International Journal of Electrical and Computer Engineering (IJECE)
Vol. 10, No. 4, August 2020, pp. 4340~4351
ISSN: 2088-8708, DOI: 10.11591/ijece.v10i4.pp4340-4351

Semi-supervised learning approach using modified self-training algorithm to counter burst header packet flooding attack in optical burst switching network

Md. Kamrul Hossain, Md. Mokammel Haque
Department of Computer Science and Engineering, Chittagong University of Engineering and Technology, Bangladesh

Article Info

Article history:

Received Nov 3, 2019
Revised Feb 25, 2020
Accepted Mar 3, 2020

Keywords:

BHP flooding attack
Classification
Optical burst switching network
Self-training algorithm
Semi-supervised learning

ABSTRACT

Burst header packet flooding is an attack on optical burst switching (OBS) network which may cause denial of service. Application of machine learning technique to detect malicious nodes in OBS network is relatively new. As finding sufficient amount of labeled data to perform supervised learning is difficult, semi-supervised method of learning (SSML) can be leveraged. In this paper, we studied the classical self-training algorithm (ST) which uses SSML paradigm. Generally, in ST, the available true-labeled data (L) is used to train a base classifier. Then it predicts the labels of unlabeled data (U). A portion from the newly labeled data is removed from U based on prediction confidence and combined with L. The resulting data is then used to re-train the classifier. This process is repeated until convergence. This paper proposes a modified self-training method (MST). We trained multiple classifiers on L in two stages and leveraged agreement among those classifiers to determine labels. The performance of MST was compared with ST on several datasets and significant improvement was found. We applied the MST on a simulated OBS network dataset and found very high accuracy with a small number of labeled data. Finally we compared this work with some related works.

Corresponding Author:

Md. Kamrul Hossain,
Department of Computer Science and Engineering,
Chittagong University of Engineering and Technology,
Chittagong-4349, Bangladesh.
Email: muhammadkamrulhossain@gmail.com

1. INTRODUCTION

Optical burst switching (OBS) [1] network is a new generation optical networking paradigm that combines the advantages of traditional optical circuit switching(OCS) and optical packet switching(OPS). In OBS network, an IP packet or optical packet which is travelling from source to destination, is first received by an ingress node (optical router). There it is buffered for a very short time which is not as big as in OCS. In this time, other packets may join with the former packet in that buffer. When the time is over, a control packet or burst header packet (BHP) is sent from the ingress node like OCS to try allocating resources for the packets in buffer to travel towards destination. After waiting for a while, the data (packets) in the buffer will be sent without waiting for confirmation from the sent BHP. In this way, this OBS scheme works like both OPS and OCS but without the associated problems of OPS/OCS like high setup latency, low bandwidth utilization or optical buffer requirement etc. In OBS network, BHP flooding attack [2] is a type of DoS attack where a malicious node send BHPs to optical switch to allocate resources but it does send any valid data after it, i.e. it wants to hold the resources of the core node to deprive legitimate nodes from using it.

To prevent BHP flooding attack, some approaches were discussed in [2-4]. These methods more or less depend on human network administrator to set rules for labeling misbehaving nodes. This prevents them from being fully automated. Machine learning (ML) approach in classification of OBS network nodes is not abundant in existing literatures as per our study. Little research is available regarding application of ML in context of OBS network. The number is fewer in terms of works that specifically deal with the application of ML to prevent BHP flooding. Among them, Rajab et al. in [5] discussed about a decision tree based rule finding method from a manually built dataset of OBS network. The authors successfully employed supervised ML approach to classify the data and hence generate some rules to govern the behavior of OBS network nodes. Though this research paved the way for supervised learning approach in countering BHP flooding attack, semi-supervised or unsupervised methods should be investigated because finding labeled data for supervised learning is not always possible. In [6] authors exploited deep learning method in supervised learning approach to detect misbehaving OBS network nodes. They compared their approach with Naïve Bayes, support vector machine, k-nearest neighbors, and found superior performance. A semi-supervised classification of OBS network dataset was done in [7] using k-means clustering technique. The authors used 20% instances of a labeled dataset to train their method. They obtained 90.2% accuracy for two class classification (using 152 labeled instances) and 41.61% accuracy for four class classification (using 215 labeled instances) of OBS network nodes.

In this study, we are proposing a semi-supervised approach to learn the malicious behavior of OBS network nodes. To the best of our knowledge, this study is the first to offer a semi-supervised solution with high accuracy in classification of OBS network nodes when the amount of labeled data is small. Semi-supervised learning (SSML) [8, 9] is a popular machine learning paradigm which makes use of unlabeled data to find a better insight about the structure of data. It is very useful when we have a large amount of unlabeled data but relatively small amount of labeled data. This case is appropriate for OBS network as fully labeled datasets are not widely available and it is relatively easy to find unlabeled/partially-labeled OBS network data. Another reason for the scarcity of properly labeled data is that OBS network is not yet widely implemented. Many algorithms have been developed to exploit the unlabeled data based on the thought that even though the classification of the unlabeled data are unknown, the data itself carries significant information about the class parameters. Some commonly applied SSML techniques are: Self training, co-training, generative models, graph-based algorithms, pseudo-labeling etc.

This study focuses on the 'Self-training' method of SSML. Self-training method is very common choice for SSML [10-12]. Classical self-training method (ST) uses the available true-labeled data (L) to train a base classifier (C). Then C predicts the labels of unlabeled data (U). A portion (P) from the newly labeled data is removed from U along with labels based on prediction confidence and combined with L. The resulting data (P+L) is then used to re-train the base classifier. This process is repeated until U is empty or a predefined number of iteration reached. The method is shown in algorithm 1. Although this method is easily understandable and yields a good result in most cases with very few data, it is much slower that other SSML methods. Another disadvantage is that early inaccuracy could reinforce themselves in future iterations [9].

In this paper, we modified this ST method to boost its performance. In existing literatures, SSML methods similar to ST are widely found. 'Pseudo-labeling' is SSML technique proposed by Lee [13]. In this method, for an unlabeled record, the class with the maximum predicted probability is chosen and called 'pseudo-label'. In effect, this process is similar to entropy regularization. Though it was proposed for deep neural networks, the principle can be applied with traditional machine learning models. Although this method has better performance than ST approach, the difference is not very large. Co-training is a variant of ST method [14]. In co-training, the features of a dataset are split into two parts. One part (labeled) is trained with a classifier and other part (labeled) with another classifier. Then the unlabeled data is labeled using the two models separately. Then the labels obtained by the former classifier is added to the labeled data of later classifier and vice versa. Then the whole process is repeated. The algorithm performs well when the feature split is good but it is challenging to find natural feature split. A modified self-training algorithm based on decision tree was proposed in [15]. The authors showed that the standard decision tree algorithm is not effective as base learner in self-training because it produces unreliable probability estimation to its predictions. They proposed some modification to the standard decision tree learning algorithm. In [16] authors proposed a methodology for selecting an appropriate base learner to be used in self-training. From among a pool of classifiers, this method chooses a learner by eliminating others through filtering with a certain threshold of prediction probability. After choosing an appropriate classifier, it is used as the base learner of standard self-training algorithm.

In this work, we are proposing a method that is built upon the idea of self-training discussed above. We name it modified self-training (MST). We exploited multiple (N) classifiers (A_1,A_2,..A_N) of different types unlike conventional ST method. We made use of agreement among multiple classifiers in prediction of labels following a two stage selection process. At first stage, some classifiers of different types are trained

on the available labeled data (L). Then they predicted the labels of unlabeled data (U). We call these newly assigned labels 'pseudo-labels'. Out of all the pseudo-labeled data, we chose those whose labels were agreed by a specified number of classifiers. Let say P is the subset of U which was selected based on label-agreement. In the second stage, some classifiers of different types were trained on L. Then they predicted the labels of P. Now, based on agreement of both prediction probability (above a certain threshold) and predicted labels (agreed by specified number of classifiers), a subset (S) of data from P was selected along with their labels. S was then added to L. This produced the final labeled dataset (F). F can be used to train any classifier and predict labels for data with unknown labels. The performance of MST was compared with ST on six commonly referred datasets and we found significant improvement. We applied MST on a simulated OBS network dataset from UCI machine learning repository and found very high accuracy. Finally, we compared this work with some related work and found that our method outperformed them. Algorithm 2 presents the proposed method. Besides, we defined some rules for blocking OBS nodes, detected as harmful by the proposed algorithm.

2. RESEARCH METHOD

In this section we briefly discussed about the basic self-training algorithm. Then we explained our proposed methodology in detail.

2.1. Semi-supervised learning setup

In semi-supervised setting, the amount of unlabeled data is very large compared to the amount of labeled data. Data is divided into L and U where L is the set of labeled data points and U is the set of unlabeled data points. That is, $L=(x_1,x_2...,x_n)$ for which labels $Y_L=(y_1,y_2,y_3,..y_l)$ are provided, and $U=(x_{n+1},x_{n+2}...,x_{n+u})$ for which label Y_U is unknown. We assume that Y_L can have two or more labels and both U and L are from same data distribution.

2.2. Self-training (ST) algorithm

Classical self-training(ST) method [15] uses a single base learning algorithm to train on available labeled dataset L. After the algorithm is trained, it predicts the unknown labels of U. Then from the newly labeled data, a portion P is selected based on prediction confidence (probability estimate here) and mixed with the original labeled data L to form a new dataset F. P is removed from U. Then F is used to re-train the learning algorithm. The newly trained model then predicts label for the remaining unlabeled data in U. This process is repeated until U is empty or a specified iteration count is reached. The basic structure of classical ST is presented in algorithm 1.

Algorithm 1 Classical Self-training algorithm

```
      //initialization
1.    N: Iteration counter; C: Base classifier; L: Labeled data; U: Unlabeled
      data; maxCount: number of iterations allowed; H: Confidence Threshold;
2.    N = 1;
      //confidence based learning
3.    do
4.    Train C on L
5.    for each di in U do
6.    Assign pseudo-label to di based on prediction confidence
7.    end for
8.    Select a set P of the high-confidence predictions from U based on threshold
      H
9.    Update N = N+1; U = U - P; L = L U P;
10.   while (U! = empty) and (n < maxCount )
11.   end do-while
12.   Output: Fully labeled dataset
```

2.3. Proposed modified self-training (MST) algorithm

In our proposed methodology, we take 'N' classifiers instead of just one. The value of N can be chosen at will, the larger the better. The N classifiers are trained by L and hence, N models are generated. The unlabeled data U will be given to each of the N models to be predicted and pseudo-labeled based on each model's default classification confidence (here it is probability estimate). As a result, N pseudolabeled datasets will be generated. Then the first stage of selection will start. Let say, $U=(x_{n+1},x_{n+2}...,x_{n+u})$ and each x_i in U has got N pseudo-labels. If all the pseudo-labels of x_i are same then we say that N models agreed on that label and this data point will be retained for the next stage. If in any case, there is no unanimous

agreement by all the N models for any of the x_i in U, then we may define a vote requirement rule for selection. That is, we can set a lower limit M1 (where $0<M1\leq N$) and if any pseudo-label for x_i is voted by at least M1 models, then the point will be selected for the next stage. Now let say, P is the subset of U which was selected (along with labels) based on label-agreement of the classifiers. In the second stage, 'C' number of classifiers of different types will be trained by L (can be new or previously mentioned classifiers). The obtained C models will then predict and pseudo-label P based on classification confidence (here it is probability estimate). So we get C pseudo-labeled copies of P. Let say, $P=(x_{p1},x_{p2}...,x_{pn})$ and each x_{pi} in P has got C pseudo-labels. Now, if M2 (where $0<M2\leq C$) models agreed on the label of x_{pi} and all those pseudo-labels were assigned by the respective classifiers with a prediction probability greater than the specified threshold H, then x_{pi} is selected. Let S be the set of all the points selected (along with labels) this way, i.e. based on agreement of prediction probability and predicted pseudo-labels. S is deleted from U and added to L. L is now the final training dataset. L is used to train any classifier in supervised way and then the classifier can be used to predict unknown labels. The detailed steps are presented in algorithm 2. In Figure 1 a block diagram is shown to illustrate the overview of the proposed modified self-training (MST) method.

Algorithm 2 Proposed Modified self-training algorithm (MST)

```
     //initialization
1.   C1: A set of N classifiers; C2: A set of M classifiers; F: Final classifier;
     L: Labeled data; U: Unlabeled data; K: Set of empty datasets; P, S: Empty dataset;
     V1: 1st stage pseudo-label agreement threshold; V2: 2nd stage pseudo-label agreement
     threshold; H: Classification confidence threshold;
2.   C1= {C11,C12,..C1N}; C2= {C21,C22,..C2M}; i = 1; K= {K1,K2,..KN}
     //first stage voting
3.   for each c in C1 do
4.   Train c by training set L
5.   for each d in U do
6.   Assign pseudo-label to d based on prediction confidence of c
7.   Save d along with the pseudo-label in set Ki
8.   end for
9.   i = i+1
10.  end for
11.  for each d in U do
12.  compute pseudo-label agreement(votes) for d in K for every possible label
13.  if for any label, votes >=V1 then
14.  copy d and save in set P
15.  end if
16.  end for
     //second stage voting
17.  i = 1; K1,K2,..KN = {};
18.  for each c in C2 do
19.  Train c by training set L
20.  for each d in P do
21.  Classify d by model c
22.  if classification confidence >= H then
23.  Save d along with the pseudo-label in set Ki
24.  end if
25.  end for
26.  i = i+1;
27.  end for
28.  for each d in P do
29.  compute pseudo-label agreement(votes) for d in K for every possible label
30.  if votes >=V2 then
31.  copy d in set S
32.  end if
33.  end for
     //preparing final training dataset
34.  Update U by removing S from U: U = U - S;
35.  Update L by joining S with L: L = L U S;
     //final model training and labeling
36.  Train a classifier F by new training set L
37.  Classify U by F and predict labels for all the points in U
38.  Output: Fully labeled dataset
```

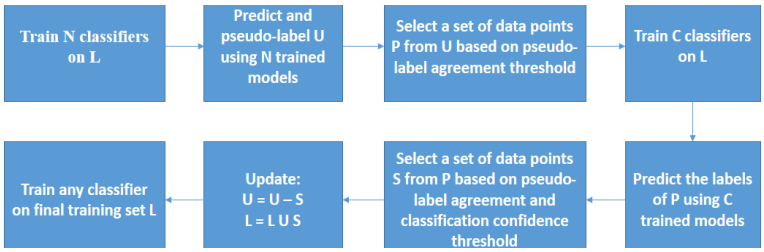

Figure 1. Block diagram of proposed modified self-training (MST) method

2.3. Dataset for OBS network's node classification

In this section, we describe a way to obtain a dataset from an OBS network simulator, which can be used by our proposed method with a few labeling. In Table 1, we presented a sample OBS network dataset. Using NCTUns network simulator software [17], we simulated an OBS network to observe the behaviors of the OBS network nodes and to record each node's behavioral data. We made an NFSNET topology and configured the network's parameters following the work in [5]. The setting had 14 core switches that follow JET protocol, edge routers linked to host-pcs, 1 Gbps maximum link bandwidth, 1 µs propagation delay, 1500 bytes burst length, 0 bit error rate, 2 DB channel, 1 BHP channel, UDP protocol at transport layer. Each simulation was run for 10 minutes by varying the bandwidth from 0.1 Gbps to 1 Gbps. One malicious node was selected in each run of the simulation in varying position. We recorded data for the following three variables related the node performance: Average percentage of packet drop rate, average bandwidth used (mbps) and average delay (second). Averaging was done by running the simulator 10 times and calculating the mean value. Ten rows of the dataset are shown in Table 1. The three mentioned attribute was chosen out of many because we found highest degree of independence and significance in them. After a dataset was recorded, the class labels were given manually for a few instance from each possible class. We considered three different labels for three different behaviors of the nodes. The first is "MALICIOUS", it means that the node is causing BHP flood. Second is 'TRUSTED', this means that the corresponding node is well-behaving. Third label is 'SUSPICIOUS', it means that the node has high packet drop rate but not as high to label as malicious. When the dataset was collected for the first time, all the possible cases were considered, i.e. the OBS network was configured in a way that the recorded data reflects not only well-behaving node's behavior, but also the malicious node's behavior in BHP flooding scenario.

Table 1. 10 rows from OBS network dataset (contains both labeled and unlabeled data)

Iteration	Node Id	Average delay	Average packet drop rate	Average bandwidth use	Class
1	1	0.0003	0.1531	0.726	unknown
1	2	0.0002	0.5262	0.3281	unknown
2	1	0.001	0.0811	0.8699	trusted
2	2	0.0005	0.4803	0.7292	unknown
3	2	0.0006	0.097	0.7185	unknown
3	1	0.0012	0.133	0.5085	trusted
4	2	0.0004	0.4052	0.668	unknown
4	1	0.0006	0.2681	0.4861	unknown
5	1	0.0002	0.3775	0.7142	unknown
5	2	0.001	0.3066	0.7349	malicious

In this way, a dataset from OBS network can be drawn and with a few labeling and can be used for the proposed method. In this work, we did not produce a fully labeled dataset, neither had we generated a large amount of data. Rather we used a readymade dataset from UCI ML repository to demonstrate the effectiveness of our method. The authors of the dataset built the dataset using the approach mentioned above. The reason for not using new and self-made data is that, we were not able to find human-experts to properly label a large OBS network dataset which is a very critical task. Another reason is the absence of real life OBS network near us.

2.4. Blocking of misbehaving nodes

After obtaining a partially labeled OBS network dataset, the proposed MST method was applied following the steps of algorithm 2. This gives a fully labeled dataset and a model trained on that dataset. We mentioned earlier that we categorized the OBS network nodes into three types: MALICIOUS',

'TRUSTED' and 'SUSPICIOUS'. If a node is classified as ‘MALICIOUS’ by our model, the node will be prevented from sending any packet in future. At the same time, we change the status of that node to ‘Blocked’. If a node is classified as 'TRUSTED', we allow it to send packets and simultaneously, we change the status of that node to ‘Trusted’. If a node is classified as 'SUSPICIOUS', we check if its current status is 'Suspicious' or not. If yes then we prevent it from sending packet for a specified time and change the status of that node to ‘Waiting’. Otherwise, we set the status of the node as 'Suspicious' and continue. Then the next node is visited. In this way, the trained model can be used to classify OBS network nodes and counter BHP flooding attack. The flowchart in Figure 2 presents the rules to block misbehaving nodes in an active OBS network. Note, the process box showing “Circularly increment i” means that, after a node is checked, the next node will be checked according to the arrangement in the given node set. When all the nodes are checked, it will return to the first node. Besides, when a node is sent to waiting state, a counter should be there to count the end of waiting time, after which, the node’s status should be labeled as suspicious. We did not show the time counter in the flowchart.

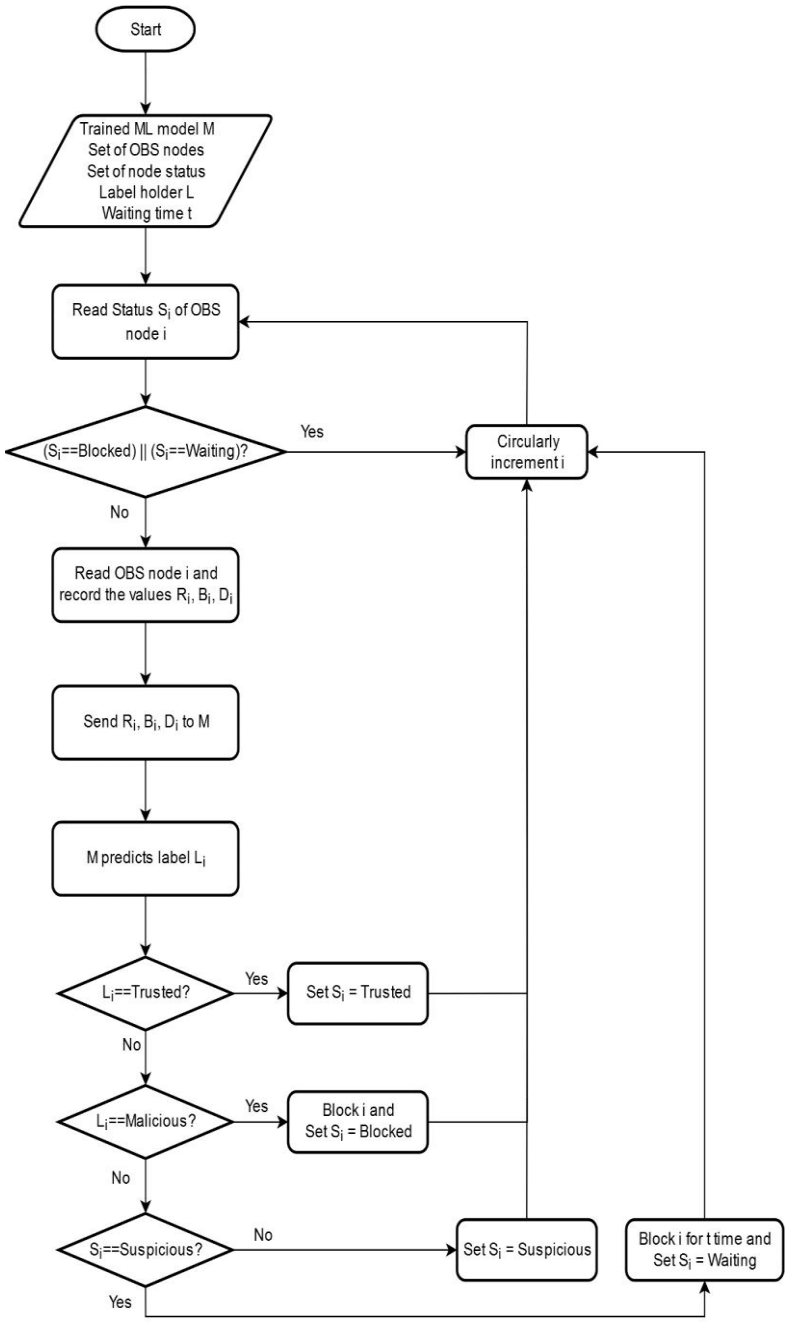

Figure 2. Flowchart of blocking rules for malicious node

3. RESULTS AND ANALYSIS

3.1. Datasets for evaluating performance

We compared the performance of our proposed algorithm with the basic self-training algorithm on six commonly referred datasets from UCI machine learning repository [18]. They are: adult dataset, breast cancer dataset, car dataset, iris dataset, glass dataset and balance dataset. The adult dataset and breast cancer dataset are binary classification datasets while others are multi-class datasets. Car is four class and glass is six class classification dataset while iris and balance are three class classification dataset. All these datasets are classification datasets with no missing labels or missing values. To test our proposed algorithm, we removed the labels of 97% data in each dataset (except for iris dataset, for which we removed 95% labels). Hence, only 3% data (for iris, 5%) are labeled and the rest 97% (for iris, 95%) are unlabeled on each of the datasets. We saved the deleted labels separately to assess performance.

3.2. Selection of classifiers

In self-training algorithm (algorithm 1, say 'ST'), for a specific dataset, we ran the algorithm six times, each time with a different base classifier. The classifiers were: Extra tree classifier [19], gradient boosting classifier [20], gaussian naive bayes [21], logistic regression classifier [21] and quadratic discriminant analysis [22] and support vector machine [21]. We used python programming language and an ML library called Scikit-learn [23] for implementation of these classifiers. In proposed modified self-training algorithm (algorithm 2, say 'MST'), at the first stage of voting, we used eight classifiers which includes the classifiers mentioned above except the Support vector machine. The other three are: Random forest classifier [21], XGBoost classifier [24, 25] and multi-layer perceptron classifier [21]. At the second stage of voting, two of the above mentioned classifiers were removed and the remaining six were used. The two removed were: Extra tree classifier and Random forest classifier. The reason for not including them in the second stage is that the standard decision tree classifier produces unreliable probability estimation to its predictions which cannot be an appropriate selection criterion in self-training [15]. Once the MST algorithm produced the final training dataset, six different classifiers were trained by it. We chose the same six classifiers as in ST for making valid comparison between ST and MST method.

3.3. Comparison metrics

We chose six commonly referred metrics to assess the performance of MST against ST algorithm. Those are: average precision, area under ROC curve, accuracy, f1-score, precision and recall. Since these metrics functions differently for binary and multi-class dataset, we computed them suitably.

Precision-recall curve (PRC) is a commonly used metric that shows the tradeoff between recall and precision. It is a measure of how many relevant results are found. The larger the area under the curve (AUC), the higher the recall and precision. Average precision [26] expresses a summary of the entire AUC as the weighted average of precision returned at each threshold in a way that the weight becomes the difference in recall from the previous threshold. In case of multi-class classification, the learning algorithm is wrapped in a one-vs-rest fashion to produce binary comparisons for each class. The PRC was then computed as the 'micro average' [27] of the recall and precision for all classes. Average precision ranges from 0 to 1, the bigger the better. Area under the receiver operating characteristics (ROC-AUC) plot [28, 29] illustrates the tradeoff between a classifier' sensitivity and specificity. Receiver operating characteristics (ROC) is a probability curve that displays the true positive rate and false positive rate on the Y-axis and X-axis respectively. The higher the ROC-AUC, the better the model's correct prediction ability is. For multi-class dataset classification, one-vs-all (macro score) [30] strategy was used. ROC-AUC ranges from 0 to 1, the bigger the better. Accuracy is the ratio of correctly predicted labels to the total labels. It is expressed in percentage. Precision, for the binary labeled dataset, is the ratio of correctly predicted positive labels to the total predicted positive labels. Recall, for the binary labeled dataset, is the ratio of correctly predicted positive labels to the total observations in actual positive label. F1 Score returns the weighted average of precision and recall. In case of multi-class dataset, precision, recall and f1 score are computed differently. We calculate the metric for each label and then computed their unweighted mean (macro average) [31]. The value of precision, recall and f1 score range from 0 to 1, the bigger the better.

3.4. Experimental settings for the proposed algorithm

There are some tuning parameters in algorithm 1 and 2. In algorithm 1 (classical self-training, ST), confidence threshold H1 is used to select high-confidence predictions at each iteration. In algorithm 2 (proposed algorithm, MST), first stage pseudo-label agreement threshold V1 was used to select data points whose labels were agreed upon by V1 models. In the second stage, classification confidence threshold H was used to select data points that had high-confidence predictions. The data points selected in this step is further filtered using pseudo-label agreement threshold V2 by choosing only those points whose labels

were agreed upon by V2 models. All the above mentioned thresholds have great impact on the performance of corresponding classifier, i.e. the model's performance is sensitive to the choice of threshold parameters. Also, performance varies from dataset to dataset based on those threshold values.

In ST, to select the confidence threshold H1, we followed the method suggested in [15]. That is, we took top 10 percent from the high-confidence predictions and used the mean of their probability estimation as threshold H1. Also, the value for maxCount was chosen as 40 based on those works. In MST, V1 is the number of classifiers need to agree for selecting a data point for next stage. V1 can be any value from 0 to N where N is the number of available classifiers in first stage. V1 should be chosen such that it yields data points of all the possible labels. Also, majority portion of the unlabeled data points should be obtained for second stage. Value of V1 starts from N and should be reduced until the two mentioned conditions are met. H2 is confidence threshold, i.e. estimated probability threshold and V2 is like V1 but it is for the second stage. V2 can be any value from 0 to M where M is the number of available classifiers in second stage. V2 and H should be as high as possible. The combination of H and V2 should be such that it results in data points with all the possible class-labels, i.e. at least one data point from each class should be present in output. The data instances found from this stage is combined with the original true labeled data for preparing final training dataset. Value of H2 and V2 should start from maximum possible value. If desired output is not found when H2 and V2 are maximum, then at first, V2 should be decremented numerically. But V2 should always be greater than half of the available classifiers in second stage, i.e. if M classifiers are available in second stage, then V2 should be greater than M/2. Now, if V2 reaches its lowest permissible value but desired output is not obtained, then we should reduce the value of H2 one step. We took step size equals to 0.5. It can be chosen at will, the smaller the better. After reducing H2, we again tune V2 by starting it from its maximum value (i.e. M). Following the steps mentioned above, we try to find V2 that gives desired output. In this way, we find a combination of H2 and V2 that gives desired output, i.e. it results in data instances with all the possible class-labels. The data instances found from this stage, are combined with the original true labeled data for preparing final dataset.

3.5. Result

In Table 2 to 7, we summarized the performance comparison between ST and MST on six commonly referred datasets. In the tables, the first column holds the names of the classifiers. ST is written on bracket to indicate that the classifier was used as base classifier for algorithm 1 and MST indicate that the classifier was used as final classifier following algorithm 2.

Table 2. Adult dataset

Name of classifier	Micro avg. precision	ROC-AUC	Accuracy	F1 Score	Precision	Recall
Extra Tree(ST)	0.507	0.729	83.610	0.575	0.757	0.463
Extra Tree(MST)	0.649	0.853	82.905	0.593	0.690	0.519
GradientBoost(ST)	0.740	0.890	82.686	0.473	0.869	0.325
GradientBoost(MST)	0.782	0.905	85.803	0.663	0.767	0.583
Gaussian(ST)	0.447	0.780	74.593	0.402	0.460	0.356
Gaussian(MST)	0.639	0.838	81.189	0.556	0.639	0.492
LogisticRegression(ST)	0.679	0.846	81.797	0.457	0.799	0.320
LogisticRegression(MST)	0.681	0.846	82.618	0.556	0.715	0.455
QDA(ST)	0.581	0.789	74.969	0.404	0.470	0.355
QDA(MST)	0.658	0.850	79.435	0.444	0.629	0.343
SVM(ST)	0.619	0.823	81.054	0.493	0.685	0.385
SVM(MST)	0.632	0.834	80.666	0.433	0.726	0.308

The Adult dataset is an imbalanced binary dataset of 48842 rows where 11687 rows are of positive label ('income exceeds 50') and rest are negative. We took 3% labeled data for our experiment maintaining the class label ratio. From Table 2, we find that, our proposed method (MST) outperformed the classical self-training (ST) in most of the metrics for all the classifiers, except for Support vector classifier which showed performance similar to ST.

The Balance dataset is an imbalanced multiclass (3 label) dataset of 625 rows of which 288 are 'L' label, 288 are 'R' label and 49 are of 'B' label. We took 3% labeled data for our experiment maintaining the class ratio. From Table 3, we find that the proposed method (MST) outperformed the classical self-training (ST) in most of the metrics for all the classifiers.

The Breast cancer diagnostic dataset is an imbalanced binary dataset of 569 rows where 212 rows are of positive label ('malignant tumor') and rest are negative. We took 3% labeled data for our experiment maintaining the class ratio. From Table 4, we find that, the proposed method (MST) outperformed

the classical self-training (ST) in most of the metrics for all the classifiers, except for Gaussian Naive Bayes and Logistic regression (which are very close to ST in performance).

Table 3. Balance dataset

Name of classifier	Micro avg. precision	ROC-AUC	Accuracy	F1 Score	Precision	Recall
Extra Tree(ST)	0.749	0.750	74.794	0.522	0.514	0.540
Extra Tree(MST)	0.833	0.789	78.089	0.545	0.529	0.564
GradientBoost(ST)	0.714	0.738	69.028	0.527	0.533	0.528
GradientBoost(MST)	0.870	0.829	77.759	0.554	0.566	0.568
Gaussian(ST)	0.668	0.699	63.427	0.438	0.427	0.458
Gaussian(MST)	0.897	0.803	81.549	0.569	0.554	0.589
LogisticRegression(ST)	0.891	0.856	81.878	0.568	0.554	0.592
LogisticRegression(MST)	0.908	0.858	82.867	0.577	0.557	0.599
QDA (ST)	0.752	0.769	68.369	0.466	0.515	0.494
QDA (MST)	0.859	0.796	81.878	0.572	0.554	0.592
SVM(ST)	0.812	0.801	70.675	0.572	0.600	0.570
SVM(MST)	0.918	0.884	83.361	0.671	0.677	0.667

Table 4. Breast cancer dataset

Name of classifier	Micro avg. precision	ROC-AUC	Accuracy	F1 Score	Precision	Recall
Extra Tree(ST)	0.969	0.964	91.486	0.931	0.946	0.916
Extra Tree(MST)	0.967	0.965	91.486	0.932	0.931	0.934
GradientBoost(ST)	0.942	0.939	87.319	0.894	0.939	0.853
GradientBoost(MST)	0.943	0.939	89.855	0.917	0.942	0.893
Gaussian(ST)	0.989	0.981	93.297	0.946	0.953	0.939
Gaussian(MST)	0.987	0.978	93.659	0.948	0.976	0.922
LogisticRegression(ST)	0.885	0.920	90.399	0.924	0.913	0.936
LogisticRegression(MST)	0.879	0.916	89.674	0.918	0.909	0.928
QDA (ST)	0.843	0.863	83.514	0.873	0.842	0.908
QDA (MST)	0.986	0.980	91.667	0.930	0.984	0.882
SVM(ST)	0.747	0.807	87.862	0.908	0.864	0.957
SVM(MST)	0.800	0.865	89.130	0.914	0.909	0.919

Table 5. Car dataset

Name of classifier	Micro avg. precision	ROC-AUC	Accuracy	F1 Score	Precision	Recall
Extra Tree(ST)	0.770	0.754	73.942	0.423	0.531	0.390
Extra Tree(MST)	0.826	0.802	77.340	0.426	0.497	0.404
GradientBoost(ST)	0.844	0.827	76.148	0.430	0.451	0.432
GradientBoost(MST)	0.750	0.876	79.905	0.490	0.553	0.468
Gaussian(ST)	0.769	0.646	68.933	0.339	0.546	0.318
Gaussian(MST)	0.795	0.683	70.543	0.285	0.451	0.296
LogisticRegression(ST)	0.764	0.719	70.006	0.206	0.175	0.250
LogisticRegression(MST)	0.778	0.734	69.827	0.223	0.299	0.257
QDA (ST)	0.252	0.570	3.936	0.046	0.028	0.250
QDA (MST)	0.606	0.638	63.089	0.292	0.365	0.323
SVM(ST)	0.746	0.695	70.364	0.277	0.357	0.284
SVM(MST)	0.775	0.770	70.125	0.356	0.406	0.363

The Car dataset is a highly imbalanced multiclass (4 label) dataset of 1728 instances where class 0 has 384 instances, class 1 has 69 instances, class 2 has 1210 instances and class 3 has 65 instances. We took 3% labeled data for our experiment maintaining the class ratio. From Table 5, we find that, the proposed method (MST) outperformed the classical self-training (ST) in most of the metrics for all the classifiers, except for Gaussian Naive bayes classifier which has similar performance to the ST.

In Table 6 the result on Glass dataset is shown. The Glass dataset is a highly imbalanced multiclass (6 class labels) dataset of 214 instances. We took 9% labeled data (19 instances) for our experiment maintaining the class ratio. From Table 5, we find that the proposed method (MST) outperformed the classical self-training (ST) in most of the metrics for all the classifiers.

In Table 7 the result on Iris dataset is shown. The Iris dataset is a balanced dataset of 150 instances with three class labels, having 50 instances for each class. We took 5% labeled data for our experiment maintaining the class ratio. From Table 5, we find that the proposed method (MST) outperformed the classical self-training (ST) in most of the metrics for all the classifiers, except for Quadratic discriminant analysis classifier.

Table 6. Glass dataset

Name of classifier	Micro avg. precision	ROC-AUC	Accuracy	F1 Score	Precision	Recall
Extra Tree(ST)	0.346	0.679	53.333	0.398	0.421	0.454
Extra Tree(MST)	0.543	0.811	60.000	0.564	0.577	0.585
GradientBoost(ST)	0.461	0.693	49.744	0.333	0.478	0.325
GradientBoost(MST)	0.574	0.786	57.436	0.483	0.528	0.523
Gaussian(ST)	0.531	0.747	46.667	0.404	0.391	0.428
Gaussian(MST)	0.542	0.758	49.744	0.493	0.591	0.490
LogisticRegression(ST)	0.367	0.731	45.641	0.279	0.274	0.325
LogisticRegression(MST)	0.540	0.788	52.821	0.456	0.519	0.476
QDA (ST)	0.278	0.569	32.821	0.164	0.165	0.234
QDA (MST)	0.543	0.662	53.333	0.405	0.621	0.409
SVM(ST)	0.342	0.760	47.692	0.493	0.531	0.498
SVM(MST)	0.414	0.793	58.462	0.528	0.550	0.536

Table 7. Iris dataset

Name of classifier	Micro avg. precision	ROC-AUC	Accuracy	F1 Score	Precision	Recall
Extra Tree(ST)	0.830	0.918	88.811	0.886	0.908	0.889
Extra Tree(MST)	0.961	0.980	90.909	0.909	0.910	0.909
GradientBoost(ST)	0.648	0.820	74.825	0.754	0.767	0.748
GradientBoost(MST)	0.836	0.965	79.021	0.796	0.813	0.790
Gaussian(ST)	0.359	0.653	35.664	0.222	0.779	0.361
Gaussian(MST)	0.732	0.732	61.538	0.543	0.812	0.618
LogisticRegression(ST)	0.847	0.974	67.832	0.571	0.832	0.674
LogisticRegression(MST)	0.926	0.989	89.510	0.894	0.907	0.896
QDA (ST)	0.888	0.953	83.916	0.838	0.839	0.838
QDA (MST)	0.668	0.682	51.049	0.442	0.776	0.514
SVM(ST)	0.871	0.950	87.413	0.870	0.894	0.873
SVM(MST)	0.984	0.979	88.112	0.879	0.893	0.882

3.6. Choosing final classifier for OBS network dataset classification

In Figure 3 F1-scores for the six classifiers (MST) is presented for each dataset. The final classifier 'F' in proposed MST is the classifier which is trained by the final labeled dataset. After careful observation of Table 2 to Table 7, we suggest 'Extra tree classifier' to be the final classifier (F) for MST, to be used for OBS network dataset classification. The reason for choosing Extra tree is that this classifier outperformed all other classifiers (in ST) when used as F in MST. Also, for small sized datasets (glass and iris), it performed better than other classifiers (in MST). Moreover, we want to keep our OBS network dataset as small as possible. Therfore, for OBS network node classification, the final classifier is selected to be 'Extra tree classifier' for MST.

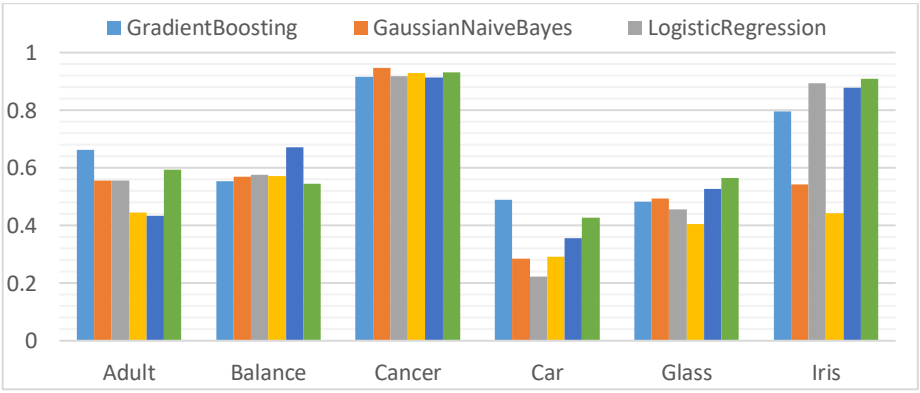

Figure 3. F1-scores of six classifiers (MST)

3.7. Classifying OBS network dataset using the proposed algorithm

We found an OBS network dataset from UCI machine learning repository which was made using NCTUns network simulator software. It has 1075 labeled instances and two target columns. In first target column, it has three labels for the instances: Behaving, not-behaving and potentially not-behaving. In the other target column, it has four types of labeling: Misbehaving-noblock, misbehaving-wait, misbehaving-block and

behaving-noblock. We made three test cases for our classes of interest. First, with two labels from first target column: Behaving and Not-behaving. Second, with an added label: Potentially not-behaving. Third, with four labels given in second target column. Using 'Extra tree classifier' mentioned in section 3.5, we applied the proposed algorithm on the dataset. The result is presented as shown in Table 8.

Table 8. Performance of MST on OBS dataset

Number of labeled instances in OBS dataset	Accuracy		
	2 Classes	3 Classes	4 Classes
23 (for 2 Class) 33 (for 3 Class) 150 (for 4 Class)	99.32%	89.45%	80.54%

3.8. Comparison with similar works

Table 9 presents a comparison between the proposed work and other similar works. The first column presents the reference of the work. Second column is for type of work. It is to indicate the paradigm of the work (ML or other). The 'Classification method' column is to indicate the algorithm used for labeling the nodes. 'Number of labeled instances' column indicates how many labeled instance required to produce the accuracy. From the table below result, we can say that, the proposed method performed very well considering the amount of labeled data instances, in comparison to other similar works.

Table 9. Comparison of proposed work with related work on OBS dataset

Works	Type of work	Classification method	Overall detection accuracy (%) and amount of labeled instances		
			2 Classes	3 Classes	4 Classes
Rajab et al. [5]	Supervised learning	Decision tree	93% with 1075 labeled instances	N/A	87% with 1075 labeled instances
Hasan et al. [6]	Supervised learning	Deep convolutional neural networks	N/A	N/A	99% with 1060 labeled instances
Patwary et al. [7]	Semi-supervised learning	K-means algorithm	90.2% with 152 labeled instances	65.15% with 215 labeled instances	41.61% with 215 labeled instances
Proposed work	Semi-supervised learning	Extra Tree classifier and modified self-training	99.32% with 23 labeled instances	89.45% with 33 labeled instances	80.54% with 150 labeled instances and 99.07% with 538 labeled instances

4. CONCLUSION

In this research, we investigated the BHP flooding attack in OBS network. We employed a SSML approach to classify OBS network dataset in order to detect malicious nodes. We modified the classical self-training algorithm using the concept of agreement of multiple classifiers in decision making. By employing two stage voting, we labeled some data points from the pool of unlabeled data and added them with original labeled data to produce a bigger labeled dataset. The performance of the proposed method was tested on six datasets against the classical self-training algorithm and we found significant improvement. Then, the proposed algorithm was used to classify an OBS network dataset using small amount of labeled data and it showed very good result. We compared the result with similar works and found that it outperforms them significantly. To the best of our knowledge, this work is among the earliest to propose a semi-supervised classification to the OBS network nodes. This method can be used to prevent BHP-flood-attack and improve the QoS for the OBS network. In near future, we aspire to further investigate the proposed method. The threshold values in the proposed algorithm need to be optimized. Besides, the choice of classifiers for agreement and defusion of their outcomes, need further study. Also, we intend to collect a real world OBS network dataset to test our method.

REFERENCES

[1] C. Qiao and M. Yoo, "Optical burst switching (OBS)-A new paradigm for an optical internet," *J. High Speed Networks*, vol. 8, no. 1, pp. 69-84, 1999.

[2] M. Sliti and N. Boudriga, "BHP flooding vulnerability and countermeasure," *Photonic Netw. Commun.*, vol. 29, no. 2, pp. 198-213, 2015.

[3] A. Rajab, C. T. Huang, M. Alshargabi, and J. Cobb, "Countering burst header packet flooding attack in optical burst switching network," *Int. Conference on Information Security Practice and Experience*, pp. 315-329, 2016.

[4] M. Sliti, M. Hamdi, and N. Boudriga, "A novel optical firewall architecture for burst switched networks," *12th International Conference on Transparent Optical Networks*, pp. 1-5, 2010.

[5] A. Rajab, C. T. Huang, and M. Al-Shargabi, "Decision tree rule learning approach to counter burst header packet flooding attack in Optical Burst Switching network," *Opt. Switch. Netw.*, vol. 29, pp. 15-26, 2018.

[6] M. Zahid Hasan, K. M. Zubair Hasan, and A. Sattar, "Burst header packet flood detection in optical burst switching network using deep learning model," *Procedia Computer Science*, vol. 143, pp. 970-977, 2018.

[7] M. K. H. Patwary and M. M. Haque, "A semi-supervised machine learning approach using K-means algorithm to prevent burst header packet flooding attack in optical burst switching network," *Baghdad Sci. J.*, vol. 16, no. 3, pp. 804, 2019.

[8] R. Sheikhpour, M. A. Sarram, S. Gharaghani, and M. A. Z. Chahooki, "A Survey on semi-supervised feature selection methods," *J. Pattern Recognit.*, vol. 64, pp. 141-158, 2017.

[9] X. Zhu, "Semi-supervised learning literature survey," *University of Wisconsin-Madison*, 2005.

[10] Y. Li, C. Guan, H. Li, and Z. Chin, "A self-training semi-supervised SVM algorithm and its application in an EEG-based brain computer interface speller system," *Pattern Recognit. Lett.*, vol. 29, no. 9, pp. 1285-1294, 2008.

[11] E. Riloff, J. Wiebe, and W. Phillips, "Exploiting subjectivity classification to improve information extraction," *Proceedings of the National Conference on Artificial Intelligence*, 2005.

[12] B. Wang, B. Spencer, C. X. Ling, and H. Zhang, "Semi-supervised self-training for sentence subjectivity classification," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 344-355, 2008.

[13] D.-H. Lee, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," *ICML 2013 Work. Challenges Represent. Learn.*, 2013.

[14] J. Tanha, M. v. Someren and H. Afsarmanesh, "Disagreement-based co-training," *IEEE 23rd International Conference on Tools with Artificial Intelligence, Boca Raton,*, pp. 803-810, 2011.

[15] J. Tanha, M. van Someren, and H. Afsarmanesh, "Semi-supervised self-training for decision tree classifiers," *Int. J. Mach. Learn. Cybern.*, vol. 8, no.1, pp. 355-370, 2017.

[16] I. E. Livieris, A. Kanavos, V. Tampakas, and P. Pintelas, "An auto-adjustable semi-supervised self-training algorithm," *Algorithms*, vol. 11, no. 9, pp. 1-16, 2018.

[17] S. Bhat and K. R. Kamath, "Effective learning with usage of simulators-a case of NCTUns simulator in Computer Networks," *Int. J. Sci. Res. Mod. Educ.*, vol. 1, pp. 415-420, 2016.

[18] M. Lichman, "UCI machine learning repository" 2013, [Online] Available at: http://archive.ics.uci.edu/ml.

[19] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Mach. Learn.*,vol 63, pp. 3-42, 2006.

[20] T. Hastie, R. Tibshirani, and J. Friedman, "The elements of statistical learning," *Springer Series in Statistics*, 2009.

[21] C. C. Aggarwal, "Data classification: Algorithms and applications," *CRC Press*, 2014.

[22] P. A. Lachenbruch and M. Goldstein, "Discriminant analysis," *Biometrics*, vol. 1, pp. 69-85, 1979.

[23] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*,vol.12, pp. 2825-2830, 2011.

[24] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. 1, 2016.

[25] P. P. Meghna Ghosh, "Empirical analysis of ensemble methods for the classification of robocalls in telecommunications," *International Journal Electrical. Computer. Engineering*, vol. 9, no. 4, pp. 3108-3114, 2019.

[26] W. Su, Y. Yuan, and M. Zhu, "A relationship between the average precision and the area under the ROC curve," *Proceedings of the ACM SIGIR International Conference on the Theory of Information Retrieval*, 2015.

[27] "Precision-recall curves," Yellobrick, 2020, [Online], Available at: https://www.scikit-yb.org/en/latest/api/classifier/prcurve.html. [Accessed: 09-Jan-2019].

[28] N. A. Obuchowski and J. A. Bullen, "Receiver operating characteristic (ROC) curves: Review of methods with applications in diagnostic medicine," *Phys. Med. Biol.*, vol. 63, no.7, 2018.

[29] P. PI and A. G, "A novel ensemble modeling for intrusion detection system," *International Journal Electrical. Computer. Engineering*, vol. 10, no. 2, pp. 1963-1971, 2020.

[30] "ROC-AUC," Yellowbrick, 2020, [Online], Available at: https://www.scikit-yb.org/en/latest/api/classifier/rocauc.html. [Accessed: 09-Jan-2019].

[31] D. Ballabio, F. Grisoni, and R. Todeschini, "Multivariate comparison of classification performance measures," *Chemom. Intell. Lab. Syst.*, vol. 174, pp. 33-44, 2018.