SLIC: Self-Learning Intelligent Classifier for Network Traffic

Dinil Mon Divakaran[1,*], Le Su, Yung Siang Liau, Vrizlynn L. L. Thing

Cyber Security and Intelligence,
Institute for Infocomm Research (I^2R),
Agency for Science, Technology and Research, Singapore.
1 Fusionopolis Way, #21-01 Connexis (South Tower), Singapore 138632

Abstract

Internet traffic classification plays an important role in the field of network security and management. Past research works utilize flow-level statistical features for accurate and efficient classification, such as the nearest-neighbor based supervised classifier. However, classification accuracy of supervised approaches is significantly affected if the size of the training set is small. More importantly, the model built using a static training set will not be able to adapt to the non-static nature of Internet traffic. With the drastic evolution of the Internet, network traffic cannot be assumed to be static. In this paper, we develop the concept of 'self-learning' to deal with these two challenges. We propose, design and develop a new classifier called Self-Learning Intelligent Classifier (SLIC). SLIC starts with a small number of training instances, self-learns and rebuilds the classification model dynamically, with the aim of achieving high accuracy in classifying non-static traffic flows. We carry out performance evaluations using two real-world traffic traces, and demonstrate the effectiveness of SLIC. The results show that SLIC achieves significant improvement in accuracy compared to the state-of-the-art approach.

Keywords: Classification, Learning, Traffic, Security, Protocol, Network

1. Introduction

Classification of application protocols in the ever changing Internet traffic is a fundamental problem in network security and management. The phenomenal growth of the Internet not only gives rise to new and different kinds of applications, but also leads to dynamic changes in network traffic over time. Accurate traffic classification is of paramount use in a number of network security problems such as, intrusion detection,

*Corresponding author

Email addresses: divakarand@i2r.a-star.edu.sg (Dinil Mon Divakaran), lsu@i2r.a-star.edu.sg (Le Su), liauys@i2r.a-star.edu.sg (Yung Siang Liau), vriz@i2r.a-star.edu.sg (Vrizlynn L. L. Thing)

detection and prevention of various network attacks (like Denial of Service attacks), anomaly detection, lawful inspection, cybercrime forensic analyses, etc. Traffic classification also plays an important role in network management, for example in traffic prediction and Quality of Service (QoS) provisioning. Network administrators find accurate classifier an indispensable tool for securing and managing their networks. The failure of the naïve and conventional port-based traffic classification in accurately identifying network application protocols has lead to the development of different solutions that exploit the flow-level statistical properties in network traffic.

Over the past many years, the research community has explored various machine learning techniques, such as supervised classification algorithms, unsupervised classification algorithms (clustering), and the combination of the two, namely semi-supervised classification, to provide accurate and efficient solutions for statistical features based flow-level traffic classification [2]. In a supervised traffic classification approach, for example the parametric Naive Bayes (NB), a number of labeled traffic flows called training instances are used to build a model for predicting the unlabeled flows. However, supervised traffic classification faces two fundamental issues.

1. Dependence on labeled training instances: One issue with supervised methods is, they are heavily dependent on the training data. When the number of available labeled training instances is small, or selected in a non-representative manner, the classification accuracy could be severely affected. Hence, most existing classifiers require large number of labeled data to achieve a high classification accuracy. On the contrary, in practice, manually labelling traffic flows for training purposes is a time-consuming task and requires strong expertise, and can yet be challenging for new applications and encrypted flows. It is thus motivating to provide an efficient classification method which could produce high accuracy with small number of training instances.

2. Inability to deal with non-static traffic: An observation from existing classification approaches (and not just supervised classification) comes to our attention. There is no utility of a test instance to the classifier: once a test instance (usually, a traffic flow) has been predicted by the model and subsequently assigned a label, it is no longer used in any further predictions. In other words, the classification model and thus its accuracy depends on the selection of the *static* set of labeled training instances. However, intuitively, an "intelligent" classification system should not be restricted to this static training set. Instead it should be able to *self-learn* from the predicted instances, to continuously update and evolve its training set as well as classification model. The aim of this learning process is not only to improve the prediction accuracy of the future instances, but also to steadily adapt to the evolution of the Internet traffic. Such an approach will bring significant improvement in accurately classifying continuously changing network traffic. Indeed, dealing with non-static data is identified as one of the 10 most challenging problems in data mining research [3]. With the prolific increase in the number of users connecting to the Internet, and the advancements being made in the connecting technologies, applications evolve rapidly over time [4]. This in turn leads to dynamic changes in Internet traffic. The need to deal with evolving network traffic is particularly important for classifiers which start with small number of training samples that might not be representative enough for application classes.

Some recent works have proposed new traffic classification algorithms, which utilize flow correlation information for more accurate classification with only a small

number of labeled training instances [5, 6]. However, designing a new classifier that not only starts with a small training set, but also learns on its own (self-learning), and yet achieves high accuracy is a challenging task. We take up this challenge in our work here, and introduce the concept of *self-learning* to solve the above mentioned two issues. We take our first step by coming up with a new classifier called Self-Learning Intelligent Classifier, or SLIC. This system is a combination of two main parts: a standard supervised classifier for predicting flow label, and a decision maker that decides and selects potentially useful flows for (re-)building the training set and the model. While previous solutions use aggregation of flows, or bag-of-flows (Section 3.1), to improve prediction accuracy, we use this concept for both prediction and self-learning. We show that SLIC performs significantly better than a k-Nearest Neighbor (k-NN) based classifier that also exploits correlation in traffic flows. This k-NN based classifier was recently developed to work with small number of labeled data, and was shown to have high accuracy [5]. The performance improvement achieved by SLIC is much more valuable when seen in the light that, unlike existing classifiers, SLIC is dynamic in nature and is capable of learning and building a model when faced with realistic dynamic traffic.

With this work, we make the following major contributions:

- We propose a new direction of research in traffic classification, where the challenge is to build a classifier that requires only a small number of labeled data to start with, and yet can learn and adapt on its own to classify application protocols in dynamically changing network traffic with high accuracy.
- We formally propose and develop a novel network traffic flow classification system, called Self-Learning Intelligent Classifier (SLIC), that learns and evolves on its own, to improve the overall classification accuracy over time.
- We carry out extensive evaluation of SLIC using two real-world traffic traces. We compare SLIC with a k-NN based classifier, where the only difference between the two classifiers is that the latter works in a traditional way with no self-learning process.

The remainder of the paper is organized as follows. In Section 2, we provide a brief review of related works on network traffic classification. Section 3 briefs on relevant preliminary concepts. We give an overview of SLIC in Section 4, and develop the proposed system in Section 5. Section 6 presents performance evaluation and results.

2. Related Work

Over the last decade, a large number of research works have been proposed on the application of machine learning techniques to statistical feature-based network traffic flow classification. Each traffic flow is represented as a vector of different features such as packet-size statistics (min, max, mean, etc.), connection-size statistics, inter-packet time, etc. We briefly review these works. For more details, interested readers may refer to the survey by Nguyen and Armitage [2].

In supervised classification, a model is built using labeled training flows, which is later used to predict or map future unlabeled flows into one of the classes. Among the pioneering works under this category, Moore and Zuev [7] uses supervised Naive Bayes techniques to classify network traffic flows based on statistical features. By extending [7], Auld *et al.* [8] utilized Bayesian neural networks for achieving more accurate results. Naive Bayesian classifier was also used to identify Skype connections with encrypted data [9]. Kim *et al.* evaluated a set of widely used supervised classifiers, including Naive Bayes, Naive Bayes Kernel Estimation, Bayesian Network, Neural Networks, k-Nearest Neighbors (k-NN) and Support Vector Machines (SVM) [10]. The two parametric classifiers, SVM and Neural Networks, and the non-parametric k-NN were the top and closely performing classifiers. Roughan *et al* used LDA (Linear Discriminant Analysis) to perform low-error class-of-service mapping of network traffic (into interactive, bulk data transfer, streaming and transactional classes) [11]

Recently, a series of work by Zhang *et al.* [6, 5, 12] utilized flow correlation information to further improve the classification accuracy with small number of training instances based on k-NN and Naive Bayes classifier. As our solution also exploits flow correlation, we compare SLIC with [5] in our study.

In contrast to supervised methods, the unsupervised approach, or known as clustering, takes as input a set of unlabeled flows and automatically groups them into different clusters, and subsequently assigns any testing flow into its nearest cluster. McGregor *et al.* [13] proposed an approach that utilized expectation-maximization algorithm to form clusters. Zander *et al.* used AutoClass [14] to group traffic flows, and Bernaille *et al.* [15] explored k-means clustering mechanism and mapped each cluster to an application through payload analysis tool. Erman *et al.* [16] provided a comparison among three clustering algorithms, namely k-means, DBSCAN and AutoClass. Their study showed clustering could produce high-purity clusters when the number of allowed clusters is set to be much larger than the number of real applications. In [17], the authors apply GMM (Gaussian Mixture Models) and HMM (Hidden Markov Models) to effectively perform early identification of application class at the beginning of a TCP connection. There are also attempts to use a hybrid combination of flow statistical feature-based clustering algorithms with other approaches. For example, Wang *et al.* integrated clustering algorithm with payload-based signature matching method [18].

However, there is one significant drawback of clustering-based approaches. As mentioned above, in order to obtain high-purity clusters, the allowed cluster number needs to be set much higher than the number of real applications, it thus creates difficulty in mapping large number of clusters to small number of applications. Without the knowledge of each application, this drawback is hard to overcome. Moreover, the fundamental difference between SLIC and clustering-based approach is that SLIC is still a supervised learning-based method, and naturally the hassle of mapping clusters to applications is not applicable to our system. In their seminal work, Erman *et al.* [19] proposed a hybrid approach by combining the ideas of both supervised and unsupervised methods, which embedded a small amount of labeled flows into large amount of unlabeled ones. The hybrid approach first performs an unsupervised clustering process and groups unlabeled flows into small clusters, subsequently with the aid of the labeled flows, clusters are then mapped into applications. Following this work, Li *et al.* proposed a semi-supervised classification based on support vector machine [20],

and Wang *et al.* [21] utilized flow-level side information instead of labeled samples during clustering. Very recently, Grimaudo *et al.* [22] proposed a learning-based semi-supervised approach using k-means clustering, to adapt to the changes in the network traffic. However their approach does not consider small size flows, besides facing the traditional clustering disadvantages, for example, of mapping large number of clusters to application classes. One drawback common to the semi-supervised approaches is that they need to aggregate a large number of unlabeled flows to produce high purity clusters. On the contrary, SLIC being a supervised approach, the above mentioned constraint does not apply to it, and can therefore be easily deployed for online classification.

A particular concept called online machine learning [23, 24] is worth a special mention here. It is a supervised learning approach where data is available in a sequential order. Although the ultimate goal for online machine learning is similar to the traditional supervised classification, which is to determine a mapping of data points to class labels, there is one major difference. In online learning the training set and its corresponding prediction model are updated after the arrival of every new data (as mentioned earlier, the data is available in a sequential order), while in traditional approach, one has access to the entire and fixed training set at once.

At first glance, online machine learning may appear to be quite similar to our proposed SLIC system: both have a dynamic training set and a continuously evolving prediction model aiming for better accuracy. However, for online machine learning, once a new data point arrives and has been predicted, its true class label is *revealed* and used for updating the training set and prediction model, before the next data arrives. This is a rather restrictive scenario as the true label of the data might not be available immediately after prediction, or even not available at all. SLIC, on the other hand, performs self-learning and updates the model *without* any knowledge of the true label. That is, the ground truth of predicted instances are never known to SLIC. Under this challenging scenario, SLIC needs to be designed in such a way that enables it to select good representatives from predicted data for learning and improving the classifier.

3. Preliminary

In this section, we briefly recall some preliminary concepts that are used throughout this paper.

3.1. Bag-of-Flows (BoF)

An important concept that we adopt for our solution is bag-of-flows—an aggregation of flows or connections based on correlation. As is well-known, a flow is usually defined by the five-tuple of source and destination IP addresses, source and destination ports and protocol. In conventional classification, traffic flows are taken into the classifier as individual and independent instances one by one. However, as mentioned and shown in [5], the correlation information between an aggregation of flows can significantly improve the classification accuracy, especially when only small training sets are available. The correlation information used, or more concretely, the way to construct a correlated aggregation of flows, is based on the "three-tuple heuristic"—in a

certain period of time, the flows sharing the same three-tuple {destination IP address, destination port, protocol} form an aggregation.

The concept of using the three-tuple heuristic for flow correlation has been considered in several previous works [25, 26, 21]. We adapt the same definition and notation as [5], and refer to a correlated aggregation as bag-of-flows (BoF).

Definition 1. *A BoF is a set of traffic flows localized in time, and having the same three-tuple of {destination IP address, destination port, protocol}.*

In subsequent content, we denote a BoF by

$$\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$$

where $\mathbf{x}_i$ is a single flow represented as a feature vector. Note, the size for a BoF is not fixed; rather the size depends on how the flows are aggregated to form BoFs, and might vary from one BoF to another.

During classification, each flow $\mathbf{x}_i$ in $\mathbf{X}$ runs through a classifier and is given a class label. Once every flow in $\mathbf{X}$ obtains a class label, the label of BoF is decided based on some criteria, for example, majority voting; and subsequently every instance in $\mathbf{X}$ is assigned the same class label. In the following, we also use *bags* to refer to BoFs.

3.2. Euclidean Distance

Our SLIC is built based on a k-NN classifier, which calculates the distances between the feature vector of a test instance and its labeled neighbours, to decide on the label for the test instance. Equally importantly, we use a distance metric in the learning phase of SLIC (as explained in Section 5.2). For both purposes, we adopt the commonly used Euclidean distance defined below.

For two vectors $\mathbf{u}, \mathbf{v}$ with the same dimension n, the Euclidean distance is computed as,

$$\text{dist}(\mathbf{u}, \mathbf{v}) = \sqrt{\sum_{i=1}^{n} (u_i - v_i)^2}, \tag{1}$$

where u_i and v_i denote the i^{th} component of vectors $\mathbf{u}$ and $\mathbf{v}$, respectively.

3.3. Centroid

We also define the concept of "application class centroid" in the proposed SLIC model. Informally, in an n-dimensional space, a centroid of a set of points is the arithmetic mean in all coordinates of all points in the set. Formally, given a set $\mathbf{S}$ of n-dimensional vectors $\mathbf{s}_j$, where $j \in [1, |\mathbf{S}|]$, the centroid $\mu_{\mathbf{S}}$ of this set $\mathbf{S}$ is calculated as,

$$\mu_S = \frac{1}{|\mathbf{S}|} \sum_{j=1}^{|\mathbf{S}|} \mathbf{s}_j$$

where the addition and average is taken component-wise.

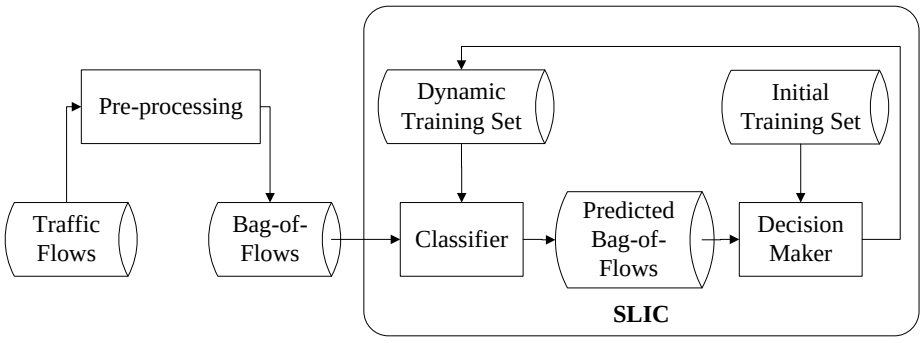


Figure 1: SLIC system model

4. System Model Overview

In this section, we provide an overview of our proposed system. Fig. 1 depicts the system. Incoming traffic flows are pre-processed to construct bag-of-flows. SLIC consists of the classifier and the decision maker. First, the classifier builds a model based on the training set, and classifies BoFs. The classifier in SLIC is different from the traditional classifier in two ways:

- The model using which the classifier predicts the application class for a given BoF is constantly updated; and this process continues as long as SLIC decides to learn.
- In SLIC, the model is built and predictions are made on individual flows. But, the final label assigned to a flow is based on the label assigned to the bag it belongs to. This is described in 5.2.

The second and the most important part of SLIC is the decision maker. The decision maker decides which among the test instances are good enough to be inducted into the training set. One naïve approach to design a self-learning classifier would be to take every predicted instance into the training set and rebuild the model based on the updated training set. However, this approach suffers from serious drawbacks: (i) this greatly affects the efficiency of the classifier, as it has to perform the process of rebuilding and regenerating a model for the classifier on a per-bag basis; (ii) wrongly predicted BoFs will be added into the training set and can subsequently contribute a negative effect on the prediction accuracy for future instances. Instead of doing self-learning in a good way to improve accuracy, the classifier might potentially perform even worse. The possibility of such a negative impact is particularly high when the initial training set size is small.

An ideal decision maker should be able to select good candidates for rebuilding the model. By setting certain decision criteria, the system needs to decide which test instance, after being predicted and assigned a class label, is potentially good for improving the prediction. Such candidates should be inducted into the training set for

Table 1: Notations

$\mathcal{I}$	Initial (labeled) training set
$\mathcal{D}$	Dynamic training set; $\mathcal{D} \supset \mathcal{I}$
$\mathcal{T}$	Testing set
A	Set of application classes
N	Maximum (permitted) size of dynamic training set $\mathcal{D}$

rebuilding the model. Designing such decision criteria is the core part of our contribution. On one side, inevitably, we should allow the system to make mistakes occasionally, but not too many. The portion of wrongly predicted BoFs that pass the decision criteria, and have subsequently been added into the training set, should be small and make no significant adverse effect on the overall accuracy. On the other hand, the decision criteria should not be too strict such that only a few instances are inducted; this would make self-learning an extremely slow process.

To minimize error, we design the decision criteria to be dependent only on the initial training set, which is a static and small set of labeled data. The initial training set consists of data with ground truth; and therefore this is the best minimal set based on which we can decide on the candidates for induction. The test instances that pass the decision criteria are inducted into and to form the dynamic training set. Each time the dynamic training set inducts a set of test instances, the model is rebuilt. We detail the system in the following section below.

5. Self-Learning Intelligent Classifier (SLIC)

SLIC builds its first model using a small set of labeled data, which we call the initial training set $\mathcal{I}$. Let A be the set of application classes, with $|A|$ denoting the number of classes. Each element of $\mathcal{I}$ is a tuple $(\mathbf{x}, a)$, where $\mathbf{x}$ is a flow (feature vector) and $a \in A$ is an application class. For each $a \in A$, $\mathcal{I}_a \subset \mathcal{I}$ is the subset of the initial training data belonging to class a. Obviously, $\mathcal{I}_a$s form a partition of $\mathcal{I}$. In implementation, $\mathcal{I}_a$ does not maintain label information (as all labels are the same, and it is the label a); therefore an element of $\mathcal{I}_a$ is not a tuple, but a flow of class a.

The dynamic training set, denoted by $\mathcal{D}$, is a superset of the initial training set $\mathcal{I}$. SLIC rebuilds its model as and when $\mathcal{D}$ changes. SLIC uses k-NN as the underlying classifier; therefore, the computational complexity of SLIC is dependent on the size of the training set. For this reason, the rebuilding of the model stops when the size of the dynamic training set $\mathcal{D}$ reaches a pre-defined limit N. The testing set is denoted by $\mathcal{T}$. As mentioned before, each instance in the testing set is a bag-of-flows. Table 1 lists the notations that are commonly referred in this paper.

In the following, we develop SLIC. After discussion the training phase in Section 5.1, we describe the classifier and the decision maker in Section 5.2.

5.1. Training

Algorithm 1 gives the steps for training and building a model for the classifier in SLIC. Comments in the algorithms are preceded by '##'.

Algorithm 1 train(m, *trace*)

1: **for** each $a \in A$ **do**
2: $\quad \mathcal{I}_a \leftarrow$ Initialize training set for class a with m unique flow instances from trace
$\quad$ ## $\mathcal{I}_a = \{\mathbf{x}_1^a, \mathbf{x}_2^a, \ldots, \mathbf{x}_m^a\}$
3: $\quad \mu_a \leftarrow$ centroid($\mathcal{I}_a$)
4: **end for**
5: $\mathcal{I} \leftarrow \bigcup_{\forall \mathbf{x} \in \mathcal{I}_a, \forall a \in A} (\mathbf{x}, a)$
6: $\mathbf{M} \leftarrow$ build_kNN_model($\mathcal{I}$)
7: **return**

The first step in the **for** loop segregates the given traffic trace with labeled data to form an initial training set for each application class $\mathcal{I}_a$ ($a \in A$) consisting of m unique flows. In line 3, we compute centroids of application classes (refer Section 3.3); centroids later (in Algorithm 5) play an important role in deciding whether a test instance is 'good' enough to be inducted into the dynamic training set $\mathcal{D}$. Line 5 constructs the initial training set by taking the union of all tuples. The initial training set $\mathcal{I}$ is then passed to the function *build_kNN_model* that builds a model based on the k-NN classifier. This model is used for predicting the test instances later, in Algorithm 2.

Strictly speaking, there is no training phase for a k-NN based classifier, as it is a non-parametric approach. Thus, the *build_kNN_model* function merely aggregates the labeled instances and forms the training set. However, for ease of exposition, we still name this process as training phase.

Algorithm 1 is executed only once, but the model $\mathbf{M}$ built in this algorithm will be rebuilt multiple times later.

5.2. Classifier and Decision maker

Once the initial model is built, SLIC performs the prediction of test instances and improves the model on its own, as per Algorithm 2. After providing a high level description of Algorithm 2 (*slic*) below, we discuss the underlying algorithms.

5.2.1. slic

In Algorithm 2, SLIC begins by initializing the dynamic training set $\mathcal{D}$ to the initial training set $\mathcal{I}$. Looping over all test instances, in line 4, the function *classifier* (Algorithm 3) assigns the test instance $\mathbf{X}$ a class label based on the model $\mathbf{M}$. Subsequently, in line 5, the function *decision_maker* (details given in Algorithm 5) decides if the predicted instance should be inducted into the dynamic training set. If the decision is positive, the test instance is considered as an element of the candidate induction set, denoted as $\mathbf{C}$, and stored in a batch (line 6). When the batch size reaches a maximum value $b^{\max}$, a subset of the stored batch of instances is inducted in line 9. The algorithm *induct* is described in Algorithm 4. After inducting new instances into the dynamic training set $\mathcal{D}$, the model $\mathbf{M}$ has to be regenerated, and this is achieved by the function *rebuild_kNN_model* in line 10.

Observe that the model built in the training phase uses data with true labels (line 6, Algorithm 1); but the model rebuilt in the testing phase here uses data with predicted

Algorithm 2 slic($\mathbf{M}, \mathcal{T}, \mathcal{I}$)

1: $\mathcal{D} \leftarrow \mathcal{I}$ ## Initialize dynamic training set
2: $b^s \leftarrow 0$ ## Initialize size of inducted batch of bags
3: **for** each $\mathbf{X} \in \mathcal{T}$ **do**
4: p_label $\leftarrow$ classifier($\mathbf{M}, \mathbf{X}$)
5: **if** (decision_maker($\mathbf{X}$, p_label) == True) **then**
6: store_batch($\mathbf{C}, \mathbf{X}$, p_label)
7: increment b^s
8: **if** $b^s == b^{\max}$ **then**
9: induct($\mathcal{D}, \mathbf{C}$)
10: $\mathbf{M} \leftarrow$ rebuild_kNN_model($\mathcal{D}$)
11: $b^s \leftarrow 0$
12: **end if**
13: **end if**
14: **end for**
15: **return**

labels. One advantage of k-NN is that, similar as the *build_kNN_model* in Algorithm 1, rebuilding is inexpensive as it does nothing more than adding more data to the existing set.

Next, we provide detailed descriptions of the three important algorithms used in Algorithm 2, namely *classifier*, *induct*, and *decision_maker*.

Remark: For simplicity, Algorithm 2 assumes a finite testing set. However, it is straightforward to modify the algorithm to work on a continuously flowing stream of data.

5.2.2. classifier

Recall, a test instance is a bag-of-flows (BoF). In Algorithm 3 for *classifier*, the class label of a test instance $\mathbf{X}$ is predicted in line 3. For deciding on the application class of a BoF, we use k-NN to predict each of the flows in the given BoF. The label to which the maximum number of flows belong to, is returned as the label of the BoF (line 6). Here we use the simple majority voting for assigning label to the BoF, and ties are broken randomly. Note, the k-NN prediction in line 3 of Algorithm 3 is based on the specific model $\mathbf{M}$ which is not static. As explained later, the model $\mathbf{M}$ is regenerated multiple times until the dynamic training set inducts no more instances.

Algorithm 3 classifier($\mathbf{M}, \mathbf{X}$)

1: label_count: Array$[1, \ldots, |A|]$
2: **for** each $\mathbf{x} \in \mathbf{X}$ **do**
3: $p \leftarrow$ kNN_predict($\mathbf{M}, \mathbf{x}$)
4: increment label_count[p]
5: **end for**
6: **return** $\arg\max_p$ label_count ## Break ties randomly

Algorithm 4 induct($\mathcal{D}$, $\mathbf{C}$)

1: $l \leftarrow \arg\min_a |\mathbf{C}_a|$ ## application class with min. no. of inducted instances
2: max_induct $\leftarrow (1 + y\%) \times |\mathbf{C}_l|$
3: $\mathbf{B} = \emptyset$ ## Set of instances that will be inducted
4: **for** each $a \in A \backslash l$ **do**
5: **if** $|\mathbf{C}_a| >$ max_induct **then**
6: $\mathbf{B} \leftarrow \mathbf{B} \cup \{$randomly selected max_induct no. of instances from $\mathbf{C}_a\}$
7: **end if**
8: **end for**
9: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathbf{B}$
10: Reset $\mathbf{C}$ and $\mathbf{B}$ to $\emptyset$

5.2.3. induct

The most important decision on inducting a test instance is taken by the function *decision_maker*; however, this in itself is not sufficient to guarantee good performance. A common problem that impedes performance in learning systems is the imbalance in class distribution [3]. Imbalanced or skewed class distribution means that the number of data in one class is significantly different from other classes. The performance of k-NN too is known to be affected by this problem, and there are solutions to tackle this [27, 28]. Thus, when SLIC inducts new test instances into the training set, Algorithm 4 ensures that the resulting dynamic training set is not considerably imbalanced. More precisely, the algorithm *induct* controls the imbalance in the dataset.

In Algorithm 4, line 1 finds the label that has the minimum number of induction candidates in the given batch $\mathbf{C}$. Set this label to be l; and the number of flows with this label (classified as l) is $|\mathbf{C}_l|$. In the following steps, the algorithm ensures that the difference in the number of data in any two classes is within $y\%$ of $|\mathbf{C}_l|$. Therefore, the maximum number of instances that is inducted in any class with label $a, a \neq l$, is $(1 + y\%) \times |\mathbf{C}_l|$. If the number of candidates of a particular label is more than that can be inducted (following the above mentioned condition), the algorithm in line 6 selects the necessary number of data randomly. We note that there are other ways of choosing from the candidates. For example, one approach is to rank the candidates according to their distances to the class centroid, and select $(1 + y\%) \times |\mathbf{C}_l|$ with the shortest distance. However, we decide to have a simple solution of choosing the candidates randomly. The computational complexity of induction for each bag is $\mathcal{O}(1)$, and hence that of a batch of candidates is $\mathcal{O}(b^{\max})$. As SLIC is a continuously learning process, the induction will not stop until the maximum number of training instances allowed is reached.

5.2.4. decision_maker

The algorithm *decision_maker* is the most important and distinguishing feature of SLIC. It decides whether a given (and predicted) test instance is a good candidate that can be inducted into the dynamic training set $\mathcal{D}$. Observe that a test instance has already been classified as belonging to one of the application classes before facing the decision on induction. It is worth noting that this label is not provided as ground truth, but

Algorithm 5 decision_maker(**X**, p_label)

1: μ_a : Centroid of flows **x**, **x** $\in \mathcal{I}_a$
2: **if** size($\mathcal{D}$) $> N$ **then**
3: **return** False
4: **end if**
5: $\mu_{\mathbf{X}} \leftarrow$ centroid(**X**)
6: **for** each $a \in A$ **do**
7: $\mathbf{d}[a] \leftarrow$ dist($\mu_a, \mu_{\mathbf{X}}$) ## refer Eq. (1)
8: **end for**
9: $l \leftarrow \arg\min_a \mathbf{d}$
10: **if** $l \neq$ p_label **then**
11: **return** False
12: **end if**
13: sort($\mathbf{d}$) ## $\mathbf{d}[1] = \mathbf{d}[l]$
14: **if** $\theta \times \mathbf{d}[1] > \mathbf{d}[2]$ **then**
15: **return** False
16: **end if**
17: **return** True

rather the one predicted by SLIC; therefore, the label need not be correct. This differs from the online learning approaches in the literature (refer to Section 2). For SLIC to improve its classification performance, the challenge is to induct test instances that are "good".

Our design of the criteria for inducting a given test instance is inspired by the k-means clustering [29]. At a high level, we measure the distance between the centroid of a given test instance (recall, a test instance is a BoF) to centroids of all classes, and induct the instance only if it is sufficiently near to the centroid of the predicted class. The distances to the centroids of different application classes are used to estimate the goodness of a test instance. The detailed steps are as given in Algorithm 5. We explain the steps.

It is important to note here that the centroids of application classes are computed using the initial training set $\mathcal{I}$ (see line 3 of Algorithm 1). That is, centroid μ_a of an application class $a, a \in A$, is dependent only on its initial training set $\mathcal{I}_a$, and is therefore pre-computed as well as static. The first conditional statement in Algorithm 5 checks if the size of the dynamic training set exceeds the limit specified by N; and if so, no more instance will be inducted. The size of $\mathcal{D}$ may still go beyond N after a round of induction. This is not a problem, because the batch size limit $b^{\max}$ can be used to ensure that exceeded size of $\mathcal{D}$ is not, say, 5% more than N.

As a test instance is a bag-of-flows, we also compute its centroid; this is done in line 5 of Algorithm 5. Next, in the **for** loop of the algorithm, using Eq. (1) we compute the Euclidean distance of the centroid of the given test instance **X** to the centroid of each of the classes. There are two conditions that need to be satisfied for a BoF instance to be inducted. First, the label of the application class that is nearest to the BoF instance should be the same as the label *p_label* assigned to the BoF by SLIC (lines 9-12).

Second, in the case where the first condition is satisfied, the distance to the second nearest application class should be θ times more than the distance to the nearest class (lines 13-16). Only if both criteria are satisfied, will SLIC decide to induct the test instance (line 17).

Though Algorithm 5 consists of a loop and a sort, they run in constant times as the number of applications and features are constant values. Therefore, the complexity of *decision_maker* is dominated by line 5, which is linear in the number of flows in a bag. However, as the maximum size of bags is also fixed at the beginning of classification, and is typically small, the algorithm takes constant time to decide on a given test instance.

We now conclude the descriptions of all algorithms in the proposed SLIC system.

6. Performance Evaluation

In this section, we evaluate SLIC over two real-world traffic datasets, and compare the performance with the BoF-based classifier proposed in [5]. The prediction of a bag-of-flows instance based on the majority of the decisions on flows (constituting the bag) is one of the specific solutions in [5]. As this naturally appears like an extension of k-NN to bag-of-flows classification, in the following, we refer to this classifier simply as k-NN.

SLIC is designed to function as an online traffic classifier; therefore, in the next section, we present a few necessary changes that need to be applied to SLIC for offline performance evaluation. The information on the datasets and the metrics used for evaluations are given in sections 6.2 and 6.3, respectively. After discussing the settings for the experiments in Section 6.4, we present and discuss the results in Section 6.5.

6.1. Offline testing

As opposed to an online scenario where there is continuous stream of network traffic flows, only finite number of flows collected as static traffic traces (detailed in Section 6.2) are available for offline evaluation. In this section, we explain the modifications applied to the algorithms described in Section 5 to effectively perform offline evaluation, and argue that these changes will have no significant impact over the online performance of SLIC.

6.1.1. Modification 1

The first modification is to create the testing set. In traditional classifiers, creating a testing set is not a concern, as once the training instances are randomly selected from the traces, the remaining data can be used for testing. However, SLIC is a continuously learning classifier that inducts predicted instances from the testing dataset. For unbiased prediction, the inducted instances have to be removed from the testing data, thereby shrinking its size. In effect, if the traditional segregation is applied, the evaluation of SLIC will be over a reducing testing data, making the evaluation difficult and even inaccurate. Such a problem is not seen in existing works—as there is no learning phase in them, the classification is performed over the testing set only once. To overcome this problem, in (each run of) an experiment, we first randomly select the

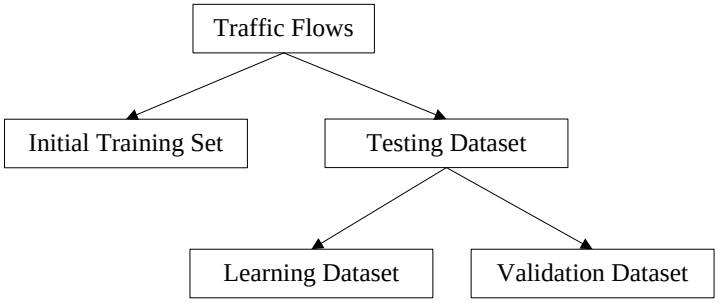


Figure 2: Separation of traffic trace into different datasets

training samples from the traces to form the initial training set $\mathcal{I}$. The remaining data is then randomly separated into two disjoint and smaller sets, which we call *learning* and *validation* datasets. Refer to Fig. 2 for an illustration of the separation procedure of a traffic trace.

The learning dataset is dynamic, and SLIC inducts test instances only from the learning dataset for self-learning. The inducted test instances are removed from the learning set; therefore, the number of instances in the learning set reduces over time. The validation set (comparable with the standard "testing set") remains static for the entire execution; and it is used for the sole purpose of evaluating the classifiers, (BoF-based) k-NN and SLIC. In the beginning of an execution, before any test instance is inducted (i.e., before the learning begins), the validation set is evaluated for classification results using the initial training set $\mathcal{I}$. As this happens before SLIC comes into play, this classification corresponds to k-NN and the results obtained are thus due to the (BoF-based) k-NN classifier. Then onwards, SLIC inducts data from the learning dataset and rebuilds its model over iterations. The last evaluation on the validation dataset using the final model gives the classification results for SLIC.

Two additional points need to be noted for the above mentioned testing setup. First, to validate that the learning and validation datasets show no particular bias, at the start of an experiment, the initial training set is also used to predict the flows in the learning dataset. The accuracy over the learning dataset is compared with the one obtained on the validation dataset. Our experiments show, the two accuracies vary only within 1% difference, which is normal due to randomness. This justifies the separation is valid. Second, the separation of the testing dataset does not necessarily need to be on a equal-sized basis. We also tested on a 60% learning - 40% validation separation, and the performance showed no significant difference. We believe, for a large dataset, as long as a separation is not too extreme (such as 90% - 10% separation), both datasets would have sufficient flows to make them representative.

6.1.2. Modification 2

The second modification is on the simulation of the batch learning process. As mentioned earlier in Section 5.2, BoFs are inducted in batches. Once a batch of test instances is inducted, the model used for prediction is regenerated. Since a newly

regenerated model is (presumably) better than its predecessor, there is a higher chance of predicting the right class labels of newly arriving BoF instances. This eventually increases the chance of inducting a good predicted instance. One can visualize this to work smoothly in an online setting, where there is continuous stream of traffic flows arriving at the classifier. However, in offline testing, there is only a finite number of flows (and hence BoFs), and the improvement attained by going through this finite test data might be limited.

To simulate an online scenario, we introduce *learning iteration*—SLIC iterates over the learning set multiple times. At the beginning of each iteration, when the predicted BoFs run through the decision criteria, all BoFs that satisfy the constraints are stored as a batch of candidates for induction. At the end of the iteration, Algorithm 4 (*induct*) is used to select a set of BoFs from this batch. These selected bags are inducted into the dynamic training set $\mathcal{D}$, and the model updated. At the same time, the inducted BoFs (or equivalently, individual flows in the bags) are deleted from the learning dataset. This completes one learning iteration. In the next round, SLIC iterates on the remaining learning dataset, to check if it can induct more BoFs with its new model, and thereby continue learning. This continuing process perfectly simulates online classification—bags that could have potentially been rejected in one iteration may pass the decision criteria in the next iteration, due to an improved model. The modification also simulates the impact of the learning process. However, as there is no continuously flowing data stream, we need to add additional stopping criteria for the learning process. In the online setting, the induction (and hence model regeneration) stops when the size of the dynamic training set $\mathcal{D}$ reaches the predefined limit N. In the offline setting, having just this condition might lead to large number of iterations and longer execution times. Hence, for offline setting, we decide to stop the iterations when one of the following conditions is met:

- when the size of the dynamic training set reaches N;
- when the number of iterations over the learning dataset reaches η, a pre-defined threshold;
- when no test instance is inducted in an iteration over the learning dataset.

Note that if the third condition is met, the model does not change, and hence there is no point iterating again. When the iteration stops due to one of the above three conditions, the learning in SLIC comes to an end, and the last generated model becomes the final one. We highlight, this modification is purely for offline testing; in online classification, the learning has no reason to stop until the maximum training size of N is reached (and this limit is for computational efficiency). Fig. 3 illustrates the working of SLIC with the above mentioned modifications.

6.2. Traffic traces

For testing, we use two large-scale real network traffic flow traces that were recently used to evaluate new classifiers [5, 6]. The *isp* trace is a full payload traffic dataset collected at a 100 Mbps edge link of a small-medium ISP located in Melbourne, Australia during Nov-Dec 2010 [21]. It contains around 200,000 traffic flows. The *wide* trace

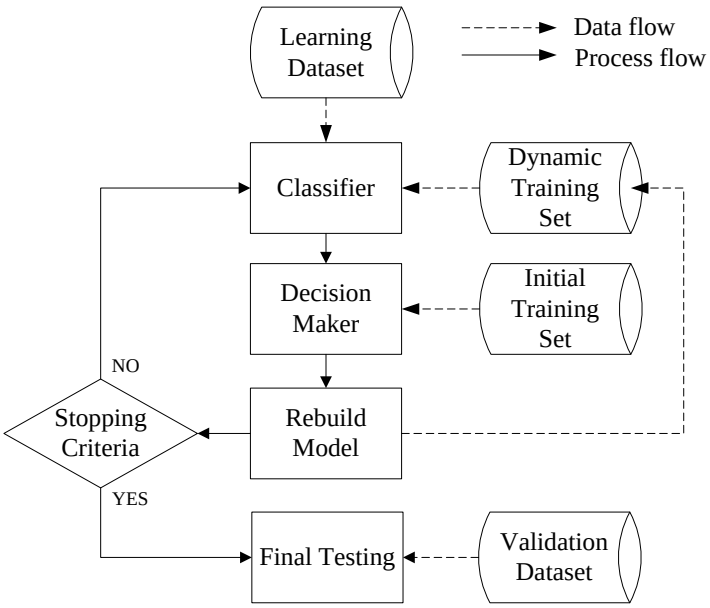


Figure 3: Illustration of offline version of SLIC

was taken from a US-Japan trans-Pacific backbone line with a 150 Mbps Ethernet link that carries commodity traffic for the WIDE organization [30]. The trace was collected at 72-hour basis from 18-20 March 2008 and 30 March - 4 April 2009. The *wide* trace contains around 1.6 million flows. Since the number of flows across applications is highly skewed in both traces (with differences in orders of magnitude), we extracted flows in such a way so as to ensure that the maximum number of flows in any class is not more than five times of any other classes. After this, protocols with insufficient flows were not considered for evaluation.

The ground truth for the flows in both traces are obtained via deep packet inspection (DPI) tools, such as l7-filter [31] and Tstat [32]. The *isp* trace has seven application protocols, each with sufficient number of flows for meaningful testing: BitTorrent, HTTP, IMAP, POP3, SMTP, SSH, XMPP. The *wide* trace has been categorized into five application classes, four of which are the protocols BitTorrent, DNS, HTTP, and FTP, and the fifth one is an aggregate class *Mail*. The Mail class combines the protocols POP3 and SMTP. The reason for combining the two mail protocols into one single application is because POP3 on its own does not have sufficient data in *wide* compared to other protocols. Moreover, as mail protocols are quite similar, grouping them into a larger Mail category is justified.

We consider bidirectional flow features for classification. In both traces, the same set of 20 statistical features are available. The features can be broadly seen to be as one of the two types, namely client-to-server (c2s) and server-to-client (s2c). These features cover wide range of properties of real network traffic flows, for example number

Table 2: Features used for the two datasets, *isp* and *wide*

isp	*wide*
c2s maximum packet size	c2s maximum packet size
c2s minimum inter-packet time	c2s packet size standard deviation
s2c minimum inter-packet time	c2s volume of bytes transferred
s2c maximum packet size	s2c maximum packet size
s2c minimum packet size	s2c packet size standard deviation
s2c packet size standard deviation	s2c volume of bytes transferred

of packets/bytes transferred, minimum/maximum/average values of packet sizes and inter-packet times, and their standard deviations. However, not all features mentioned above are used for testing. Feature selection is an important data pre-processing step to remove redundant and irrelevant features. Feature reduction is also useful in reducing the running time of the solutions, especially for the nearest-neighbor-based classifiers due to the "curse of dimensionality". Random forest (RF) has been shown to be an efficient and accurate approach for finding the important features [33]. We run RF on both traces and on each trace RF produces a list of *(feature, importance)* pairs. We select six features with the highest importance from these two lists and use them for evaluations. The six selected features for *isp* and *wide* traces are given in Table 2.

6.3. Metrics for evaluation

To evaluate the performance of SLIC, we use two commonly and widely adopted metrics in classification, namely overall accuracy and F_1 score. The overall accuracy is the ratio of the sum of correctly predicted BoFs (across all application classes) to the size of the population of BoFs. The F_1 score for a class is defined as:

$$F_1 \text{ score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}.$$

Precision and recall for an application class are defined as:

$$\text{precision} = \frac{\#\text{True Positive}}{\#(\text{True Positive + False Positive})};$$

$$\text{recall} = \frac{\#\text{True Positive}}{\#(\text{True Positive + False Negative})}.$$

Precision gives the fraction of correctly predicted instances of all those predicted for (and as) a class; recall is the fraction of correctly predicted instances of the true instances of a class.

6.4. Settings

We use the unweighted k-NN version and set the number of neighbors to three; i.e., $k = 3$. We utilize the Python *scikit-learn* library [34] to implement the k-NN classifier as well as for data pre-processing such as feature selection using Random Forests. The

value of N, the maximum size of the dynamic training set $\mathcal{D}$, is set to 2500 flows (and not bags), and is the same for experiments on both traces and also independent of the number of application classes. As the training set size influences the efficiency of k-NN classifier, a larger N slows down the experiments. The threshold to the number of iterations (over the learning dataset), η, is set to five; we further discuss on this parameter in Section 6.5.3.

Another important parameter which is worth a special mention is the size of BoFs. Intuitively, the larger the number of flows in a bag, the better the prediction accuracy, and this has been validated in [6]. In our experiments, we set the allowed range for bag-size for *isp* trace to be 3-20, and for *wide* we reduce the minimum bag-size to one. Later, in Section 6.5.3, we experiment the performance by varying the bag-sizes.

Experiments are distinguished by the number of instances per application class, m, used for building the initial training set $\mathcal{I}$. Note, $|\mathcal{I}_a| = m, \forall a \in A$. The number of experiments is therefore equal to the number of values that m takes; and the list of values taken by m is $[5, 10, 20, 30, 40, 50]$. Each experiment is run 100 times, sufficient enough to get 95% confidence interval on accuracy. In each run of the experiment, the initial training set is chosen randomly. The learning and validation datasets are also created randomly for each run.

6.5. Results

We present the results on the *isp* and the *wide* traces. We also evaluate the classifiers on different bag-sizes.

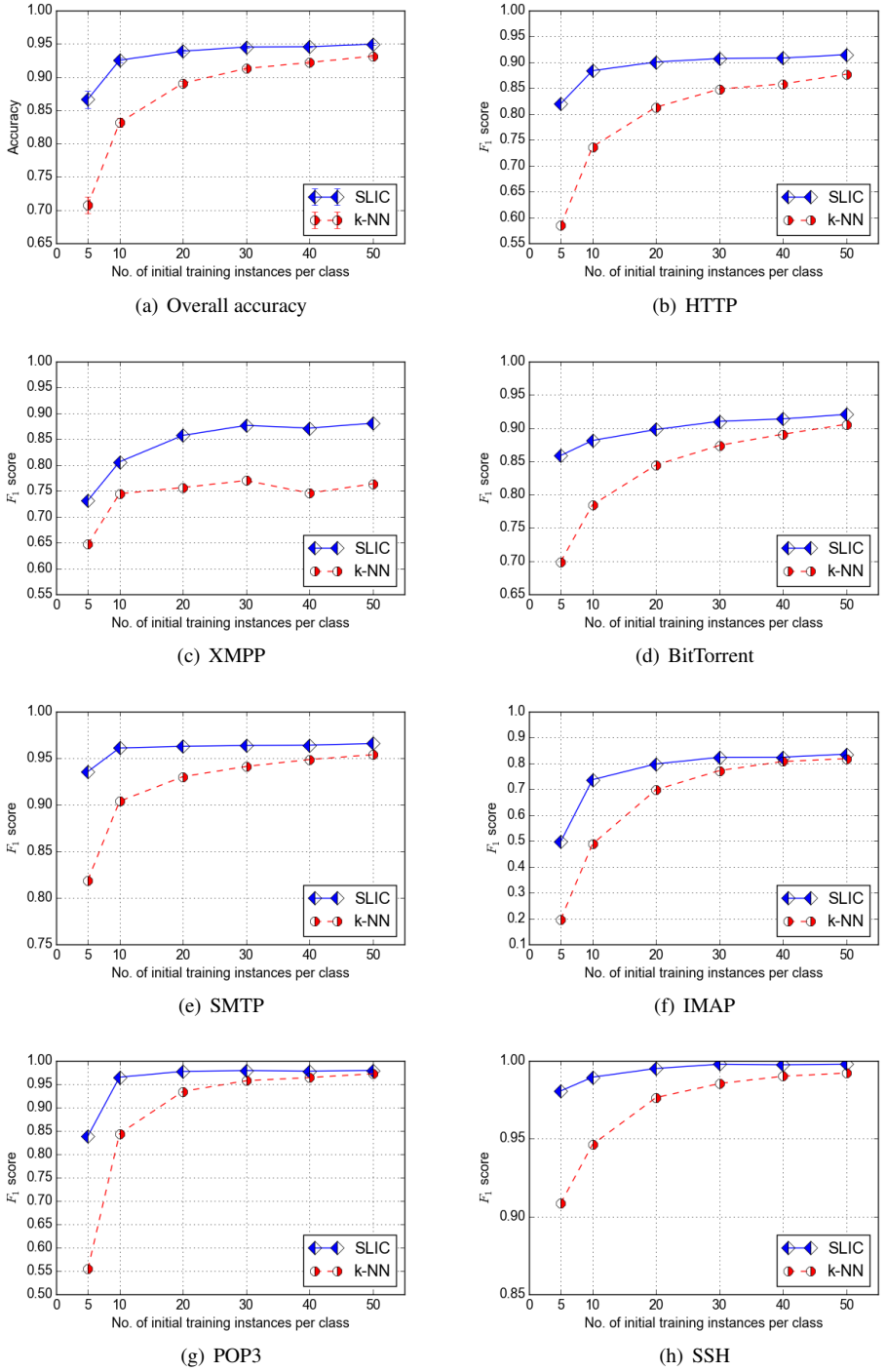
Each point on a plot is the mean value of 100 runs. 95% confidence intervals are marked for the figures plotting accuracies. For the quantitative analyses below, we use rounded values.

6.5.1. Trace isp

Fig. 4(a) gives the overall classification accuracy achieved on the *isp* trace with different number of training instances per class (m). The bag-size range was set to 3-20. The value of θ in Algorithm 5 is set to one. This means, if a bag predicted as class a, finds the centroid of $\mathcal{I}_a$ nearest to it (among all application classes $\mathcal{A}$), the bag is inducted into the dynamic training set $\mathcal{D}$.

We observe that the (BoF-based) k-NN, which does not learn using the predicted data, achieves a moderate overall accuracy of 71% when it uses five instances per class for training (i.e., in this experiment $|\mathcal{I}| = m \times |A| = 35$). With self-learning at its core, SLIC is seen to improve this performance significantly. The accuracy achieved by SLIC is 87%, which is 23% higher than that achieved by k-NN. While the accuracies achieved by both the classifiers increase with increasing m, SLIC further improves the accuracies achieved by BoF-based k-NN. In particular, SLIC is able to achieve more than 92% accuracy with just 10 training instances per class.

The remaining figures, Fig. 4(b)-4(h), plot the F_1 score for the seven protocols. For HTTP, observe that, the best classification result obtained using k-NN is 87%; and this required a large initial training size of 50 per class. SLIC achieves 88% F_1 score with just 10 training instances per class. The best F_1 score achieved for XMPP by k-NN is 77%, whereas SLIC is able to achieve close to 90% F_1 score, thereby increasing the



(a) Overall accuracy

(b) HTTP

(c) XMPP

(d) BitTorrent

(e) SMTP

(f) IMAP

(g) POP3

(h) SSH

Figure 4: Classification results on *isp* trace. Overall accuracy with 95% confidence interval is plotted in (a); (b)-(h) plot the F_1 scores for the different protocols. The metrics are plotted against m, the number of training instances (flows) per application class.

classification performance by almost 17%. For BitTorrent, SMTP, IMAP, POP3 and SSH, SLIC brings in significant performance improvement over k-NN when the initial training data is less. The F_1 score for POP3, with $m = 5$, improved from a mere 55% to an impressive 84% due to SLIC. For SSH (in Fig. 4(h)), we also observe that SLIC achieves close to the best classification result of 98% with just 5 training instances per class.

6.5.2. Trace wide

As mentioned earlier, the *wide* trace has been categorized into five application classes, namely HTTP, DNS, BitTorrent, Mail and FTP. Also recall that we set the minimum bag-size to be one, thereby allowing bags of all sizes to be considered for classification. The value of θ in Algorithm 5 is set to three here. This is to restrict the induction in each iteration, as (i) *wide* testing dataset had imbalance in its class distribution, and (ii) small size bags may lead to decrease in accuracy. Fig. 5(a) plots the overall accuracy. The accuracies achieved on *wide* is slightly lower than that achieved on *isp*. The reason for this might be that, in these experiments the minimum bag-size considered is one; we experiment on the effect of bag-sizes in the next section and provide further analysis.

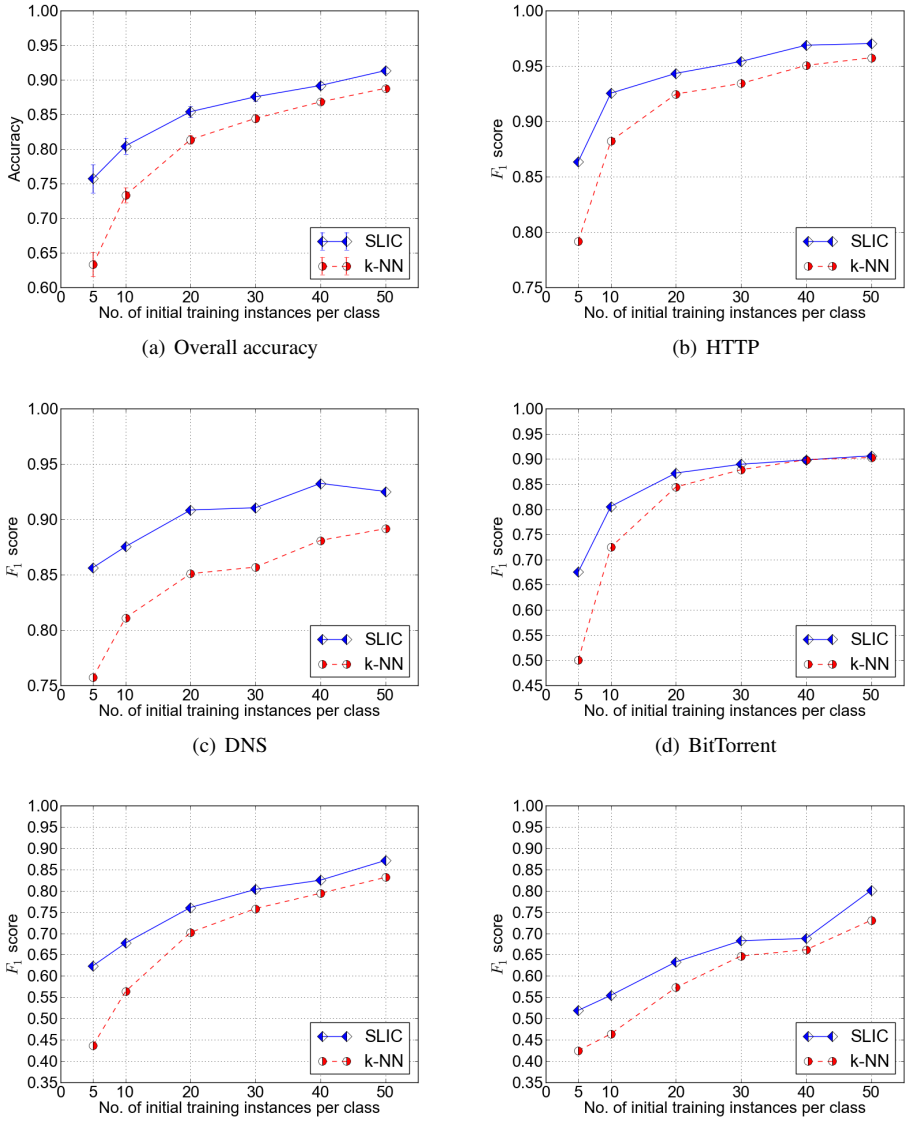
With five instances per class ($m = 5$), the improvement in accuracy brought about by SLIC over k-NN is 22%. From Fig. 5(a), we observe that SLIC achieves an overall accuracy of 76%. Also note that with increasing m, SLIC is still able to bring in further improvements in accuracies over k-NN.

Figures. 5(b)-5(f) plot the F_1 scores for the different protocols. The F_1 scores achieved by SLIC for HTTP and DNS are always above 85%, while this is not the case with k-NN. The accuracy achieved by SLIC with five instances per class is moderate for BitTorrent and Mail; yet the relative improvement over k-NN is huge—35% for BitTorrent, and 43% for Mail. For FTP, though SLIC improves the classification performance attained by k-NN, the absolute F_1 score reaches 80% only when m is 50. We suspect this may be due to the less number of test instances available under the FTP class—FTP had 85% lesser number of BoFs than most other classes.

6.5.3. Discussions

Bag-size: As mentioned earlier, BoF size plays an important role for the improvement of classification accuracy. This was validated in [6]—the larger the bag-size, the higher the classification accuracy. However, a disadvantage is that, bag-size also dictates the time the (pre-processor of the) classification system needs to wait in order to gather sufficient number of flows to form bags. This waiting time can be too long, possibly indefinite in the worst case. Consider the case where a particular user in an enterprise network visits a suspicious site whose IP address keeps changing. Unless other users visit the same site at almost the same times, the bags formed will always be of size one. Thus, it is important for a classification system to be able to predict on a single flow (or equivalently, bags of size one).

Motivated by the above reasoning, we further evaluate the classification performance on *isp* trace when the bag-size is varied. We set the number of initial training instances per class to be five. The upper bound on the bag-size is set fixed to 20, as it is already known that increasing bag-sizes leads to higher accuracy. So, we vary the



(a) Overall accuracy

(b) HTTP

(c) DNS

(d) BitTorrent

(e) Mail

(f) FTP

Figure 5: Classification results on *wide* trace. Overall accuracy with 95% confidence interval is plotted in (a); (b)-(f) plot the F_1 scores for the different classes. The metrics are plotted against m, the number of training instances (flows) per application class.

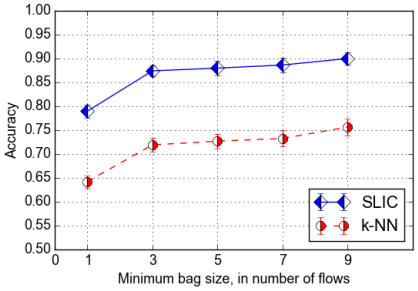


Figure 6: Accuracy when the minimum bag-size is varied

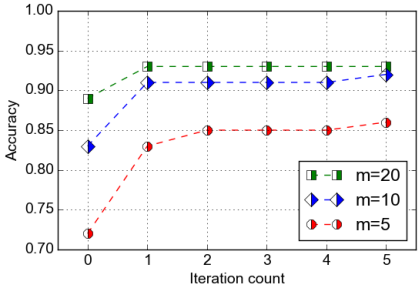


Figure 7: Improvement in accuracies over iterations for SLIC

minimum bag-size in this set of experiments. Note that, when the minimum bag size is one, bags of all sizes are considered for testing; whereas if the same is set to three, bags of sizes one and two are ignored.

Fig. 6 shows the experiment result. The X-axis marks the minimum bag-size for each set of experiments. From the figure we make the following observations:

- Regardless of the lower bound, SLIC consistently achieves significant improvement in accuracy, compared to the BoF-based k-NN classifier—SLIC achieves 20% or higher accuracy. Recall that the number of initial training instances is only five per class.
- With a lower bound of one for the bag-size, the accuracy achieved by SLIC is 79%. This is higher than the best accuracy obtained by k-NN (75%, when the minimum bag-size is nine). This again shows the strength of SLIC: instead of setting a high lower bound and missing out flows in small-sized bags in order to achieve better accuracy, SLIC, with the lowest bound of one, ignores no flow, and yet achieves high accuracy. This makes SLIC a practical online classifier, as it does not have to wait indefinitely, or even for long time, to be able to predict the class labels accurately.
- Bags of sizes one and two indeed affect the classification results of both the classifiers. We observe a decrease of roughly 10% for SLIC, and 12% for k-NN between when the minimum bag-size is reduced from three to one. This is the reason for the diminished improvement for *wide* trace in comparison to the *isp* trace.

Number of iterations (η): The number of iterations η was set to five to allow enough iterations for SLIC to learn and stabilize. Fig. 7 plots the improvement in accuracies per iteration over the validation datasets of *isp* trace. These plots are for different sizes of the initial training set (5, 10 and 20). The accuracy for iteration zero is obtained using the initial training set, and without any self-learning. Observe that SLIC stabilizes quite quickly, achieving significant improvement in the first iteration.

7. Conclusions and future directions

This work proposed a new direction to solve the two fundamental issues in supervised traffic classification: (i) learning using small number of labeled data, and (ii) adapting to non-static network static. We proposed and developed SLIC, a classification system that starts with a small set of labeled data and self-learns by inducting selected test instances (without ground truth) into its training set and rebuilds the model for classification. In short, SLIC evolves on its own and achieves significant improvement over the initial prediction accuracy achieved without self-learning. We demonstrated the effectives of SLIC by evaluating it against a BoF-based k-NN on two real-world traffic traces.

This is our first step towards developing self-learning traffic classification solutions. With this work, we see a number of interesting directions to advance the research on self-learning classifiers. We discuss a few important ones here. As SLIC is built on k-NN classifier, the size of the dynamic training set is constrained by computational complexity. This means, if SLIC is to be continuously learning, the growing training size would slow down the classifier. There are two ways to overcome this limitation. One way is to continuously prune the dataset to remove obsolete instances from the training set, thereby making space for more inductions; and this process can be repeated infinitely. As long as the size of the dynamic training set is large enough to allow instances of a class to be a proper representation of the population, this solution will be effective. We envision such a pruning process will allow SLIC to achieve its best performance with an initial training set of just five instances per class. A second interesting approach is to develop self-learning over a parametric classifier such as Naive Bayes. As the computational complexity of Naive Bayes classification is independent of the size of the training set, the self-learning system has freedom to induct any number of good candidates. Another potential research direction is to adapt SLIC to detect and classify even unknown application protocols. Detection of zero-day protocols is not possible using the current supervised approach, but the core self-learning concept of SLIC can be used to develop a new solution.

8. Acknowledgement

We sincerely thank Dr. Jun Zhang and his research team from Deakin University, Australia, for generously sharing their labeled traffic traces with us for performance evaluation.

References

[1] D. M. Divakaran, L. Su, Y. S. Liau, V. L. Thing, Slic: Self-learning intelligent classifier for network traffic, Computer Networks 91 (2015) 283–297.

[2] T. Nguyen, G. Armitage, A survey of techniques for internet traffic classification using machine learning, IEEE Communications Surveys Tutorials 10 (4) (2008) 56–76.

[3] Q. Yang, X. Wu, 10 Challenging Problems in Data Mining Research, International Journal of Information Technology and Decision Making 5 (4) (2006) 597–604.

[4] Cisco visual networking index: Forecast and methodology, 2013-2018, http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf (2014).

[5] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang, Y. Guan, Network Traffic Classification Using Correlation Information, IEEE Transactions on Parallel Distributed Systems (TPDS) 24 (1) (2013) 104–117.

[6] J. Zhang, C. Chen, Y. Xiang, W. Zhou, Y. Xiang, Internet Traffic Classification by Aggregating Correlated Naive Bayes Predictions, IEEE Transactions on Information Forensics and Security (TIFS) 8 (1) (2013) 5–15.

[7] A. W. Moore, D. Zuev, Internet Traffic Classification Using Bayesian Analysis Techniques, in: Proc. of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, 2005, pp. 50–60.

[8] T. Auld, A. W. Moore, S. F. Gull, Bayesian neural networks for internet traffic classification, IEEE Trans. on Neural Networks.

[9] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, P. Tofanelli, Revealing Skype Traffic: When Randomness Plays with You, ACM SIGCOMM Comput. Commun. Rev. (CCR) 37 (4) (2007) 37–48.

[10] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, K. Lee, Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices, in: Proc. of the 2008 ACM CoNEXT Conference, 2008, pp. 11:1–11:12.

[11] M. Roughan, S. Sen, O. Spatscheck, N. Duffield, Class-of-service Mapping for QoS: A Statistical Signature-based Approach to IP Traffic Classification, in: Proc. of the 4th ACM SIGCOMM Conference on Internet Measurement, IMC '04, 2004, pp. 135–148.

[12] J. Zhang, C. Chen, Y. Xiang, W. Zhou, A. Vasilakos, An Effective Network Traffic Classification Method with Unknown Flow Detection, IEEE Transactions on Network and Service Management 10 (2) (2013) 133–147.

[13] A. McGregor, M. A. Hall, P. Lorier, J. Brunskill, Flow Clustering Using Machine Learning Techniques, in: Proc. 5th International Workshop Passive and Active Network Measurement (PAM), 2004, pp. 205–214.

[14] S. Zander, T. Nguyen, G. Armitage, Automated traffic classification and application identification using machine learning, in: Proc. IEEE Conference on Local Computer Networks (LCN), 2005, pp. 250–257.

[15] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, K. Salamatian, Traffic Classification on the Fly, ACM SIGCOMM Comput. Commun. Rev. (CCR) 36 (2) (2006) 23–26.

[16] J. Erman, M. Arlitt, A. Mahanti, Traffic Classification Using Clustering Algorithms, in: Proc. of the 2006 SIGCOMM Workshop on Mining Network Data, MineNet '06, 2006, pp. 281–286.

[17] L. Bernaille, R. Teixeira, K. Salamatian, Early Application Identification, in: Proc. of the 2006 ACM CoNEXT Conference, CoNEXT '06, 2006, pp. 6:1–6:12.

[18] Y. Wang, Y. Xiang, S. Yu, An automatic application signature construction system for unknown traffic, Concurrency and Computation: Practice and Experience 22 (13) (2010) 1927–1944.

[19] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, C. Williamson, Offline/Realtime Traffic Classification Using Semi-supervised Learning, Perform. Eval. 64 (9-12) (2007) 1194–1213.

[20] X. Li, F. Qi, D. Xu, X. song Qiu, An Internet Traffic Classification Method Based on Semi-Supervised Support Vector Machine, in: Proc. IEEE International Conference on Communications (ICC), 2011, pp. 1–5.

[21] Y. Wang, Y. Xiang, J. Zhang, S. Yu, A novel semi-supervised approach for network traffic clustering, in: Proc. 5th International Conference on Network and System Security (NSS), 2011, pp. 169–175.

[22] L. Grimaudo, M. Mellia, E. Baralis, R. Keralapura, SeLeCT: Self-Learning Classifier for Internet Traffic, Network and Service Management, IEEE Transactions on 11 (2) (2014) 144–157.

[23] M. Opper, On-line learning in neural networks, Cambridge University Press, 1998, Ch. A Bayesian Approach to On-line Learning, pp. 363–378.

[24] S. A. Solla, O. Winther, Optimal perceptron learning: an online Bayesian approach, On-Line Learning in Neural Networks.

[25] J. Ma, K. Levchenko, C. Kreibich, S. Savage, G. M. Voelker, Unexpected Means of Protocol Inference, in: Proc. of the 6th ACM SIGCOMM Conference on Internet Measurement, IMC '06, 2006, pp. 313–326.

[26] M. Canini, W. Li, M. Zadnik, A. W. Moore, Experience with High-speed Automated Application-identification for Network-management, in: Proc. of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS '09, 2009, pp. 209–218.

[27] Y. Li, X. Zhang, Improving k Nearest Neighbor with Exemplar Generalization for Imbalanced Classification, in: Proc. of the 15th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, PAKDD'11, 2011, pp. 321–332.

[28] H. Dubey, V. Pudi, Class Based Weighted K-Nearest Neighbor over Imbalance Dataset, in: Proc. 17th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD), 2013, pp. 305–316.

[29] J. A. Hartigan, M. A. Wong, Algorithm AS 136: A k-means clustering algorithm, Applied statistics.

[30] K. Cho, K. Mitsuya, A. Kato, Traffic Data Repository at the WIDE Project, in: Proc. of the Annual Conference on USENIX Annual Technical Conference, ATEC '00, 2000, pp. 51–51.

[31] l7-filter, http://l7-filter.clearfoundation.com.

[32] TCP STatistic and Analysis Tool, http://tstat.tlc.polito.it/index.shtml.

[33] L. Breiman, Random Forests, Machine Learning 45 (1) (2001) 5–32.

[34] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in Python, The Journal of Machine Learning Research 12 (2011) 2825–2830.