Hindawi Publishing Corporation
Applied Computational Intelligence and Soft Computing
Volume 2016, Article ID 1465810, 13 pages
http://dx.doi.org/10.1155/2016/1465810

Research Article

Online Incremental Learning for High Bandwidth Network Traffic Classification

H. R. Loo, S. B. Joseph, and M. N. Marsono

Faculty of Electrical Engineering, Universiti Teknologi Malaysia, 81310 Johor Bahru, Johor, Malaysia

Correspondence should be addressed to M. N. Marsono; nadzir@fke.utm.my

Received 31 October 2015; Revised 27 January 2016; Accepted 31 January 2016

Academic Editor: Jun He

Data stream mining techniques are able to classify evolving data streams such as network traffic in the presence of concept drift. In order to classify high bandwidth network traffic in real-time, data stream mining classifiers need to be implemented on reconfigurable high throughput platform, such as Field Programmable Gate Array (FPGA). This paper proposes an algorithm for online network traffic classification based on the concept of incremental k-means clustering to continuously learn from both labeled and unlabeled flow instances. Two distance measures for incremental k-means (Euclidean and Manhattan) distance are analyzed to measure their impact on the network traffic classification in the presence of concept drift. The experimental results on real datasets show that the proposed algorithm exhibits consistency, up to 94% average accuracy for both distance measures, even in the presence of concept drifts. The proposed incremental k-means classification using Manhattan distance can classify network traffic 3 times faster than Euclidean distance at 671 thousands flow instances per second.

1. Introduction

Network traffic classification is a critical network processing task for network management. Traffic measurement and classification enable network administrators to understand the current network state and reconfigure the network such that the observed network state can be improved over time. The complexity and dynamic characteristic of today's network traffic have necessitated the need for traffic classification techniques that are able to adapt to new concepts. This includes the ability to classify types of traffic almost instantaneously to avoid outdating the knowledge gained from the learning of new concepts.

Data stream mining algorithms [1–6] have been introduced to overcome the shortcoming of conventional data mining algorithms. They are designed to handle concept drift, to forget old irrelevant data, and to adapt to new knowledge. References [7–9] have proposed the use of data stream mining algorithms for traffic classification such as Very Fast Decision Tree [3] and Concept-Adaptive Very Fast Decision Tree [4]. Reference [10] proposed a new algorithm named Concept-Adaptive Rough Set based Decision Tree (CRSDT) to classify network traffic. These algorithms have successfully demonstrated the ability of data stream mining to handle dynamic and fast changing network data streams with sustained accuracy. However, the decision tree based implementation requires intensive training process and causes high memory consumption for model building [11].

References [2, 12] proposed the use of incremental clustering for data stream classification. Although both works show high classification accuracy for evolving data stream, the processing rate of such algorithms is low. One of the reasons is due to the use of Euclidean distance in both works as the distance metric. Euclidean distance computation that requires multiple square and square root functions contributes to high overhead and limited speed. Another distance measure is the Manhattan distance that does not require heavy multiplications [13], which can be efficiently implemented on reconfigurable hardware such as Field Programmable Gate Array (FPGA). Unlike batch data mining, conversion of distance metric from Euclidean distance to Manhattan distance in k-means incremental learning cannot be directly applied. Certain modifications on the incremental k-means algorithm need to be done. In incremental clustering

algorithms, clusters information is normally stored as a summary of points and the raw data are discarded after training. Radius recomputation in Euclidean distance can be totally based on the summary of points, but raw data are needed in Manhattan distance. Thus some modification on the summary of points is needed.

This paper analyzes online incremental k-means network traffic classification in [16] with two distance measures (Euclidean and Manhattan). The proposed method using Manhattan distance requires less computation than Euclidean distance and, hence, achieves higher running speed. This algorithm has been verified with real network traces to evaluate the accuracy when using these distance measures and possible implementation trade-off for high throughput network traffic classification. The rest of the paper is organized as follows. The incremental k-means proposed method based on Euclidean distance is described in Section 3. Modification of the proposed method to suit Manhattan distance is discussed in Section 4. The experimental setup and results are discussed in Section 5. Section 6 concludes the paper and states suggestions for future works.

2. Related Works

The simplicity of clustering algorithm such as k-means makes it well adapted for network traffic classification. References [17–19] proposed k-means algorithm implementations for classifying network traffic with high accuracy. Reference [20] proposed the use of feature selection method with k-means algorithm to enhance the classification accuracy. Reference [11] proposed a new initialization method for centroid selection in k-means to further improve the classification accuracy of network traffic, whereas [21] proposed enhancement to k-means algorithm to prevent the diverse impact of attributes on clustering output. The aforementioned works successfully show the suitability of clustering based algorithms in network traffic classification, although they could not adapt to changes in network concept in today's network traffic.

References [22–27] proposed incremental clustering algorithm which could adapt to new knowledge over time. Reference [22] proposed the microclustering concept where only the summary of clusters is kept throughout the learning process. The proposed algorithm can learn new concept incrementally by updating the clusters summary. Reference [23] proposed adapting the microclustering concept originally proposed in [22] and proposed macroclustering stage with pyramidal time frame. Not all microclusters are saved that reduce overall memory consumption. Reference [24] proposed the adaptation of microclustering and macroclustering concepts from [22, 23] and is customized for trajectory data. Although the method in [24] demonstrates the ability to update classification model incrementally, it does not support streaming data.

Reference [25] proposed graph based incremental clustering although it is not suitable for online network classification due to its long processing rate and large memory consumption. Reference [26] proposed incremental DBSCAN for data warehousing. The proposed algorithm is based on density-based clustering. The radius of cluster and minimum number of points in clusters are assumed to be fixed. Reference [27] proposed incremental clustering based real-time anomaly detection. Incremental training is initiated based on the false alarm threshold that requires continuous feedback from the network administrator. Incremental clustering can continuously learn new knowledge and reduce misclassification caused by outdated knowledge. For online real-time network traffic classification, the processing rate of software implementation of such algorithms is not sufficient to support current network speed. Implementation of such algorithms in reconfigurable hardware such as Field Programmable Gate Array (FPGA) can accelerate the processing rate.

To the best of our knowledge, only [28] has proposed the implementation of incremental clustering algorithm in FPGA for multimedia traffic classification. The proposed method uses Hamming distance instead of Euclidean distance for the distance measurement. It uses an extra bit appended to the data to indicate training or testing dataset and incrementally updates the model when training bit is detected. However, the proposed method requires large training set ($10k$ instance) to achieve high accuracy. On the other hand, implementations of nonincremental k-means algorithm in FPGA are more common, for example, [29–32]. However, implementation of Euclidean distance based k-means algorithm on hardware consumes high hardware resources. References [30, 31] reported that 90% of the hardware resources were required to implement k-means algorithm. Modification of such distance was proposed as in [33] using distance squared. References [13, 34–36] proposed the use of Manhattan distance as a distance measure for k-means. The implementation not only can reduce the hardware cost, it also can be fully configurable and easily pipelined to support high degree of parallelism. Hence, towards the implementation of hardware acceleration of incremental online network traffic classifier, the proposed incremental k-means classifier based on Manhattan distance is a better option than using Euclidean distance.

3. Online Incremental k-Means Algorithm

We proposed online incremental k-means clustering in [16] for online network traffic classification. It consists of two main processes: classification and learning. Some of the terms and terminology used in this paper are as follows:

(1) *Flow*: the network traffic that belongs to a process-to-process communication.

(2) *Instance*: a tuple of attributes.

(3) *Flow features*: the attributes or statistical features that are extracted from a flow, for example, number of bytes in payload.

(4) *Flow instance*: the instance made up of flow features which represent a flow.

The classification process performs online classification on flow instances, while the learning process simultaneously performs incremental learning to update the classification model. Both processes will be discussed in detail in Sections

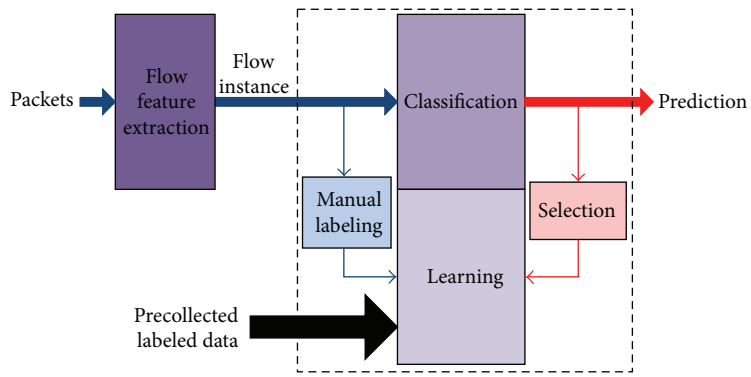

Figure 1: Overview of the proposed method.

3.2 and 3.4, respectively. Figure 1 shows the overview of the proposed method. The selection module performs the selection of flow instances to be learned to avoid mislearning (see Section 3.3). However, manual labeling is not covered in this paper. An example of such technique is the groundtruth method which is discussed in [15].

3.1. Classification Model Initialization. Before online classification can be performed, the classification model needs to be initialized. This process is performed once during start-up to prepare the *base* classifier model. In this stage, the supervised k-means technique is used to cluster the batch labeled flow instances into k initial clusters. In order to increase classification efficiency, the classification model M is made up of B smaller micromodels that are located in different location in the Euclidean space. To perform this, precollected flow instances are distributed to B chunks according to the distance to origin D_o, such that

$$D_{o_i} = \sqrt{\sum_{m=0}^{d} \left(x_i^m\right)^2}, \tag{1}$$

where x_i is a flow instance with d flow features. Each micromodel is then built using respective chunk of flow instances.

The initializations of centroids are based on the method suggested in [12] where the initial class of clusters are assumed to be proportional to the data distribution. The clusters are then compressed to sufficient statistics known as clustering features (CF). CF is a 3-tuple information that summarizes information about a cluster as proposed in [22]. Given N_j d-dimensional data points $(\vec{x_i})$ in a cluster j where $i = 1, 2, \ldots, N_j$, the CF of a cluster is defined as three-tupple: $\mathrm{CF}_j = \langle N_j, \overrightarrow{\mathrm{LS}_j}, \overrightarrow{\mathrm{SS}_j} \rangle$, where

$$\begin{aligned} \overrightarrow{\mathrm{LS}} &= \sum_{i=1}^{N} \vec{x_i}, \\ \overrightarrow{\mathrm{SS}} &= \sum_{i=1}^{N} \left(\vec{x_i}\right)^2. \end{aligned} \tag{2}$$

(1) x_i: Pre-collected flow instances (array of features)
(2) y_i: True labels for x_i
(3) D_o: Distance of x_i to origin
(4) B: Number of micro-model
(5) N: Number of Pre-collected flow instances
(6) B_k: Number of cluster in a micro-model
(7) **for** x_i in range of $(0, N)$ **do**
(8) Calculate $D_o(x_i)$
(9) **end for**
(10) Quicksort($D_o[N]$)
(11) Distribute x_i into B micro-model
(12) **for** each micro-model **do**
(13) Generate B_k cluster
(14) Summarize B_k cluster into Clustering Feature, CF
(15) Store clusters in time-series and set timestamp to 0
(16) **end for**

Algorithm 1: Classification model initialization.

In our classification model, we modify and add the timestamp T_j and class information y_j to represent each cluster; $\mathrm{CF}_j = \langle N_j, \overrightarrow{\mathrm{LS}_j}, \overrightarrow{\mathrm{SS}_j}, T_j, y_j \rangle$. Merging new $(\vec{x_i})$ to cluster j is based on the Additivity Theorem [22], where T_j and y_j are unchanged, such that

$$\mathrm{CF}_j = \left\langle \left(N_j + 1\right), \left(\overrightarrow{\mathrm{LS}_j} + \vec{x_i}\right), \left(\overrightarrow{\mathrm{SS}_j} + \left(\vec{x_i}\right)^2\right), T_j, y_j \right\rangle. \tag{3}$$

Raw data are discarded in order to save memory space. The k clusters that are represented by k clustering features are used for classification and the clusters may be modified based on newly received data.

Algorithm 1 shows the overall steps for classification model initialization. During model initialization, the precollected flow instances are divided into B sets, and the supervised k-means method is used to create a micromodel for each set of flow instances. Created clusters are summarized as clustering features, CF with timestamp 0.

3.2. Online Classification. Classification starts upon receiving an incoming flow instance $(\vec{x_i})$. The distance to origin $D_o(x_i)$ is calculated to find the respective micromodel. In

Table 1: Prediction confidence level.

Confidence level	Conflicting neighbour	In-boundary
L_0	Yes	—
L_1	No	No
L_2	No	Yes

the micromodel, the distance between cluster's centroid and $\overrightarrow{x_i}$ is computed using (4). The nearest cluster (C_1) and second nearest cluster (C_2) are then determined. Assuming that the real class label of a flow instance is unknown (unlabeled flow instance), the predicted class $y'(x_i)$ will be classified to class of C_1 with respect to $\overrightarrow{x_i}$:

$$D\left(x_i, \mu_j\right) = \sqrt{\sum_{m=0}^{d} \left(x_i^m - \mu_j^m\right)^2}, \tag{4}$$

where μ_j is the centroid of cluster j and $\mu_j = \overrightarrow{\mathrm{LS}}/N$.

3.3. Selection of Learning Instance. As self-training method is applied in the learning algorithm, all predicted labels are assumed to be accurate. While this is not entirely true in incremental learning process, certain threshold of false positive must be expected. Since learning on falsely predicted flow instances can cause false learning, only flow instances with high confidence of prediction are chosen to be learned. The selection criteria are designed such that they make use of the information from classification process so that they do not need extra computation. Extra computation will make longer model learning and incremental learning inefficient.

Confidence level is divided into three levels $\{L_0, L_1, L_2\}$ as shown in Table 1. The criteria are to determine confidence level in terms of label conflict within the two nearest clusters (conflicting neighbor) and condition when a flow instance is within the nearest cluster's boundary (in-boundary). Conflicting neighbors are set to true when both nearest clusters belong to different classes or set to false when they belong to the same class.

The boundary of the nearest cluster C_1 is determined by the cluster's average radius R defined by (5) [22]. However, for clusters with $N = 1$, (5) is not valid as the subtraction in numerator will results in zero denominator. This consideration was not analyzed in [37]. In [2], the calculation of maximum boundary is based on the nearest neighbor's maximum boundary. However, the boundary of such clusters is not always similar, and to determine the nearest neighbors will increase the system computation complexity by $O(kd)$.

We propose that the boundary of a cluster with only one flow instance can be determined by the similarity of attributes. Each attribute of a cluster centroid is compared with the attributes of incoming flow instance. An attribute is considered similar to other attributes if the ratio between them is within 10% of each other ($0.9 \geqslant x_1/x_2 \geqslant 1.1$). A parameter boundary threshold is needed to define the maximum nonsimilar attributes between flow instances and the cluster centroid that can be tolerated to include an incoming flow instance in the respective cluster. If the number of nonsimilar attributes is lower than the boundary threshold, it is considered to be within the boundary of the particular cluster:

$$R\left(\mu_j\right) = \sqrt{\frac{\sum_{m=0}^{d} \mathrm{SS}_{\mu_j}^m - \left(\sum_{m=0}^{d} \left(\mathrm{LS}_{\mu_j}^m\right)^2 / N_{\mu_j}\right)}{N_{\mu_j}}}. \tag{5}$$

The confidence level can be determined as follows: let C_1 be the nearest cluster and C_2 the second nearest cluster, y_{C_i} the class of C_i, R_{C_i} the radius of C_i, and μ_{C_i} the centroid of C_i. Confidence level is set to L_0 by default. In the case of $y_{C_1} = y_{C_2}$, confidence level will increase by one (L_1). If $x_i \in R_{C_1}$ for C_1 with more than one flow instance or $x_i \approx \mu_{C_1}$ for C_1 with only one flow instance, confidence level will increase to two (L_2) if the condition $y_{C_1} = y_{C_2}$ is satisfied.

(1) x_i': L_2 flow instance
(2) C_1: Nearest clusters from x_i
(3) **while** new x_i' **do**
(4) merge x_i' to C_1
(5) **end while**
(6) **—Labeled Instance Injection (upon receiving)—**
(7) **if** receive (x_j, y_j) **then**
(8) - Inject x_j
(9) **end if**
(10) **—Micro-model Reconstruction (after one chunk)—**
(11) N: Number of clusters
(12) k_d: Desired number of clusters
(13) **while** $N > k_d$ **do**
(14) delete clusters with timestamp 0
(15) reduce clusters timestamp by 1
(16) **end while**
(17) gather all clusters in classification model
(18) reconstruct micro-model

Algorithm 2: Proposed algorithm.

3.4. Semisupervised Incremental Learning. Only flow instances $\overrightarrow{x_i}$ with confidence level L_2 are used in incremental learning. As shown in Algorithm 2, flow instances with confidence level L_2 are merged with the nearest cluster C_1 based on (3). The learning will then update the classification model. As the input flow instances are in streams, changes in distribution and concept are expected. Outdated knowledge needs to be deleted and the micromodel needs to be reconstructed based on recent clusters. This process will be discussed in Section 3.4.2.

3.4.1. Injecting Labeled Instances. In our previous work [16], we assumed that labeling of flow instances can be done immediately after the learning process. Our extended experiment suggests that this is not possible since labeling of flow instances involves manual steps. Thus, we can only assume that some flow instances will be labeled externally and fed into the model once they are ready. Thus, the flow instances will be treated as a new flow instance and reclassified in

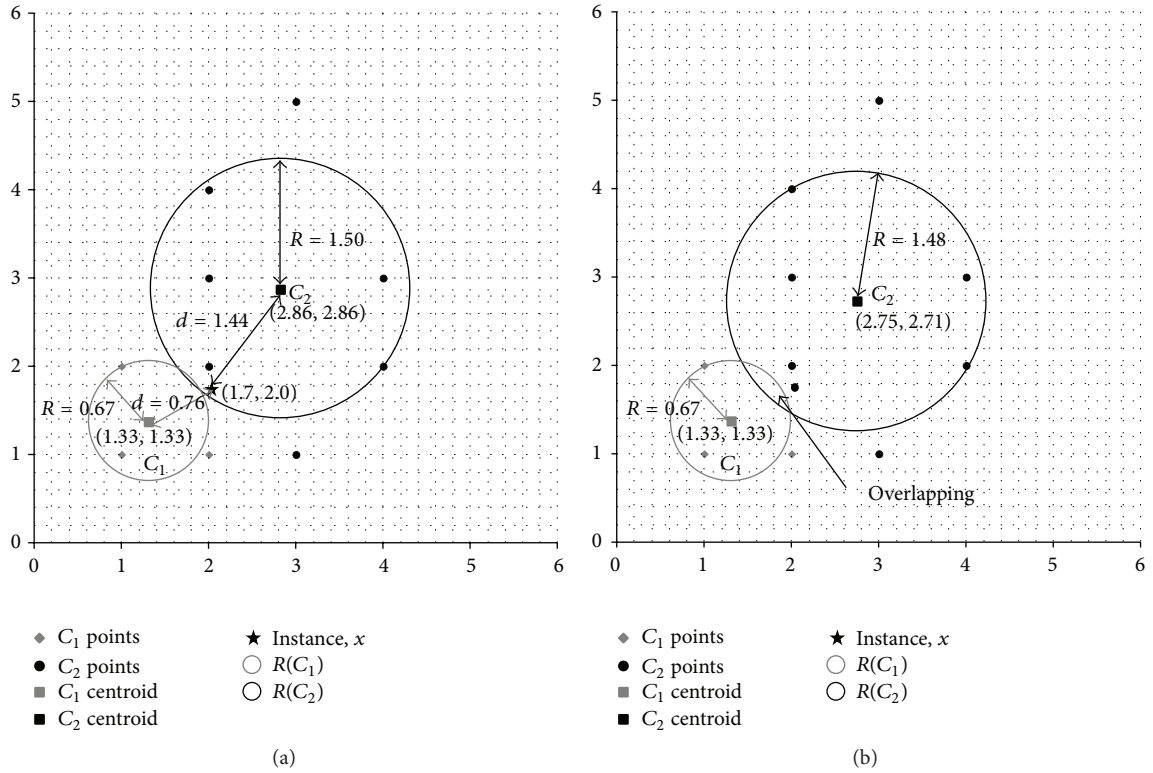

Figure 2: Example of boundary condition (a) before model merging with C_2 (b) after merging with C_2.

Table 2: Injection handling method.

Scenario	Confidence level	Prediction	Procedure
1	0	True	Merge x with C_1 if $x \in R_{C_1}$, else add new cluster for x
2	0	False	Merge x with C_2 if $x \in R_{C_2}$ and $y_x = y_{C_2}$, else add new cluster for x Delete C_1
3	1	True	If x is not in $R(C_2)$, add new cluster for x
4	1	False	Add new cluster for x
5	2	True	Do nothing
6	2	False	Erase C_1 and add new cluster for x

order to get the necessary information such as confidence level and prediction results. In order to achieve minimum effort in injection, different handling methods based on the confidence level and prediction results are proposed for handling different possible scenarios. Table 2 shows the injection handling method for several scenarios. False prediction in trained flow instances will cause the trained cluster to be deleted immediately since the cluster is no longer reliable. Besides, new cluster with x_i will be added if x_i is not in the boundary of C_1. This is true except for scenarios 2 and 3, since there is a possibility of x_i to be within the boundary of C_2. Figure 2(a) illustrates the example of boundary condition when x_i is nearer to C_1 but is in the boundary of C_2. Merging x_i to C_2 will shift boundary of C_2 towards C_1 and can result in an overlap (Figure 2(b)). Thus, when C_1 and C_2 are of different classes, one of these clusters needs to be deleted. If they belong to the same class, x_i will be ignored since its learning brings no significant changes to the model.

3.4.2. Micromodel Reconstruction. In order to prevent the storing of all outdated clusters that may result in imbalanced micromodel, a micromodel reconstruction process is performed after a user-predefined number of flow instances (chunks) have been received. Clusters that are not utilized or underutilized will be deleted as they do not contribute to the classification decision. In addition, the micromodel reduction also aims to reduce memory footprint and classification time. This reduction process includes the following steps:

(1) All clusters are structured as a time-series based on the time they were created.

(2) All clusters are given zero timestamps.

(3) When a cluster is nominated as the nearest cluster in the classification process, the timestamps is incremented by one.

(4) When a chunk is received, the timestamps of each cluster is checked from the beginning of the series. Clusters with timestamps zero will be deleted until the number of clusters, k, is reduced to a user-predefined number, k_d/B. Then, the timestamps of remaining clusters will be decremented by one. If in a situation where $k > k_d/B$ even after deleting all

zero timestamps clusters, the deleting process will be repeated for timestamps one and so on until it reaches the required number of clusters.

After the deletion of unused clusters, the remaining clusters will be repartitioned into B micromodels based on their centroid locations in the Euclidean space.

4. Analysis of Euclidean and Manhattan Distance Measures

The conversion from Euclidean distance to Manhattan distance for the incremental k-means is not direct. Modifications on the clustering features and equations are needed to suit the used distance measure.

4.1. Euclidean Distance versus Manhattan Distance. Euclidean distance and Manhattan distance are the special variant of Minkowski distance. The Minkowski distance (see (6)) is the distance of order p between two points X and Y, where $X = (x_1, x_2, \ldots, x_d)$ and $Y = (y_1, y_2, \ldots, y_d)$. Manhattan distance is Minkowski distance in first order ($p = 1$) ($L1$-Norm), while Euclidean distance is Minkowski distance in second order ($p = 2$) ($L2$-Norm):

$$D = \left(\sum_{m=1}^{d} \left| x^m - y^m \right|^p \right)^{1/p}. \tag{6}$$

4.2. Affected Elements. By changing the distance measures, the following elements in the proposed incremental k-means algorithm need to be changed as well. These include the calculation of distance to origin D_o (see (1)), distance between incoming instance and centroid (see (4)), and radius of a cluster (see (5)). The changes in R indirectly change the elements in CF. By applying $p = 1$, the new equation for D_o is as in (7), while new equation for D is as in (8). Consider the following:

$$D_o = \sum_{m=0}^{d} \left(x_i^m \right), \tag{7}$$

$$D\left(x_i, \mu_j \right) = \sum_{m=0}^{d} \left| x_i^m - \mu_j^m \right|. \tag{8}$$

For the case of R, the original equation is in (9) before it can be translated in terms of $\overrightarrow{\mathrm{LS}}, \overrightarrow{\mathrm{SS}}$,

$$R_j = \sqrt{\frac{\sum_{n=0}^{N_j} \left(\sum_{m=0}^{d} \left(x_n^m - \mu_j^m \right)^2 \right)}{N_j}}, \tag{9}$$

where N_j is the number of instances in cluster j, μ_j is the centroid of cluster j, and d is the dimension. After substituting $p = 2$ to $p = 1$, the new equation for R will be

$$R_j = \frac{\sum_{n=0}^{N_j} \left(\sum_{m=0}^{d} \left| x_n^m - \mu_j^m \right| \right)}{N_j}. \tag{10}$$

Note that (10) is not able to be represented in $\overrightarrow{\mathrm{LS}}, \overrightarrow{\mathrm{SS}}$; hence the calculation of radius is no longer possible by only keeping the cluster summary; $\mathrm{CF}_j = \langle N_j, \overrightarrow{\mathrm{LS}_j}, \overrightarrow{\mathrm{SS}_j}, T_j, y_j \rangle$. For example, let a cluster j be with 3 instances (x_1, x_2, x_3) in one dimension, $d = 1$, and centroid,

$$\mu_j = \frac{x_1 + x_2 + x_3}{N_j}. \tag{11}$$

Expanding (10) will result in

$$R_j = \frac{\left| x_1 - \mu_j \right| + \left| x_2 - \mu_j \right| + \left| x_3 - \mu_j \right|}{N_j}. \tag{12}$$

When a new instance x_4 is added into cluster j, the centroid will change to

$$\mu_j' = \frac{x_1 + x_2 + x_3 + x_4}{N_j}. \tag{13}$$

The new value of R will be

$$R' = \frac{\left| x_1 - \mu_j' \right| + \left| x_2 - \mu_j' \right| + \left| x_3 - \mu_j' \right| + \left| x_4 - \mu_j' \right|}{N_j}. \tag{14}$$

Each absolute element in (14) requires recalculation due to the changes of centroid, but each instance (x_1, x_2, x_3) is not accessible (i.e., raw data are not kept). In this case, R' could not be recalculated. In this paper, we suggest to calculate an approximation of R by storing the previous value of R in each dimension and the accumulation direction of the centroid change. New R can be calculated based on these values. Since a centroid is located in the middle of instance in the cluster and adding of new instance will only happen when it is within the radius, $(x_i \in R_{i-1})$, we can assume that the changes of $\mu_{i-1} \rightarrow \mu_i$ are as small as possible, such that $(x_{i-1} - \mu_i)$ will be in the same direction with $(x_{i-1} - \mu_{i-1})$. By taking into consideration the direction of change, we assume another instance in the cluster will have the change in the opposite direction. Thus, the changes will be canceled out. With this assumption, we can calculate the approximate of R as in (15). Consider the following:

$$R_i \approx \frac{\sum_{m=0}^{d} \left(R_{i-1}^m + U_{i-1}^m * \left| \left(x_i^m - \mu_{i-1}^m \right) / N_i \right| + \left| x_i^m - m_i^m \right| \right)}{N_n}, \tag{15}$$

$$U_i = U_{i-1} + U_{\mathrm{tempi}}, \tag{16}$$

$$U_{\mathrm{tempi}} = \begin{cases} 1; & x_i > m_i, \\ -1; & x_i < m_i, \\ 0; & x_i = m_i. \end{cases} \tag{17}$$

With these changes in the calculation of new R, $\overrightarrow{\mathrm{LS}}$ and $\overrightarrow{\mathrm{SS}}$ are no longer needed. New clustering features (CF) will be changed accordingly to $\mathrm{CF}_j = \langle N_j, \overrightarrow{R_j}, \overrightarrow{U_j}, T_j, y_j \rangle$. Merging

Table 3: Computation complexity for different distance measures.

Desc.	Euclidean	Manhattan
$D_o(x_i)$	$\sqrt{\sum_{m=0}^{d} (x_i^m)^2}$	$\sum_{m=0}^{d} (x_i^m)$
$D(x_i, \mu_j)$	$\sqrt{\sum_{m=0}^{d} (x_i^m - \mu_j^m)^2}$	$\sum_{m=0}^{d} \lvert x_i^m - \mu_j^m \rvert$
$R(C_j)$	$\sqrt{\dfrac{\sum_{m=0}^{d} (\mathrm{SS}_{\mu_j}^m) - \sum_{m=0}^{d} (\mathrm{LS})^2 / N_{C_j}}{N_{C_j}}}$	$\dfrac{\sum_{m=0}^{d} (R_{i-1}^m + U_{i-1}^m * \lvert (x_i^m - \mu_{i-1}^m)/N_i \rvert + \lvert x_i^m - m_i^m \rvert)}{N_n}$
CF additivity	$\mathrm{CF}_{j_i} = \langle (N_{j_{i-1}} + 1), (\overrightarrow{\mathrm{LS}_{j_{i-1}}} + \overrightarrow{x_i}), (\overrightarrow{\mathrm{SS}_{j_{i-1}}} + (\overrightarrow{x_i})^2), T_{j_{i-1}}, y_{j_{i-1}} \rangle$	$\mathrm{CF}_{j_i} = \langle (N_{j_{i-1}} + 1), (\overrightarrow{R_{j_i}}), (\overrightarrow{U_{j_i}}), T_{j_{i-1}}, y_{j_{i-1}} \rangle$

Table 4: List of online features selected for online classification.

ID	Name	Long description	Short description as in [14]
01	Source port	Source port number	Server port
02	Destination port	Destination port number	Client port
03	TL IP	Total bytes in IP packet	q1_data_ip
04	UL IP	Total bytes in IP packet (uplink)	q1_data_ip_a b
05	DL IP	Total bytes in IP packet (downlink)	q1_data_ip_b a
06	TL Eth	Total bytes in Ethernet packet	q1_data_wire
07	UL Eth	Total bytes in Ethernet packet (uplink)	q1_data_wire_a b
08	DL Eth	Total bytes in Ethernet packet (downlink)	q1_data_wire_b a
09	TL Ctr	Total control bytes in packet	q1_data_control
10	UL Ctr	Total control bytes in packet (uplink)	q1_data_control_a b
11	DL Ctr	Total control bytes in packet (downlink)	q1_data_control_b a

new $(\overrightarrow{x_i})$ to cluster j will cause recalculation of R based on (15), additive of U based on (16), and increment of N by 1, and T and y remain unchanged:

$$\mathrm{CF}_{j_i} = \left\langle \left(N_{j_{i-1}} + 1\right), \left(\overrightarrow{R_{j_i}}\right), \left(\overrightarrow{U_{j_i}}\right), T_{j_{i-1}}, y_{j_{i-1}} \right\rangle. \quad (18)$$

4.3. Complexity Comparison. Table 3 shows the overall changes of our proposed method using Euclidean distance and Manhattan distance. For calculation of distance to origin, D_o Euclidean distance requires $O(N^2 + dN^2)$ compared to Manhattan distance that only requires $O(d)$. D_o is used in the determination of partition, which is required once every instance and affected the most in the reconstruction stage. During reconstruction, D_o for all clusters in the classification model need to be calculated. Hence, it will increase the complexity of reconstruction by $O(k)$.

Similar to D_o, the distance to centroid D for Euclidean distance also requires $O(N^2 + dN^2)$ compared to Manhattan distance that only requires $O(d)$. The increase in complexity in the calculation of D affects classification time by $O(Nk)$.

The radius R is used during selection for learning and during injection of labeled instances. For calculation of R in Euclidean distance, $O(dN^2 + 3N^2)$ are required compared to $O(2dN^2 + N^2)$ for Manhattan. In this case, the complexity is dependent on the dimension. As long as the dimension of dataset is greater than 2, the complexity of R in Manhattan distance is greater. Increase in complexity for the calculation of R directly affects the time for learning by $O(N)$.

The updates of CF in Euclidean distance are slightly simpler than Manhattan distance due to the recalculation of R for each dimension involved. This is the only trade-off of using Manhattan distance over Euclidean distance. However, since the updates of CF only happen during the learning and model reconstruction, it will increase the complexity of reconstruction by $O(k)$.

Overall, although the computations of R and CF are more complex for Manhattan distance, they are not as frequent as the calculation of the distance D, which requires k calculations of D on every incoming flow instance.

5. Experimental Results

This section describes the simulation setup and results of our proposed work. The experiment is conducted to analyze and compare the performance of our proposed method using Euclidean distance and Manhattan distance measures.

5.1. Datasets. Two real network traffic datasets Cambridge [14] and UNIBS [15] are chosen for the experiment. The Cambridge dataset was captured from University of Cambridge network. It contains 248 attributes and 12 classes. A total of 11 online features are selected from the attributes as listed in Table 4. The data of the minimal class, games, and interactive

Table 5: Dataset used.

	Cambridge [14]		UNIBS [15]
	Original	Selected	Extracted
# flow features	248	11	11
# classes	12	10	6
# flow instances	397,152	397,030	82752

are not used as they are not sufficient for training and testing (less than 10 flow instances). UNIBS dataset was captured in University of Brescia. It was collected in three consecutive days and the traces are in pcap format and come with a groundtruth. The traces were processed to extract online features with only the first 5 packets of each observed flow as in [38]. By using the provided groundtruth labels, we labeled the flow instances into 5 classes (Web, Mail, P2P, SKYPE, and MSN). The details of the used datasets are summarized in Table 5.

5.2. Experimental Setup. The model parameters used in our experiment are as stated below, unless specified otherwise:

(1) Percentage of labeling, $P = 10$.

(2) Chunk size = 1000.

(3) Number of micro-model, $B = 10$.

(4) Desired number of cluster, $k_d = 100$.

(5) Boundary threshold = 1.

In our experiment, the first chunk of data (the first 1000 flow instances) are treated as precollected flow instances and they are used for model initialization to generate the base model. The rest of the data are randomly labeled for different percentage P. The accuracy of the proposed model is verified using the interleaved test-then-train method where the data were first tested before being trained incrementally [39]. Each experiment was repeated 100 times, and the average performance indicators are reported in this paper.

The performance indicators used in this paper are the accuracy, cumulative accuracy, time, classification speed, and memory requirement. *Accuracy* refers to the accuracy of each chunk, while *cumulative accuracy* is the cumulative accuracy after the classification of each chunk. *Running time* is defined as the time to process one chunk, including the time for model reconstruction. In our experiment, the running time does not include data labeling time as in [12], since data labeling is usually done offline and is beyond the scope of discussion in this paper. *Classification time* is measured based on the time needed to classify one flow instance not including the feature extraction time.

5.3. Performance. This subsection discusses the overall performance of the proposed algorithm. The accuracy in time-series is shown in Figure 3. As we only use the first chunk in the dataset for model initialization, the classes which were not seen in the first chunk are treated as new concepts. A drift detection experiment was conducted on our dataset by using Drift Detection Method [40] provided in MOA tools [41].

A series of detected drifts are plotted in Figure 4. We found out that Cambridge dataset has more detected drifts than the UNIBS dataset. In order to visualize the drifts clearly, we show the Cambridge dataset in different chunks range. In the figure, we observed that when concept drifts occur (drift detected = 1), our proposed algorithm (using either Euclidean distance or Manhattan distance) can learn from the new knowledge and is able to maintain network classification accuracy compared to the model without incremental learning. In order to clearly show the accuracy difference of our proposed method between both distance measures, the difference of accuracy when using Manhattan distance over Euclidean distance is shown in Figure 5. The results show that the proposed method using Manhattan distance can provide slightly higher accuracy than using Euclidean distance. This shows that simple distance measures can provide better performance.

Table 6 shows the overall performance of our proposed algorithm. In this paper, we assume that the classifier is the bottleneck of the network traffic classification system, as reported in several works such as [38], flow features extraction in FPGA can function in very high speed. We compute the performance of the classification in terms of millisecond per flow instances in software as that is the lower bound of the system performance. The classification time of our proposed algorithm with Euclidean distance is 4.45 ms to classify 1,000 flow instances and 1.49 ms to classify 1,000 flow instances when using the Manhattan distance. This shows that using Manhattan distance can increase the classification speed by almost 3 times. Besides, we found that the time for injecting a labeled flow instance was similar to the time for classifying a new flow instance. Thus, injecting labeled flow instance will not cause long delay to affect the overall online classification process. Time for reconstruction is less than 1% of the total running time and it is almost negligible for both methods. Our proposed system does not require large memory as only the summary of cluster's information, CF, is kept. It only requires in average 140 KiB RAM to complete the processes. Since both of the proposed method store similar amount of data, $\mathrm{CF}_j = \langle N_j, \overrightarrow{\mathrm{LS}_j}, \overrightarrow{\mathrm{SS}_j}, T_j, y_j \rangle$ for Euclidean distance and $\mathrm{CF}_j = \langle N_j, \overrightarrow{R_j}, \overrightarrow{U_j}, T_j, y_j \rangle$ for Manhattan distance, respectively, the overall memory consumption of both methods is similar.

5.4. Impact of Different Parameters Setting on Classification Performance. In this subsection, we analyze the effect of changing algorithm parameters on the overall classification performance. The experiments are done by changing one parameter and fixing the others. Figures 6 and 7 show how labeling percentage, P, is affecting the accuracy and running time, respectively. Increases in labeled flow instances provide more class information to the classification model for better accuracy. Our experimental results also show that the use of Manhattan distance provides better accuracy than Euclidean distance. The percentage of labeling does not affect classification time, but it will increase running time that is more significant for high labeling percentage. This is due to the fact that the more the labeled data are injected to

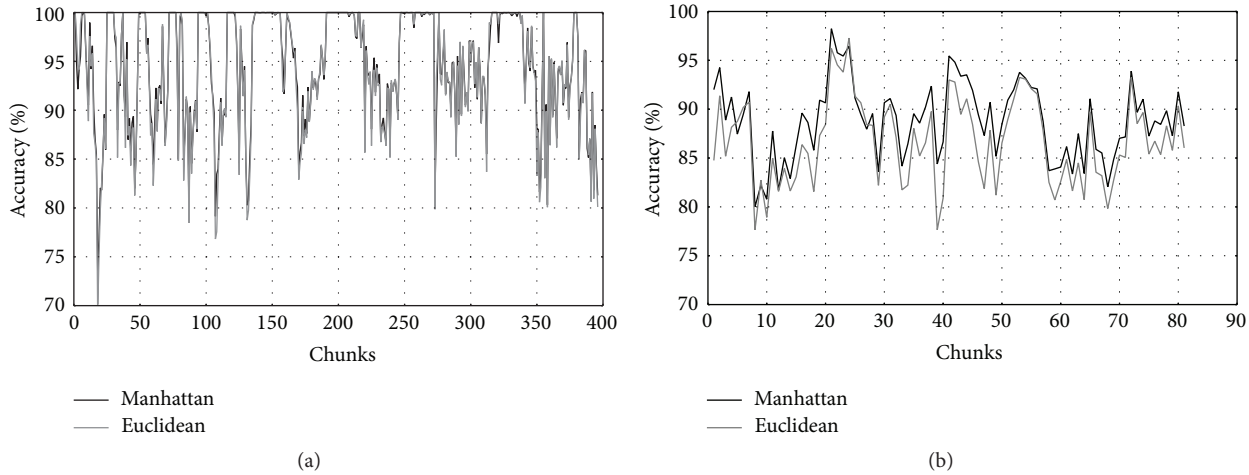

Figure 3: Average accuracy of (a) Cambridge and (b) UNIBS datasets.

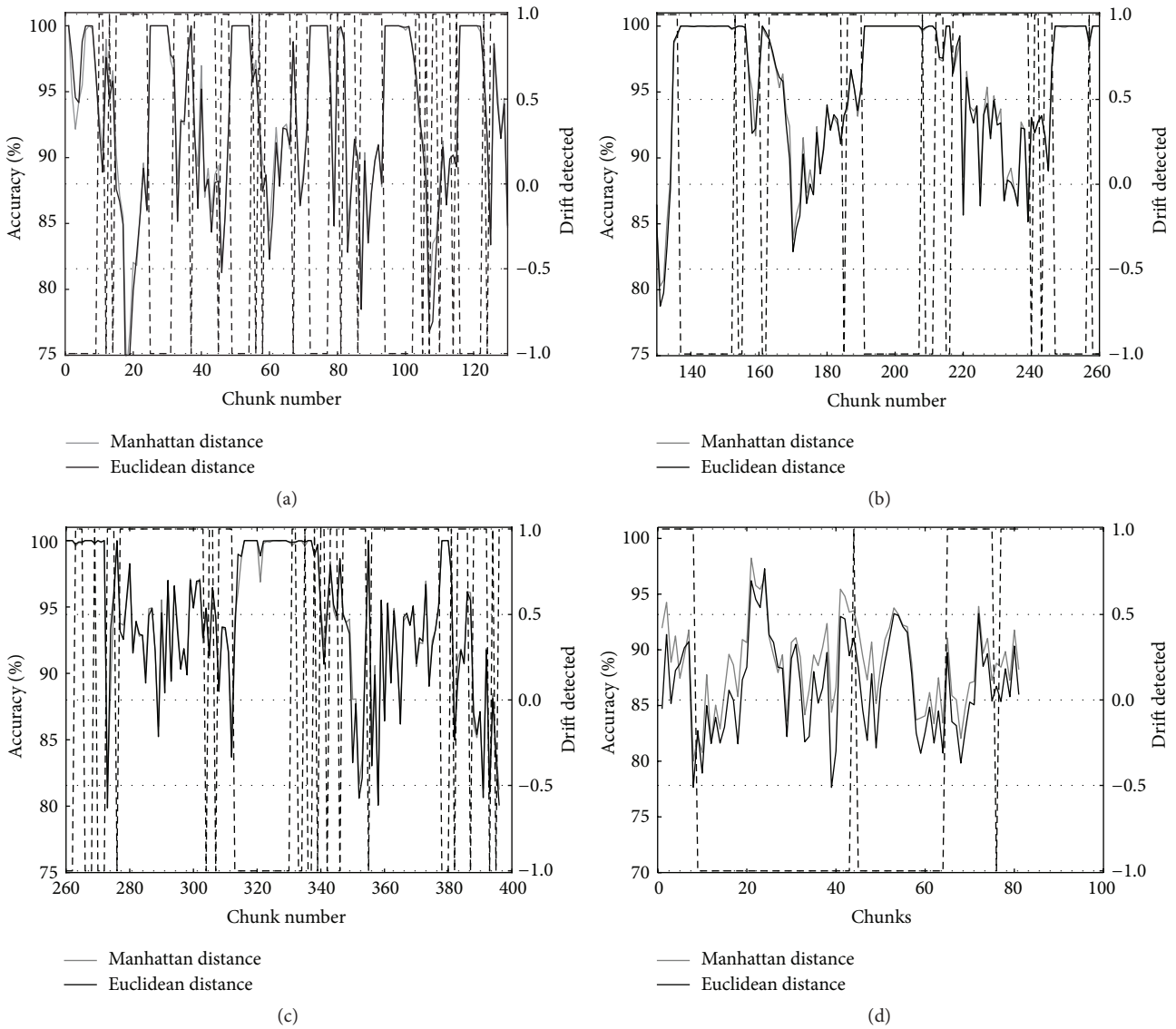

Figure 4: Accuracy of proposed method during concept drift occurrence: (a) Cambridge (chunks 0–130), (b) Cambridge (chunks 130–260), (c) Cambridge (chunks 260–400), and (d) UNIBS datasets.

Table 6: Overall performance comparison.

Dataset	Cambridge		UNIBS	
Distance measure	Euclidean	Manhattan	Euclidean	Manhattan
Average accuracy (%)	94.15	94.38	86.71	88.79
Running time (ms/1,000 flow instances)	5.14	1.75	6.93	2.39
Classification time (ms/1,000 flow instances)	4.45	1.49	6.03	2.02
Injecting labeled instance time (μs/labeled flow instances)	4.68	1.57	6.47	2.32
Reconstruction time (ms/chunk)	0.18	0.09	0.19	0.11
Memory required (KiB)	140	140	140	140

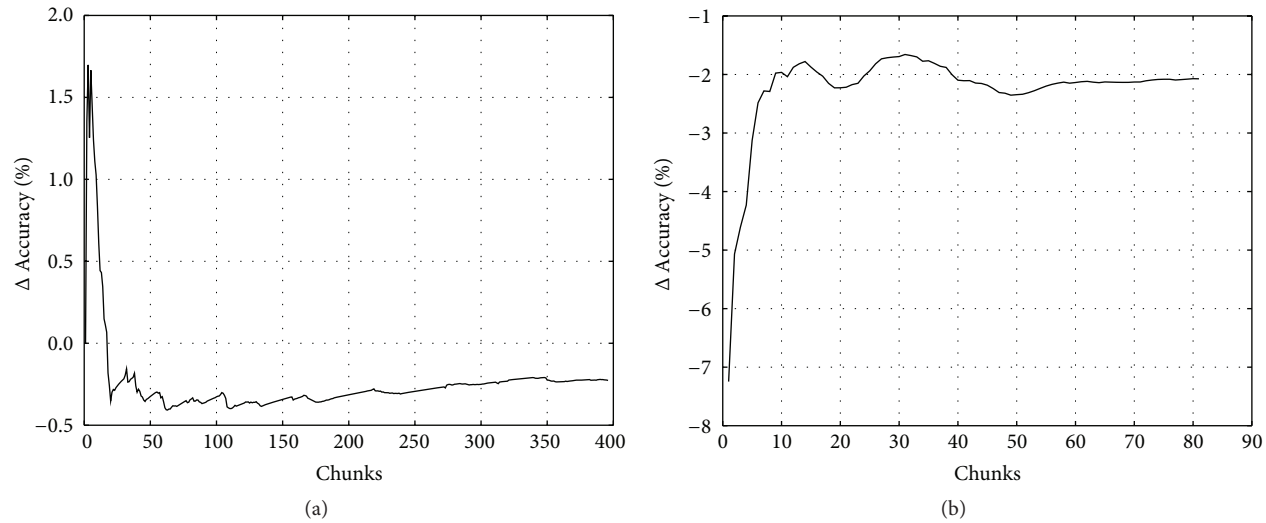

Figure 5: Accuracy difference between Manhattan distance and Euclidean distance for (a) Cambridge and (b) UNIBS datasets.

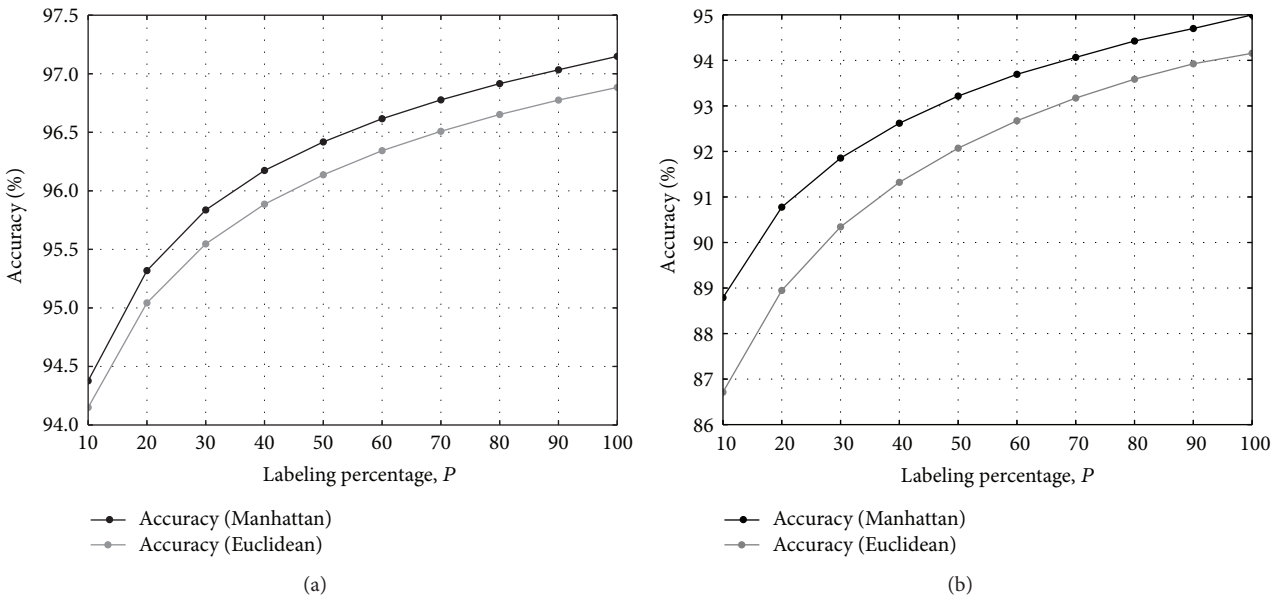

Figure 6: The impact of labeling percentage, P on accuracy (a) Cambridge and (b) UNIBS datasets.

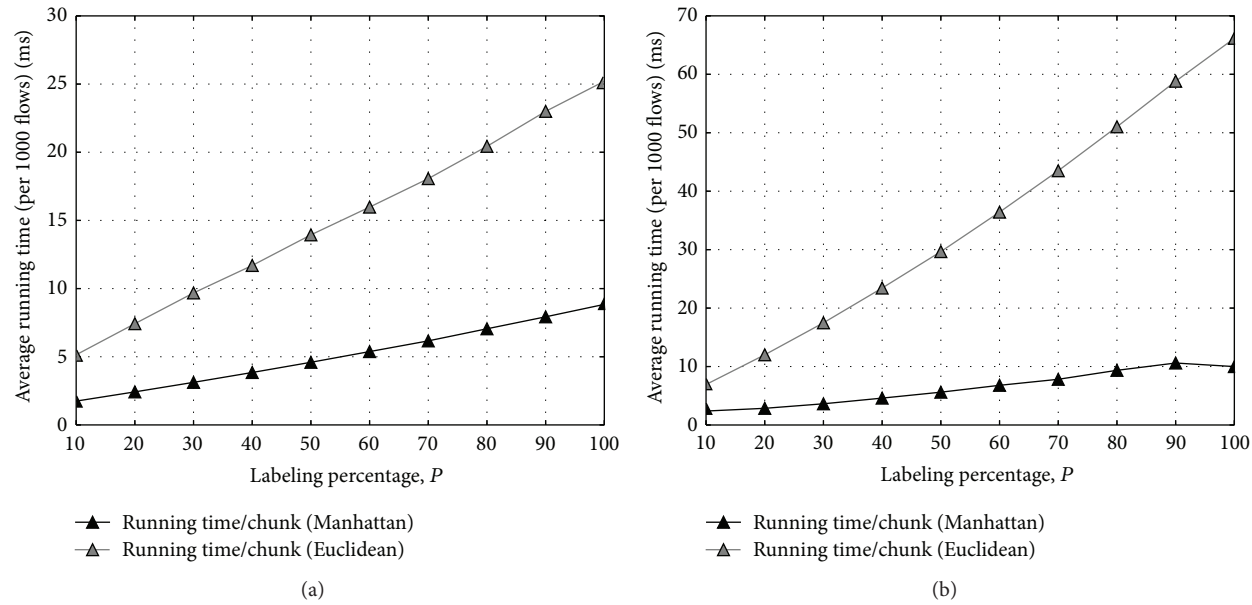

Figure 7: The impact of labeling percentage, P on classification time (a) Cambridge and (b) UNIBS datasets.

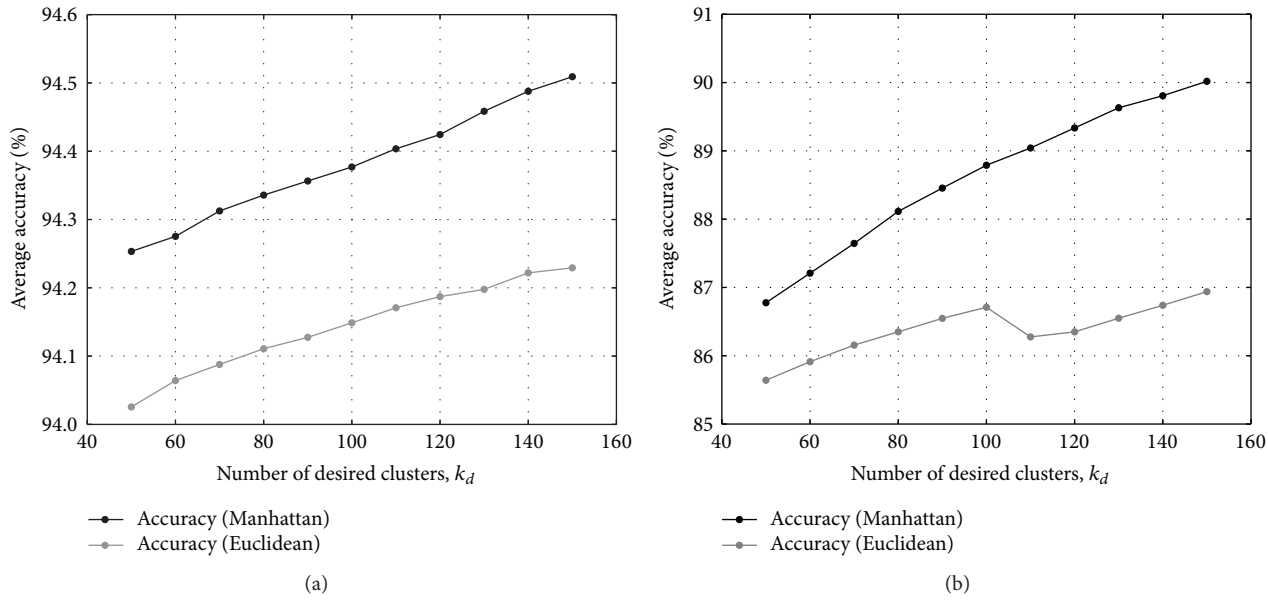

Figure 8: The impact of different number of desired cluster, k_d on accuracy (a) Cambridge and (b) UNIBS datasets.

the classification model, the more distance the measure needs to be calculated. Hence the running time difference becomes more significant.

The number of desired clusters, k_d, is the parameter used in model reconstruction process. It is used to maintain the number of clusters in the classification model. Figures 8 and 9 show the impact of different number of desired clusters on accuracy and reconstruction time, respectively. The increasing accuracy and reconstruction time are more consistent with the increase in k_d for proposed method using Manhattan distance. Reconstruction time for our proposed method with Euclidean distance is higher due to the fact that origin distance calculation required for Euclidean distance in the reconstruction stage is more complex than Manhattan distance.

6. Conclusion

This paper proposed and analyzed an online incremental learning high bandwidth network traffic classification method with two different distance measures (Euclidean and Manhattan distance). The use of Manhattan distance not only provides improvement in running and classification time, it

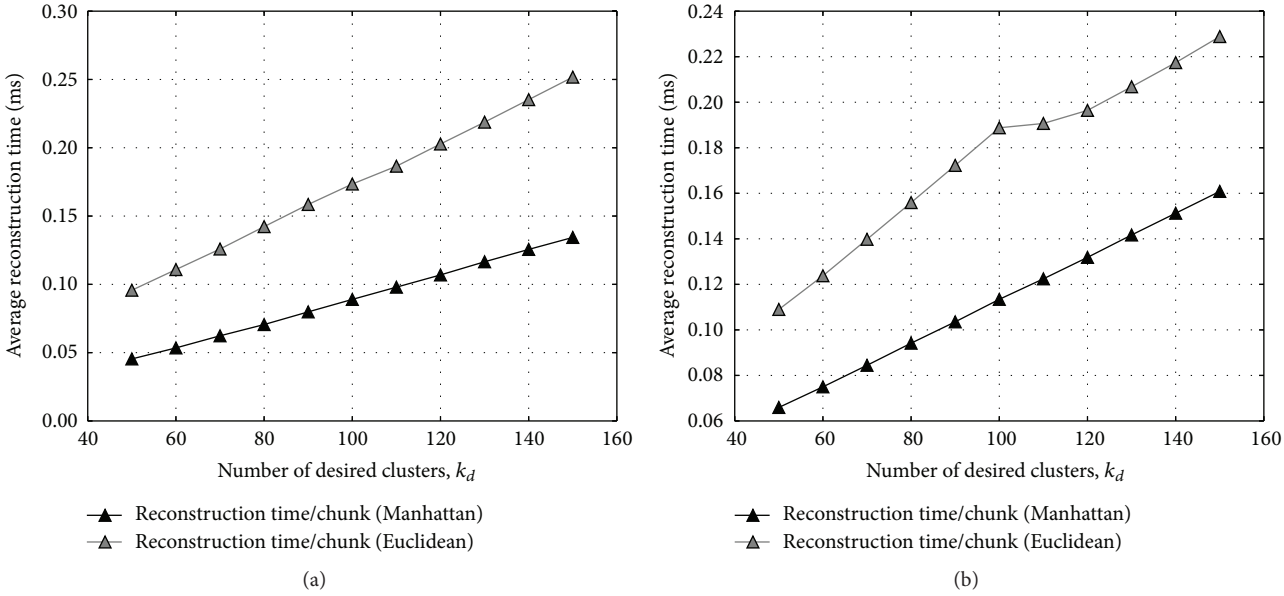

Figure 9: The impact of different number of desired cluster, k_d on classification time (a) Cambridge and (b) UNIBS datasets.

also slightly improves the average accuracy. In future, we will implement this work in reprogrammable hardware so that it can perform inline classification of live network traffic.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The first author is funded by UTM Zamalah schorlaship. This work is funded by Ministry of Science, Technology, and Innovation Science Fund grant (UTM vote no. 4S095).

References

[1] H. Abdulsalam, *Streaming random forest [Ph.D. Dissertation]*, School of Computing, Queen's University, Kingston, Canada, 2008.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "On demand classification of data streams," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '04)*, pp. 503–508, ACM, Seattle, Wash, USA, August 2004.

[3] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00)*, pp. 71–80, ACM, New York, NY, USA, 2000.

[4] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01)*, pp. 97–106, ACM, San Francisco, Calif, USA, August 2001.

[5] W. N. Street and Y. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," in *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01)*, pp. 377–382, ACM, San Francisco, Calif, USA, August 2001.

[6] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03)*, pp. 226–235, ACM, Washington, DC, USA, August 2003.

[7] X. Tian, Q. Sun, X. Huang, and Y. Ma, "Dynamic online traffic classification using data stream mining," in *Proceedings of the International Conference on MultiMedia and Information Technology (MMIT '08)*, pp. 104–107, Zigui, China, December 2008.

[8] G. Mingliang, H. Xiaohong, T. Xu, M. Yan, and W. Zhenhua, "Data stream mining based real-time highspeed traffic classification," in *Proceedings of the 2nd IEEE International Conference on Broadband Network & Multimedia Technology (IEEE IC-BNMT '09)*, pp. 700–705, Beijing, China, October 2009.

[9] B. Raahemi, W. Zhong, and J. Liu, "Peer-to-peer traffic identification by mining IP layer data streams using concept-adapting very fast decision tree," in *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '08)*, pp. 525–532, IEEE, Dayton, Ohio, USA, November 2008.

[10] S. Li, *Towards ultra high-speed online network traffic classifcation enhanced with machine learning algorithms and openflow accelerators [Ph.D. thesis]*, University of Massachusetts Lowell, 2012.

[11] M. Zhang, H. Zhang, B. Zhang, and G. Lu, "Encrypted traffic classification based on an improved clustering algorithm," in *Trustworthy Computing and Services*, vol. 320 of *Communications in Computer and Information Science*, pp. 124–131, Springer, Berlin, Germany, 2013.

[12] M. M. Masud, C. Woolam, J. Gao et al., "Facing the reality of data stream classification: coping with scarcity of labeled data," *Knowledge and Information Systems*, vol. 33, no. 1, pp. 213–244, 2012.

[13] M. Estlick, M. Leeser, J. Theiler, and J. J. Szymanski, "Algorithmic transformations in the implementation of K-means clustering on reconfigurable hardware," in *Proceedings of the 9th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '01)*, pp. 103–110, ACM, Monterey, Calif, USA, February 2001.

[14] A. Moore, D. Zuev, and M. Crogan, "Discriminators for use in flow-based classification," Tech. Rep., Department of Computer Science, Queen Mary University of London, London, UK, 2005.

[15] F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, F. Risso, and K. C. Claffy, "GT: picking up the truth from the ground for internet traffic," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 5, pp. 12–18, 2009.

[16] H. R. Loo and M. N. Marsono, "Online data stream classification with incremental semi-supervised learning," in *Proceedings of the 2nd IKDD Conference on Data Science (CODS '15)*, pp. 132–133, ACM, Bangalore, India, March 2015.

[17] G. Münz, S. Li, and G. Carle, "Traffic anomaly detection using k-means clustering," in *Proceedings of the GI/ITG Workshop MMBnet*, September 2007.

[18] J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms," in *Proceedings of the SIGCOMM Workshop on Mining Network Data (MineNet '06)*, pp. 281–286, ACM, Pisa, Italy, September 2006.

[19] G. Maiolini, G. Molina, A. Baiocchi, and A. Rizzi, "On the fly application flows identification by exploiting K-means based classifiers," *Journal of Information Assurance and Security*, vol. 4, no. 2, pp. 142–150, 2009.

[20] L. Yingqiu, L. Wei, and L. Yunchun, "Network traffic classification using K-means clustering," in *Proceedings of the 2nd International Multi-Symposiums on Computer and Computational Sciences (IMSCCS '07)*, pp. 360–365, IEEE, Iowa City, Iowa, USA, August 2007.

[21] C. Yang and B. Huang, "Traffic classification using an improved clustering algorithm," in *Proceedings of the International Conference on Communications, Circuits and Systems (ICCCAS '08)*, pp. 515–518, IEEE, Fujian, China, May 2008.

[22] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: an efficient data clustering method for very large databases," *ACM SIGMOD Record*, vol. 25, no. 2, pp. 103–114, 1996.

[23] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB '03)*, vol. 29, pp. 81–92, VLDB Endowment, 2003.

[24] Z. Li, J.-G. Lee, X. Li, and J. Han, "Incremental clustering for trajectories," in *Database Systems for Advanced Applications*, vol. 5982 of *Lecture Notes in Computer Science*, pp. 32–46, Springer, Berlin, Germany, 2010.

[25] S. Lühr and M. Lazarescu, "Incremental clustering of dynamic data streams using connectivity based representative points," *Data & Knowledge Engineering*, vol. 68, no. 1, pp. 1–27, 2009.

[26] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu, "Incremental clustering for mining in a data warehousing environment," in *Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB '98)*, pp. 323–333, August 1998.

[27] K. Burbeck and S. Nadjm-Tehrani, "Adaptive real-time anomaly detection with incremental clustering," *Information Security Technical Report*, vol. 12, no. 1, pp. 56–67, 2007.

[28] W. Jiang and M. Gokhale, "Real-time classification of multimedia traffic using FPGA," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '10)*, pp. 56–63, IEEE, Milan, Italy, September 2010.

[29] F. Winterstein, S. Bayliss, and G. A. Constantinides, "FPGA-based K-means clustering using tree-based data structures," in *Proceedings of the 23rd International Conference on Field Programmable Logic and Applications (FPL '13)*, pp. 1–6, IEEE, Porto, Portugal, September 2013.

[30] T. Maruyama, "Real-time K-means clustering for color images on reconfigurable hardware," in *Proceedings of the 18th International Conference on Pattern Recognition (ICPR '06)*, vol. 2, pp. 816–819, IEEE, Hong Kong, August 2006.

[31] A. G. S. Filho, A. C. Frery, C. C. de Araújo et al., "Hyperspectral images clustering on reconfigurable hardware using the k-means algorithm," in *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design (SBCCI '03)*, pp. 99–104, IEEE, São Paulo, Brazil, September 2003.

[32] J. Singaraju and J. A. Chandy, "Active storage networks for accelerating K-means data clustering," in *Reconfigurable Computing: Architectures, Tools and Applications*, vol. 6578 of *Lecture Notes in Computer Science*, pp. 102–109, Springer, Berlin, Germany, 2011.

[33] Z. Lin, C. Lo, and P. Chow, "K-means implementation on FPGA for high-dimensional data using triangle inequality," in *Proceedings of the 22nd International Conference on Field Programmable Logic and Applications (FPL '12)*, pp. 437–442, Oslo, Norway, August 2012.

[34] D. Lavenier, "Fpga implementation of the k-means clustering algorithm for hyperspectral images," in *Los Alamos National Laboratory*, Citeseer, Laur, Philippines, 2000.

[35] V. Bhaskaran, *Parametrized implementation of K-means clustering on reconfigurable systems [Ph.D. Dissertation]*, University of Tennessee, Knoxville, Tenn, USA, 2003.

[36] J. S. S. Kutty, F. Boussaid, and A. Amira, "A high speed configurable FPGA architecture for k-mean clustering," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '13)*, pp. 1801–1804, IEEE, Beijing, China, May 2013.

[37] H. R. Loo, T. Andromeda, and M. N. Marsono, "Online data stream learning and classification with limited labels," *Proceeding of the Electrical Engineering Computer Science and Informatics*, vol. 1, no. 1, pp. 161–164, 2014.

[38] A. Monemi, R. Zarei, and M. N. Marsono, "Online NetFPGA decision tree statistical traffic classifier," *Computer Communications*, vol. 36, no. 12, pp. 1329–1340, 2013.

[39] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: massive online analysis," *The Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.

[40] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence-SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29-Ocotber 1, 2004. Proceedings*, vol. 3171 of *Lecture Notes in Computer Science*, pp. 286–295, Springer, Berlin, Germany, 2004.

[41] Waikato, MOA Massive Online Analysis, 2015, http://moa.cs.waikato.ac.nz/.