. The parameters $\mu_i, i = 1, \ldots, K$ and $\mathbf{\Sigma}$ are estimated from the data in ref_win$_x$ and we refer to them as $\hat{\mu}_i$ and $\hat{\Sigma}$ respectively.

The next step of the DDTE algorithm is to simulate synthetic observations with characteristics similar to (ref_win$_x$, ref_win$_y$). Let the number of observations in ref_win$_x$ be n. We first simulate n class labels from Multinomial $(K; p_1, \ldots, p_k)$. For each of the simulated labels $\tilde{y}_i$, we simulate the associated set of predictors $\tilde{x_i}$ from MVN($\hat{\mu}_{\tilde{y}_i}, \hat{\mathbf{\Sigma}}$). This simulated predictors are stored in win$_t$(x) and simulated class labels are stored in win$_t$(y).

Algorithm 1: Optimal Transport based Based Drift Detection (OTDD) Algorithm

Input: DataStream $_x$, DataStream$_y$,*win_size*

Output: Updated probabilistic classifier ref_model at every ρ

Set $\rho = 0$

Set ref_win$_x$=Datastream$_x$[0 : *win_size*]

Set ref_win$_y$=Datastream$_y$[0 : *win_size*]

Set ref_model=ProbClassifier(ref_win$_x$, ref_win$_y$)

Set ref_prob=ClassProbability(ref_model,ref_win$_x$)

Set ref_bary=Barycentre(ref_prob)

Set *threshold*=FindThreshold(ref_win$_x$, ref_win$_y$)

while *t=1,2,3...* **do**

win$_t$(x)=DataStream$_x$[t**win_size*: (t+1) *win_size*]

win$_t$(y)=DataStream$_y$[t**win_size*: (t+1) *win_size*]

prob$_t$=ClassProbability(ref_model ,win$_t$(x))

bary$_t$=Barycentre(prob$_t$)

Compute KL$_t$=KLDivergence(ref_bary,bary$_t$)

if *KL$_t$ >threshold* **then**

Concept Drift is detected

Set $\rho = t$

Update reference entities and measures

ref_win$_x$=win$_t$(x)

ref_win$_y$=win$_t$(y)

ref_model=ProbClassifier(win$_t$(x),win$_t$(y))

ref_prob=ClassProbability(ref_model,ref_win$_x$)

ref_bary=Barycentre(ref_prob)

threshold=FindThreshold(win$_t$(x),win$_t$(y))

else

No drift detected

Algorithm 2: Data-Driven Threshold Estimation (DDTE) Algorithm

Input: ref_win$_x$,ref_win$_y$,ref_bary, *m*

Output: *threshold*

c=$\{1, \ldots, K\}$

KL_array$\leftarrow \emptyset$

forall *elements of c* **do**

  Estimate parameters of MVN (μ_i,$\mathbf{\Sigma}$)

  Compute $p_i = \frac{\text{Number of observations with label i in ref\_win}_y}{\text{Number of observations in ref\_win}_y}$

Define n = number of observations in ref_win$_x$

while *t=2,3,. . . , m* **do**

  win$_t$(y)=Sample n class labels from Multinomial$(K, p_1, \ldots, p_K)$

  win$_t$(x) = ϕ

  for *k=1,2,. . . ,n* **do**

    x_k=Random sample from MVN($\hat{\boldsymbol{\mu}}_{\boldsymbol{y_k}}$, $\hat{\mathbf{\Sigma}}$)

    win$_t$(x) = win$_t$(x) $\cup\{x_k\}$

  ref_model=ProbClassifier(ref_win$_x$,ref_win$_y$)

  prob$_t$=ref_model(win$_t$(x))

  bary$_t$=Barycentre(prob$_t$)

  KL$_t$=KLDivergence(ref_bary,bary$_t$)

  KL_array=KL_array $\cup$ KL$_t$

threshold=95thpercentile(KL_array)

The classifier (ref_model) is trained on the current 'reference' i.e. (ref_win$_x$,ref_win$_y$). This trained model is used for prediction with the simulated data. For each observation in *win$_t$(x)*, we obtain a probability histogram on the set of all class labels. We call this collection of probability histograms prob$_t$. The Wasserstein barycentre *bary$_t$* is used to summarise prob$_t$. The KL-Divergence measure is computed between *bary_ref* and *bary$_t$*. Since *bary$_t$* is associated with the simulated data window that has characteristics similar to the current 'reference' this exercise repeated several times would yield a distribution of the KL-Divergence values when there is no 'concept drift' present in the data stream. This process is repeated for a specified number of times(m) and the 95^{th} percentile of the distribution of the KL-Divergence values is taken as the *threshold* for drift detection. For the purpose of this paper, we use 1,000 iterations to determine the *threshold*. The pseudocode of the DDTE Algorithm is given in Algorithm 2.

Although we demonstrate all results of OTDD used in conjunction with DDTE, it may be noted that the underlying idea of DDTE can be adapted for other drift detection algorithms with appropriate modifications. This algorithm is called by Algorithm 1 whenever a drift is detected to update the classifier to adapt to the current concept.

4.3. Evaluation Scheme

The purpose of drift detection in a predictive task is to raise a signal that the model needs an update. It is generally observed that if the model is not updated, the predictive performance degrades. Thus, after the update signal is obtained it is appropriate to update the model using the latest available data. A commonly used evaluation metric is accuracy which is used by several authors to compare among drift detection algorithms (Ditzler & Polikar (2011); Gama et al. (2004); Baena-Garcıa et al. (2006)).

However, in this paper, we focus on real-life situations where acquiring labels is costly and not instantaneous. While methods that assume complete access to labels are likely to have better accuracy than methods with restricted access to labels, the former methods are not usable in this set-up. Using accuracy as the evaluation metric disregards this challenge encountered in many applications. Thus, we need an evaluation metric that incorporates accuracy-labeling requirement trade-off.

We propose a new metric called Trade-off Criterion (TC) to facilitate the trade-off between the accuracy of the method and the labeling requirement. The formula for the TC is given below:

$$TC = \lambda * Acc - (1 - \lambda) * LabelsReqd$$

where *Acc* is the accuracy of method and *LabelsReqd* is the fraction of total windows in a data stream for which labels are required by the method. $\lambda \in [0, 1]$ is a constant that controls the relative importance given to either term and can be appropriately chosen by the decision maker. An ideal classifier in the streaming context, would be one that achieves an accuracy of 1 without requiring any labels subsequent to training on initial data window when no concept drift is present. Thus, the maximum value of $TC \approx \lambda$ would be achieved for a long data stream having a large number of windows but without presence of any concept drift. On the other hand, the worst classifier would have an accuracy

of 0 despite having access to labels of all windows. This would have $TC = -(1 - \lambda)$. Hence, the range of TC is $[-(1 - \lambda), \lambda)$.

Choosing the value of λ allows the decision-maker to tune the requirements on the performance of the model appropriately to the decision context. For example: while building a model for classifying medical images into malignant or benign tumor categories, the decision maker is likely to emphasize on accuracy as a misclassified observation would imply a patient either receiving inadequate or unnecessary treatment. Since the performance of the OTDD algorithm would be dependent on the choice of the value of λ, in this paper we report the performance for different values of λ to understand the impact of the choice of λ (see Section 6).

In the *batch* setup, it is implicit that there would be no concept drift and hence the dataset under analysis would be representative of all the future data. Under this assumption, the entire data is split into training and test datasets and the model is trained using the training dataset and the performance of the model is evaluated on the test data. However, this important underlying assumption is not valid when working with streaming data that may be subject to concept drift. Hence, the well-known evaluation techniques for methods working with batch data would not be appropriate for use with methods designed to work with streaming data.

Two main evaluation strategies exist for methods working with streaming data: (i) Holdout and (ii) Prequential (Gama et al. (2014)). In the Holdout scheme, a common subset of data is excluded from the training process and only used to evaluate and compare different algorithms. However, in practice identifying such a subset whose characteristics are representative of the entire data stream is a challenging task. This is so because of the unpredictable nature of the concept drift. As a result, the concept drifts that may be present within the training data may be quite different from those in the Holdout data. This is especially difficult in real-life deployments of drift detection algorithms as in most cases, there is no way to concretely establish the presence/ absence of drift and its associated attributes like speed, magnitude, and recurrence. In the Prequential scheme, as data instances are arriving continuously, each new arriving window is first used for testing performance of the learner. This ensures that model evaluation is always done on the latest concept contained in most recent data window which can be further used to decide whether the model needs an update. In this paper, we adopt a Prequential strategy for algorithm performance evaluation in conjunction with the TC.

4.4. Experimental Design

In order to demonstrate the performance of the proposed OTDD algorithm, we compare its predictive performance with respect to three methods: *batch*, *stream* and Hellinger Distance based Drift Detection Method (*HDDDM*) (Ditzler & Polikar (2011)). The *stream* and *batch* approach represent two extreme approaches for prediction in data streams with regard to labeling requirements. They are chosen as they provide the bounds on the labeling requirement against which the *OTDD* algorithm would be compared. The *batch* approach has no drift awareness as it doesn't update the classifier once it has been trained. While the *stream* approach is 'passively' drift-aware as it updates the classifier at every new window but doesn't have an explicit drift detection mechanism. The *HDDDM* and the *OTDD* algorithms

are drift aware and they update the classifier only after a drift is detected. Further details of these approaches are given below.

The dominant paradigm in machine learning involves splitting the available data into training and test subsets. The training subset is used as the basis for the classifier to learn the relationship between predictor and outcome variable. The test subset contains observation unseen by the classifier and is used to measure its predictive performance which indicates its generalisability. For the purpose of this paper, we train the classifier on the initial data window and use it for prediction on each incoming window without any update. The labeling requirements for this approach are minimal as the classifier is only trained once and never updated. However, the performance of a classifier trained under this method is expected to deteriorate in presence of concept drift. The performance of this approach is further explored in section 6. We call this the *batch* approach and report results for logistic regression and decision tree classifier. Figure 4 gives the graphical description of the *batch* approach.

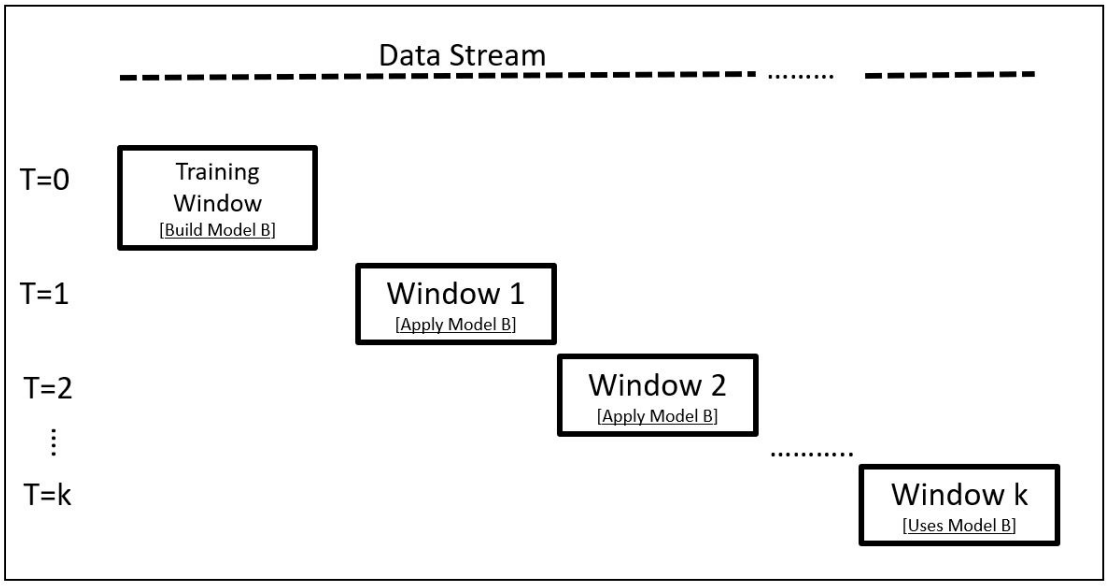


Figure 4: Graphical description of *batch* approach.

Since, concept drift in a datastream is a common occurrence an alternative approach called the *sliding window* approach is often used (Aggarwal (2007)). In this approach, the incoming instances are divided into windows of fixed/variable size. In order to adapt to the latest 'concept', the classifier is trained on each incoming data window and the old data is discarded. In this paper, we call this the *stream* approach (see Figure 5). Thus, the *stream* approach requires labeled data for every data window. As mentioned before, the *stream* approach is on the opposite end of the spectrum in terms of labeling requirements with respect to *batch* method. However, under the *stream* approach, since the classifier is updated at every incoming data window, we expect it to maintain its performance even in the presence

of concept drift.

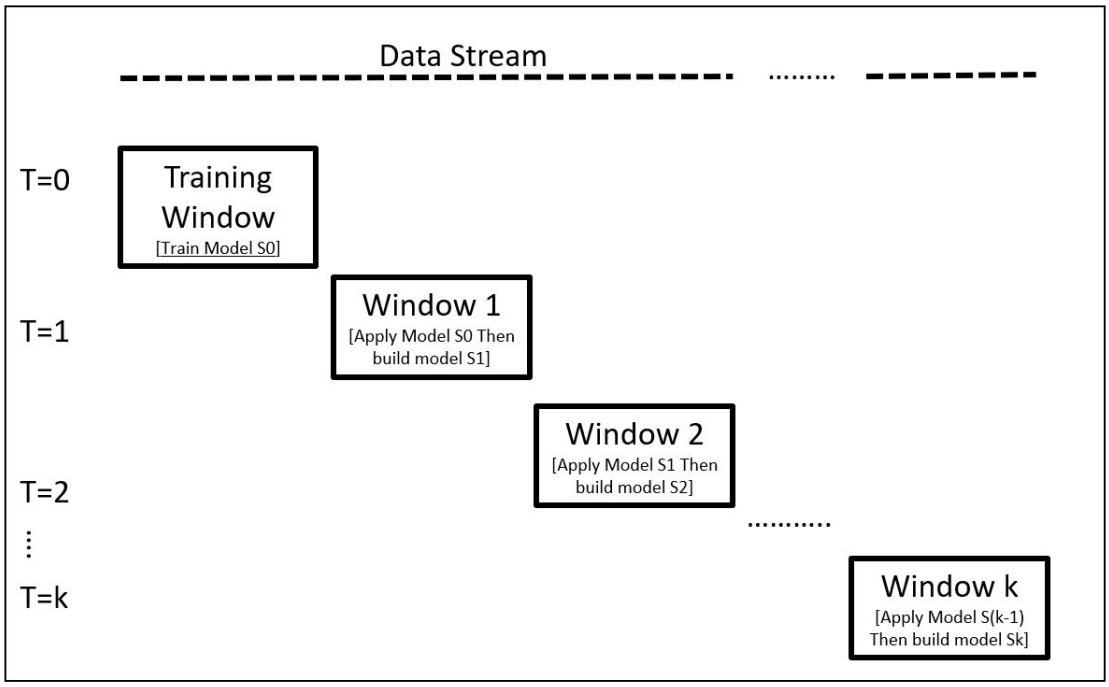


Figure 5: Graphical description of *stream* approach.

In the *HDDDM* algorithm, data is divided into windows and a drift tracking metric based on the Hellinger distance between histograms of successive windows is used to detect drift. If the drift tracking metric exceeds the threshold, old training data is discarded and the classifier is trained on the latest data window. Otherwise, the latest data window is appended to the training set and the classifier is retrained. In essence, it is an incremental learning algorithm with drift awareness. Consequently, *HDDDM* requires access to labels for all data windows. A graphical description of HDDDM and the OTDD algorithms are given in Figures 6 and 7 respectively.

This gives a set of four methods (*batch, stream, OTDD, HDDDM*) for performance comparison. In order to understand the dependence of these four algorithms to the choice of the classifier, we implement these approaches for logistic regression and decision tree classifier (section 3.1). We examine the performance of these approaches for different window sizes as their performance may be sensitive to the choice of window size. The results of these studies are provided in section 6.

5. Data

In this section, we describe the artificial and real-life datasets used for evaluating performance of the proposed OTDD algorithm. We also provide the motivation behind choosing the datasets that aid in rigorous evaluation of the proposed algorithm.

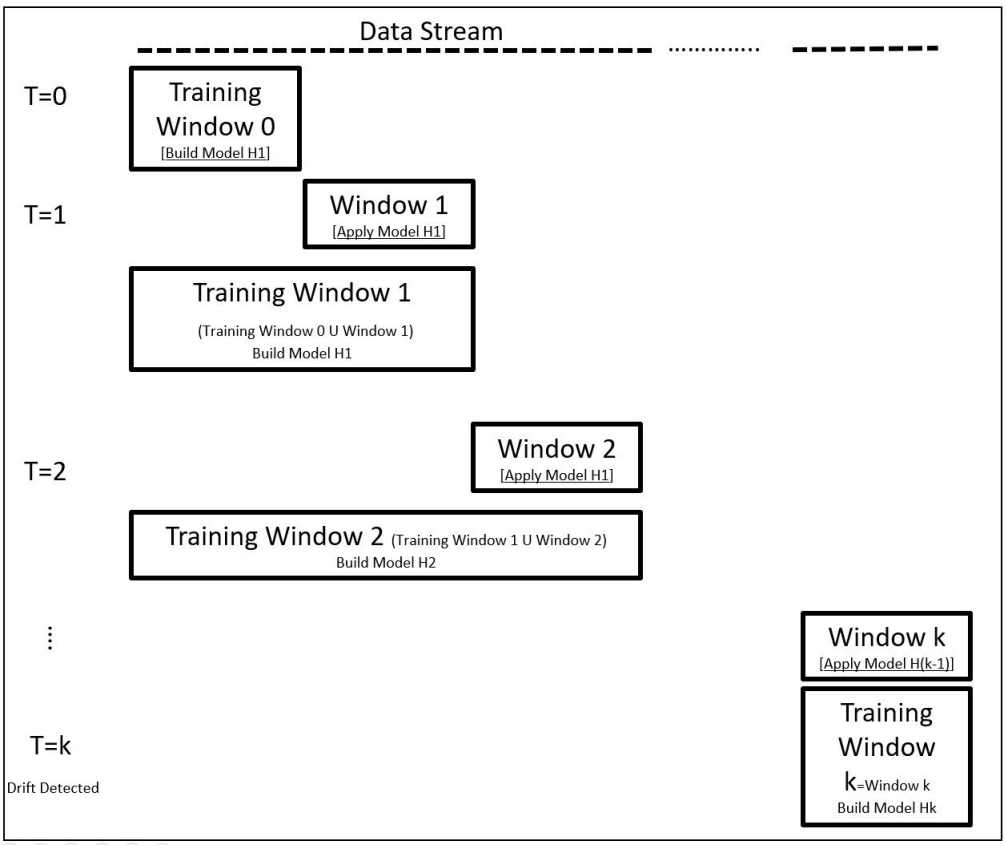


Figure 6: Graphical description of *HDDDM* approach.

5.1. Artificial Datasets

By the virtue of its design, The OTDD algorithm is expected to perform well in detecting abrupt drifts. However, the nature of drift (i.e. abrupt or gradual) are hard to establish a priori when drift detection algorithms are deployed in practice.

In order to demonstrate the performance of OTDD and its efficacy in practice, we use two artificial datasets: SYN and HYP. The SYN dataset which is described in 5.1.1, is expected to inform us about the performance of the OTDD algorithm when only abrupt drift is present. In the HYP dataset (Hulten et al. (2001)), it is known that there is gradual drift. Though, this dataset doesn't originate in a transportation context, the evaluation of the OTDD algorithm on HYP dataset informs us about its efficacy in presence of gradual drifts. Further details about the HYP dataset can be found in section 5.1.2.

5.1.1. Synthetic Datasets (SYN)

In this paper, we deal with drift detection in a classification setup. Thus, we not only need to simulate the predictors but also need to link with the class labels. Further, the predictors may not be mutually independent which needs to be taken into account. Moreover, the concept drift in this set-up can occur when the distribution of one or more of the

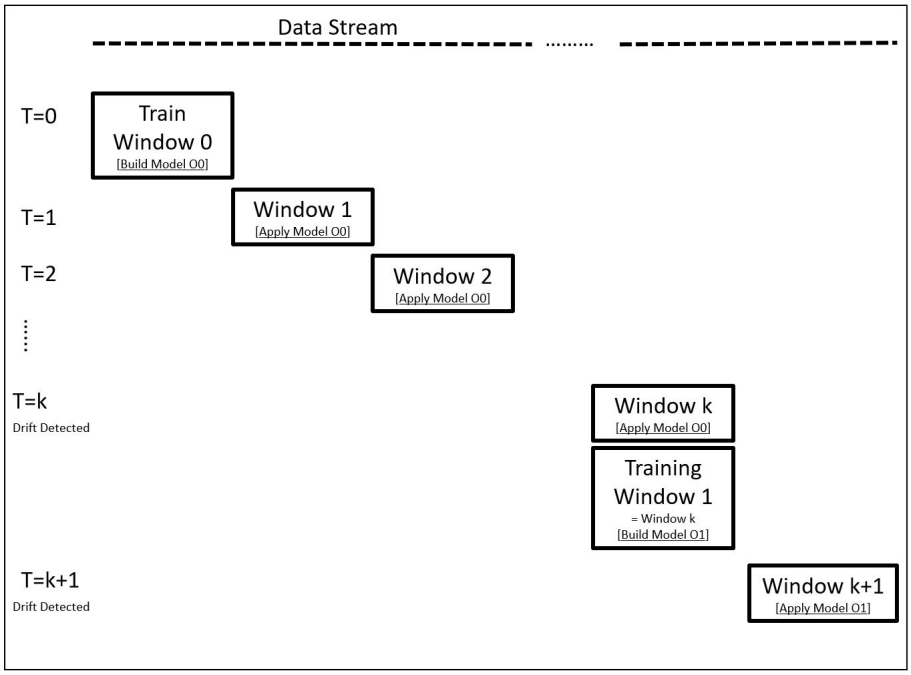


Figure 7: Graphical description of *OTDD* approach.

predictors may change at a time point.

In view of the above, to test the efficacy of the OTDD-algorithm,we use synthetic data streams that are generated as per Algorithm 3. Here please note that for any two vectors $A = (a_i)$ and $B = (b_i)$, $A * B$ denotes the vector $A * B = (a_i b_i)$.

In this paper, we consider 9 scenarios by varying P and M and generate 9 synthetic datasets with $stream\_size = 50000$. The details are given in Table 1 and 2 below.

5.1.2. Hyperplane(HYP)

Hyperplane is a publically available synthetic dataset where a drift is created by changing parameters for orientation and position of hyperplane in d-dimensions (Hulten et al. (2001)). It has been used widely in streaming data literature to benchmark performance. A hyperplane in d-dimension is denoted by the equation:

$$\sum_{i=1}^{d} w_i x_i = w_0 \tag{7}$$

where x_i is the ith coordinate of x and w_i. Observations for which $\sum_{i=1}^{d} w_i x_i \geq w_0$ are labelled as 1 and observations with $\sum_{i=1}^{d} w_i x_i < w_0$ are labelled as 0, making it a binary classification problem. Drift is induced in data by changing position and orientation of the hyperplane controlled by the weight parameter. In this paper, we consider three instances of hyperplane with k and t, where k represents the number of dimensions drifting and t represents the magnitude

Algorithm 3: Synthetic Data Generation Algorithm

Input: P (position of the concept drift in data stream)

M (magnitude of drift)

I(Set of predictors to be drifted)

K (number of classes),$p_1, \ldots, p_k$ (class probabilities)

$\boldsymbol{\mu_1}, \ldots, \boldsymbol{\mu_K}$ (Mean of predictors for each class)

$\sum$ (Variance-covariance matrix for all classes)

$stream\_size$ (number of observations in data stream)

Output: Synthetic data stream **DS** with drift of magnitude **M** at position **P**

if *observation number(n)* $<$*P* **then**

Randomly generate class label y_n from multinomial distribution with parameters $(K, p_1, \ldots, p_k)$

Sample predictor values $\boldsymbol{x_n}$ from multivariate normal distribution of class $y_n = c$ with mean $\boldsymbol{\mu_c}$ and variance covariance matrix. $\sum$

else

Randomly generate class label y_n from multinomial distribution with parameters $(K, p_1, \ldots, p_k)$

Sample predictor values $\boldsymbol{x_n}$ from multivariate normal distribution of class $y_n = c$ with mean $\boldsymbol{\mu_c} * \mathbf{M}$ and variance covariance matrix $\sum$

$DS = \{(\boldsymbol{x_n}, y_n) : n = 1, \ldots, stream\_size\}$

of change. Results are reported for HYP3 (k=2,t=1), HYP5 (k=5,t=0.5) and HYP8 (k=8,t=0.5). In all these three datasets, value of d=5. The datasets can be downloaded from https://www.win.tue.nl/~mpechen/data/DriftSets/

5.2. Real-Life Datasets

In this section, we introduce two real-life datatsets chosen to demonstrate the performance of the OTDD algorithm for problems arising in the transportation domain. The Transport Mode Detection (TMD) dataset consists of sensor readings from accelerometer, gyroscope and sound sensors of a smartphone, which are used to predict whether the subject is using bike, bus, car, train or is standing still. The Ford StayAlert dataset uses observations from vehicular, environmental and physiological state of the driver to predict whether the driver is alert or not.

5.2.1. Transport Mode Detection (TMD)

Transport Mode detection refers to identifying travel mode (car/bus/walking) of a commuter via smartphone sensor or GPS stream. It falls under the broad umbrella of human activity recognition tasks which deals with extracting knowledge and awareness of human activities from external and wearable sensors(Lara & Labrador (2012)).

Mode choice is an important defining characteristic of travel behavior(McFadden (1974)) as it gives insight into the preferences of commuters in the presence of alternatives. Accurate estimates of proportions of commuters using

Scenarios	Dataset Name	Position(P)	Magnitude(**M**)
1	SYN_P10_M30	$\lceil$0.1*stream_size$\rceil$	$[1, 1, 1.3, 1.3, 1]$
2	SYN_P10_M40	$\lceil$0.1*stream_size$\rceil$	$[1, 1, 1.4, 1.4, 1]$
3	SYN_P10_M50	$\lceil$0.1*stream_size$\rceil$	$[1, 1, 1.5, 1.5, 1]$
4	SYN_P30_M30	$\lceil$0.3*stream_size$\rceil$	$[1, 1, 1.3, 1.3, 1]$
5	SYN_P30_M40	$\lceil$0.3*stream_size$\rceil$	$[1, 1, 1.4, 1.4, 1]$
6	SYN_P30_M50	$\lceil$0.3*stream_size$\rceil$	$[1, 1, 1.5, 1.5, 1]$
7	SYN_P50_M30	$\lceil$0.5*stream_size$\rceil$	$[1, 1, 1.3, 1.3, 1]$
8	SYN_P50_M40	$\lceil$0.5*stream_size$\rceil$	$[1, 1, 1.4, 1.4, 1]$
9	SYN_P50_M50	$\lceil$0.5*stream_size$\rceil$	$[1, 1, 1.5, 1.5, 1]$

Table 1: SYN Dataset scenarios.

various alternative modes of transport can help public authorities in transport demand estimation and planning. For example, if such an analysis of a particular region shows heavy dependence on private modes of transportation, it can be a useful input into the transport authority's decision problem of whether to increase the reach and frequency of public transportation or to promote alternative sustainable choices. It can also be leveraged by private entities for building mobility-aware applications and data collection capabilities.

The data for the travel mode detection task was conventionally collected via travel surveys and travel diary. Travel diary involves the respondents self-reporting their trips in the entire day along with its attributes such as start and end timestamps, start and end locations, the purpose of the trip, mode of travel, etc.

Collecting data via surveys is a time-consuming, costly process that suffers from low response (Richardson et al. (1996)) rates, recollection bias and induces high respondent burden (Stopher et al. (2008)). To put the costly and time-consuming nature of travel surveys in perspective, for the 2010 California Household Travel Survey (California Department of CDOT (2013)), over 19,692 households were surveyed for a period of one year and monetary incentives to them ranged from $20-$40, amounting to expenses of approximately $ 0.5 million only in respondent payouts.

There has been a marked shift in data collection for travel behavior research towards automated data collection methods. These sources for automated data collection include GPS, smartphone sensor and network data (Kanarachos et al. (2019); Huang et al. (2019); D'Andrea & Marcelloni (2017); Liu et al. (2014)), sound sensors(Cevher et al. (2008)), road-side loop detectors (Bajwa et al. (2011)), CCTV(Lee et al. (2017),), and social media data (Dabiri & Heaslip (2019)).

In addition to academic research being focussed on leveraging big data, many transport authorities have also started collecting data from a small percentage of respondents using GPS loggers and wearable sensors. The reader may refer to NREL (2019) for a list of transportation studies and surveys in the United States that have used GPS loggers and wearable sensors. In addition to enabling cheap data collection at scale, automated data collection methods also allow

Distribution	Parameters	Values
Multivariate Normal (MVN)	$\boldsymbol{\mu_1}$	$[1, 1, 1.4, 1.4, 1]$
	$\boldsymbol{\mu_2}$	$[1, 1, 1.5, 1.5, 1]$
	$\boldsymbol{\mu_3}$	$[1, 1, 1.3, 1.3, 1]$
	$\boldsymbol{\mu_4}$	$[1, 1, 1.4, 1.4, 1]$
	$\boldsymbol{\mu_5}$	$[1, 1, 1.4, 1.4, 1]$
	$\sum$	$\begin{bmatrix} 1.1 & 0.3 & 1.8 & -0.9 & 0.1 \\ 0.3 & 0.7 & 0.3 & -0.3 & 0.0 \\ 1.8 & 0.3 & 3.5 & -1.6 & 0.1 \\ -0.9 & -0.3 & -1.6 & 1.2 & -0.1 \\ 0.1 & 0.0 & 0.1 & -0.1 & 0.1 \end{bmatrix}$
Multinomial	p	$[0.259, 0.133, 0.203, 0.403, 0.002]$

Table 2: Parameters used for generating SYN Datasets.

continuous data collection as opposed to once-in-a-decade travel surveys.(Nitsche et al. (2014))

The transport mode detection task is dependent on data collected by sensors that operate in dynamic environments with multiple interacting variables. As per Taylor & Fink (2013), travel patterns are impacted by demographic and socio-economic factors, spatial factors, pricing, service quality, and quantity. A change in any of these factors can cause a resultant change in travel patterns, some of which may be hard to isolate. This creates the need for concept drift detection algorithms that raise an alert whenever the incoming data distribution is 'significantly different' from the older data distribution.

The TMD dataset contains readings from the accelerometer, gyroscope, and sound sensors of the smartphone. Data collection was done for 13 subjects in which the subject at any point of time, may be using any one of the transport modes among bike, bus, car, train or is stationary (still). Data was collected via a smartphone application that runs passively in the background and ground truth labels of the actual transport mode being used were collected via a web interface. The dataset contains 5893 observations. The TMD dataset along with relevant details is available at http://tempesta.cs.unibo.it/projects/us-tm2017/.

5.2.2. Ford StayAlert

The pervasive presence of smartphones and vehicle sensors has enabled more granular, accurate, and unobtrusive mapping of driver behavior unlike travel surveys and naturalistic driving experiments (Nitsche et al. (2014); Düh et al. (2010)). Real-time detection of dangerous behavior like lane changing, braking, overtaking can be utilized to profile individual driving styles that can feed into connected intelligent transportation systems (ITS). Cellphone usage is one of the major causes of distraction among drivers with either handheld or connected device usage. Despite the ban on

hand held mobile phone usage while driving by multiple states in the United States of America (USA), the National Highway Traffic Safety Administration estimated 620,000 drivers on the road with cellphones on their ear on a typical daylight moment in the USA alone (Pickrell et al. (2015)). Early warning systems for driver alertness detection can help mitigate the enormous economic and social costs of car crashes. For example, the overall societal cost of motor vehicle crashes was estimated to be $836 billion in 2010 for the USA with 32,999 fatalities (Blincoe et al. (2015)). In India, 4,37,396 road accidents were reported in 2019, leading to 1,54,732 fatalities (National Crime Records NCRB (2019)).

Driver attention analysis is of particular interest for road safety as a timely alert of an inattentive driver can help avoid a potential accident. We use the StayAlert dataset which contains predictors of environmental, vehicular, and driver physiological data to predict driver alertness. It is a binary classification dataset with two classes (Alert/Not Alert) with sequential data collected over multiple driving trials, hence constituting a data stream. The dataset contains 6,04,328 observations. The StayAlert dataset and relevant details can be found at `https://www.kaggle.com/c/stayalert/data`.

There is no ground truth for the presence of concept drift in real-world data streams, so the accuracy difference between static and online learning algorithms is often used to establish the presence of concept drift (Klinkenberg & Joachims (2000)).

6. Results

In this section, we report the results of the comparative experiments, following the design described in section 4.4. In section 6.1.1, we report results on SYN and HYP datasets. In section 6.2, we report results on TMD and StayAlert datasets.

6.1. Artificial Datasets

In this section, we compare the performance of four competing algorithms on their TC values for different values of λ on artificial datasets SYN and HYP. We find that OTDD outperforms other methods as increasing emphasis is given to labelling requirements, both for abrupt drift in SYN and gradual drift in HYP.

6.1.1. SYN

In Table 3, we examine the performance of *batch*, *stream*, *HDDDM* and *OTDD* in terms of average TC values for different values of λ for the logistic regression classifier with Window size=1000 on SYN datasets. These TC values are obtained by averaging TC for SYN datasets with common value of P parameter. We observe that all the drift aware algorithms namely *stream*, *HDDDM* and *OTDD* perform better than *batch* in terms of TC for all the values of λ considered in this experiment. In the specific case when $\lambda = 1$ (i.e. when the trade-off criterion reduces to accuracy) we find that all the drift aware algorithms perform better than the *batch*. This result is along expected lines as the SYN

datasets contain concept drift that is not accounted for by *batch* since the classifier in this algorithm is not retrained after the occurrence of the concept drift.

As more weightage is given to proportion of labeled windows required, which is captured by decreasing value of λ from 1 to 0.7, the performance of the OTDD algorithm is seen to become increasingly superior in comparison to the *stream* and *HDDDM*.

Table 3 further allows us to examine the impact of the position of the concept drift on the performance of the algorithm. This is important since in real-life situations the position of the concept drift would be unknown and it is desirable to use an algorithm that performs well irrespective of the position of the concept drift. In this regard we find that when accuracy is the sole concern (i.e. λ=1) we find that the HDDDM is the method of choice as it performs the best for all the values of P considered in this study. As the value of λ is decreased, we find that OTDD generally performs better than all the other algorithms considered in this paper.

Method	P	λ =1	λ =0.9	λ =0.8	λ =0.7
batch	10	0.52	0.46	0.41	0.36
batch	30	0.52	0.47	0.41	0.36
batch	50	0.51	0.46	0.41	0.35
stream	10	0.87	0.71	0.54	0.38
stream	30	0.88	0.72	0.55	0.39
stream	50	0.88	0.71	0.55	0.39
HDDDM	10	0.99	0.80	0.60	0.40
HDDDM	30	0.99	0.80	0.60	0.40
HDDDM	50	0.99	0.80	0.60	0.40
OTDD	10	0.99	0.88	0.77	0.65
OTDD	30	0.85	0.75	0.65	0.55
OTDD	50	0.99	0.88	0.77	0.66

Table 3: Average TC values of four methods used with logistic regression classifier for different values of λ on SYN dataset. The TC values are averaged over SYN datasets with same value of P.

In Table 4, we report the average TC values of the four algorithms for different values of M. The average TC values are obtained by averaging TC values for SYN datasets with common value of M parameter. This study is important as in most real-life situations the magnitude of the concept drift would not be known in advance and hence an algorithm that performs generally well for different magnitudes of concept drift would be desirable. We notice that the drift aware algorithms (*stream, HDDDM, OTDD*) perform better than *batch* for all values of λ due to their ability to adapt to changing characteristics of data. As increasing weightage is given to labeling requirements by decreasing λ from 1 to 0.7, *OTDD* outperforms the competing algorithms. It is also worthwhile to notice that the performance gains of

OTDD become more pronounced for higher values of magnitude of drift. This may possibly be due to the fact that it becomes easier for a drift detection algorithm to identify a drift accurately when the drift magnitude is high.

Method	M	λ =1	λ =0.9	λ =0.8	λ =0.7
batch	30	0.52	0.46	0.41	0.36
batch	40	0.52	0.46	0.41	0.36
batch	50	0.52	0.46	0.41	0.36
stream	30	0.87	0.71	0.54	0.38
stream	40	0.88	0.72	0.55	0.39
stream	50	0.88	0.72	0.55	0.39
HDDM	30	0.99	0.80	0.60	0.40
HDDM	40	0.99	0.80	0.60	0.40
HDDM	50	0.99	0.80	0.60	0.40
OTDD	30	0.92	0.81	0.71	0.60
OTDD	40	0.92	0.82	0.71	0.61
OTDD	50	0.99	0.88	0.77	0.66

Table 4: Average TC values of four methods used with logistic regression classifier for different values of λ on SYN dataset. The TC values are averaged over SYN datasets with same value of M.

6.1.2. HYP

As discussed in section 5.1.2, the HYP dataset contains gradual drift. When $\lambda = 1$, we compare the methods on accuracy alone where *stream* and *HDDDM* outperform *OTDD* (see Table 5). This is expected as both *stream* and *HDDDM* require labelled data for every window while *OTDD* demands it only for windows where drift is detected. As we move from $\lambda = 1$ to $\lambda = 0.7$, increasingly more weightage is given to labeling, and the superior performance of the *OTDD* gets more pronounced. Similar conclusions about performance of OTDD can be drawn for HYP5 (see Figure 9) and HYP8 dataset .

Figure 8 shows the variation of TC for the OTDD algorithm on varying λ and window size for the three HYP datasets. Among the three datasets, HYP3 has the least number of dimensions drifting and a relatively high magnitude of drift. On the other hand, HYP5 and HYP8 have more number of dimensions drifting with a lesser magnitude of drift (see section 5.1.2 for details). Among the three datasets, it is likely that the concept drift is relatively easier to detect in HYP3 as compared to HYP5 and HYP8. This intuition is substantiated by the performance gains of OTDD on HYP3 (Figure 8), which are higher than those on HYP5 and HYP8. This is further supported by the fact that *stream* and *HDDDM* also have superior performance on HYP3.

It may be noted that the *stream*, *HDDDM* and *OTDD* methods can be applied with different probabilistic classi-

fiers. Hence, it is worthwhile to investigate the performance of these methods with different classifiers. Towards this objective, we consider the performance of these methods with the logistic regression and decision tree classifiers for the HYP3 dataset. We observe that the OTDD algorithm outperforms the other methods on the TC criterion for both the classifiers (see Tables 5 and 6).

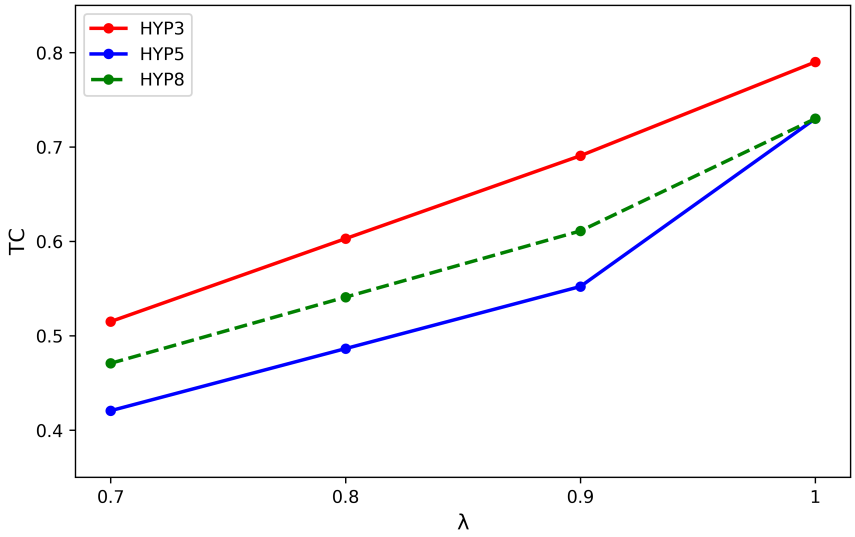


Figure 8: Variation of TC with λ for all HYP Datasets for the OTDD Algorithm.

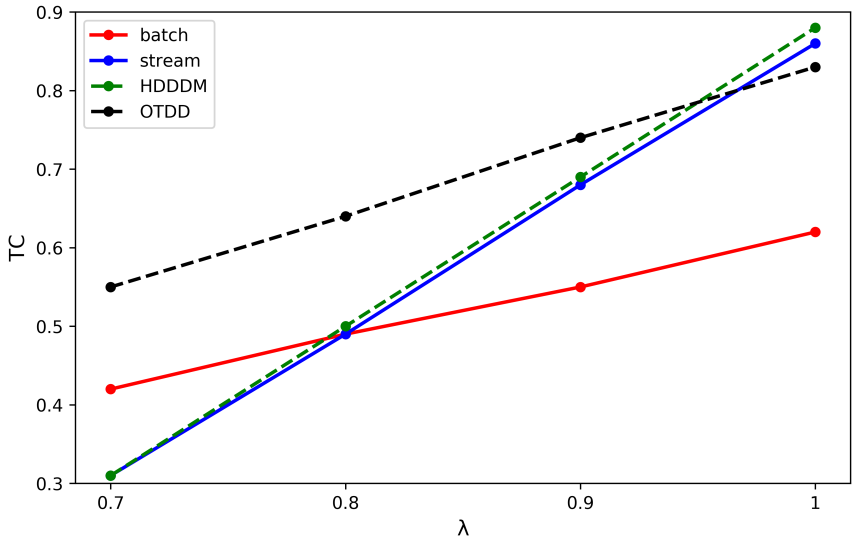


Figure 9: Variation of TC with λ for HYP5 Dataset. Logistic Regression classifier with window size 100 is considered.

Based on the results of SYN and HYP datasets, we conclude that the OTDD algorithm performs better than the competing algorithms in presence of concept drift, irrespective of whether it is abrupt or gradual in nature.

Method	*Window Size*	λ =1	λ =0.9	λ =0.8	λ =0.7
batch	50	0.67	0.60	0.53	0.47
stream	50	0.76	0.59	0.41	0.24
HDDDM	50	0.78	0.60	0.42	0.25
OTDD	50	0.76	0.67	0.59	0.50
batch	100	0.61	0.55	0.49	0.42
stream	100	0.77	0.59	0.42	0.24
HDDDM	100	0.78	0.60	0.42	0.25
OTDD	100	0.73	0.65	0.57	0.49
batch	200	0.67	0.60	0.53	0.46
stream	200	0.79	0.61	0.44	0.26
HDDDM	200	0.79	0.61	0.43	0.25
OTDD	200	0.78	0.69	0.60	0.52

Table 5: TC values of four methods used with decision tree classifier for different values of λ on HYP3 dataset.

Method	*Window Size*	λ =1	λ =0.9	λ =0.8	λ =0.7
batch	50	0.66	0.46	0.53	0.60
stream	50	0.84	0.29	0.47	0.66
HDDDM	50	0.88	0.31	0.50	0.69
OTDD	50	0.82	0.56	0.64	0.73
batch	100	0.69	0.48	0.55	0.62
stream	100	0.88	0.32	0.51	0.69
HDDDM	100	0.75	0.31	0.50	0.69
OTDD	100	0.88	0.55	0.64	0.74
batch	200	0.71	0.49	0.57	0.64
stream	200	0.90	0.33	0.52	0.71
HDDDM	200	0.88	0.32	0.51	0.69
OTDD	200	0.71	0.49	0.57	0.64

Table 6: TC values of four methods used with logistic regression classifier for different values of λ on HYP3 dataset.

6.2. Real-Life Datasets

In this section, we report the performance of the four competing algorithms in terms of their TC values for the two real-life datasets, TMD and StayAlert. For both datasets, we find that *OTDD* generally performs better than the other methods, particularly when labeling requirements are an important concern.

6.2.1. TMD

Since TMD is a real-life dataset, we first examine if concept drift is present in the dataset. One possible way of identifying the presence of concept drift is to observe a sharp difference in predictive accuracy of *batch* and *stream* paradigms after a point in time. When no concept drift is present in the data stream, we can expect similar predictive performance of *batch* and *stream* methods on successive data windows. In Figure 10, we track the performance of decision tree classifier over successive data windows for a window size of 300. We observe that *stream* performs better than *batch* for all windows, suggesting the presence of concept drift. This is further substantiated in Figure 11 where *stream* outperforms *batch* for different window sizes and classifiers.

From Tables 7 and 8, *OTDD* is seen to generally perform better than the competing methods across window sizes and the values of λ. Since, in real life situations we would not know the optimal window size in advance it would be desirable to have a method that performs well for different choices of the window size. By examining Tables 7 and 8, we also note that the performance of *OTDD* and the other methods are better for decision tree classifier than the logistic regression classifier.

Method	*Window Size*	λ =1	λ =0.9	λ =0.8	λ =0.7
batch	300	0.5	0.44	0.39	0.33
stream	300	0.56	0.41	0.26	0.11
HDDDM	300	0.70	0.53	0.36	0.19
OTDD	300	0.64	0.55	0.46	0.37
batch	400	0.41	0.36	0.31	0.26
stream	400	0.56	0.41	0.26	0.11
HDDDM	400	0.67	0.52	0.35	0.18
OTDD	400	0.64	0.54	0.45	0.36
batch	500	0.47	0.41	0.36	0.30
stream	500	0.60	0.44	0.30	0.15
HDDDM	500	0.67	0.50	0.34	0.17
OTDD	500	0.67	0.54	0.43	0.32

Table 7: TC values of four methods used with decision tree classifier for different values of λ on TMD dataset.

Method	*Window Size*	λ =1	λ =0.9	λ =0.8	λ =0.7
batch	300	0.38	0.34	0.29	0.25
stream	300	0.54	0.39	0.24	0.09
HDDDM	300	0.60	0.43	0.27	0.11
OTDD	300	0.47	0.39	0.32	0.25
batch	400	0.39	0.34	0.29	0.25
stream	400	0.54	0.39	0.24	0.10
HDDDM	400	0.59	0.43	0.27	0.11
OTDD	400	0.47	0.40	0.33	0.26
batch	500	0.40	0.35	0.30	0.25
stream	500	0.53	0.39	0.24	0.10
HDDDM	500	0.59	0.43	0.27	0.11
OTDD	500	0.46	0.39	0.31	0.24

Table 8: TC values of four methods used with logistic regression classifier for different values of λ on TMD dataset.

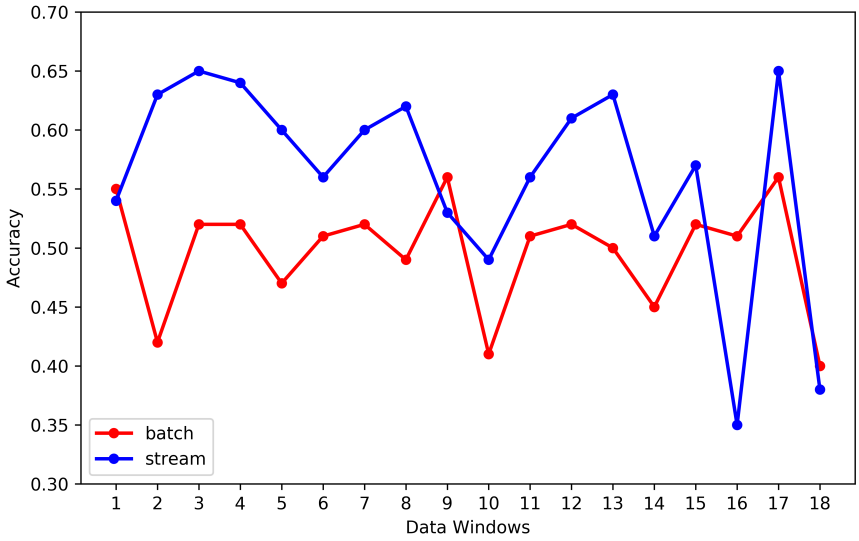


Figure 10: Accuracy of *stream* and *batch* for successive data windows of TMD dataset using decision tree classifier with window size of 300.

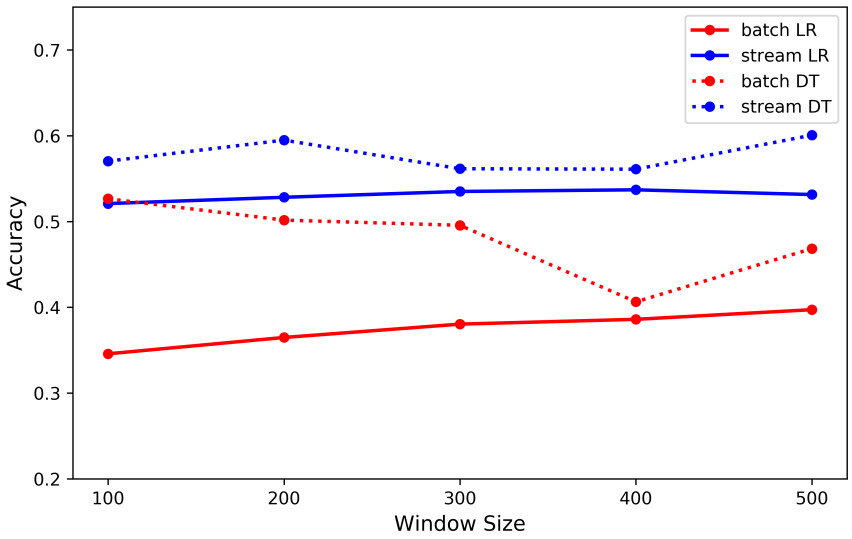


Figure 11: Accuracy of *stream* and *batch* for decision tree and logistic regression classifiers for different window sizes on TMD dataset.

6.2.2. StayAlert

The StayAlert dataset being a real-life dataset it is not clear to us apriori whether there is concept drift present in this dataset. Thus, as in the case of TMD dataset(section 6.2.1), we first examine the StayAlert dataset for the presence of concept drift by tracking the performance of *batch* and *stream* over successive data windows using a decision tree classifier and window size of 1000. We observe in Figure 12 that stream outperforms *batch* by a substantial margin for a large proportion of data windows, indicating the presence of concept drift in this dataset.

In Table 9, we examine the performance of *batch*, *stream*, *HDDDM* and *OTDD* in terms of TC for window sizes 1000, 3000 and 10,000. For λ=1, *HDDDM* and *OTDD* perform better than both *batch* and *stream*. As we place more weightage on labelling requirements by decreasing the value of λ, *OTDD* emerges as the method of choice for window sizes 1000 and 3000.

Window size is one of the parameters that influence the ability of the algorithm to detect drifts. If monitoring statistics are computed over very wide windows, the effects of drift might get averaged out, leading to missed detection. Moreover, for the *OTDD* algorithm, a large window size would impose significantly larger cost of acquiring labels. On the other hand, if the window size is too small, it leads to excessive computation at each set of newly arriving observations. This is particularly true for the *HDDDM* algorithm. In this context, we observe in table 9 that for the window size of 10000, and $\lambda = 0.7$ and 0.8 the TC values of all the drift aware algorithms are lesser than the *batch*.

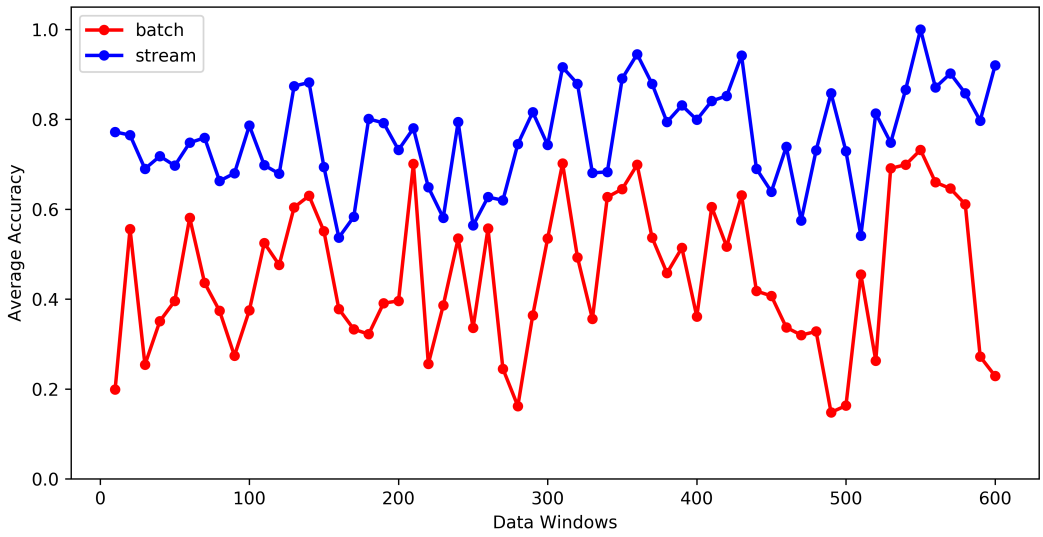


Figure 12: Accuracy of *stream* and *batch* for decision tree and classifier with window size=1000 on StayAlert dataset. Each data point represents average accuracy over 10 successive windows.

Method	Window Size	λ =1	λ =0.9	λ =0.8	λ =0.7
batch	1000	0.45	0.40	0.36	0.34
stream	1000	0.76	0.58	0.40	0.23
HDDDM	1000	0.77	0.59	0.42	0.24
OTDD	1000	0.80	0.66	0.52	0.38
batch	3000	0.49	0.44	0.39	0.34
stream	3000	0.68	0.52	0.35	0.18
HDDDM	3000	0.74	0.57	0.39	0.22
OTDD	3000	0.75	0.61	0.48	0.34
batch	10000	0.57	0.51	0.46	0.33
stream	10000	0.66	0.50	0.33	0.16
HDDDM	10000	0.68	0.52	0.35	0.20
OTDD	10000	0.60	0.50	0.41	0.30

Table 9: TC values of four methods used with decision tree classifier for different values of λ on StayAlert dataset.

7. Discussion

In this paper, we propose a new algorithm, the OTDD algorithm, which is suitable for detecting drifts in environments with scarcity of labels. We compare the performance of this algorithm with some widely used algorithms and demonstrate its superior performance on two synthetic datasets. The real life performance of the OTDD algorithm is analyzed by applying it to two datasets from the transportation domain. The OTDD algorithm is found to provide better results compared to the other algorithms on both these datasets. The performance of OTDD algorithm bears potential of use in other domains where streaming data is pervasive such as non-invasive medical monitoring, manufacturing control and human activity recognition. The low labeling requirements of the algorithm is an attraction to its wide-spread use. In view of this, we intend to investigate in future, if active learning strategies can be used to further reduce the labeling requirements through identification of the most 'informative' instances for which to acquire labels for. Another interesting possibility is to devise a strategy for dynamic window sizing that may lead to quicker detection of concept drift. We plan to explore these directions in a future work.

References

Abou-Moustafa, K. T., & Ferrie, F. P. (2012). A note on metric properties for some divergence measures: The gaussian case. In *Asian Conference on Machine Learning* (pp. 1–15). PMLR.

Aggarwal, C. C. (2007). *Data streams: Models and Algorithms* volume 31. Springer Science & Business Media.

Aggarwal, C. C., & Philip, S. Y. (2007). A survey of synopsis construction in data streams. In *Data Streams: Models and Algorithms* (pp. 169–207). Springer.

Baena-Garcıa, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavalda, R., & Morales-Bueno, R. (2006). Early drift detection method. In *Fourth International Workshop on Knowledge Discovery from Data Streams* (pp. 77–86). volume 6.

Bahri, M., Bifet, A., Gama, J., Gomes, H. M., & Maniu, S. (2021). Data stream analysis: Foundations, major tasks and tools. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, (p. e1405).

Bajwa, R., Rajagopal, R., Varaiya, P., & Kavaler, R. (2011). In-pavement wireless sensor network for vehicle classification. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks* (pp. 85–96). IEEE.

Blincoe, L., Miller, T. R., Zaloshnja, E., & Lawrence, B. A. (2015). *The Economic and Societal Impact of Motor Vehicle Crashes, 2010 (Revised)*. Technical Report United States. National Highway Traffic Safety Administration.

CDOT (2013). *2010-2012 California Household Travel Survey Final Report*. Technical Report.

Cevher, V., Chellappa, R., & McClellan, J. H. (2008). Vehicle speed estimation using acoustic wave patterns. *IEEE Transactions on Signal Processing*, *57*, 30–47.

Chen, S., Wang, W., & Van Zuylen, H. (2009). Construct support vector machine ensemble to detect traffic incident. *Expert systems with applications*, *36*, 10976–10986.

Dabiri, S., & Heaslip, K. (2019). Developing a twitter-based traffic event detection model using deep learning architectures. *Expert systems with applications*, *118*, 425–439.

D'Andrea, E., & Marcelloni, F. (2017). Detection of traffic congestion and incidents from gps trace analysis. *Expert Systems with Applications*, *73*, 43–56.

Deza, M.-M., & Deza, E. (2006). *Dictionary of Distances*. Elsevier.

Ditzler, G., & Polikar, R. (2011). Hellinger distance based drift detection for nonstationary environments. In *2011 IEEE symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE)* (pp. 41–48). IEEE.

Düh, J., Hufnagl, H., Juritsch, E., Pfliegl, R., Schimany, H.-K., & Schönegger, H. (2010). *Data and Mobility: Transforming Information Into Intelligent Traffic and Transportation Services. Proceedings of the Lakeside Conference 2010* volume 81. Springer Science & Business Media.

Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. In *Brazilian Symposium on Artificial Intelligence* (pp. 286–295). Springer.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, *46*, 1–37.

Ghofrani, F., He, Q., Goverde, R. M., & Liu, X. (2018). Recent applications of big data analytics in railway transportation systems: A survey. *Transportation Research Part C: Emerging Technologies*, *90*, 226–246.

Han, J., Kamber, M., & Pei, J. (2011). Data mining concepts and techniques third edition. *The Morgan Kaufmann Series in Data Management Systems*, *5*, 83–124.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data mining, Inference, and Prediction*. Springer Science & Business Media.

Hinkley, D. V. (1971). Inference about the change-point from cumulative sum tests. *Biometrika*, *58*, 509–523.

Hou, Y., Edara, P., & Sun, C. (2015). Situation assessment and decision making for lane change assistance using ensemble learning methods. *Expert Systems with Applications*, *42*, 3875–3882.

Huang, H., Cheng, Y., & Weibel, R. (2019). Transport mode detection based on mobile phone network data: A systematic review. *Transportation Research Part C: Emerging Technologies*, *101*, 297–312.

Hulten, G., Spencer, L., & Domingos, P. (2001). Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 97–106).

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning* volume 112. Springer.

Kanarachos, S., Mathew, J., & Fitzpatrick, M. E. (2019). Instantaneous vehicle fuel consumption estimation using smartphones and recurrent neural networks. *Expert Systems with Applications*, *120*, 436–447.

Khamassi, I., Sayed-Mouchaweh, M., Hammami, M., & Ghédira, K. (2018). Discussion and review on evolving data streams and concept drift adapting. *Evolving systems*, *9*, 1–23.

Klinkenberg, R., & Joachims, T. (2000). Detecting concept drift with support vector machines. In *ICML* (pp. 487–494).

Kolouri, S., Park, S. R., Thorpe, M., Slepcev, D., & Rohde, G. K. (2017). Optimal mass transport: Signal processing and machine-learning applications. *IEEE Signal Processing Magazine*, *34*, 43–59.

Krempl, G., Žliobaite, I., Brzeziński, D., Hüllermeier, E., Last, M., Lemaire, V., Noack, T., Shaker, A., Sievi, S., Spiliopoulou, M. et al. (2014). Open challenges for data stream mining research. *ACM SIGKDD explorations newsletter*, *16*, 1–10.

Kumar, U. D. (2017). *Business Analytics: The Science of Data-Driven Decision Making*. Wiley.

Kuncheva, L. I. (2009). Using control charts for detecting concept change in streaming data. *Bangor University*, (p. 48).

Laha, A. K., & Putatunda, S. (2018). Real time location prediction with taxi-gps data streams. *Transportation Research Part C: Emerging Technologies*, *92*, 298–322.

Lara, O. D., & Labrador, M. A. (2012). A survey on human activity recognition using wearable sensors. *IEEE Communications Surveys & Tutorials*, *15*, 1192–1209.

Larose, D. T. (2015). *Data Mining and Predictive Analytics*. John Wiley & Sons.

Lee, G., Mallipeddi, R., & Lee, M. (2017). Trajectory-based vehicle tracking at low frame rates. *Expert Systems with Applications*, *80*, 46–57.

Liao, B., Zhang, J., Cai, M., Tang, S., Gao, Y., Wu, C., Yang, S., Zhu, W., Guo, Y., & Wu, F. (2018). Dest-resnet: A deep spatiotemporal residual network for hotspot traffic speed prediction. In *Proceedings of the 26th ACM international conference on Multimedia* (pp. 1883–1891).

Liu, F., Janssens, D., Cui, J., Wang, Y., Wets, G., & Cools, M. (2014). Building a validation measure for activity-based transportation models based on mobile phone data. *Expert Systems with Applications*, *41*, 6174–6189.

Markou, I., Kaiser, K., & Pereira, F. C. (2019). Predicting taxi demand hotspots using automated internet search queries. *Transportation Research Part C: Emerging Technologies*, *102*, 73–86.

Marz, N., & Warren, J. (2013). *Big Data: Principles and best practices of scalable real-time data systems*. Manning.

McFadden, D. (1974). The measurement of urban travel demand. *Journal of Public Economics*, *3*, 303–328.

Mitchell, T. (1997). Machine learning. macgraw-hill companies. *Inc., Boston*, .

Moreira-Matias, L., Cats, O., Gama, J., Mendes-Moreira, J., & De Sousa, J. F. (2016). An online learning approach to eliminate bus bunching in real-time. *Applied Soft Computing*, *47*, 460–482.

Moreira-Matias, L., Gama, J., Ferreira, M., Mendes-Moreira, J., & Damas, L. (2013). Predicting taxi-passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems*, *14*, 1393–1402.

Nabian, M. A., Alemazkoor, N., & Meidani, H. (2019). Predicting near-term train schedule performance and delay using bi-level random forests. *Transportation Research Record*, *2673*, 564–573.

Nair, R., Hoang, T. L., Laumanns, M., Chen, B., Cogill, R., Szabó, J., & Walter, T. (2019). An ensemble prediction model for train delays. *Transportation Research Part C: Emerging Technologies*, *104*, 196–209.

NCRB (2019). *Accident Deaths Suicides in India 2019*. Technical Report. URL: https://ncrb.gov.in/en/accidental-deaths-suicides-india-2019 accessed: 2021-04-11.

Nitsche, P., Widhalm, P., Breuss, S., Brändle, N., & Maurer, P. (2014). Supporting large-scale travel surveys with smartphones–a practical approach. *Transportation Research Part C: Emerging Technologies*, *43*, 212–221.

NREL (2019). Transportation secure data center. URL: https://www.nrel.gov/transportation/secure-transportation-data/tsdc-cleansed-data.html accessed: 2021-04-11.

Oentaryo, R., Lim, E.-P., Finegold, M., Lo, D., Zhu, F., Phua, C., Cheu, E.-Y., Yap, G.-E., Sim, K., Nguyen, M. N. et al. (2014). Detecting click fraud in online advertising: A data mining approach. *The Journal of Machine Learning Research*, *15*, 99–140.

O'Leary, D. E. (2013). Artificial intelligence and big data. *IEEE Intelligent Systems*, *28*, 96–99.

Page, E. S. (1954). Continuous inspection schemes. *Biometrika*, *41*, 100–115.

Pickrell, T. M. et al. (2015). *Driver Electronic Device Use in 2013*. Technical Report United States. National Highway Traffic Safety Administration.

Richardson, A., Ampt, E., & Meyburg, A. (1996). Nonresponse issues in household travel surveys. In *Conference proceedings* (pp. 79–114). volume 10.

Sebastiao, R., Silva, M. M., Gama, J., & Mendonça, T. (2011). Contributions to an advisory system for changes detection in depth of anesthesia signals. *Learning from Medical Data Streams*, (p. 1).

Stopher, P. R., Kockelman, K., Greaves, S. P., & Clifford, E. (2008). Reducing burden and sample sizes in multiday household travel surveys. *Transportation Research Record*, *2064*, 12–18.

Sussman, J. S. (2008). *Perspectives on Intelligent Transportation Systems (ITS)*. Springer Science & Business Media.

Taylor, B., & Fink, C. (2013). Explaining transit ridership: What has the evidence shown? *Transportation Letters*, *5*, 15–26.

Wang, Y., Zhang, D., Liu, Y., Dai, B., & Lee, L. H. (2019). Enhancing transportation systems via deep learning: A survey. *Transportation Research Part C: Emerging Technologies*, *99*, 144–163.

Žliobaite, I. (2010). Change with delayed labeling: When is it detectable? In *2010 IEEE International Conference on Data Mining Workshops* (pp. 843–850). IEEE.

Žliobaitė, I., Pechenizkiy, M., & Gama, J. (2016). An overview of concept drift applications. In *Big Data Analysis: New Algorithms for a New Society* (pp. 91–114). Springer.