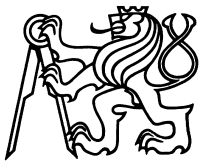
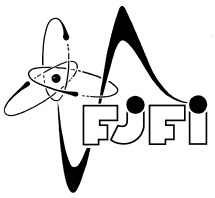


Czech Technical University in Prague
Faculty of Nuclear Sciences and Physical Engineering



Supervised classification of network traffic

Řízená klasifikace síťového provozu

Bachelor's Degree Project

Author: **Patrik Urban**

Supervisor: **Michal Sofka, MSc., Ph.D.**

Academic year: 2014/2015

- Zadání práce -

- Zadání práce (zadní strana) -

Acknowledgment:
My greatest thanks belongs to my supervisor Michal Sofka Ph.D. for all his advice, help with revisions of the text and also for his great attitude. I would also like express my deepest gratitude to my parents and family for the support in my studies, without which all of this would be impossible.

Candidate's declaration:
I hereby certify that:

- this bachelor's degree project represents my own work;
- the contribution of any supervisors or others to the research or the dissertation itself was consistent with normal supervisory practice;
- external contributions to the research are properly acknowledged and all used sources of information are listed in the bibliography.

Prague, July 7, 2015 Patrik Urban

Název práce:
Řízená klasifikace síťového provozu

Autor: Patrik Urban

Obor: Matematické inženýrství

Zaměření: Matematické modelování

Druh práce: Bakalářská práce

Vedoucí práce: Michal Sofka, MSc., Ph.D., Cisco Systems Inc.

Abstrakt: Tato práce představuje techniku klasifikace přenosu multimédií (videa a audia) motivované síťovou bezpečností. Způsob přenosu obsahu zachytitelný v záznamech z proxy serveru je popsán novým příznakem založeným na opakujících se hodnotách velikosti přenesených dat ze serveru na klienta. Tento příznak je použit ve dvou metodách klasifikace multimédií: klasifikace každého proxy logu nebo klasifikace IP adres serverů. Výsledky jsou porovnány s naivním klasifikátorem založeném na manuálně řízených pravidlech. Představené techniky jsou vhodné pro klasifikaci multimediálního přenosu i v šifrovaném `https` provozu. Nejlepší klasifikátory zde dosáhly detekce 51% všech proxy logů multimediálního provozu s přesností 77% a detekce 94% IP adres přenášející multimédia s přesností 85%. S využitím příznaků, v `https` provozu nedostupných, dosáhly nejlepší klasifikátory detekce až 55% všech multimediálních proxy logů s přesností 64% a nebo detekce 93% IP adres s přesností 94%.

Klíčová slova: klasifikace mediálního přenosu, klasifikace síťového provozu, síťová bezpečnost, síťová komunikace, strojové učení s učitelem

Title:
Supervised classification of network traffic

Author: Patrik Urban

Abstract: This work proposes a technique for classifying media streaming traffic (audio and video) for the network security. The streaming strategy captured by server proxy logs is described by a new feature computed from repetitive server-to-client byte values. This feature is then used by a binary classifier in two methods of media classification: classification of each proxy log or of the server IP address. The results are compared with a naive classification method based on manually-derived rules. The proposed methods are suitable for classifying media requests in https traffic, where the the best classifiers achieved the recall of 51% with the precision 77% in the case of the proxy log classification and the recall of 94% with the precision 85% in the case of classifying IP addresses transferring media traffic. The best classifiers using additional features not available in `https` achieved the recall 55% with the precision 64% in the case of the proxy log classification and recall 93% with the precision 94% in the case of classifying IP addresses.

Key words: media streaming traffic, network security, network traffic, streaming data classification, supervised classification

Contents

List of Tables

Chapter 1

Introduction

1.1 Motivation for this work

One of many threats nowadays is a PC infection by a malicious software. Such software is in most cases programmed to gain information, sensitive data or anything else from the infected computer and then to send it over the internet to the attacker. The attacker - the one who controls such software - is in most cases motivated by some form of gain or political interest. The infection may start with a click on an advertisement or on a link in an email or with a file download. The software then begins to communicate over the internet to ex-filtrate information or to receive additional orders. The computer resources may be even sold to someone else for Bit Coin mining or cyber-attacking. Any of these threats are dangerous and apparently unwanted. Such (and also many other) unwanted http requests are detectable through the analysis of the internet communication.

This bachelor degree project was supervised by Cisco Systems Inc. (to which I will further refer simply as Cisco). Cisco is one of many companies, which specializes themselves for network and internet security. A product of Cisco offered to their customers is a cloud web security solution. To protect the customer, Cisco provides a proxy server service, through which the customer connects to the internet. Thanks to a proxy server Cisco can monitor and record all internet communication between the customer and internet, these records (also denoted as **proxy logs**) stores, analyses and checks for any malicious http request.

With the size of a customers network rises not only the probability of existence of malicious communication, but also the time needed to check all proxy logs. This checking procedure can become - and in most cases also becomes - unmanageable due to the huge amount of data to be checked. Therefore, prior analysis and classification of all the internet entries has to take place. The goal of this analysis and classification is to reduce the amount of data, which has to be manually reviewed or further analysed. The roughest output of such classification is labelling the communication as:

- **Safe** - communication that cannot harm the client
- **Malicious** - communication that is known to be harmful or unwanted
- **Unknown** - communication which does not have a clear label assignment

The goal of this project is finding and classifying media streaming. According to Cisco measurements and estimates [6], around 64% of volume of global internet traffic was video internet traffic in the year 2014 and this number will rise up to 84% in the year 2019. Although this estimate does not say how many requests are or will be generated, it can be assumed, that the percentage of the requests will be considerable given the large video transmission volumes. Being able to sort out video traffic will greatly

reduce the amount of data to be further analysed. Audio streaming - either self contained music or radio streaming, or files sent parallel to videos, is also included in media streaming traffic.

In addition to the rising video traffic, the amount of encrypted internet traffic will rise according to recent estimates [5]. Encrypted `https` traffic improves the security and privacy of web page visitors by making it impossible to read detailed information about the communication and therefore proxy logs include only a little information, namely IP address of both client and server, the amount of data transferred both ways and time, when the communication happened. Such encrypted information is extremely difficult to analyse. A solution to this problem may be training a classifier on unencrypted data only using information available in `https` traffic.

1.2 Proxy service, proxy logs analysis and classification

Proxy server is a server (computer or an application) which stands between a client and other servers and provides communication. Along with providing the communication, proxy server also keeps record of it in a form of proxy logs. Each proxy log is a line of information about one particular client's request. Proxy log can include various information about the request, which can be used for an analysis of the communication.

The more of the communication is described the easier it is for Cisco to provide security for their clients. One of many goals of the analysis is to recognize as much of the internet traffic as possible to be able to decide which communication is malicious and which is not. The classification result *safe* has to be very accurate, not to have many **false positives** - that is not to label *unknown* and *malicious* traffic as *safe*. Such misclassification causes a security risk. It is therefore better to classify some *safe* traffic as *unknown* or *malicious* than the other way around.

The first step of designing a classifier is to manually look into the proxy logs to find a typical feature for a particular communication (e.g. advertisement, internet radio or news reading). When a feature is found, a rule can be designed and tested whether it suffices to describe the chosen communication. These manually designed rules are not the best option to use in real practice, because they lack flexibility and scalability. That means, that if the traffic slightly changes or not all of the traffic follows the same rule, the rule might not be applicable. An example of such rule might be filtering of the communication by the amount of data sent from a client to a server.

Because the internet keeps evolving, classification has to be of different form then of using rigid manually designed rules. Machine learning methods takes the leading role on this part. The first steps of the analysis are the same, only the procedure does not stop with manually designed rules. Instead of trying to find a threshold on the identified features, that would differentiate the chosen traffic, we create data-driven algorithms, that will do the job. The goal is then to:

1. Find descriptive features
2. Find a set of proxy logs containing only our chosen communication
3. Find a set of proxy logs containing everything else but our chosen communication
4. Train a classifier on these two sets of data, so that it:
   (a) Finds the best values of features
   (b) Is be able to distinguish previously-unseen traffic and label it either as a member of one or the other set

If the found features are sufficient, the classifier can always be updated by training on new sets of data. If the features divide the input space into more than two regions, the classifier can be used to distinguish other types of requests, for example text or Java applications, only by training on different sets of proxy logs.

1.3 Supervised machine learning, classification

Supervised machine learning is a process of creating a data driven algorithm based on pre-labelled data - so called **training set**. Machine learning can also be **unsupervised** (the set is not pre-labelled). This type of machine learning includes for example **clustering**, which means that the algorithm separates the data into distinct groups of subsets itself.

Classification and **regression** are two main domains of supervised machine learning. The difference between these two is in the predictions, that can be made about the data. Classification predictions are elements of a finite set of categories (e.g. {"safe", "malicious", "unknown"}), while for regression, the predictions are elements of infinite or continuous set of values (e.g. threat severity). This bachelor's thesis will be devoted to classifying media streaming and the predictions made will be { "media", "nomedia" ≡ "mediaC" } (C meaning complement).

The **classifier** (classification algorithm) learns itself on training data and creates rules which separate the training set into the same number of subsets as is the number of predictions. It can afterwards classify any other data on the basis of these rules into one of the categories. It is obvious that a sufficient set is essential for training a robust classifier.

1.4 Mathematical formulation and used classification methods

1.4.1 General formulation

Classification is a process of assigning **categories** (also **classes**) to certain input values. Therefore I begin be denoting X the **input space** of finite dimension N with elements $\mathbf{x} = (x_1, \ldots, x_N)^T$ called **input vectors** or simply **inputs**. We will make use of terms known from linear spaces and so we assume X is a linear vector space. For every **input** there is an **output** y. In a classification problem y is a category $\mathcal{G}_k$ which form a finite disjoint system of categories $(G_k)_{k=1}^K$ for some natural number K. We can denote each pair of input and output as $(\mathbf{x}, \mathcal{G}_k)$, or $(\mathbf{x}, y_k)$ when we assign $y_k \sim \mathcal{G}_k$ (y_k can be for example a numerical value). For the purpose of this work I can put $K = 2$ and for convenience map the two categories $\mathcal{G}_1$ and $\mathcal{G}_2$ to a set of numbers $\{+1; -1\}$ with y_k taking values of ± 1. The bijection between an input $\mathbf{x}$ in input space X and output ± 1 divide the input space into regions $\mathcal{R}_1$ and $\mathcal{R}_2$ for which holds $(\forall(\mathbf{x}, y))(\mathbf{x} \in \mathcal{R}_k \Rightarrow y \sim \mathcal{G}_k)$ for $k \in \{1; 2\}$. These are called **decision regions** and they are divided by **decision boundaries**.

Considering $X \times (G_k)_{k=1}^K$ as a probability space we have a joint probability distribution $p(\mathbf{x}, \mathcal{G}_k) \equiv p(\mathbf{x}, y_k)$ (it is not a system of probabilities indexed by k, instead $1 = \int_X \sum_k p(\mathbf{x}, \mathcal{G}_k) d\mathbf{x}$). Finding the joint probability from data is a very difficult task but solving this problem would almost completely resolve the classification problem. It is possible to obtain the joint probabilities from the product rule given by (1.2) but as will be shown, the posterior probabilities will be sufficient classification.

Let there be a set of M inputs to which all outputs are given: $((\mathbf{x}_i, y_{k_i}))_{i=1}^M$. This is called a **training set**. For a two-class problem we can further distinguish a **positive** training set ($y_{k_i} = 1 \sim \mathcal{G}_1$) and **negative** training set ($y_{k_i} = -1 \sim \mathcal{G}_2$). The goal now is to find a function, which predicts the output of a new input $\mathbf{x}$ using the information from the training set. Such function is to be precise, in other words it

should make as few mistakes as possible. The probability of a mistake [4] ($\mathbf{x} \in \mathcal{R}_1$ classified as a member of class $\mathcal{G}_2$ and visa versa):

$$\begin{aligned} p(\text{mistake}) &= p(\mathbf{x} \in \mathcal{R}_1, \mathcal{G}_2) + p(\mathbf{x} \in \mathcal{R}_2, \mathcal{G}_1) \\ &= \int_{\mathcal{R}_1} p(\mathbf{x}, \mathcal{G}_2) d\mathbf{x} + \int_{\mathcal{R}_2} p(\mathbf{x}, \mathcal{G}_1) d\mathbf{x} \end{aligned} \tag{1.1}$$

should be as small as possible. To achieve small classification errors $\mathbf{x}$ is assigned to such category for which the integrand is smaller that the other, in other words: if $p(\mathbf{x}, \mathcal{G}_1) > p(\mathbf{x}, \mathcal{G}_2)$ then $\mathbf{x}$ is of category $\mathcal{G}_1$. From the probability theory is given that

$$p(\mathbf{x}, \mathcal{G}_k) = p(\mathcal{G}_k|\mathbf{x})p(\mathbf{x}) \tag{1.2}$$

and so it is clear, that the inequality implies:

$$\text{if } p(\mathcal{G}_1|\mathbf{x}) > p(\mathcal{G}_2|\mathbf{x}) \text{ then } \mathbf{x} \text{ is of category } \mathcal{G}_1$$

Our aim is than to find the posterior probabilities $p(\mathcal{G}_k|\mathbf{x})$. The Bayes' theorem states that

$$p(\mathcal{G}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{G}_k)p(\mathcal{G}_k)}{p(\mathbf{x})} \tag{1.3}$$

and so the problem is in a way equivalent to finding conditional probability distributions $p(\mathbf{x}|\mathcal{G}_k)$ which can estimated from the training set.

Suppose now there exists a function G (also called **predictor**), which would assign proper category to a certain $\mathbf{x}$: $\mathcal{G}(\mathbf{x}) = y_k \in (\mathcal{G}_k)_{k=1}^K$ and minimize the probability of mistake given by (1.1). It shall be further assumed, that for every category $\mathcal{G}_k$ there exists a function $\delta_k(\mathbf{x})$ so that $\mathbf{x}$ is a member of category $\mathcal{G}_k$, when $\delta_k(x) > \delta_j(x)$ for all $j \neq k$. Such function δ_k is called a **discriminant function** of class $\mathcal{G}_k$ ($p(\mathcal{G}_k|\mathbf{x})$ itself falls into this category of functions) and the predictor can be constructed using discriminant functions. The decision boundary between two categories $\mathcal{G}_k$, $\mathcal{G}_j$ is then a set of points in X for which $\delta_k(x) = \delta_j(x)$.

A special case of discriminant functions which model linear dependencies between inputs and outputs are linear functions of the form $\delta_k(x) = \mathbf{w}_k \cdot \mathbf{x} + w_{k0}$. The decision boundary between two classes is than a set of point

$$\{\mathbf{x} \in X | (\mathbf{w}_k - \mathbf{w}_j) \cdot \mathbf{x} + (w_{k0} - w_{j0}) = 0\} \tag{1.4}$$

which is an affine space in X and thus the decision boundary is linear too. We will loosen the linearity assumption by assuming existence of monotonous transformation $\mathbf{\Phi}$ of δ_k such that $\mathbf{\Phi}(\delta_k)$ is linear function of $\mathbf{x}$ (monotonousness provides the existence of inversion). In that case the decision boundary $\{\mathbf{x} \in X | \delta_k(x) = \delta_j(x)\} = \{\mathbf{x} \in X | \mathbf{\Phi}(\delta_k(x)) = \mathbf{\Phi}(\delta_j(x))\}$ which is again a set of the form (1.4) and thus an affine space in X. The loosened assumption in other words states, that the decision boundary is generally a monotonous transformation of an affine space in the mapping of $\mathbf{\Phi}^{-1}$ in X.

1.4.2 Linear and quadratic discriminant analysis

In the following will be discussed only a two-class problem. So far no assumption has been made about the probability distribution of the training data. For the derivation of quadratic discriminant analysis and linear discriminant analysis classifiers normal distribution of the training data will be assumed. As already mentioned the discriminant function can be $p(\mathcal{G}_k|\mathbf{x})$ itself and for a

multivariate Gaussian distribution the probability density is given by

$$f_k(\mathbf{x}) = \left(\frac{1}{(2\pi)^{M_k}|\mathbf{\Sigma}_k|}\right)^{1/2} \exp\Big(-\frac{1}{2}(\mathbf{x}-\mu_k)^T\mathbf{\Sigma}_k^{-1}(\mathbf{x}-\mu_k)\Big) \tag{1.5}$$

where $\mathbf{\Sigma}_k$ is the covariance matrix of training data from category k, μ_k is the mean value of of training data from category k and M_k is the number of points labelled as of category $\mathcal{G}_k$. Considering monotonous transformation by logarithm and using (1.3), the decision boundary takes the form:

$$\ln\left(\frac{f_1(\mathbf{x})p(\mathcal{G}_1)}{p(\mathbf{x})}\right) = \ln\left(p(\mathcal{G}_1|\mathbf{x})\right) = \ln\left(p(\mathcal{G}_2|\mathbf{x})\right) = \ln\left(\frac{f_2(\mathbf{x})p(\mathcal{G}_2)}{p(\mathbf{x})}\right)$$

$$\begin{aligned} 0 = \ln\left(\frac{f_1(\mathbf{x})}{f_2(\mathbf{x})}\right) + \ln\left(\frac{p(\mathcal{G}_1)}{(p(\mathcal{G}_2)}\right) = \ln\left(\frac{p(\mathcal{G}_1)}{(p(\mathcal{G}_2)}\right) + \frac{1}{2}\ln\left((2\pi)^{M_2}|\mathbf{\Sigma}_2|\right) - \\ - \frac{1}{2}\ln\left((2\pi)^{M_1}|\mathbf{\Sigma}_1|\right) - \frac{1}{2}(\mathbf{x}-\mu_1)^T\mathbf{\Sigma}_1^{-1}(\mathbf{x}-\mu_1) + \frac{1}{2}(\mathbf{x}-\mu_2)^T\mathbf{\Sigma}_2^{-1}(\mathbf{x}-\mu_2) \end{aligned} \tag{1.6}$$

This is a quadratic function in $\mathbf{x}$ and with the discriminant function

$$\delta_k(x) = \ln\left(p(\mathcal{G}_1)\right) - \frac{1}{2}\ln\left((2\pi)^{M_k}|\mathbf{\Sigma}_k|\right) - \frac{1}{2}(\mathbf{x}-\mu_k)^T\mathbf{\Sigma}_k^{-1}(\mathbf{x}-\mu_k) \tag{1.7}$$

it defines the **quadratic discriminant function** which forms a basis of **quadratic discriminant analysis** or shortly **QDA**.

When further assuming univariate distribution $\mathbf{\Sigma}_1 = \mathbf{\Sigma}_2 =: \mathbf{\Sigma}$, we can simplify the form of the decision boundary to

$$\ln\left(\frac{p(\mathcal{G}_1)(2\pi)^{M_2/2}}{(p(\mathcal{G}_2)(2\pi)^{M_1/2}}\right) - \frac{1}{2}(\mu_1+\mu_2)^T\mathbf{\Sigma}^{-1}(\mu_1-\mu_2) + \mathbf{x}^T\mathbf{\Sigma}^{-1}(\mu_1-\mu_2) = 0 \tag{1.8}$$

which is a linear function in $\mathbf{x}$ and thus the decision boundary is a hyperplane (an affine space) in X perpendicular to $\mathbf{\Sigma}^{-1}(\mu_1-\mu_2)$. The discriminant function has a form:

$$\delta_k(\mathbf{x}) = \mathbf{x}^T\mathbf{\Sigma}^{-1}\mu_k - \frac{1}{2}\mu_k^T\mathbf{\Sigma}^{-1}\mu_k + \ln\left(\frac{p(\mathcal{G}_1)}{(2\pi)^{M_1/2}}\right) \tag{1.9}$$

and it defines the **linear discriminant function** which forms a basis of **linear discriminant analysis** or shortly **LDA**.

A final remark should be made: the assumption of multivariate Gaussian distribution is not necessarily needed and LDA can be derived also using the method of least squares. The equivalence of these two approaches is described in detail for example in [4].

1.4.3 Support vector machines

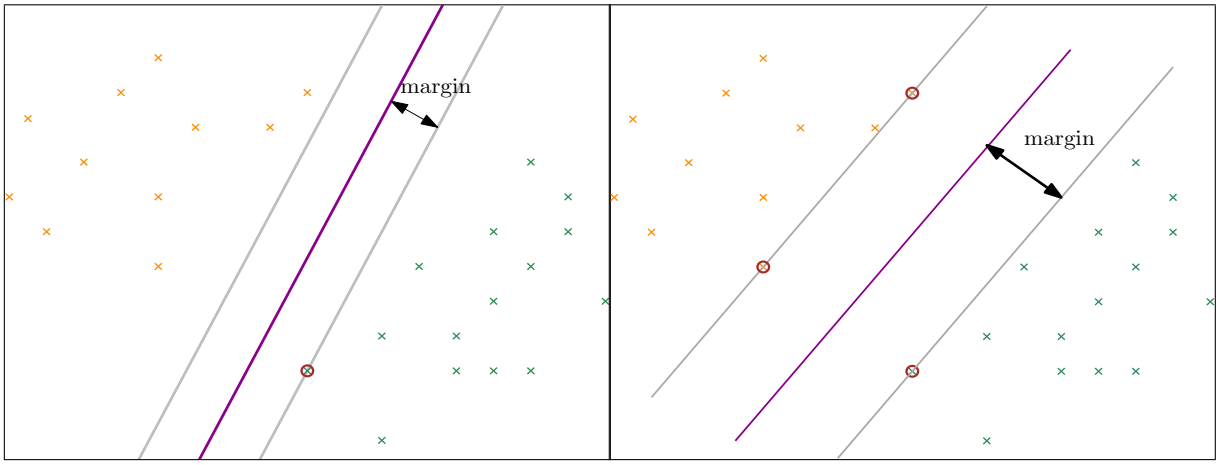
This section on **support vector machines** or shortly **SVM**s will use more general form of a linear model of classification. At first, the two-class problem of linearly separable data will be discussed. The assumption of linear separability is equivalent to the existence of a hyperplane (affine space) of the form

$$y(\mathbf{x}) = \mathbf{w}^T \cdot \mathbf{\Phi}(\mathbf{x}) + b \tag{1.10}$$

which separates the two sets of data for some $\mathbf{w}$, b and spatial transformation Φ (from the assumption of transformation follows, that for Φ exists also inverse transformation, which is defined everywhere

Figure 1.1: An example of the difference between an arbitrary separating hyperplane and the optimal separating hyperplane found by the SVM algorithm. The separating hyperplane is purple, grey lines parallel to the separating hyperplane are in a distance of a margin. Support vectors are marked by circles.

(a) Arbitraty separating hyperplane (b) Optimal separating hyperplane



on X). If there exists more than one pair of parameters $\mathbf{w}$ and b then there exist an infinite number of parameters defining separating hyperplane and we can choose a subset. Example of such situation in two dimensional feature space is given in Figure 1.1a. The purple line makes a linear decision boundary but there exists a better one as shown in Figure 1.1b. We can quantify the quality of a separating hyperplane by the size of a **margin**, which is defined to be the smallest distance between the decision boundary and any of the sample points [4]. Sample point in the distance of margin from the separating hyperplane are **called support vectors** and the SVM algorithm finds such support vectors which maximize the size of margin.

Suppose again there is a training set that comprises of M inputs $(\mathbf{x}_1, \ldots, \mathbf{x}_M)$, each assigned to one of two categories $\mathcal{G}_1$ and $\mathcal{G}_2$. Numerical values ± 1 can be assigned to output categories so that for every input $\mathbf{x}_n$ there is t_n having value either 1 or -1 corresponding to $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively. The linearity of (1.10) provides the freedom of choice $\mathbf{w}$ so that all $\mathbf{x}_n$ are assigned to classes according to the sign of $y(\mathbf{x}_n)$ or equivalently so that the training data satisfies $t_n = \text{sign}(y(\mathbf{x}_n))$ for all samples. From this choice follows $t_n y(\mathbf{x}_n) > 0$ for all samples in training data.

From simple algebra follows it that the distance of any point in a feature space from the separating hyperplane is given by

$$\frac{|y(\mathbf{x})|}{||\mathbf{w}||} = \frac{|\mathbf{w}^T \cdot \mathbf{\Phi}(\mathbf{x}) + b|}{||\mathbf{w}||} \tag{1.11}$$

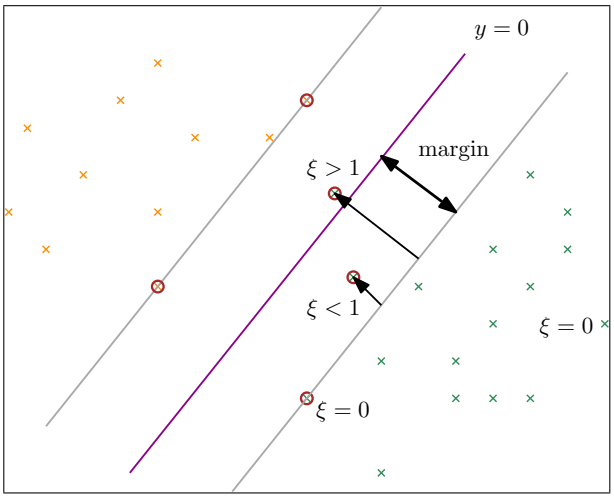
We want to find the smallest distance between the hyperplane and any of the sample points. We are interested in the distance of a sample point about which we already know that $t_n y(\mathbf{x}_n) > 0$. Thus the distance of $\mathbf{x}_n$ from the decision boundary is given by

$$\frac{t_n y(\mathbf{x}_n)}{||\mathbf{w}||} = \frac{t_n(\mathbf{w}^T \cdot \mathbf{\Phi}(\mathbf{x}) + b)}{||\mathbf{w}||} \tag{1.12}$$

Margin is defined to be the smallest of these distances and we want to maximize the margin with respect to $\mathbf{w}$ and b. The optimization problem for $\mathbf{w}$ and b can be than written as

$$\arg\max_{\mathbf{w},b} \left\{ \frac{1}{||\mathbf{w}||} \min_n \left(t_n(\mathbf{w}^T \cdot \mathbf{\Phi}(\mathbf{x}) + b) \right) \right\} \tag{1.13}$$

Figure 1.2: Illustration of using slack variables for SVM using non separable data



Solution of this problem is analytical and leads to a problem of quadratic programming. The solution is discussed in more detail for example in[4].

What has not been mentioned yet is a problem of applying SVM to data which are not linearly separable. In that case *slack variables* $\xi_n \geq 0$ will be added for all sample points to express the loss for incorrect classification. In addition the transformation $b \to \kappa b$ and $\mathbf{w} \to \kappa \mathbf{w}$ do not change the distance between a point and the decision boundary given by (1.12) and it allows the choice of setting

$$t_n(\mathbf{w}^T \cdot \mathbf{\Phi}(\mathbf{x}_n) + b) = 1 \tag{1.14}$$

for all support vectors. Slack variables are set to be zero for all correctly classified samples having the distance from the decision boundary larger than 1. For all other points, the slack variable are defined to satisfy $\xi_n = |t_n - y(\mathbf{x}_n)|$. Thus all support vectors will have $\xi_n = 0$ and points laying on the decision boundary (for which $y(\mathbf{x}_n) = 0$) will have $\xi_n = 1$ as shown in Figure 1.2. The problem is then very similar to the one with linearly separable data and it again leads to a problem of quadratic programming. Detailed solution is given for example in [4] and it will not be discussed here further.

A problem of dividing non-linearly separable data can by approached by a feature space transformation. In that case a **kernel** is used to transform the feature space into other high dimensional space where data become linearly separable. Kernel is a function of a form

$$k(\mathbf{w}^T, \mathbf{x}) = \mathbf{\Phi}(\mathbf{w})^T \cdot \mathbf{\Phi}(\mathbf{x})$$

for some fixed nonlinear feature space mapping $\mathbf{\Phi}$. The linear model (1.10) then takes the form

$$y(\mathbf{x}) = k(\mathbf{w}^T, \mathbf{x})$$

In this project will be used two kinds of kernels for SVM. The first one uses $\mathbf{\Phi} = \mathbf{I}_X$, the identity mapping on the feature space, the second one will be a polynomial of the form

$$k(\mathbf{x}, \mathbf{w}) = (\mathbf{w}^T \cdot \mathbf{x} + b)^p$$

for some b and $p \in \mathbb{N}$ being the order of the polynomial. The performance of these two classifiers will be compared in Chapter 4.

Chapter 2

Background

A work on traffic patterns of particular http application [2] studied 20 different applications (among which was also Youtube, Dailymotion or Radioways - web radio streaming) covering 12 different types of applications. The traffic was made artificially by using the application for a certain amount of time and recorded. The information about the patterns of Youtube or web-radio streaming traffic agree with the observation of stable, repetitive traffic which was used to classify media streaming in this work. For more details see Figure 2 and 3 in [2].

Proxy server service was also used to classify users of a university network in order to control and better use the networks total bandwidth. The work is not focused on classifying media streaming, but the on identification of users with large amount of transferred data. High bandwidth users are restricted during peak hours to enable others to use the internet [12].

Media streaming classification using keywords detection and statistical properties of server-client data traffic was studied in [11]. The proposed method achieved both high precision and recall although it might lack robustness and is not applicable to `https` traffic.

In a research described in [14] was applied SVM classifier to detect different internet applications communication over a wide range of protocols, such as `http`, `https`, ftp, pop3, smtp etc. Our work has a different aim - classify different types of content transferred over one or two protocols.

Chapter 3

Classifying media streaming

3.1 Data and proxy log attributes, what is media streaming

The data I was working with consisted of millions of proxy logs. Each proxy log had 12 proxy log attributes. The description of each attribute is given in Table 3.1. An example of a proxy log is given in Table 3.2. In the rest of this text I will refer to the proxy log attributes as they are given in Table 3.1. These logs come from a communication record of a whole network of one client of Cisco. From the description of the proxy log attributes it is clear that these data do not provide much information about one particular request. Considering that not all attributes are always available (for example because of encrypted `https` traffic) finding a feature that would describe particular traffic is challenging. The easiest way of finding a request belonging to some category of traffic is to search the *url* for string patterns that are known to fall into that category or the *content-type-RS*. Unfortunately this approach is not applicable for example for `https` traffic.

Table 3.1: Description of proxy log attributes

Feature	**Description**
timestamp	Time based attribute in milliseconds.
httpstatus	Numerically coded status of connection to a webpage.
scb	Size of data transferred form server to client.
csb	Size of data transferred form client to server.
url	Target internet address.
userID	Hashed ID of PC in client's network.
elapsedtime	Time interval the request took.
s-IP	IP address of a server to which client has conntected.
c-IP	IP address of a client.
content-type-RS	Sent content is classified by the server into groups to identify the communication. These groups are registered and managed by IANA[1]. Groups are organized into *types* and *subtypes* and can include additional information. The classification is, however, unreliable and therefore cannot be used as a feature of further classification.
referrer	URL of web site, which sent client to current web page.
user-agent	Web browser and additional information about clients PC

[1] https://www.iana.org

Table 3.2: Proxy log example. Content-type-RS, referrer and user agent and are not always available.

Feature	**Value**
timestamp	1382058540001
httpstatus	200
scb	0
csb	13470
url	https://remote.astpl.com/
userID	06a3c2f922cbf5f00213298723be88b0
elapsedtime	47012
s-IP	203.59.96.242
c-IP	[2]
content-type-RS	[][3]
referrer	[]
user-agent	Mozilla/4.0

My task was to find media (audio and video) streaming traffic. Because of its today's popularity, identifying such traffic can greatly reduce the amount of unknown requests that would have to be searched for unwanted or malicious traffic. It is also assumed, that media streaming traffic is not malicious and can be labelled as *safe*.

Example sets containing one or other type of traffic had to be created to find distinctive features. For that purpose *content-type-RS* feature and *url* was used to filter certain type of traffic and also to search for names of well-known video streaming websites. Filtered flows were further manually analysed and cleared. Not all approaches to the problem of identifying media streaming were equally successful, as will be seen in further chapters.

For the purposes of this work media streaming will be defined defined by *content-type-RS* values. Flows categorised as media streaming are those having the value of *content-type-RS* any subtype of `video` excluding `video/x-ms-asf` and `video/vnd`, `application/octet-stream` containing string "youtube.com" and not containing string "google" in *url* (to prevent including Google search results), any subtype of `audio`, `application/x-shockwave` and `application/x-fcs` having *scb*>10kB and *csb*< 100B.

The prefix "www" was omitted from the string "www.youtube.com" because *url* of the form `https://www.youtube.com/watch?v=IvdYyil6IEk` will link you to the web page where you can see a video of funny animals, but the actual content of the page - the video and audio itself, comes from a Youtube content delivery server with an address of a form `http://r17-sn-aiglln7y.c.youtube.com/videoplayback...`. More about the delivery cloud system of Youtube is given for example in [1].

Some types such as `video/x-ms-asf` or `video/vnd` had to be excluded, even though they are a subtype of `video`. The first subtype was excluded because *url* of flows of this type contained patterns indicating advertisement and it was not possible to check the actual content by opening the *url* address in a web browser. The second subtype was excluded, because these subtypes are registered under various vendors according to [8], which makes them unreliable.

Probably the most controversial is the type `application/x-fcs` and its conditioning. This content type was added to the media streaming traffic after manual analysis and it is considered to be video traffic for example in [3]. According to [9] it really should be a video traffic transferred over the real-

[2]Sensitive data

[3]Empty value - not all feature values are always available.

time-messaging protocol . However, flows with this content type make up to 10% of all internet traffic and it is highly improbable to have so many media streaming requests only of this type. In addition, most of the traffic of this type has *scb*=1B, which is too little to contain a video (or anything else). After thorough analysis of the `application/x-fcs` content type I decided to categorise as media streaming only those flows having *scb* value greater than 10kB and *csb* value smaller than 100B. This content type will not be included in the training sets, because the flows are difficult to label even by manual analysis.

For the matter of consistency and simplicity I will further refer to media streaming traffic only as to *media* and to any other traffic as to *mediaC*.

3.2 Naive methods (based on simple models)

The first approach to *media* identification is using statistical properties of given features of *media* traffic and comparing them to the statistical properties of the *mediaC* traffic. It is obvious from the description of proxy log attributes in Table 3.1 that only few features are actually giving some useful information about the traffic. *Timestamp*, *c-IP* and *user-agent* cannot describe any particular traffic in principle and *referrer* was not always available. Moreover both mode and median of *csb* are zero in random traffic (see Table 3.3). The only useful features for any statistical analysis left are *scb* or **length of url** (number of characters in the URL address) for which I will use the notation of *lurl*.

Before sorting logs by *content-type-RS* or *url* I removed logs with values of both *scb* and *csb* equal to zero (logs cleared of those having zero values will be in further text called **nonZero**). Example sets were created by filtering the rest of the data for *content-type-RS* values to form a *media* set as defined in the previous section with the only difference, that the type `application/x-fcs` was excluded from the set.

Figure 3.1 shows the distribution of *csb*, *scb* and *lurl* values within sets *media* and *mediaC* and also within subsets of *media* and *mediaC* of request with zero *csb* value (the distribution will be later used in Chapter 4).

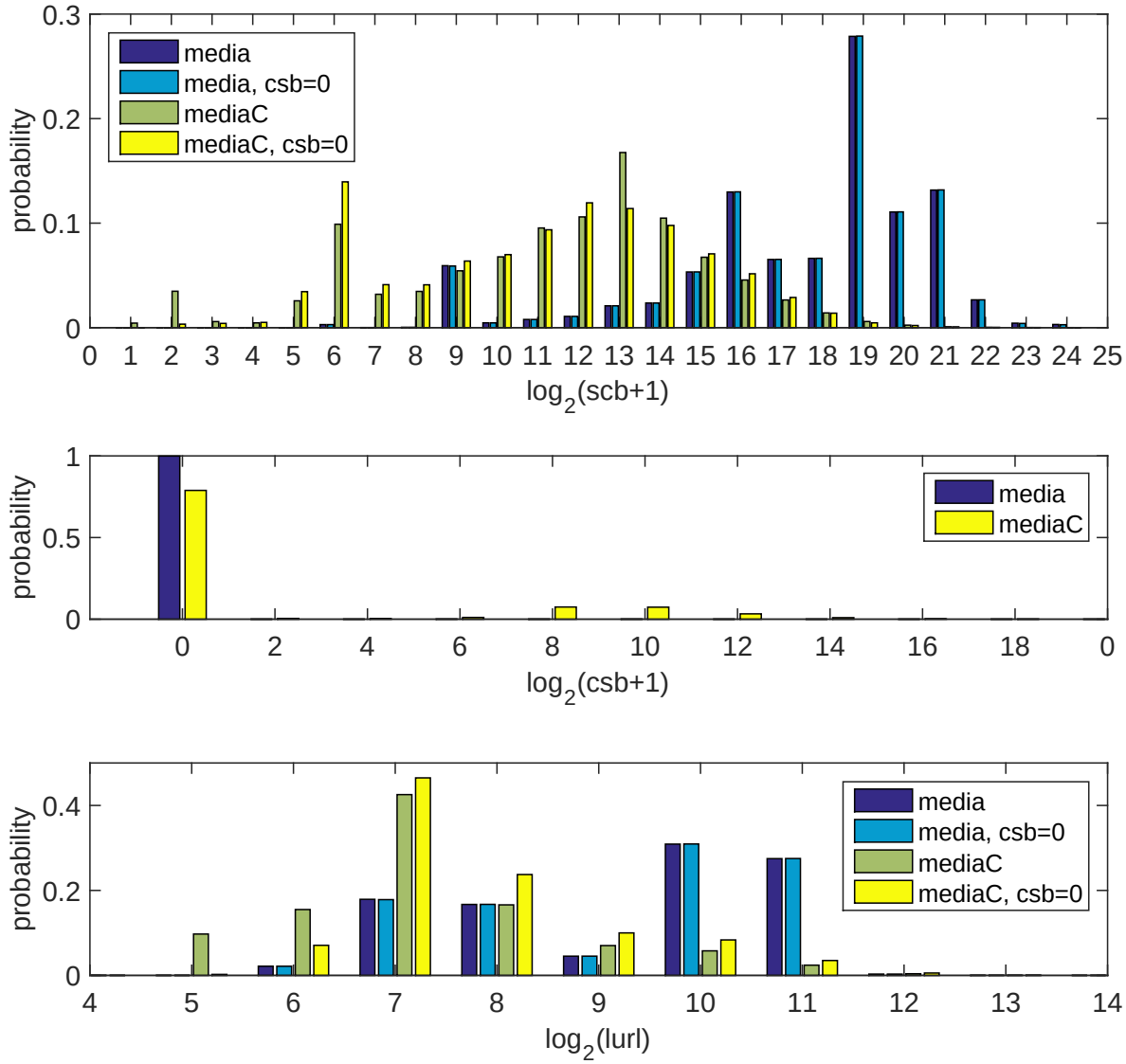
Table 3.3: Statistical properties of datasets used for naive classification obtained from more than 7 million flows. Note, that `application/x-fcs` was excluded from the set *media*.

Dataset (size)	*media* (211120)			*mediaC* (7191571)		
feature	*scb*	*csb*	*lurl*	*scb*	*csb*	*lurl*
mean	461.294kB	0.0003kB	477.8	19.240kB	3.362kB	133
mode	237.568kB	0kB	135	0.043kB	0kB	22
median	237.568kB	0kB	677	1.868kB	0kB	66

Dataset (size)	*media* with zero *csb* value (210912)			*mediaC* with zero *csb* value (4927570)		
feature	*scb*	*csb*	*lurl*	*scb*	*csb*	*lurl*
mean	458.223kB	0kB	478.2	18.195kB	0kB	177.6
mode	237.568kB	0kB	135	0.043kB	0kB	90
median	237.568kB	0kB	677	1.487kB	0kB	87

As we can see in the figures, it is obvious, that some simple conditioning by chosen features will not be much precise. Histograms of features distribution within requests with zero *csb* values were added because the conditioning on zero *csb* value was used in experiments. This value is typical for *media* traffic, as is shown in Table 3.3.

Figure 3.1: Distribution of *scb*, *csb* and *lurl* values within the sets *media*, *mediaC* and their subsets of request with zero *csb* value. Note, that `application/x-fcs` was excluded from the set *media*.



3.3 Single log behavioural analysis, repetitive features

3.3.1 Repetitiveness, used features

After rejecting simple statistical approach the data had to be further analysed. I used already constructed sets *media*[4] and *mediaC* and tried to find some rule or pattern which would differentiate them. I sorted the set *media* by *userID* to get a record of continuous communication over the internet of one particular user at a time. I discovered most *media* flows seemed to follow a repetitive pattern. This repetitiveness means that equally sized (or nearly equally sized) packages of data are being sent over and over again from the server. Some video websites send long videos in more than a hundred packages.

[4]Content type `application/x-fcs` was still considered to belong in the set *mediaC*.

The same was observed and called "short ON-OFF" strategy of video streaming in [13]. This behaviour is typical for Youtube, forming almost a half of all *media* requests. However, not all *media* follow this streaming strategy. Another possibility seems to be sending only one file which size is from 10^2 to 10^5 bigger than the mean *scb* of other *media* (again described also in [13]). Because of this behaviour only a very small percentage of proxy logs belong to this type of video communication and they are usually easily distinguishable by their size. My further aim became detecting media streaming using repetitiveness. I created a data extraction script[5] which groups flows by *user-ID* and *s-IP* and calculates additional data for them such as repetitiveness, length of *url* and other statistics. I sorted the data by the repetitiveness greater than 10 (more than 10 proxy logs of communication between the same *s-IP* and the same *userID* in a series has the same or similar *scb* value). Surprisingly more than 30% of all communication has the repetitiveness greater than 10. This was mainly because of very popular *scb* values - almost 30% of all traffic comprises of flows with *scb* smaller than 1kB, furthermore over 50% of these have one of following values: $\{0, 1, 2, 5, 13, 15, 16, 20, 35, 42, 43, 44, 49, 204, 202\}$. It is obvious, that the repetitiveness by itself (not even high) is not descriptive enough.

Another feature of all *media* communication seemed to be that no data is sent back to the server and so *csb* value is equal to 0. This feature is not very descriptive, because as was shown in Table 3.3 zero value of *csb* is the most common value of all. However, the value of *csb* helps filtering out unwanted repetitive flows.

A typical feature of *media* is also large *scb* value. Mean *scb* value of *media* is around 20 times greater than mean *scb* value of *mediaC* mode and median of *scb* was even much greater, as was seen in Table 3.3. Again, this feature itself cannot describe media traffic distinctively enough, but it helps with cleaning the repetitive flows belonging to *mediaC*.

Another feature, that is typical for *media* is that a connection to the website was successful and therefore *httpstatus* was set to 200. Unfortunately, this feature is not accessible for `https` traffic and therefore will not be included in all used feature vectors. The same applies to *lurl*, which is a good feature but unfortunately *url* is not available in `https`.

Last feature could be *url* itself. Searching through *url* for strings and patterns is very trivial and impractical way of identifying traffic. A list of video or audio streaming website domains has to be created and regularly updated to achieve a stable detection performance. Even then would be some *media* requests unrecognised, because their *url* does not contain anything else but IP address and they are therefore difficult to distinguish. This approach was not a goal of this work.

Statistical features can be added, having flows grouped in repetitive series. The similarity of *scb* values is already used but there is also a similarity of *lurl*. Many *media* flows in repetitive series have almost identical *url* address, which differ only by a number of a part of transferred file.

I succeeded in sorting out *media* from random traffic by suitable conditioning of values of only three given features. The first condition was zero *csb* value, the other two constraints on *scb* and repetitiveness were:

- *scb* > 200kB
  The value is large and filters out not only unwanted repetitive traffic but also around 65% of all *media* requests.

- repetitiveness > 10
  Repetitiveness proved to be the key feature. Although conditioning flows to zero *csb* and *scb* bigger than 200kB is already highly restrictive, repetitiveness reduces the number of different *content-type-RS* values by another 90% as shown in Table 3.4.

[5] Detailed description of this script will be given in Section 3.3.3 on page 25.

An example of descriptive potential of these features is shown in Table 3.4. In around 600000 nonZero flows (cleared of flows with both *scb* and *csb* value equal to zero) there is more than a hundred different types of *content-type-RS*. After restricting this dataset by *csb* and *scb* values, we get around 14000 logs. As we can see from the table these logs still have *content-type-RS* values in a wide range of types. Conditioning these 14000 logs by repetitiveness value resulted in around 7000 logs with only 5 *content-type-RS* values. Moreover, we get only those types, which belong to *media*. Using other datasets of proxy logs, `image/jpeg`, `application/ms-dos` or `text/` types appeared but they were very rare (less than 1% of all logs) and some of those logs included in *url* patterns suggesting they actually belonged to *media* (e.g. ".flv" file extension , word "video" etc.).

As we can see, there is a possibility of filtering out some part of media streaming only by using *s-IP*, *userID* (*c-IP*), *csb* and *scb*, which are usually the only attributes available in `https` traffic. Adding *httpstatus* and *lurl* can only improve the identification performance.

It seems that *timestamp* could be used to create some time window in which flows would have to follow one another. In practice, however, the time delay between two flows in a series was apparently dependent on the download speed of each particular user and therefore did not provide widely applicable information.

3.3.2 Training sets

The next step is to use the descriptive properties of the selected features to train a classifier. In order to do so appropriate training sets need to be created and descriptive feature vector designed. I will now discuss creating of the training sets (positive set will include only *media*, negative will include only *mediaC*).

The first positive training set satisfies the definition of the set *media* given in Section 3.1., only excluding `application/x-fcs`. It was excluded because it might pollute the training set resulting in decrease of classification performance.

The second positive training set was created by searching for strings in *url* in addition to searching through *content-type-RS* values for video types. It was searched for names of well-known video websites: "youtube.com", "vimeo", "metacafe", "hulu.com", "veoh.com", "abc.com", "dailymotion" and "stream.cz". Unfortunately, most of these were not found. Only Youtube and Vimeo proved to be popular and also some flows containing string "hulu.com" and "dailymotion" appeared. The results of searching in *content-type-RS* for `video` were added to form a set of almost 200 000 logs found in more than 9 million nonZero logs. These logs were than put for data extraction by designed extraction script[6] and manually cleared of *content-type-RS* types with the help of the list of content types [8] not to include any type of traffic unrelated to media streaming. In the end I had around 140 000 logs. The second training set was aiming on the detection of video streaming only.

The third positive training set was based on the second one, but the logs were further restricted by *scb* and *csb*. I wanted to have a training set similar to what I found only by using rules as discussed in Section 3.3.1. I set the minimum *scb* to 150kB, *csb* again equal to 0, and repetitiveness greater than 10. The condition on *scb* was slightly loosened to get more data so that the classifier could be trained to distinguish more video communication than just the most evident.

All three positive sets will be further referred to by their names given in Table 3.5 or commonly as *media*sets, even though they do not follow the exact definition of *media* given in Section 3.1.

To form a negative set I used again information from *content-type-RS* to filter out request corresponding to *media*. This dataset was further cleared of very frequent communication with *scb*=1. These obviously do not belong to the type of communication I was trying to find and they might interfere

[6] See Section 3.3.3 on page 25.

Table 3.4: Proxy log conditioned by *scb*, *csb* and repetitiveness values and grouped by *content-type-RS* values. Set1 (≈14000 logs) obtained by conditioning more than 600000 logs only by *scb* and *csb*, Set2 (≈7000 logs) is conditioned also by repetitiveness.

content-type-RS	**Set1**	**Set2**
application/font-woff	2	0
application/java-archive	1	0
application/javascript	61	0
application/msword	1	0
application/octet-stream	10507	6739
application/pdf	70	0
application/pkix-crl	6	0
application/vnd.ms-fontobject	5	0
application/x-font-woff	2	0
application/x-javascript	142	0
application/x-msdos-program	104	0
application/x-pkcs7-crl	1	0
application/x-shockwave-flash	316	22
application/x-silverlight-app	1	0
application/x-swz	1	0
application/x-unknown	3	0
application/xml	1	0
application/zip	2	0
audio/mp4	192	105
audio/mpeg	46	0
audio/x-wav	1	0
font/eot	2	0
image/bmp	1	0
image/gif	170	0
image/jpeg	929	0
image/png	262	0
image/svg+xml	6	0
image/vnd.microsoft.icon	1	0
image/x-icon	1	0
multipart/byteranges	65	0
text/css	144	0
text/html	172	0
text/javascript	82	0
text/plain	79	0
text/xml	16	0
unknown/unknown	10	0
video/f4f	20	0
video/mp4	230	103
video/quicktime	1	0
video/x-flv	244	163
video/x-ms-wmv	1	0

Table 3.5: Training sets used for single log classification - summary

Name	**Brief description**	**Size**
trSet0	Flows searched for string patterns in *url* and for `video` in *content-type-RS*. Then manually cleared using *content-type-RS* values to include only video related traffic.	142290
trSet1	Flows from trSet0 restricted by *scb*>150kB, *csb*=0 and repetitiveness > 10.	87845
trSet2	*media* without `application/x-fcs`	92744
mediaC	Flows searched for *content-type-RS* values related to *mediaC* and restricted by *scb*>1.	468823

with the training of a robust classifier because of their common large repetitiveness values (the requests followed one after another dozens of times).

Some statistical information about all four sets is given in Table 3.6, list of *content-type-RS* values of positive training sets is given in Table 3.7 and a summary about obtaining training sets is provided in Table 3.5. Histograms of *scb*, *csb*, *lurl* and repetitiveness are given in Figure 3.2 to show the distribution of these features within the training sets. The basis of the logarithm is in the figure stated, because the features were not yet normalised.

In further text I will refer to all training sets by their name given in Table 3.5.

Table 3.6: Statistical properties of training sets used for single log classification. Length of URL is measured by the number of characters, rep (repetitiveness) does not have a unit.

Dataset	trSet0				trSet1			
feature	*scb*	*csb*	*lurl*	rep	*scb*	*csb*	*lurl*	rep
mean	529.642kB	0.006kB	632	111	559.056kB	0kB	718	88
mode	237.568kB	0kB	135	1	237.568kB	0kB	785	12
median	241.644kB	0kB	704	28	249.856kB	0kB	719	38

Dataset	trSet2				mediaC			
feature	*scb*	*csb*	*lurl*	rep	*scb*	*csb*	*lurl*	rep
mean	400.735kB	0kB	514	79	18.404kB	1.249kB	146	68
mode	237.568kB	0kB	135	1	43kB	0kB	22	1
median	237.568kB	0kB	686	20	1.972kB	0kB	77	4

Figure 3.2: Distribution of *scb*, *lurl* and repetitiveness within the training sets used for single log classification.

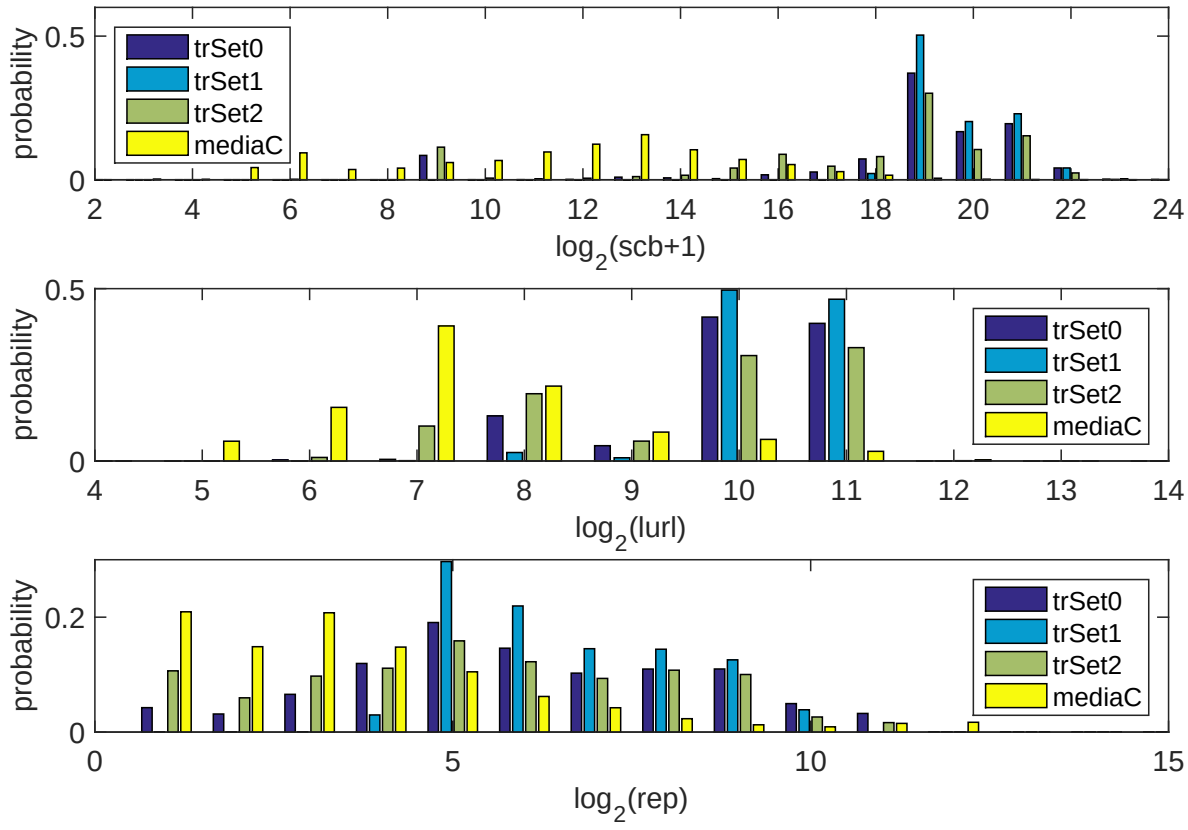


Table 3.7: List of *content-type-RS* values of training sets used for single log classification. Negative training set mediaC includes 71 different *content-type-RS* values which will not be listed.

content-type-RS	**trSet0**	**trSet1**	**trSet2**
application/octet-stream	104110	79766	52236
application/x-shockwave-flash	1545	0	19340
audio/mp4	0	0	1671
audio/mpeg	0	0	825
audio/ogg	0	0	24
video/3gpp	8	2	4
video/abst	12000	0	10436
video/f4f	5795	791	3838
video/f4m	0	0	16
video/flv	16	15	0
video/mp2t	998	0	757
video/mp4	14216	5880	2660
video/quicktime	2	0	2
video/webm	23	2	4
video/x-f4f	19	0	0
video/x-flv	3300	1389	931

3.3.3 Feature extraction algorithm

Require: *data*[7], *lim*, *memlen* $\in \mathbb{N}$, *interval* $\in [0; 1]$
Ensure: *flows with computed features*

find nonZero requests (cleared of requests having zero both *scb* and *csb* value)
for each unique *userID* **do**
  find all logs belonging to *userID*
  find all unique *s-IP* values
  for each *s-IP* **do**
    find all logs belonging to this *s-IP*
    if # flows $\geq$ *lim* **then**
      dataset $\leftarrow$ all logs with current *userID* and *s-IP*
      create array *memory* of size *memlen*×2, first column will contain *scb* values, second column will contain position vectors of flows relative to *dataset*
      for each flow $\in$ *dataset* **do**
        current $\leftarrow$ *scb* value of current flow
        current_memvalues = current first column of memory
        if $\exists$ *value* $\in$ *current_memvalues* : *current* $\in$ (1-*interval*;1+*interval*)**value* **then**
          add to the second column of *memory* on the row containing *value* the position of current flow relative to *dataset*
          value $\leftarrow \frac{\text{(length of position vector)}*value+current}{\text{length of position vector+1}}$
        else
          if *memory* is not full $\Leftrightarrow \exists$ empty row in memory **then**
            save *current* as a new value in *current_memvalues*
            add to the second column of *memory* on the same row position of *current* flow relative to *dataset*
          else
            save *current* to *current_memvalues* instead of the oldest record of *memory*
            return the position vector of the oldest record in *memory*
            add to the second column of *memory* on the row containing *current* the position of current flow relative to *dataset*
          end if
        end if
      end for
      use returned position vectors to add length of the position vector to every flow on a position given by this vector
      add length of url to every proxy flow
      find additional statistics on *scb* and *lurl* within flows grouped by position vectors
    else
      do not use these logs and jump to next *s-IP*[8]
    end if
  end for
end for
return logs with additional statistics and repetitiveness value

[7]The data stream would be buffered for a certain amount of time and than processed, making it possible to treat *data* as a finite set of proxy logs.

[8]The algorithm discards a certain amount of flows. This problem can be solved in practice for example by keeping these flows and adding them to the next buffered dataset.

3.3.4 Enriching feature vector for single log classification

Feature vector had to be designed using chosen features, to be as much successful in the traffic identification as possible. That required further analysis and tests to see, which features help with the description. The feature vector had to include more than just the five features found.

For many values it is better to add a logical value so that the classifier is more sensitive to it. This is also because of numerical imprecision but mainly because of the way the classifier works. It naturally makes the separating boundary somewhere near the descriptive value so that it maximizes the separation of the data (the size of the margin as seen in Section 1.4.3) and minimizes all penalizations from outlying points. This is unfortunate when only one value of many is important. For example *httpstatus* 200 is of my interest and status values range from 100 to slightly under 600 and there are of course values such as 201, 202,..., 208. Moreover these values are not uniformly distributed, because they are categorical and so the number is not related to the position on any axis. They are rather concentrated over the value of every hundred. The value 200 might be almost undistinguishable from 201 or 202 and so I added logical value: *httpstatus*==200 (if *httpstatus* = 200 then 1; else 0). Following the same logic I also added logical value *csb*==0 because of its importance.

In order to make some values more comparable (for example *scb* ranges from 0B to 10^{10}B) log values were added (before that I added 1 to *scb* to have all *scb* values greater than 1 and thus positive log values). The basis of logarithm is not stated, because it is not fixed. Further normalization of all feature values renormalizes also the basis of logarithm along the rule:

$$(\forall a, b, c > 0) \quad \log_a(b) = \frac{\log_c(a)}{\log_c(b)}.$$

Repetitiveness plays two important roles in the classification. First of all it clearly separates flows with no repetitiveness from the repetitive ones by a trivial limit: repetitiveness > 1. The second role is the quantification of how much certain communication is repetitive. Communication with a repetitiveness smaller than 7 is very different from that with a repetitiveness over 30. To cover all possible applications of repetitiveness, several additional features were created. To help the classifier with comparison, logarithm was added (again positive) and also two types of logical bins (if the repetitiveness value belongs to an interval 1; otherwise 0).

Another element of the final feature vector is *lurl*. In addition to thinking of flows as of groups of flows in the same series of similar *scb*, we can add some statistics of *lurl* within the group of flows. The most straightforward would be the standard deviation but it ranged from 10^2 to 10^{-8} and eventually 0. Mean value of this feature was somewhere around 10^{-5}. Uniformly distributed features are easier to separate and so I transformed the feature by logarithm after addition of 1.

Speaking of statistics within a group of flows with a similar *scb* it is apparent to use some statistics of *scb* itself - I chose mean of distances from the mean *scb* within the group. I am using this statistics and not for example standard deviation, because one of the parameters in my data extraction script is interval of values into which repetitive flows have to fall and I wanted to have these two values more intuitively comparable.

The final feature vector comprises of 25 features, all elements normalized to range from 0 to 1. The entire vector is given in Table 3.8. Because of the similarity of logical values 11-16 and 17-24 I decided to test their difference. Every classifier will therefore be trained in three versions depending on what elements of the feature vector are used. I will also make a comparison between classifiers which are trained using *httpstatus* and *lurl* and those which do not have these values available. The results will show, how much more difficult it is to identify media traffic in `https` traffic than in `http` traffic.

The final six versions of the feature vectors are given in Table 3.9.

Table 3.8: Final feature vector for single log classification. Some feature names were abbreviated in this table: http = *httpstatus*, rep = repetitiveness.

1	http	10	Δ rep	18	log(rep+1)≤1
2	http==200	11	log (rep+1)	19	1<log(rep+1)≤2
3	*lurl*	12	rep < 3	20	2<log(rep+1)≤3
4	log (std(*lurl*)+1)	13	3 ≤rep<9	21	3<log(rep+1)≤4
5	*scb*	14	9 ≤rep<20	22	4<log(rep+1)≤5
6	log (*scb*)	15	20 ≤rep<40	23	5<log(rep+1)≤6
7	*csb*	16	40 ≤rep<60	24	6<log(rep+1)≤7
8	*csb*==0	17	60 ≤rep	25	7<log(rep+1)≤8
9	rep				

Table 3.9: All versions of the feature vector for single log classification. Using the notation (n:m) = (n,n+1,...,m-1,m)

Using *lurl* and *httpstatus*	**Name**	**Elements of feature vector with respect to Table 3.8**
YES	vector11	(1:25)
	vector12	(1:17)
	vector13	(1:11,18:25)
NO	vector01	(5:25)
	vector02	(5:17)
	vector03	(5:11,18:25)

3.4 IP behavioural analysis, repetitive features

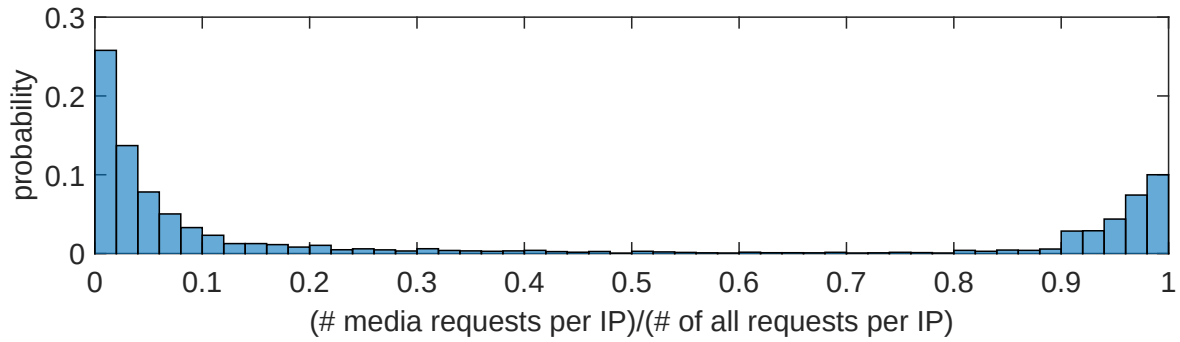
Another option of repetitive traffic classification will be discussed in this section. So far only a classification of the traffic flow by flow was discussed. The aim now will be to classify each *s-IP* as either transferring *media* or *mediaC*. This approach requires the assumption that very specific traffic goes from one particular IP address. Under this assumption, it should be possible to sort the content coming from one website only by server IP addresses.

I slightly altered the labelling script described in Section 3.3.3 so that it groups proxy log only by *s-IP* address (the differentiation of *userID* is no longer needed). This approach provides different data to work with and it is possible to use more descriptive statistics for each *s-IP* than for one log in the case of a single log classification.

3.4.1 Training sets

The first step is to create new training sets. Using the same training sets as for single log classification would not lead to training of a robust classifier. It seems that traffic of very different types (images, text, video, JS applications etc. all together) comes through one IP address. The classification then does not filter out *media* requests from *mediaC* requests, it rather distinguishes IP addresses of two types. IP addresses of the first type transfer *media* content, IP addresses of the second type transfer *mediaC* content only. The training sets used for the single log classification, however, included strictly only one category of traffic and therefore all data for particular IP would be either of *media* category or of *mediaC* category. The testing data, on the other hand, contain unsorted data and a dataset of flows with the same *s-IP*

Figure 3.3: Distribution of ratios of *media* requests relative to all requests per one IP. The distribution covers only 7679 IP addresses containing at least one *media* request. No *media* requests were found in 88% of all IP addresses. Note, that `application/x-fcs` was excluded from the set *media*.



transferring *media* streaming includes also requests unrelated to *media* according their *content-type-RS* values.

While creating training sets, flows were grouped into sets of flows with the same *s-IP*. These sets were then sorted by *content-type-RS* values into subsets of *media* and *mediaC* requests, excluding the content type `application/x-fcs` from *media* not to pollute the training sets. Figure 3.3 shows a percentage of *media* flows relative to all flows per one IP (data from all IP addresses with the ratio greater than zero, which makes around 12% of all IP addresses). The threshold on the ratio to describe IP addresses transferring *media* traffic was set to 70%. That means, that an IP address will be labelled as belonging to the class *mediaIP*, when at least 70% of all requests coming from this address are *media* requests, otherwise it is labelled as an element of class *mediaCIP*.

Two obvious options how to form a negative training set (set of IPs transferring *mediaC* traffic) suggest itself. Either to take any IP which does not belong to *mediaIP* category, or only those, from which none media requests come. Because both these options are viable, both will be taken and tested, only with a little difference. The first training set will include all IP addresses, for which the ratio of the number of *media* flows to all flows is smaller than 10%. According to the histogram in Figure 3.3 only little data will be lost and we will not have to be afraid of polluting the negative set. Brief description of the training sets is given in Table 3.10 some interesting distributions of features within training sets are given in Figures 3.4-3.5.

Table 3.10: Training sets used for IP classification - summary

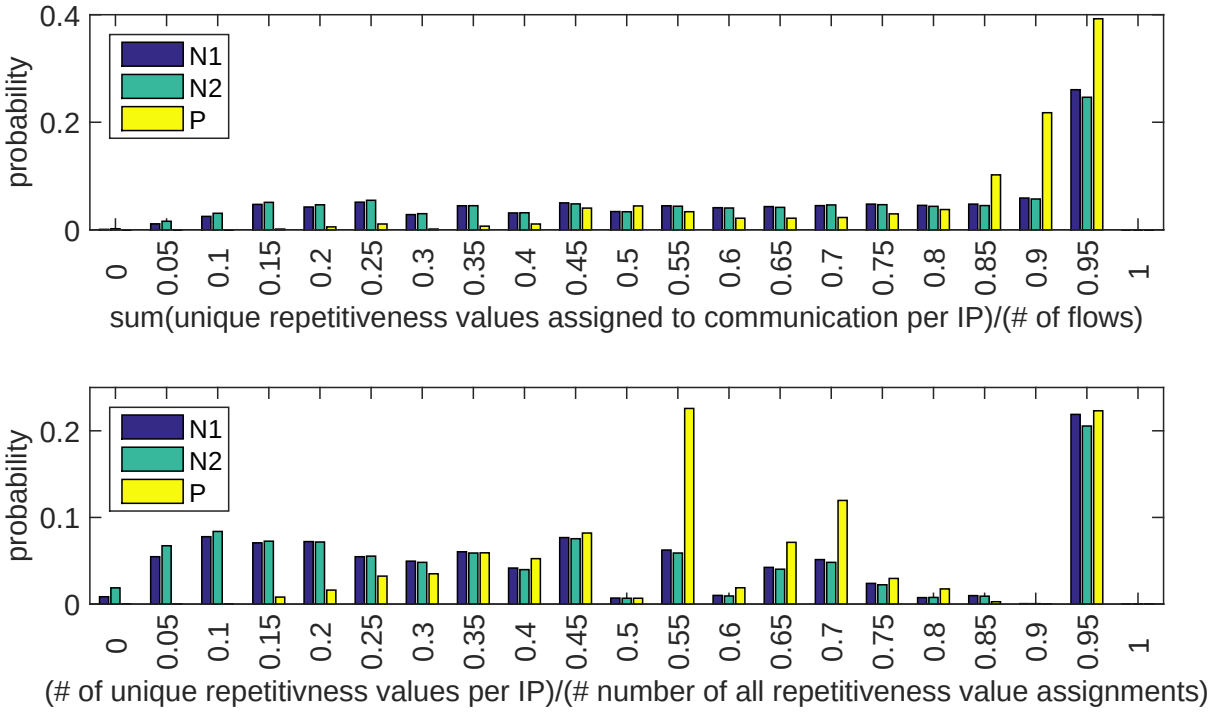
Name	**Brief description**
P	Class *mediaIP* - Proxy logs grouped into dataset of request from one *s-IP*. Then sorted by *content-type-RS* values to either *video* or *mediaC* traffic. IP address belongs to class `mediaIP` if at least 70% of all logs with the *s-IP* belong to *media* category.
N1	Class *mediaCIP* - Proxy logs grouped and sorted the same way as for P set. IP address belongs to this class, if all requests send from the *s-IP* were from *mediaC* class.
N2	Class *mediaCIP* - Proxy logs grouped and sorted the same way as for P set. IP address belongs to this class, if 10% or less requests send from the *s-IP* were from *media* class.

3.4.2 Used features, final feature vector

Feature vector for this type of communication uses all values available. Having a set of communication coming though one IP address it is possible to use standard statistics such as mean, mode, and median on all vectors of data (*scb*, *csb* etc.) from the set. These statistics will be used for repetitiveness, *scb*, *csb* and *lurl*. Transformed log values of repetitiveness and *scb* will be added after addition of 1. Mean, mode or median of *httpstatus* would not be very useful, instead logical value mode(*httpstatus*)==200 can be descriptive in the same sense as *httpstatus*==200 for the single log classification. Another logical values were used for mode(*scb*)==0 and mean(*csb*)==0 (both mode and mean are used because zero mean is much more restrictive than zero mode as shown in Figure 3.5), min(repetitiveness)>1, and the same logical bins for log(repetitiveness+1) as for the single log classification.

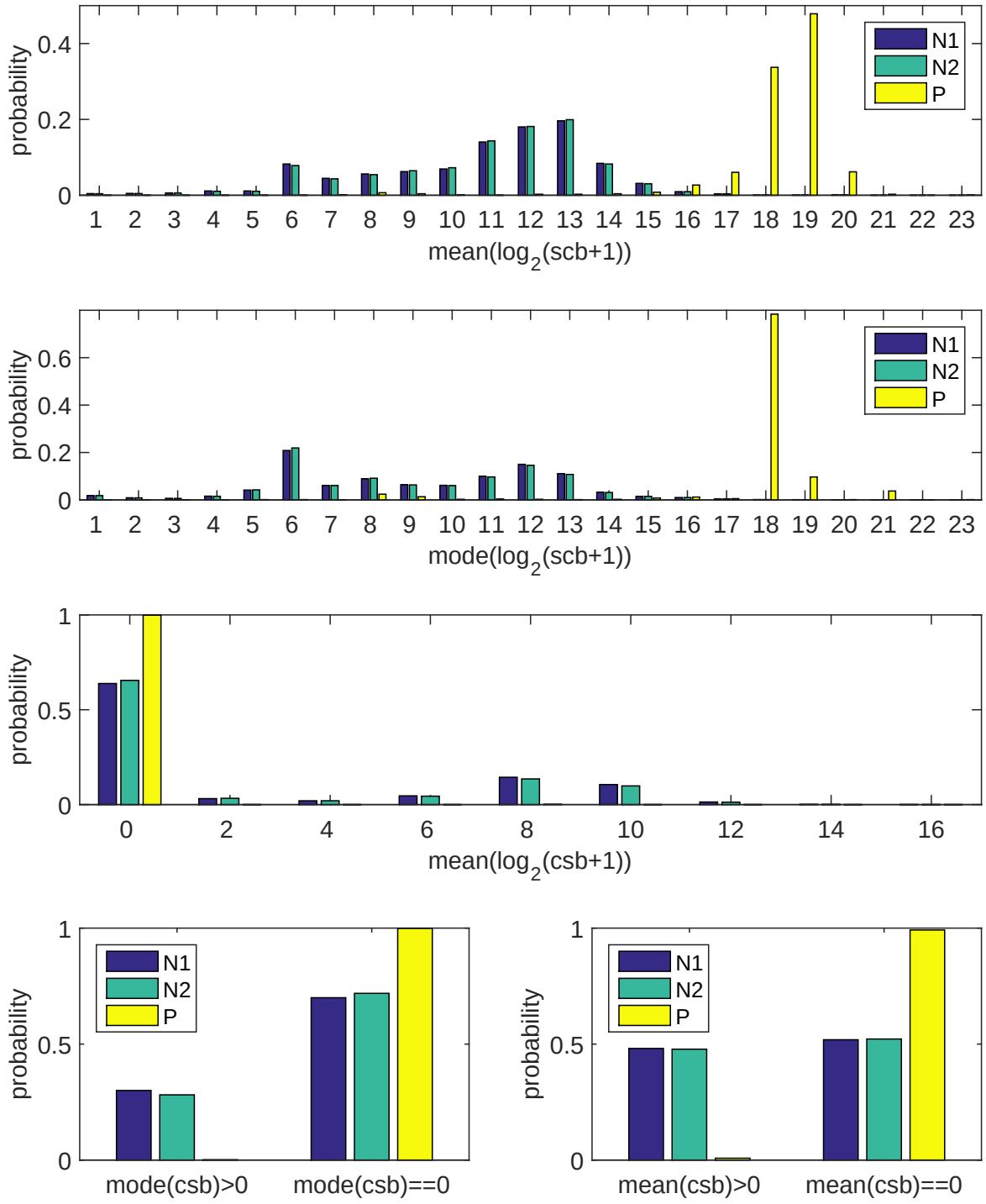
What is definitely new in this type of classification is a vector of repetitiveness values assigned during the feature extraction process and the number of flows per *s-IP*. The vector of repetitiveness values is not a list of all repetitiveness values per IP. Large values would repeat themselves, because they would be assigned to all flows in the series. Instead, the vector of repetitiveness values has the same number of elements as is the number of series of repetitive requests per IP. For example requests with *scb* values (15,20,15,20,10,15,10,20) would have the vector of the form (3,3,2). A vector of unique repetitiveness values can be created using this vector of repetitiveness. The sum of unique repetitiveness values divided by the total number of flows per *s-IP* states, how many times the unique values repeated themselves during the communication. Almost the same is given by the ratio of the number of unique repetitive values to the length of the vector of repetitiveness values. Repetitive traffic would have these values close to 1 and nonrepetitive traffic would have it close to 0 because requests being sent should have different *scb* values. The distributions of these two values within the training sets are given in Figure 3.4.

Figure 3.4: Distribution of the described ratios within the training sets used for IP classification. The bars represent the number of occurrences in the interval between the ticks so that the last bar is for the interval (0.95;1]



The final feature vector comprises of 49 features described above. The feature vector is again in two versions, one including *url* and *httpstatus* values and one without these two features.

Figure 3.5: Distribution of several selected features within the training sets used for IP classification.



Chapter 4

Experiments

Experiments and results will be described and discussed in this chapter. An experiment of naive classification was conditioning flows by certain values of features discussed in Section 3.2.

The single log behavioural classification was tested using 72 different classifiers varying by the feature vector, with which they were trained, by the training set and - what is the most important - by the use of *url* and *httpstatus*. Classifiers trained not having these two features available are ready to be used for classification of `https` traffic. Classifiers are be named systematically following a rule:

sl_clas_sX_lhY_vZ

where clas takes values in {LDA, QDA, SVM, SVM3} (SVM3 means using a polynomial kernel of the third order), X can by 0, 1 or 2 and it defines used training set (trSet0, trSet1, trSet2), Y is a logical value of the usage of *lurl* and *httpstatus* features and Z takes values 1,2 or 3, depending on the version of the feature vector (see Table 3.9).

The IP classification method was tested using 24 classifiers grouped into four sets varying in the usage of *httpstatus* and *url* and also in the used training set. Classifiers are again named systematically following a rule:

ip_clas_sX_lhY

where clas takes values in {LDA, QDA, SVM, SVM3, SVM5, SVM7} (the number following "SVM" again defines the order of the used polynomial kernel), X takes values either 1 or 2 defining used training set (N1, N2) and Y is again logical value of the usage of *lurl* and *httpstatus* features.

All classifiers were tested on the same evaluation dataset. The dataset comprises of around 2 million proxy logs and 27000 unique *s-IP* values. We want to compare these classifiers and therefore we need to quantify their performance. Three main performance measures of the classification will be counted.

The first one is **recall** (also called **TPR** - **true positive rate**) and is defined to be the number of *media* flows classified as *media* divided by all *media* flows in the evaluation dataset. It is percentage of *media* requests identified.

The second measure is **FPR** - **false positive rate** which is defined to be the the number of *mediaC* flows classified as *media* divided by all *mediaC* flows in the evaluation dataset.

The third measure is **precision** and it is defined to be the number of correctly classified *media* flows divided by the number of flows classified as *media* flows. 1 - precision quantifies the error of the first type, which is in our case classifying a request belonging to *mediaC* class as a member of *media* class.

The higher recall and precision are numbers are, the better the classifier is. They are, however, not sufficient for comparing the performance of two classifiers. Having two classifiers, one with recall 70% and precision 80%, and second classifier with recall 80% and precision 70%, we cannot be certain, which

Table 4.1: Specifications of the evaluation dataset.

Classification method	Naive	Single log	IP
Total number of elements	1974101	1974101	27178
Elements used for classification	1586574	1402484	11775
Elements in *media* class	46469	42938	378
Elements of Youtube media	20835	20457	X

of these two is better, because we do not know, how precise would be the first classifier, with operating point shifted to 80% recall.

To be able to compare the performance of two classifiers, we need to introduce the **receiver operating characteristic** or shortly **ROC curve**. It is a graph of TPR (recall) as a function of FPR. The performance of a classifier represented by a curve in a square graph of size 1 (ROC space). The higher the curve goes for small FPR values, the better. The curve is not important for values of FPR bigger than 0,1 because for such value the mistake of the first type is too big.

Obtaining the described measures involves one problem, that needs to be discussed. The number of *media* and *mediaC* flows in the dataset occurs in all definitions. The problem is that these numbers are unknown and it was actually the goal of this project to find them. Bottom estimate of the number of *media* flows in the dataset was made to obtain the values of the measures. The exact definition of the *media* traffic applies here. As was seen, the content type was never included in any training set but some requests of this type (namely those having *scb* greater than 10kB and *csb* value smaller than 100B) are considered to be true positives of the classification. Thus any such request classified as *media* will show that the classifier is able to detect previously unseen media streaming traffic.

It is clear, that not all *media* flows are found by the conditioning of *content-type-RS* values described in the definition of *media* in Section 3.1. Some flows do not include *content-type-RS* value (for example `https` traffic, which is also included in the evaluation dataset). After manual analysis of a representative example of flows in *media* can be, however, expected that more than 98% of flows labelled as *media* by this conditioning are really media streaming requests. All classifiers are on the same evaluation dataset and it will be shown, that supervised classifiers are better than the conditioning, because they find new media streaming traffic.

Because Youtube video and audio flows made almost a half of all *media* requests, special statistics measuring the performance of the classifier was added - percentage of Youtube video or audio flows found.

Overview of the used evaluation set is given in Table 4.1.

The performance of all classifiers will be measured considering some `application/x-fcs` flows as media streaming. This type was not included in any training set although it makes more than 30% of all *media* requests, because of its controversy and because it does not follow the same statistical rules as all other media streaming.

In this chapter will be used further notations for **TP** - **true positives**, which are defined to be correctly classified *media* records (requests or IP addresses) and **FP** - **false positives**, which are defined to be *mediaC* records classified as *media* records.

4.1 Naive classification experiment and results

TPR and FPR will be calculated twice - once using the the same definition of *media* as for all others classifiers and once using the definition of the *media* set with excluded `application/x-fcs` to see, how does the performance change. All results computed using the training definition of the media streaming will be labelled by number 2. Precision, number of false positives or true positives will not change and therefore will be calculated only once.

The evaluation set used for testing was only cleared of flows having both *scb* and *csb* equal to zero. The measurement itself consisted of finding the *media* and *mediaC* flows in the evaluation dataset and then finding the *media* and *mediaC* set of flows conditioned by given features.

Naive classification means conditioning by *csb*, *scb* and *lurl* values. The value of *csb* was put equal to zero, which leaves two parameters for the classification. To be able to compare the results I conducted a manual grid search. As seen in Figure 3.1, important decision values for $\log_2(lurl)$ is between 8 and 10 and for $\log_2(scb)$ it is somewhere around 15. The intervals chosen for the grid search are therefore [8;11) for $\log_2(lurl)$ and [14;18) for $\log_2(scb)$ with a step 0,1 for both features. I obtained a contour plots of recall (TPR) and FPR and used these to create an ROC curve. I mapped FPR values back to the two-dimensional feature space of $\log_2(lurl)$ and $\log_2(scb)$ and than found the values of TPR in the corresponding points. Because the mapping from the feature space to FPR is not injective, the mappings had to be put more specifically to obtain an injective mapping after the composition of the inversion from FPR to the feature space and than to TPR.

For every value of FPR there exists one or more points in the feature space with this FPR value. When the point is only one, the mapping FPR→TPR is injective and we have a one point of the ROC curve. When there are more points in the feature space with the same FPR value, the resulting value of the mapping FPR→TPR will be put to be the maximum of all TPR values, maximizing over all points with the particular FPR value.

After the construction of the mapping, we can plot obtained points into the ROC space and compare them with the results of all other classifiers. The resulting ROC curve is given in Figure 4.1. Recall, precision and other interesting results of the classification for 3 work points (marked also in the figure) of the classifier are given in Table 4.2.

The performance measured relative to the exact definition of *media* is worse, because the naive classifier could not identify correctly any request of the type `application/x-fcs`. This content type does not satisfy the condition of zero *csb* value and therefore including `application/x-fcs` into the media streaming only increases the number of all *media* requests found in the evaluation set.

We can see, that the naive classifier had very high precision in all operating points. The recall, however, decreases very quickly with decreasing false positive rate. It also found a

Figure 4.1: ROC curves of the naive classifier. The three marked points define the operating point, for which Table 4.2 gives explicit statistical results.

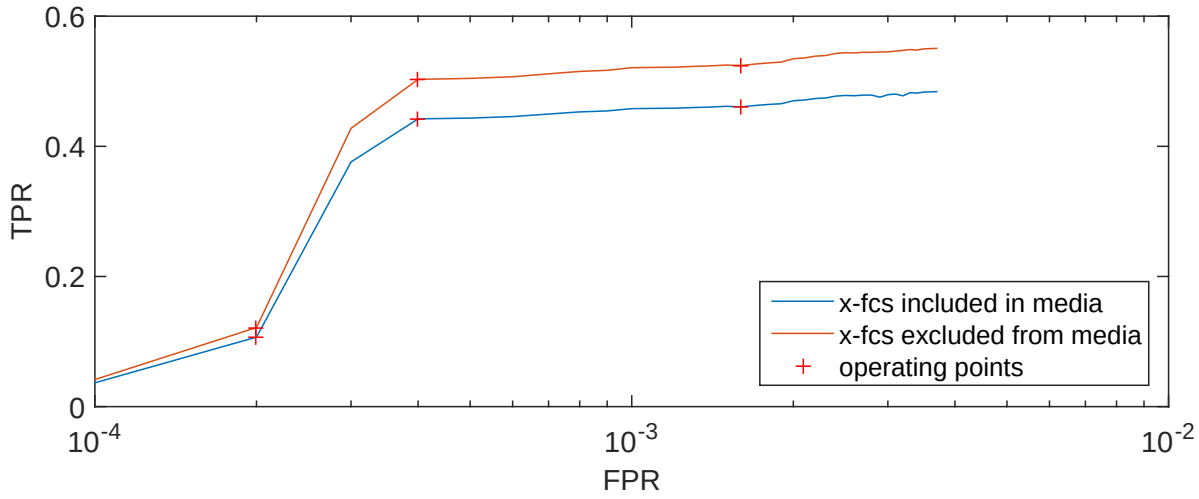


Table 4.2: Statistical results of naive classification. Operating points 1, 2 and 3 are marked on the ROC curve in Figure 4.1, ordered by the increasing value of FRP. All statistics denoted by number "2" were obtained with `application/x-fcs` excluded from *media*.

classifier	Operating point 1	Operating point 2	Operating point 3
Flows classified as *media*	5329	21229	23825
Flows classified as *mediaC*	1581245	1565345	1562749
% of Youtube media flows found	22.57%	96.18%	96.67%
True Positives	4965	20547	21410
False Positives	364	682	2415
Recall (TPR)	10.68%	44.22%	46.07%
FPR	0.02%	0.04%	0.16%
Precision	93.17%	96.79%	89.86%
Recall (TPR) 2	12.15%	50.29%	52.41%
FPR 2	0.02%	0.04%	0.16%

4.2 Single log behavioural classification and results

Feature values on the evaluation set ware obtained by the labelling script (see description in Section 3.3.3 on page 25). The parameters of the script were set to 5 for the memory length and 0.1 for the interval. The first parameter implies, there have to be at least 5 flows per *userID* and per *s-IP* to use the flows for feature extraction. The second parameter states, that for adding a new flow to a repetitive series of flows, the flow must not have the *scb* value further from the mean of the values in the series than 10% of the mean. An example: repetitiveness values assigned to a series of *scb* values (10,10,9,11,10,13) is (4,4,1,4,4,1), because the mean value is computed only for those values, that are already considered to be repetitive. The mean of the numbers 10, 10 and 9 is 9.666 and 11 does not belong to the interval 9.6666±0.96666 and therefore will not be added to this series[1].

The number of flows used for the testing was over 1,4 million. Recall, precision and ROC curves were obtained using the classification results. The ROC curves are given in Figure 4.2. Comparison of the best classifiers from each group (training set and usage of *lurl* and *httpstatus*) is given in each figure. FPR is shown on a logarithmic scale, because this region is the most important. Recall, precision, FPR and other statistics quantifying the performance of chosen classifiers are given in Table 4.3 and the operating points given in the table are also marked in the figure with ROC curves.

As was mentioned, some requests of `application/x-fcs` are considered to belong in *media*, although they were not present in the training sets. Any such request found is therefore another proof of better performance in comparison with the naive classifier or manually designed rules. An example of the detection of requests of this type were found the numbers of flows of the type `application/x-fcs` among the true positives of classifiers given in the table. Classifiers trained using the features *lurl* and *httpstatus* found 260 flows of this type on average with the minimum of 93 flows, classifiers which do not use *lurl* and *httpstatus* found 745 flows of this type on average with the minimum of 442 flows.

A discussion about false positives should be made. Among false positives were very frequently found images, text and javascript applications. For example the website www.flightradar24.com regularly sends an updated view on the flight traffic all around the world. Another javascript application probably regulates the streaming of some media - many requests with the domain www.siriusxm.com were found among false positives. Images get misclassified, when every file is sent more than once (it is not rare to send the file 3 or 4 times in a row, some images are sent more than 10 times) or when the files come from a gallery or it is a content of an image advertisement banner, that changes images with time. The images in the galleries are usually of similar size (and also their size is similar to the size of packages of media content) and as a client proceeds through the gallery, repetitive pattern occurs.

Some requests of *https* traffic were also found among false positives, which were evaluated as *mediaC*, because *content-type-RS* value is not available. These requests had an IP address for example 72.246.91.18, 23.66.160.251 or 95.100.203.164 and its domain was akamaitechnologies.com, which is known for serving audio and video streaming. The classifier is more precise in classifying these requests than the script made for the evaluation labelling. It is exactly the case of what was discussed at the end of paragraph preceding Section 4.1.

As a summary of the discussion about false positives it shall be said, that 20%-40% of false positives were images, around 10% makes `text` and 0%-30% false positives has empty *content-type-RS* value. The distribution varies from classifier to classifier but it is almost the same for the classifiers using *lurl* and *httpstatus* and for the classifiers not using them.

[1]The relation to the mean of *scb* values was included, because it is common, that the first values in the series of repetitive flows are smaller than the most frequent values or the mean value. The repetitiveness of such traffic would be smaller, although the *scb* values are similar.

Table 4.3: Statistical results of single log classification. The first part of the table shows results of classification using *lurl* and *httpstatus*, the second part shows results of classification which do not use these features. Total number of classified flows was 1402484, number of *media* flows included was 42938 and number of Youtube media requests was 20457. The operating points are marked on the ROC curves in Figures 4.2a and 4.2b.

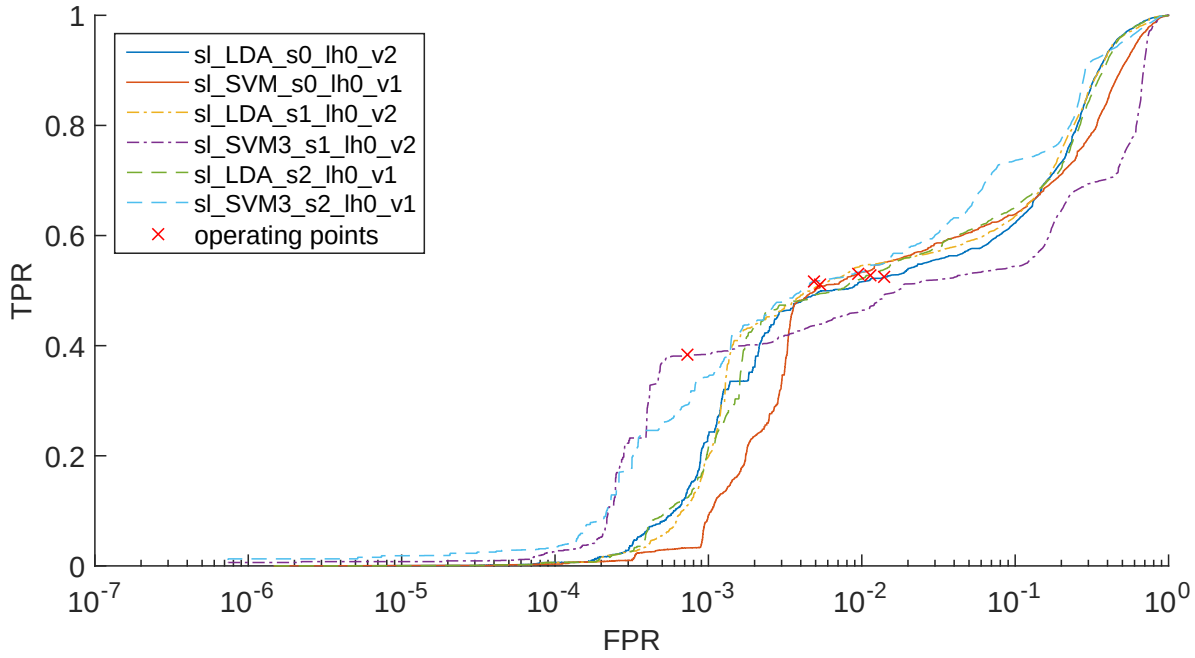
Classifier name	Classified as *media*	% of Youtube media found	TP	FP	TPR	FPR	Precision
sl_LDA_s0_lh1_v3	39911	93.46%	22272	17639	51.87%	1.30%	55.80%
sl_SVM_s0_lh1_v1	41913	97.40%	23523	18390	54.78%	1.35%	56.12%
sl_SVM_s0_lh1_v1	28569	96.55%	21341	7228	49.70%	0.53%	74.70%
sl_SVM_s1_lh1_v3	18076	76.27%	17197	879	40.03%	0.06%	95.14%
sl_LDA_s2_lh1_v3	40197	96.93%	22915	17282	53.37%	1.27%	57.01%
sl_SVM3_s2_lh1_v1	37059	97.18%	23794	13265	55.41%	0.98%	64.21%
sl_LDA_s0_lh0_v2	41413	92.77%	22516	18897	52.44%	1.39%	54.37%
sl_SVM_s0_lh0_v1	35835	93.52%	22825	13010	53.16%	0.96%	63.69%
sl_LDA_s1_lh0_v2	29089	91.47%	21899	7190	51.00%	0.53%	75.28%
sl_SVM3_s1_lh0_v2	17439	74.35%	16446	993	38.30%	0.07%	94.31%
sl_LDA_s2_lh0_v1	38117	93.12%	22705	15412	52.88%	1.13%	59.57%
sl_SVM3_s2_lh0_v1	28799	91.65%	22189	6610	51.68%	0.49%	77.05%

The comparison of classifiers according to their ROC curves is difficult, because the curves intersect each other. Among classifiers applicable on `https` traffic are probably the best sl_SVM3_s1_lh0_v2 and sl_SVM3_s2_lh0_1. The polynomial kernel of SVM classifier was the most flexible in dividing the features space in correct regions. Something about the used feature vectors could be assumed, because the vector03 did not appear among the best 6 classifiers in this group. Another interesting thing is, that LDA classifiers scored so well, even though they assume a univariate distribution. It was supposed, that QDA classifiers should perform better than LDA, because the assume only multivariate Gaussian distribution, but they usually scored worst.

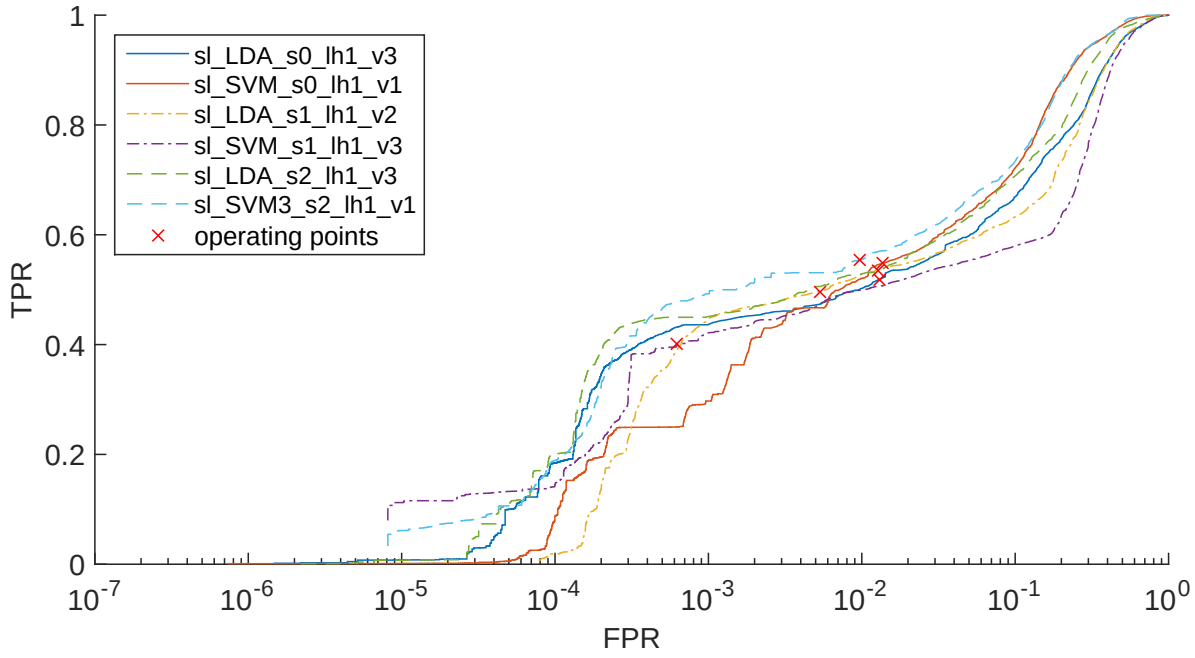
Among the best six classifiers using *lurl* and *httpstatus* were again the LDA classifiers and one of them appeared even among the best two. The two best classifiers according to their ROC curve are sl_SVM3_s2_lh1_v1 and sl_LDA_s2_lh1_v3. Nothing particular can be assumed about the use of the feature vector in the basis of the comparison of classifiers from this group. Comparing all ROC curves in Figure 4.2b with those in Figure 4.2a, we see, that using *lurl* and *httpstatus* improves the performance.

Figure 4.2: ROC curves of the best single log classifiers from each group for comparison of their performance. The marked points define the operating point, for which Table 4.3 provides further statistical results.

(a) Comparison of the best classifiers trained without the use of features *lurl* or *httpstatus*



(b) Comparison of the best classifiers trained with the use of features *lurl* and *httpstatus*



4.3 IP classification and results

The experiments with IP classification were conducted in the same manner as were those with single log classification. Feature values on the evaluation set ware obtained by the altered script and the flows were grouped into sets of requests with the same *s-IP*. To obtain performance measures, IP addresses had to be categorised into *mediaIP* or *mediaCIP* on the basis of the ratio of *media* flows to all flows, as was discussed in Section 3.4.1. Two baselines for the number of *mediaCIP* addresses were actually set. The first way of labelling an IP address as *mediaCIP* is, that absolutely no *media* request was made from this address. The second way, how to classify an IP address as *mediaCIP* is, that it is simply does not belong to *mediaIP* class. Because of these two ways of classification IP address as *mediaCIP*, we get two measurements of number of false positives and also two precisions. These are given in Table 4.4 denoted by the numbers in the same order, they were explained here.

The second option of labelling leaves some IP addresses unlabelled. These might be further analysed to see, which class they should be assigned to.

The feature extraction algorithm was run with different parameter, than for the single log classification. The interval stayed the same - 10% - but the length of the memory was set to 10. The reason for this is the following: most features are statistical and classifying the IP on the basis of statistics such as mean, mode or median from 5 values might be imprecise. The parameter set to 10 implies, that IP will be classified, if the number of flows assigned to it is greater than or equal to 10.

The number of classified IP addresses put was more than 11000. Recall, precision and ROC curves were obtained using the classification results. ROC curve of the best classifier from each group is compared with others in Figure 4.3. Recall, precision and other statistics quantifying the performance of chosen classifiers are given in Table 4.4.

IP classification has very interesting false positives. IP false positives are in most cases IP addresses with flows transferring almost strictly nothing to the server. Among misclassified IPs were some similar cases, as for single log classification - my.pinkfroot.com - a website for searching a watching the position of airplanes or ships online, a weather forecast website, which regularly updates information. These both were examples of mainly javastcript applications. Some websites transferred many images in a form of a banner (panorama-tours.com) or only one image dozens of times (www.click-n-print.com). Some interesting false positives include for example `application/octet-stream` with the domain googlevideo.com, definitely belonging to *media* traffic. Another example of better performance of classifier than of the evaluation labelling can be a type `application/binary` sent from a domain bbcfmhds.vo.llnwd.net, which is an audio and video streaming domain (the same domain under the same IP sent also data with content values `video/f4f` and *url* addresses of flows with the `binary` type contained string patterns indicating media streaming). The last example was not classified as *videoIP*, because flows with known video type made too small fraction to all flows, containing the mentioned binary files. Among false positives also appeared windowsupdate.com, with high repetitiveness and large amount of data transferred from server to client, or Facebook.com.

After the analysis of false positives it was no surprise they were classified as *mediaIP*, because they had the same features and followed the same patterns as *media*. The solution to this problem is a future work.

The comparison of best classifiers clearly confirms, that using *lurl* and *httpstatus* greatly improves the performance of the classification. It is moreover obvious, that better classification results have classifiers trained on the training set N2 - the training set including also IP addresses with *media* requests. The training set is probably better, because it also gives examples of traffic that is not of *media* type, but has more diversity in the flows.

Figure 4.3: ROC curves of the best IP classifiers from each group for comparison of their performance. The two pairs of curves vary by the usage of the features *lurl* and *httpstatus*, the difference between the curves within the pairs is the usage of training set N1 or N2.

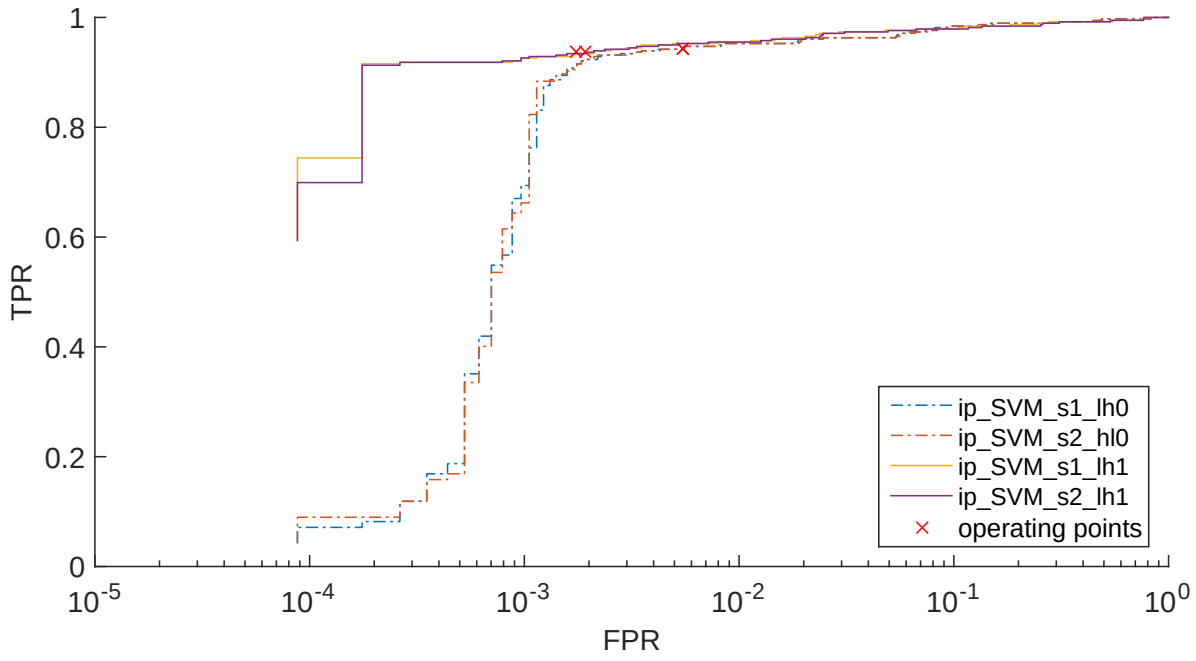


Table 4.4: Statistical results of IP classification. The operating points described in this table are marked on the ROC curve in the figure above. Performance measures are given in two variants for two baselines of false positives as described in the first paragraph of this section. The number of classified IP addresses in the evaluation dataset was 11775 containing 378 *mediaIP* addresses.

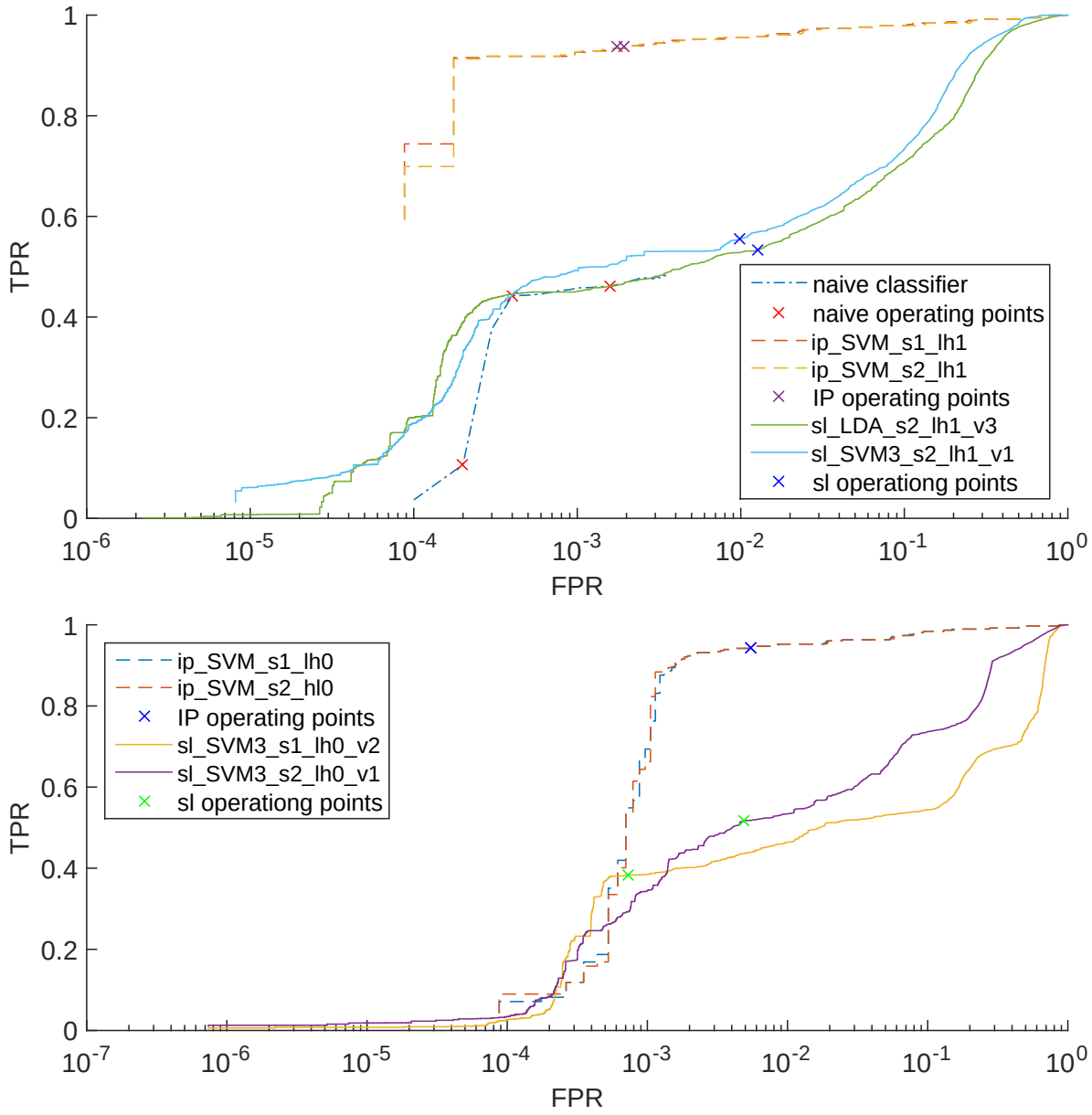
Classifier name	Classified as *mediaIP*	TP	FP 1	FP 2	TPR	FPR 1	FPR 2	Precision 1	Precision 2
ip_SVM_s1_lh0	419	357	44	62	94.44%	0.39%	0.54%	89.03%	85.20%
ip_SVM_s2_lh0	419	357	44	62	94.18%	0.39%	0.54%	89.03%	85.20%
ip_SVM_s1_lh1	376	354	16	22	93.65%	0.14%	0.19%	95.68%	94.15%
ip_SVM_s2_lh1	374	354	15	20	93.65%	0.13%	0.18%	95.93%	94.65%

4.4 Comparison of classification methods

The best classifiers of the same type were plotted into one graph to compare all methods. Naive classifier is included in only one figure, because it uses *lurl* and therefore inapplicable on encrypted traffic. Two best classifiers representing each method and the applicability on encrypted traffic are compared in Figure 4.4. The first figure compares classifiers trained using *lurl* and *httpstatus*, the second figure compares the rest.

It is easy to notice, that the best designed classifiers perform better, than the naive classifier. The best method proposed is the IP classification method, which performed worse than single log classification in only in a very small interval of FPR values.

Figure 4.4: ROC curves of the best classifiers representing all classification methods for comparison. Classifiers using *lurl* and *httpstatus* are compared in the upper figure.



Chapter 5

Discussion

In the previous chapter the performance of the three proposed classification methods was compared. Other differences between theses methods shall be discussed now.

The first difference between the naive method and the behavioural methods is, that only the naive classification can be used for the real-time traffic classification. Conditioning proxy log features can be done using only one proxy log at a time. The behavioural classification on the other hand, can be used either for classification of stored proxy logs or for "pseudo" real-time classification. Such "pseudo" real-time classification would have to use a buffer to collect enough data to be able to classify requests or IP addresses.

IP classification could be used to create a list of *mediaIP* addresses. Such list would be then applicable as a real-time classifier. It might only need regular updating.

What must not be forgotten is that all behavioural classifiers are implicitly dependent on the parameters of the labelling script. Their performance can be improved by training on and classifying data labelled with different parameters. The parameter of the interval was set to 10% after manual analysis of the media traffic and this value produced encouraging results. Smaller interval would cause some flows to be missed and not assigned to the repetitive series of flows and the mean repetitiveness of the traffic would decrease. That would make it more difficult to distinguish media traffic having the repetitiveness larger. Larger interval, on the other hand, would cause grouping of requests, that would not have anything in common, but a similar *scb* value.

The second parameter of the script defines the number of flows, that will be discarded and also the length of the memory. The longer the memory is, the greater the probability, that a flow will be assigned high repetitiveness. It was found experimentally, that increasing the length of the memory for example to 10 does not cause an increase of mean repetitiveness by more than 2 or 3, but the number of flows discarded increases by almost 30%. The longer the length of the memory, the fewer requests get classified or the bigger the buffer would have to be.

I would also like to discuss possible improvements of the proposed methods. The first improvement could be training on a bigger and more homogeneous dataset. Training sets used here had unbalanced size (negative sets were 3-5 times bigger than positive sets) and the data of *media* traffic was made mainly of Youtube media flows.

Another improvement could be achieved by clearing the feature vector of features, that do not help much in the differentiating of the two categories of traffic. There is also an opportunity in searching for other features and using them to divide the feature space to more than two regions.

More sophisticated classifiers could be more successfully trained on larger training sets. Some classifiers got overtrained on the amount of data I was working with (that is probably why using polynomial

kernel of the degree 5 or 7 for SVMs did not score better in IP classification, than the order those with the degree 3 or linear SVM).

There is also a huge space left for further analysis of content types such as `application/x-fcs` and `application/octet-stream`. There is also a question of the reliability of this feature.

Some more sophisticated tool for reviewing *url* addresses to help with the manual analysis of the data would also increase the precision of the analysis. There is also a possibility of creating the data artificially by recording a communication with particular site or using particular web applications. The range of different applications, however, makes it almost impossible to create enough data to be sure of its representativeness.

The performance of the proposed classification method can also be improved by combining them with some manual data pre-filtering or combining the classifiers together (classifying the data by more classifiers at once).

5.1 Main contribution and possible extensions

The main contribution of this work is the proposal of new method of traffic analysis. The extraction of repetitiveness of *scb* values is dependent only on the knowledge of server IP address and client ID (client ID is not used in IP classification). The most important property of described methods are the features, they are based on. Although the whole process of manual analysis, designing the naive rules, creating the training sets was based on features *content-type-RS* and *url*, the final classifiers do not use these. The classifiers designed to be used in encrypted `https` traffic use only *scb*, *csb*, *s-IP* and *userID* (*userID* is used only in the case of single log classification), those trained for classification of unencrypted traffic use also *lurl* and *httpstatus*. Although the methods use only so few features, they are successful in detecting a considerable amount of media streaming traffic. Moreover they accomplish the goal of finding previously unseen media and they are therefore even better than manual conditioning on *content-type-RS* values.

The applicability of classification on encrypted `https` traffic is very important. As was said in the introduction, the volume of data transferred over the internet using encryption rises and will continue to rise. Training classifiers on unencrypted traffic with the ability of extension to `https` traffic is needed. As an example of achievement could be taken the classification results of one trained classifier, which was able to detect up to 93% of Youtube media request (naive classifier found 96%, but the naive classification method is based on using features unavailable in `https` traffic). Youtube has recently changed from `http` to encrypted `https` traffic and any known method of identifying one of the top 5 most favourite video streaming web sites is very important for the network traffic classification.

This work can be further extended by further analysis of all false positives and finding another key feature for distinguishing them from true positives. Deeper understanding of content types such as `application/octet-stream`, `application/x-fcs` or `application/javascript` and their relationships to misclassified requests of Windows update, weather forecasts or images would also help in the improvement of both recall and precision. Further analysis could also aim on increasing recall by going through true negative requests (*media* flows classified as *mediaC*) and finding some new feature helping with their description and identification.

The repetitiveness feature could be also applied to malware detection. Some malware is typical by periodic communication with an external server and this pattern could be recognised.

Conclusion

This work is dedicated to classifying media streaming (particularly audio and video) using logs captured by a proxy server. Large portion of the media streaming can be easily described by its typical streaming strategy: sending equally sized packages of data one after another. Such traffic creates a typical pattern of repeating server-to-client byte values (the amount of data sent form the server to the client), which and used for classification. Media streaming traffic is further typical by the size of the transferred data, with its mean more than 10 times bigger than the mean of rest of the traffic. Last crucial feature of media streaming is that very small amount of data (usually no data at all) are sent from the client back to the server. Other used features are the status of the connection (media streaming is always "successfully connected"), or the length of the URL address. These are, however, available only in unencrypted `http` traffic.

One option of media streaming detection is manual setting of all parameters and searching for the best combination to achieve the classification results. This option is not viable since multiple features are used and the datasets have large sizes. The second option, which is more robust and flexible, is to create representative sets of media traffic and its complement and train a classification algorithm to find the best parameters separating the media traffic from the rest of the traffic. The classification algorithm is capable of discovering relationships between the features, which would be impossible to find by manual analysis, and therefore achieve better classification results. The algorithm can be also much more easily updated by training on new sets of data.

This work introduces new feature describing media streaming, which is based on the streaming strategy. The feature was used in two methods of classification - classification of each proxy log and classification of the server IP address. The first method aims on classifying all proxy logs as either of media streaming type or other traffic, the second method classifies a server IP address as transferring media traffic, if at least 70% of the communication with this IP address is composed of media streaming. Both methods were designed and tested in many variations, of which the most important is the difference in applicability on encrypted `https` traffic.

These two methods were compared with each other and also with a naive classifier based on manually designed rules. Two performance measures will be defined here to present the results. The first measure is TPR, which is defined to be the number of correctly classified media streaming records divided by the total number of media streaming records in the dataset and the second measure is FPR, which is defined to be the number of other records then media streaming misclassified as media streaming divided by the total number of all other records than media streaming in the dataset (the set of other records is a complement to a set of media streaming records). For the fixed value of FPR 0.1%, the naive classifier scored 45.79% TPR compared to the best classifiers of the proxy log and IP address classification method, which scored 49.24% and 92.61%, respectively. Results of the best classifiers using only features available in `https` traffic are 34.40% for the proxy log classification and 66.23% for the IP address classification. The naive classifier can not be used `https` traffic.

This work has many possible extensions. Further analysis of false positives (requests or IPs incor-

rectly classified as media streaming or transferring media streaming) would help increase the precision of the classification. Additional analysis of other content types and proxy log requests would make it possible to create better training sets. There is also a space for improvement of used feature vector and for searching for other features. The proposed feature, repetitiveness, could also be used for identifying malware, because some types of malware communicate periodically with a command and control server.

Bibliography

[1] Adhikari, Vijay Kumar and Jain, Sourabh and Chen, Yingying and Zhang, Zhi-Li. How do you 'Tube'. *SIGMETRICS Perform. Eval. Rev.*.

[2] Augustin, B. and Mellouk, A. On Traffic Patterns of HTTP Applications. In: *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*.

[3] Bujlow, T. and Riaz, T. and Pedersen, J.M. Classification of HTTP traffic based on C5.0 Machine Learning Algorithm. *Computers and Communications (ISCC), 2012 IEEE Symposium on*.

[4] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer-Verlag New York, 2006. ISBN 978-0-387-31073-2

[5] *Cisco technology radar*. 2014, https://techradar.cisco.com/pdf/cisco-technology-radar.pdf

[6] *Cisco Visual Networking Index: Forecast and Methodology, 2014-2019*. 2015, http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf,

[7] *http://www.techsupportalert.com/5-Best-Free-Video-Streaming-Sites.htm*. accessed July 7, 2015

[8] *https://en.wikipedia.org/wiki/Internet_media_type*. accessed July 7, 2015

[9] *https://en.wikipedia.org/wiki/Real_Time_Messaging_Protocol*. accessed July 7, 2015

[10] *https://www.senderbase.org/*. accessed July 7, 2015

[11] Kaoprakhon, S. and Visoottiviseth, V. Classification of audio and video traffic over HTTP protocol. In: *Communications and Information Technology, 2009. ISCIT 2009. 9th International Symposium on*.

[12] Sait, S.Y. and Kumar, M.S. and Murthy, H.A. User traffic classification for proxy-server based internet access control. In: *Signal Processing and Communication Systems (ICSPCS), 2012 6th International Conference on*.

[13] Rao, Ashwin and Legout, Arnaud and Lim, Yeon-sup and Towsley, Don and Barakat, Chadi and Dabbous, Walid. Network Characteristics of Video Streaming Traffic. *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*.

[14] Zhu Li and Ruixi Yuan and Xiaohong Guan. Accurate Classification of the Internet Traffic Based on the SVM Method. *Communications, 2007. ICC '07. IEEE International Conference on*.