Comparison of multi-instance classification methods applied on P2P traffic classification

March 8, 2012

Abstract

Peer-to-Peer stream(P2P-TV) applications offer the capability to watch real time video over the Internet at low cost. Some applications have started to become popular raising the concern of Network Operators that fear the large amount of traffic they might generate. Unfortunately, most of P2P-TV applications are based on proprietary and unknown protocols, which makes the detection of such traffic much complicated. Some accurate classification methods are based on the content of the packet which would consume lot of time. In this paper, we propose a novel methodology to accurate classify P2P-TV traffic. Our proposal relies on the bytes of each packet and the time internal between two successive packets during small time-window: the rationale is that these two characters convey a wealth of useful information concerning several aspects of the application and its inner workings, such as video chunk size and signaling activities.

Several multi-instance classification method using machine learning, such as Multi-instance Support Vector Machines and CitationkNN are applied in this paper. Lots of experiments are taken to compare those methods and to find out what parameters including the size of time window and the limit of packets during each time window are best for the performance. By means of a large experimental campaign, which uses both testbed and real network traffic, we show that it is actually possible to reliably discriminate between P2P-TV applications based on the characters of size of the packet and the time interval.

1 Introduction

The Internet has proven to have an awesome capability of adapting to new services, migrating from the initial pure datagram paradigm to real multi-service infrastructure. One of the most recent step of this evolution is P2P-TV, i.e., large-scale real-time video-streaming services exploiting the peer-to-peer communication paradigm. There are several currently deployed P2P-TV systems [1–4], which feature low-quality and low-bitrate streaming, with high-quality systems just beyond the corner. In

2 Classification framework

2.1 The rationale

Our aim is to classify P2P-TV *end-pints*, which can be identified by IP address and transport layer port pair *(IP, port)*. Typically, P2P-TV applications rely on UDP as the transport protocol. During installation, a single UDP port is selected at random, over which all the signaling and video traffic exchanged with other peers is multiplexed. Therefore, all the traffic going to/coming from a given *(IP, UDP-port)* end-point is actually destined to/source from the same P2P-TV application running on the host. This holds true for P2P-TV applications like PPLive [1], PPStream [2], SopCast [3], TVUPlayer [4], which we take as examples throughout this paper. Because of the continuous development of new applications, the choice of a representative set is definitely a difficult one. We decide to use the most popular applications at the time of experiments.

As mentioned before, we design a P2P-TV classification methodology that relies only on the evaluation of the *statistical information*, such as size of each packet and the time interval between two consecutive packets during a small time window with a lower limit of packet amount. The rationale is that a raw statistical character of exchanged data conveys useful information concerning several aspects of P2P-TV applications.

A human analogy may help in clarifying the intuition. Suppose peers in the network are people in a party room: people generally have different behavior, e.g., they will they will be more or less talkative. As much, somebody may prefer lengthy talks with a few other people, whereas somebody else may prefer very brief exchanges with a lot of people. This is similar to what happens with P2P applications: some applications continuously perform peer discovery by sending few packets to a previously not-contacted peer; others tend to keep exchanging most of packets with the same peers.

Additionally, most P2P-TV applications have been designed around the concept of ”chunks” of video, i.e., small units of information whose size is a typical parameter of each application. Download of video content is thus performed using several chunks, and the size of flows carrying the video content is roughly a multiple of the chunk size. Moreover, P2P-TV video service has an almost constant downlink throughout, due to the nature of the video stream. By tracking the *breakdown* between the different contributors it is possible to highlight different policies that a particular application can adopt, namely, fetching chunks from many neighbors, or downloading from a restricted list of preferential peers. Yet, while any P2P-TV peer consumes equally, the amount of uploaded data can be significantly different from peer to peer, due to different configuration, such as upload capacity. For example, in [**?**], it is shown that uplink to downlink throughput ration for PPLive varies in the [0,10] Mbps range, for the same downlink throughput of about 400Kbps. In reason of the above observation, we assume that the classifier is located at the *edge* of the network (where all traffic exchanged by a given end-point transits), and consider only the *downlink* direction, i.e., traffic coming from the Internet and crossing the edge of the network into the end-point direction. Notice that once an endpoint has been identified by means of downlink traffic, the uplink traffic is classified as well. However, the additional use of the characteristics of the uplink traffic to support the classification represents an interesting direction for future work.

In the following, we restrict our attention to UDP traffic, although endpoint identification can be extended to applications relying on TCP at the transport layer as well. In case TCP is used, the client TCP port is ephemeral, i.e., randomly selected by the Operating System for each TCP connection. The TCP case thus requires more complex algorithms in case of traffic *generated* from a specific peer, since ephemeral ports differ among flows generated by the same peer. However, we point out that the ephemeral port problem vanishes if we focus on the downlink direction as we do in this work(i.e., since we need in this case to aggregate all traffic received by a TCP server port, that remains the same for all flows of any given peer).

2.2 Behavioral P2P-TV signatures

According to [6], we know that different P2P-TV applications receive different amount of packets in a small time window, which means the time interval between two successive packets is different in a specific time window for different P2P-TV applications. Besides, size of each packet, which could convey the information about the trunk size which is diverse for different P2P-TV applications varies for different applications. Thus, we could retrieve these information from the packets to form a signature which could distinguish a P2P-TV application from others.

We retrieve information from the captured data to generate the signature in the following way. Let us consider the traffic received by an arbitrary end-point $P = (IP, port)$ during an interval of duration ΔT. Assuming there are several remote end-point, $P_1, P_2, ..., P_n$ which send data to the local end-point during the small time window. There is a lower limit of the packet amount L for the connection between P and $P_i, i \in [1, n]$. Only those packets the count of which is beyond L could be taken into consideration to generate the signature, otherwise, just throw these packets away. If the count of the packets is more than the lower limit, L, just take the former L packets as the resource of the signature(e.g., if the count of packets between P and P_1 during the time window ΔT is 38, however, the lower limit of the amount is 40, then the packets are thrown away. If the count of packets between P and P_2 is 42 while the limit is 40, just take the former 40 packets and leave the last 2 packets alone). It is time to retrieve the signature after the resource is determined. Two important characteristics are size of the packet and the time interval between two consecutive packets as mentioned before. The format of the signature is

$$S_1, T_1, S_2, T_2, ..., S_i, T_i, ..., S_{L-1}, T_{L-1}, S_L.$$

Here, S_i is the size of the *i*th packet and T_i denotes the time interval between the *i*th packet and *L*th packet. L and ΔT are two parameters that will be determined through experiments.

Figure 1 shows the relationship between the parameter L, ΔT and the accuracy of classification by different multi-instance machine learning algorithm. $X - axis$ represents the parameter L, lower limitation of packets' count while the parameter ΔT is reported on $y - axis$. The accuracy of classification is located on the grid with points when L and ΔT are settled.

//————————figure and explanation———

3 Methodology and dataset

The classification of P2P-TV traffic can be performed by exploiting the signature described before through any supervised learning machine. In this paper, we resort to several methods of machine learning algorithms such as Multi-instance Support Vector Machine, CitationkNN, etc., which are famous for their discriminative power in both the learning machine community [7], and have more recently peered in the Internet traffic classification literature as well [8]. Specifically, these algorithms(Multi-instance Support Vector Machine(MISVM), CitationkNN, Multi-instance Diversity Dense(MIDD), Multi-instance Expectation-Maximization version of Diverse Density(MIEMDD), Multi-instance Wrapper(MIWrapper), Multi-instance Boost(MIBoost)) are all implemented in Weka3.6.5 [9] which are applied in this article.

3.1 Workflow overview

From a high-level perspective, the P2P-TV classification workflow consists of three phases: the *training* phase, whose output is a model that can be used for the second phase, the *classification* phase. The third phase is to *validate* the classification results against a reference ground truth. We will explain the workflow in detail with the help of Fig.2.

4 Methods of Multi-instance classification using machine learning

4.1 Citation kNN

The popular k Nearest Neighbor (k-NN) approach can be adapted for MIL problems if the distance between bags is defined. In [Wang and Zucker, 2000], the *minimum Hausdorff distance* was used as the bag-level distance metric, defined as the shortest distance between any two instances from each bag.

$$Dist(A,B) = \min_{\substack{1\leq i\leq n \\ 1\leq j\leq m}} \Big(Dist(a_i,b_j)\Big) = \min_{a\in A}\min_{b\in B}\|a-b\|$$

where A and B denote two bags, and a_i and b_j are instances from each bag. Using this bag-level distance, we can predict the label of an unlabeled bag using the k-NN algorithm.

However, in a MIL setting, sometimes the majority label of the K nearest neighbors of an unlabeled bag is *not* the true label of that bag, mainly because the underlying prediction-generation scheme of kNN, *majority voting*, can be easily confused by the false positive instances in positive bags. The citation approach is used to overcome this weakness, which considers not only the bags as the nearest neighbors(known as references) of a bag B, but also the bags that that count B as their neighbors(known as citers) based on the minimum Hausdorff distance. Thus, *citation-kNN* predicts the label of a bag based on the labels of both the references and citers of that bag, which is

empirically proved to be more robust than the *kNN* based on only references. Another alternative of the majority voting scheme is the Bayesian method, which computes the posterior probabilities of the label of a unknown bag based on labels of its neighbors. Experiments on the drug activity problem showed comparable performance of citation *kNN* and Bayesian *kNN* with the DD and APR algorithm. But note that unlike the DD method, *kNN* methods are unable to predict the labels of the instances.

4.2 MIDD Diverse Density

Diversity Density(DD) was proposed in [Maron and Lozano-Perez, 1998] as a general framework for solving multi-instance learning problems. The main idea of DD approach is to find a concept point in the feature space that are close to at least one instance from every positive bag and meanwhile far away from instances in negative bags. The optimal concept point is defined as the one with the maximum diversity density, which is a measure of how many different positive bags have instances near the point, and how far the negative instances are away from that point.

A probabilistic derivation of Diverse Density is given below. We denote positive bags as B_i^+, and the *j*th instance in that bag as B_{ij}^+. Suppose each instance can be represented by a feature vector(or a point in the feature space), and we use B_{ijk}^+ to denote the value of the *k*th feature of instance B_{ij}^+. Likewise, B_i^- denotes a negative bag and B_{ij}^- is the *j*th instance in that bag. The true concept is a single point *t* defined by maximizing the diverse density defined as $DD(f) = P(t|B_1^+, ..., B_n^+, B_1^-, ..., B_m^-)$ over the feature space. Using Bayes rule and assuming uniform prior over the concept location, this is equivalent to maximizing the following likelihood:

$$\arg\max_t P(B_1^+, ..., B_n^+, B_1^-, ..., B_m^-|t)$$

By making additional assumption that the bags are conditionally independent given the concept point *t*, this decomposes to:

$$\arg\max_t \prod_i P(B_i^+|t) \prod_i P(B_i^-|t)$$

Using Bayes rule once more with the uniform prior assumption, this is equivalent to:

$$\arg\max_t \prod_i P(t|B_i^+) \prod_i P(t|B_i^-)$$

which gives a general definition of the Diverse Density. Given the fact that boolean label(say, 1 and 0) of a bag is the result of "logical-OR" of the labels of its instances, $P(t|B_i)$ is instantiated using the noise-or model:

$$P(t|B_i^+) = 1 - \prod_j \big(1 - P(B_{ij}^+)\big) \text{ and } P(t|B_i^-) = \prod_j \big(1 - P(B_{ij}^-)\big)$$

Finally, $P(t|B_{ij}^+)$(or $P(t|B_{ij}^-)$) is estimated(though not necessarily) by a Gaussian-like distribution

$$P(t|B_{ij}^+) = \exp\big(-\|B_{ij}^+ - t\|^2\big) = \exp\Big(-\sum_k w_k (B_{ijk}^+ - t_k)^2\Big)$$

where w_k is a non-negative scaling factor that reflects the degree of relevance of different features. Without close-form solution to the above maximization problem, *gradient ascent* method is used to search the feature space for the concept point with(local) maximum DD. Usually the search is repeated using the instances from every positive bag as the starting points.

4.3 EM-DD Expectation-Maximization version of Diverse Density

In the MIL setting, the label of a bag is determined by the "most positive" instance in the bag, i.e., the one with the highest probability of being positive among all the instances in that bag. The difficulty of MIL comes from the ambiguity of not knowing which instance is the most likely one. In [Zhang and Goldman, 2001], the knowledge of which instance determines the label of the bag is modeled using a set of *hidden variables*, which are estimated using the Expectation Maximization style approach. This results in an algorithm called EM-DD, which combines this EM-style approach with the DD algorithm.

EM-DD starts with an initial guess of the concept point *t*(which can be obtained using original DD algorithm), and then repeatedly performs the following two steps: in E-step, the current hypothesis of concept *t* is used to pick the most likely instance from each bag given a generative model; in M-step, the a new concept *t'* is estimated by maximizing a transformed DD defined on the instances selected in the E-step using the gradient search. Then, the old concept *t* is replaced by the new concept *t'* and the two steps are repeated until the algorithm converges. This EM-DD algorithm implements a "hard" version of EM, since only one instance per bag is used for estimating the hypothesis. It can be also regarded as a special case of the K-means clustering algorithm, where only one cluster is considered.

By removing the noise-or(or a "softmax") part in the original DD algorithm, EM-DD turns a multi-instance problem into a single-instance one, and thus greatly reduces the complexity of the optimization function and the computational time. It is also believed to help avoid local maxima since it makes major activity problem, Em-DD outperforms other MIL algorithms like DD, Citation-kNN and APR by a decent margin [Zhang and Goldman, 2001].

4.4 MISVM Supprot Vector Machine for multi-instance learning

This SVM based approach suggested by Stuart Andrews et al.(2002) follows the standard MI assumption. The main idea of this approach is to transform the MI data setting into a single-instance data setting by properly assigning the unobserved class label to each individual instance in the positive bags(all instances in negative bags are assumed to be negative). The standard single-instance SVM learning scheme is applied to assign these labels. Andrews et al. call this method "maximum pattern margin formulation of MI learning". The aim is to find the maximum margin MI-separating hyperplane, in which all instances in every negative bag are located on one side of the hyperplane and at least one instance in every positive bags is located on the other side of the hyperplane.

The key problem in the *MISVM* approach is how to determine and assign the proper class label to the individual instances within each positive bag. The *MISVM* algorithm first initializes all instances in the positive bags with the positive class labels and then iteratively adjusts these labels until they converge. The algorithm is described as following:

Build Classifier:
For each bag B
    If B is a positive bag
        Initialize class label for each instance x_i within *B* as $y_i = 1$;
    Else
        Initialize the label of each instance x_i within *B* as $y_i = 0$;
Do
    Build standard single-instance SVM model based on the labeled data;
    For each positive bag B^+
        For each single-instance x_i within B^+
            Compute SVM output $f(x_i) = \sum_j \alpha_j K(x_j, x_i) + b$;
            If $(f(x_i) \leq 0)$ $y_i = 0$;
            Else $y_i = 1$;
        If $(\sum_i y_i == 0)$ //no positive classification
            Find instance x_{i^*} within bag B^+ where $i^* = \arg\max_i f(x_i)$;
            Set $y_i^* = 1$;
While (single-instance labels have been changed)

Classify:
Initialize $distribution$;
For each single-instance x_i within the unknown bag
    Compute $f(x_i) = \sum_j \alpha_j K(x_j, x_i) + b$;
    If $(f(x_i \leq 0)$ $y_i = 0$;
    Else $y_i = 1$;
If $(\sum_i y_i == 0)$
    $distribution[0] = 1.0$;//predicted as a negative bag
Else
    $distribution[0] = 0$;//predicted as a positive bag
$distribution[1] = 1 - distribution[0]$;
return *distribution*;

4.5 MIWrapper A simple wrapper for multi-instance learning

The approach we take in the section is to assume that all instances contributes equally and independently to the bag's label. This approach allows us to apply standard propositional learning algorithms by breaking a bag up into its individual instances and labeling each instance with its bag's label. If there are *n* instances in a bag we give each instance the weight $1/n$(because it assumes that the instances contribute equally to the example's class label). It takes two steps to generate a classification for a new

bag of instances. First step is to filter each instance through the classification model built at training time to obtain a class probability estimate for each of the possible classes. Then, in second step, these class probability estimates are combined to form a prediction for the bag as a whole(just average the class probability estimates obtained in the first step because it assumes that all instances contribute equally to the bag's class label). Let *b* be a bag of instances. Then it computes the probability of class *c* given that bag as follows:

$$Pr(c|b) = E_X\big(Pr(c|x)|b\big) = \int_X Pr(c|x)Pr(x|b)$$

In other words, it marginalizes *x* out and assume that the probability of class *c* is conditionally independent of *b* given *x*(i.e. knowledge of *b* does not affect the class probability for a particular *x*). Given a concrete bag *b* of size *n*, the estimated value of $Pr(x|b)$ for each instance in the bag is *1/n*, and zero otherwise, and $Pr(c|b)$ becomes the average of the class probability estimates for the instance in the bag.

The question remains as to how to estimate $Pr(c|x)$. If we had a class label for each individual instance in a bag, we could use a standard propositional learning algorithm to estimate $Pr(c|x)$ by giving each instance the weight $1/n$. Assuming an instance-level loss function $Loss(\beta, C)$ for a model parameterized by β(e.g. the negative log-likelihood), the learner would then minimize the following expected loss:

$$E_B\Big[E_{X|B}\Big[E_C\big(Loss(\beta, C)|X\big)|B\Big]\Big] = \sum_i \frac{1}{N}\sum_j \frac{1}{n_i} E_C\big(Loss(\beta, C)|x_{ij}\big)$$

where *N* is the number of bags and n_i the number of instance in bag *i*.

However, in typical multi-instance problems there is only one label for the whole bag. Consequently the wrapper method performs a further simplification of the problem: it assigns the bag label to each instance in the bag and uses this as the training data for the propositional learner. This is a heuristic solution and does not enable the propositional learner to recover the true function $Pr(c|x)$. It will necessarily produce a biased estimate of the class probabilities. However, there are special cases where the correct *decision boundary* can be recovered(i.e. the classification performance is not affected).

4.6 MIBoost

Boosting sequentially generates a set of weak classifiers by reweighting the training instances and as a result a much stronger classifier can be obtained by combining these weak classifiers. If we have a weak learner that can deal with MI problems and handle bag weights, the *AdaBoost* algorithm can be applied directly to MI problems.

However, due to the limited number of MI learners, it is worthwhile to find a way to wrap the boosting algorithm around a single-instance weak learner. Motivated by this point, Xu and Frank (2004) upgraded the *AdaBoost* algorithm into an MI learner called *MIBoost* based on the collective assumption. It has been shown that boosting can be viewed and understood under some statistical principles, namely additive modeling and maximum likelihood (Friedman, Hastie & Tibshirani, 2000). The *MIBoost*

algorithm follows the same statistical principles. Like *AdaBoost*, *MIBoost* strives to find an additive model which can be expressed iteratively as

$$F(B) = F(B) + cf(B)$$

Here, *B* denotes an arbitrary bag, $f()$ denotes a weak classifier and *c* is a constant that needs to be learned in each boosting iteration. The details of the derivation of the *MIBoost* algorithm are described in (Xu & Frank, 2004). In the following, we describe the basic steps of the *MIBoost* algorithm.

1. Initialize weights of each bag to $W_i = 1/N, i = 1, 2, ..., N$.
2. Repeat for $m = 1, 2, ..., M$:
   (a) Set $W_{ij} \leftarrow W_i/n_i$, assign the bag's label to each of its instances, and build an instance-level model $h_m(x_{ij}) \in \{-1, 1\}$.
   (b) Within the *i*th bag(with n_i instances), compute the error rate $e_i \in [0, 1]$ by counting the number of misclassified instances within that bag, i.e. $e_i = \sum_j 1_{(h_m(x_{ij}) \neq y_i)}/n_i$.
   (c) If $e_i < 0.5$ for all i's, STOP iterations, Go to step 3.
   (d) Compute $c_m = \arg\min \sum_i W_i \exp[(2e_i - 1)c_m]$ using numeric optimization.
   (e) If ($c_m \leq 0$), go to step 3.
   (f) Set $W_i \leftarrow W_i \exp[(2e_i - 1)c_m]$ and renormalize so that $\sum_i W_i = 1$.
3. return $sign[\sum_i \sum_m c_m h_m(x_{test})]$.

Here, *N* is the total number of bags, n_i denotes the number of instances in a bag and the subscript i denotes the ith bag. h_m denotes a single-instance weak classifier built in the mth iteration. The value of $h_m()$ is either -1 or 1. Note that, for convenience, Xu and Frank assume the class label of a bag is either 1 or -1 rather than 1 or 0 here. x_{ij} denotes the jth instance in the ith bag and y_i denotes the class label of the ith bag.

As we know, in each boosting iteration, *MIBoost* adds a classifier $f()$ with the best value for *c* to the model. The aim is to minimize the exponential loss $E_B E_{Y|B}[\exp(-yF(B))]$. More precisely, in the mth iteration, $f_m(B)$ is obtained from the weighted version of the training data, and then c_m is derived by minimizing the exponential loss function(Xu & Frank, 2004):

$$\begin{aligned} E_B E_{Y|B}\Big[\exp\big(-yF(B) + c_m(-yf_m(B))\big)\Big] &= \sum_i W_i \exp\Big[c_m\big(-\frac{y\sum_i h(x_{ij})}{n_i}\big)\Big] \\ &= \sum_i W_i \exp\big[(2e_i - 1)c_m\big] \end{aligned}$$

In the implementation, c_m is found using a Quasi-Newton method (Xu & Frank, 2004). After the best value for *c* has been obtained, the bag-level weights can be updated. Each W_i is multiplied by $\exp[(2e_i - 1)c_m]$ and all the bags' are normalized so that the total weight is 1.

In *MIBoost*, the additive model obtained by minimizing the exponential loss function estimates the log-odds $\frac{1}{2}\log\frac{Pr(Y=1|X)}{Pr(Y=-1)|X}$, which is the same in *AdaBoost*. Hence, a prediction is made by treating the obtained model $F(B)$ as a bag-level log-odds function.

References

[1] PPLive. http://www.pplive.com/

[2] PPStream. http://www.ppstream.com/

[3] SopCast. http://www.sopcast.com/

[4] TVUPlayer. http://www.tvunetworks.com/

[5] X. Hei, C. Liang, J. Liang, Y. Liu, K.W. Ross, A measurement study of a large-scale P2P IPTV system, IEEE Transactions on Multimedia (2007).

[6] P. Bermolen, M. Mellia, M. Meo, D. Rossi, S. Valenti, Abacus: Accurate behavioral classification of P2P-TV traffic (2010.11).

[7] N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, Cambridge University Press, New York, NY, 1999

[8] N. Williams, S. Zander, G. Armitage, A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification, ACM Computer Communication Review 36 (5) (2006) 5C16.

[9] Weka. http://weka.wikispaces.com/