Detecting user actions from HTTP traces: Toward an automatic approach

Detecting User Actions from HTTP Traces: Toward an Automatic Approach

Luca Vassio, Idilio Drago and Marco Mellia
[1] Politecnico di Torino, Italy
name.surname@polito.it

***Abstract*—Detecting explicit user actions, i.e., requests for web pages such as hyper-link clicks, from passive traces is fundamental for many applications, such as network forensics or content popularity estimation. Every URL explicitly visited by a user usually triggers further automatic URL requests to obtain all objects that compose the web page. HTTP traces provide a summary of all URLs requested by users, but no information that could be used to separate explicit from automatic requests. Previous works have targeted this problem and ad-hoc heuristics have been proposed. Validation has been typically done using synthetic traces. This paper investigates whether an approach based solely on machine learning can successfully detect user actions from HTTP traces. A machine learning approach would come with many advantages – e.g., it minimizes manual tuning of parameters and can easily adapt to page structure changes. We build both real and synthetic traces to assess the performance and gain insights on the features that bring most advantages in classification. Our results show that machine learning reaches similar or better performance as previous heuristics. Furthermore, we show that models built with machine learning algorithms are robust, presenting consistent performance in different scenarios.**

***Index Terms*—Machine Learning, Feature Selection, Passive Monitoring, Clickstream.**

I. Introduction

A *user action* is a URL explicitly requested by a user to fetch a web page. The request can be done either by clicking on a hyperlink, by typing the URL in the browser address bar, or through bookmarks. Knowing the URLs visited by users is fundamental for many applications. For instance, network security and forensics applications [1] profit from logs of HTTP requests to detect when and how users get infected by malware or virus. Equally, URLs requested by users can unveil web content popularity [2] and be used for ranking and promoting content [3]. One possible way of obtaining user actions is by passively observing network traffic [4], [5]. One can extract HTTP requests and save requested URLs. Similar information can be obtained from proxy or firewall logs too. Many off-the-shelf flow meters are able to export HTTP traces along with traditional flow-level information [6], thus making HTTP traces sometimes readily available for processing.

However, identifying the URLs explicitly requested by users in HTTP traces is very hard. Modern web pages are rather complex [7] and include many HTML files, JavaScript, multimedia objects and dynamically generated content. These are all *automatic* objects fetched by browsers. This complexity results in hundreds of automatic objects being fetched by browsers, but very few user actions. Furthermore, non-interactive web applications (e.g., cloud storage clients, OS update agents, etc.) rely on HTTP to exchange data too, and all those requests are logged together with users' activity.

Some methods for identifying explicit user actions in HTTP traces have already been introduced in previous works. StreamStructure [7] exploits the `referer` field[1] in HTTP requests and the widespread deployment of *Google Analytics beacon* to reconstruct web page structures from HTTP traces and to determine the URL originally requested by users. Authors claim both precision and recall above 80% among pages using Google Analytics. Authors of [4] follow a similar approach, exploiting the `referer` field to group requests into HTTP streams. A series of manual rules – e.g., based on size of HTTP response, Content Type and the number *children* of requests – are used to decide whether a request is explicitly made by the user or not. Precision and recall above 90% are claimed. Finally, authors of [3] present another heuristic to identify user actions that operates only with the HTTP requests. The proposed heuristic is shown to scale well in high-speed networks, and it achieves 66%–80% precision and 91%–97% recall, depending on parameter choices.

Previous works however present some drawbacks. First, they are all based on manually tuned heuristics. Whereas the heuristics are shown to produce good results, they require time-consuming work to be configured, and the procedure might even need to be performed periodically to adapt parameters as web pages evolve. Second, given the difficult to obtain HTTP traces simultaneously to ground truth of actual users' requests, previous works have mostly validated proposals using limited synthetic datasets. In fact, ground truth datasets are generally built by automatically visiting arbitrary links, which may miss (create) artefacts (not) seen in real traces. More important, synthetic datasets do not contain the variate of browsers and behaviors of real traces. This is worrisome, given that some browsers even skip filling the `referer` fields in some cases [5], thus potentially affecting heuristics' precision when deployed in practice.

In this paper we study whether machine learning algorithms can successfully be used to detect user actions on HTTP traces. An approach based on machine learning would have many advantages. It would allow automatic tuning of parameters, could automatically learn which features are the best candidates for

[1]The `referer` is an HTTP header field that identifies the address of the web page that linked to the resource being requested.

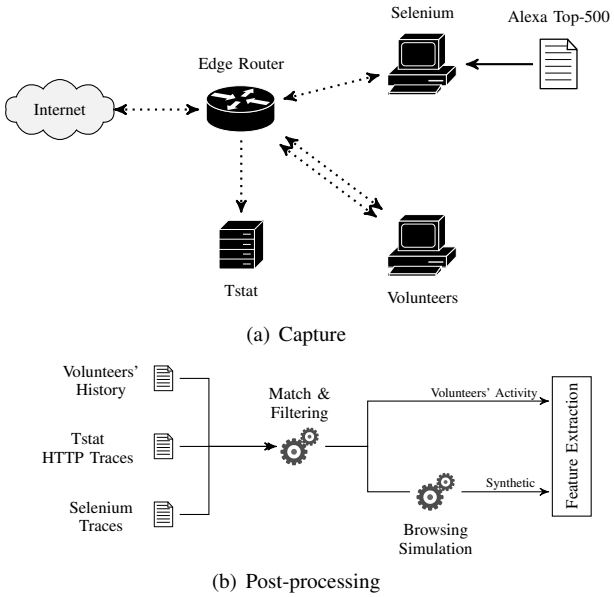


Fig. 1. Workflow of the data capture and preparations.

solving the problem, and could easily adapt to the evolution of web pages, and to different scenarios.

For testing, we build up two types of datasets to train and validate the algorithms. First, we collect browsing histories of 10 volunteers for several months, while also recording all HTTP requests of their web navigation. Second, we create synthetic datasets by revisiting a large number of arbitrary links and websites. We post-process the synthetic traces to vary parameters of users' navigation, such as the time between consecutive requests, to simulate different navigation scenarios. We then extract a large number of features from both real and synthetic datasets, which are used in experiments to select the most informative ones for the problem. We discuss results of using different datasets for training and testing and, finally, explore the performance of classification algorithms when trained with datasets of increasing sizes, by varying the number of users for training and validation.

Our results show that machine learning can reach similar performance of the previous heuristics without requiring manual set up of parameters. Moreover, we show that models built with machine learning algorithms are robust, presenting consistent performance in different scenarios.

Aiming to foster further researches and re-validations of our results, we make our datasets available (in Weka's format).[2]

II. Datasets and Ground Truth

Fig. 1 summarizes our data collection and preparation methodology. We simultaneously collect datasets by instrumenting web browsers and by passively observing traffic at edge routers. Two types of datasets are prepared as follows.

A. Volunteers' Activity

We collect browsing histories of 10 volunteers in the campus. We extract the history from three major browsers directly from volunteers' machines (i.e., Safari, Chrome and Firefox). Browsing histories include (i) timestamps of actual page visits; (ii) the URL requested on every visit; and (iii) codes describing page transitions – e.g., whether the visit resulted in a redirection to another page. In total, we observed more than 12 000 visits to more than 2 000 domains in up to 3 months of browsing activity.

In parallel, we collect HTTP traces by observing volunteers' traffic from edge routers of our university network. We use the HTTP monitoring plug-in of Tstat [8]. The plugin exports information present in HTTP headers for each HTTP request and response. This includes (i) timestamps of requests and responses; (ii) URLs requested by client browsers; (iii) User Agent and `referer` fields in the requests; and (iv) Content Type, Content Length and Status Code in the responses.

Next, we post-process HTTP traces and browsing histories. We first match requests and responses (when available) in HTTP traces. Then, we label every entry in the resulting dataset as *user-action*, if it matches an entry in browsing histories, or *automatic*, otherwise. This step however requires ingenuity. For example, we observe situations in which users' visits are redirected, and the redirection is already cached at users' browsers. Thus, we label as *user-action* any non-cached request in a redirection chain if it appears on both HTTP traces and browsing histories.

Finally, we filter out any HTTP requests coming from web browsers we did not capture data (i.e., Internet Explorer and mobile browsers). We however leave requests of background applications that we mark as *automatic*, such as Dropbox or Windows Update, since our goal is to train models able to discern such requests. Tstat recorded more than 800 000 HTTP requests related to volunteers during the captures and, after filtering, around 600 000 HTTP requests remain for our analysis. The resulting HTTP Trace annotated with the classes of our problem is the input for the feature extraction algorithms described in the next section.

B. Synthetic Datasets

We create synthetic datasets by instrumenting a Firefox browser with Selenium.[3] Selenium revisits a list of pre-provided URLs, and automatically follow links found in the visited pages. We provide Selenium a list with the top-500 Alexa HTTP domains, and instruct it to follow up to 10 random links in each domain, thus generating around 5 000 simulated user actions. For these experiments, we have disabled caching on the instrumented browser. Selenium saves the list of visited URLs, together with timestamps of the visits. As for the captures with volunteers, all activity in the testing machine is observed by Tstat and all URLs are logged.

Next, we match HTTP traces with Selenium actions following the same approach used for post-processing real traces. Then, we simulate web browsing behaviors by replacing the timestamps of user actions in synthetic datasets according to a simple model of user behavior. More precisely, the user behavior is modeled as a first-order time continuous Markov process with exponentially distributed dwell times.

[2]https://lucavassio.wordpress.com/useractions

[3]http://www.seleniumhq.org/

For experiments, we select dwell times from an exponential distribution with mean of 32 s, corresponding to the average dwell time found in the volunteers' activity dataset. The inter-arrival time of automatic HTTP requests is kept as originally recorded by Selenium. In total, Selenium made around 300 000 HTTP requests during our captures.

III. Methodology

Classification is the problem of identifying to which class a new observation belongs. In our case, an observation is a URL, which belongs to one of two classes: *user* or *automatic* action.[4] In the field of machine learning, several supervised algorithms, i.e., classifiers, have been proposed. They are based on two steps: training and classification. During *training*, the system is given a labeled dataset where the class of observations is known. Observations are characterized by *features*, i.e., explanatory variables that describe observations. The classifier uses the knowledge of the class to build a *model* that, from features, allows it to better separate objects into classes. During *classification*, the classifier uses features as input and the model built during training to return a class as output. Depending on the adopted model, classification may be much faster than training.

A. Machine Learning Classifiers

Different classifiers have been proposed, based on the model they adopt. Here, we are not interested in providing a complete assessment of which classifier performs the best, but rather to coarsely observe if there are significant differences for our specific case. We arbitrarily consider three types of models that we briefly summarize below. We use the implementations offered by Weka for experiments.[5]

Decision Trees and Random Forests (RF): A decision tree is a tree-like classifier for making sequential decisions on features [9]. Internal nodes represent tests on features, branches are the outcomes of tests, and leafs represent classes. We use `J48` – an open source implementation of the `C4.5` decision tree. Random Forests [10] are an improvement of trees. They construct a multitude of decision trees at training time using subsets of features, outputting the class that is the mode among those trees. RF are more robust to over-fitting.

Bayesian Network (BN): BN [11] is a probabilistic graphical model that represents a set of features and their conditional dependencies via a Directed Acyclic Graph (DAG). Such networks are factored representations of probability distributions that generalize the naive Bayesian Classifier, one of the simplest models.

Multi-Layer Perceptron (MLP) Neural Network: MLP is a feedforward neural network that maps input features into classes [12]. It consists of multiple layers of nodes in a directed graph, where each node is a processing element with a nonlinear activation function. We use the backpropagation algorithm to train the network.

[4]In this paper we use data from volunteers which are uniquely identified by client IP addresses and User Agents. The deployment of any approach to identify user actions requires the isolation of traffic per user. IP addresses and User Agents may be insufficient if many users are aggregated in NATs. Discussing alternatives to isolate traffic is out of our scope.

[5]http://www.cs.waikato.ac.nz/ml/weka/

B. Performance Evaluation

For a given class c, we may have a True Positive – $TP(c)$ – when the returned class c is correct; a False Positive – $FP(c)$ – when incorrectly returned as c; a False Negative – $FN(c)$ – when non identified c; and True Negative – $TN(c)$ – when returned correctly as not c.

Given we are particularly interested in *user-action* classification, we evaluate four performance metrics: (i) $Accuracy$: the fraction of requests that are correctly classified regardless of their classes; (ii) $Precision(c) = TP(c)/(TP(c) + FP(c))$: the fraction of requests correctly classified considering class c =*user-action*; (iii) $Recall(c) = TP(c)/(TP(c) + FN(c))$: the fraction of *user-actions* that the classifier is able to classify; and (iv) $F - Measure(c)$: the harmonic mean of Precision and Recall for the class c =*user-action*.

After the model has been built on a dataset, we want to estimate its performance for selecting the best approach. We use stratified 10-fold cross-validation. The training dataset is partitioned into 10 sub-samples of equal size. Of the 10 sub-samples, one is retained for measuring the performance, and the remaining 9 sub-samples are used for training the model. This process is repeated 10 times, with each of the 10 sub-samples used exactly once as the validation data. Finally, the 10 results are averaged to produce a single estimation.

Once the best method has been chosen with this validation procedure, we assess the final performance through different test sets, with data never seen during training. In particular, we explicitly analyze testing data from two independent ground truths (i.e., synthetic and volunteers' data). This is key for networking applications, given the very heterogeneous scenarios the classifier may be called to face [13]. After assessing the final model on the test set, it is important to not tune the model any further, otherwise it could bias the expected performance.

C. Feature Extraction and Selection

Features play a key role in classification. In previous works, ad-hoc and custom features have been proposed. Following the best-practice of machine learning instead, we define and extract a large number of possibly generic features. Then, we use feature selection algorithms to determine the most informative ones.

1) Extracted Features: Table I summarizes the features we extract from traces. We consider 18 features that can be roughly grouped into four non-independent categories: (i) based on referring relations among requests; (ii) based on timestamps; (iii) describing properties of HTTP responses; and (iv) describing properties of URLs. Features are sorted by their *Information Gain* with respect to the *user-action* class in our dataset of volunteers' activity (see next section). Previous works that use some of these features as part of the manually tuned heuristics are reported in the Table.

Features are extracted by tracking URL and `referer` fields, and timestamps of requests. Given a URL, we first extract the time interval (Δ_t) from the previous request from the same user. We then check if the request has a `referer`. We call the URL in the `referer` the *parent* of the request.

TABLE I
FEATURES AND THEIR INFORMATION GAIN WITH THE *user-action* CLASS.

Feature	`referer`	Time	Object	URLs	IG
No. Children [3][4][7]	x				0.2706
Content Type [3][4][7]			x		0.0287
Δ_t – Previous Request		x			0.0140
HTTP Status Code [4]			x		0.0061
URL length				x	0.0060
Δ_t – Sibling	x	x			0.0048
Ads in URL				x	0.0040
Δ_t – Parent [4][7]	x	x			0.0036
Content Length [4]			x		0.0027
Parent Status Code	x		x		0.0016
Has `referer`?	x				0.0014
Parameters in URL				x	0.0014
Max Δ_t – Child	x	x			0.0010
Parent Content Type	x		x		0.0007
Ads in `referer`	x			x	0.0005
Max Length – Child	x				0.0005
Min Δ_t – Child	x	x			0.0003
Parent Content Length	x		x		0.0002

We then determine whether the request has *children* – i.e., the set of other requests that have the `referer` pointing to it. Based on such relations of parents and children, we extract: the number of children, the time interval between the request and its eventual parent, and the time interval between the request and its last *sibling* – i.e., other request sharing the same parent. If the request has children, we compute the minimum and maximum time to see a child, and the maximum Content Length among all its children.

We consider also features that are straightforwardly available in our data, representing statistics of HTTP responses, such as the Status Code, Content Type and Content Length. We augment the feature set of a request with statistics of its parent too (if it exists), such as by retrieving the Content Length, Content Type and Status Code of the parent request.

Considering directly the URL string, we include features that describe it. We manually create a blacklist of terms associated with advertisement domains (e.g., ads, adserver, etc.) and include a feature that shows whether the URL contains any of the terms in the blacklist. Similarly, another feature marks whether those terms are present in the `referer` of the request. Finally, we add the length of the URL and the number of parameters passed on the URL in the feature set.

2) Feature Selection: Feature selection is the process of selecting a subset of relevant features for use in model construction. The central idea when doing feature selection is that the data may contain features that are irrelevant, and thus can be removed to reduce the complexity of classification models without incurring in loss of information. We compute the Information Gain (IG) of each feature, that is the reduction in entropy caused by partitioning the dataset according to the values of a specific feature. Then we rank features: the higher is the information gain, the higher the information about the class that the specific feature carries. In Table I features are sorted by their IGs with the *user-action* class, considering volunteers' data. Table I gives an initial overview of discriminating power of each feature in isolation. Later we will also discuss the performance of classifiers trained with sub-sets of features.

TABLE II
PERFORMANCE OF DIFFERENT CLASSIFIERS TRAINING WITH VOLUNTEERS' DATASET. CROSS-VALIDATION RESULTS SHOWN.

(a) Accuracy

Training Set	Tree	RF	BN	MLP
All Features	0.9956	0.9962	0.9887	0.9923
Top-5 Features	0.9944	0.9943	0.9916	0.9905
No `referer`	0.9936	0.9931	0.9866	0.9831

(b) F-Measure

Training Set	Tree	RF	BN	MLP
All Features	0.899	0.912	0.774	0.822
Top-5 Features	0.872	0.866	0.819	0.752
No `referer`	0.846	0.831	0.700	0.529

We see that Number of Children is by far the feature with the highest gain. Next, Content Type is well ranked as well. These results confirm the intuition of the previous works [3], [4], [7] that `referer` relations and the Content Type of responses provide strong indications about user actions. Next, the time interval (Δ_t) between consecutive requests of a single user appears. We see some other features that are independent of `referer` among the top features, such as HTTP Status Code and Size of URL. Note that these features may help in solving artifacts related to lack of `referer` in HTTP requests [5]. To this ends, classifiers that do not use `referer` are worth to investigate. We will train classifiers that use only non-`referer`-based features in the next section.

IV. SUPERVISED CLASSIFICATION RESULTS

A. Comparisons of Models and Feature Sets

We first evaluate how the different classifiers perform. We consider 85% the dataset of volunteers' activity for the training and the 10-fold cross validation, and leaving the remaining 15% for the final testing. Table II(a) reports the classification accuracy, whereas Table II(b) reports the F-Measure for the *user-action* class. Best results are marked in gray cells.

Focusing on classifiers using all features – first row in Table II(a) – notice how accuracy is higher than 98.8%. Random Forest and decision trees reach the best classification results. Recalling that we have 12 000 user actions and 600 000 automatic actions, this is expected, since the two classes in our problem are strongly unbalanced. A naive classifier that always returns "automatic" would have similar accuracy.

As such, it is important to focus on the class *user action*. Table II(b) reports the F-Measure for this class. RF is the best classifier again, with F-Measure equal to $91,2\%$, thanks to its precision of 94% and recall of 89% when using all features.

Focusing now on the second rows of results in the tables, notice how numbers for the classification using only the top-5 features (see Table I) are very similar to the ones with all features. For instance, the F-Measure for the Random Forest is reduced from 91.2% to 86.6%. These numbers confirm that automatic feature selection is important. Then the mix

TABLE III
RANDOM FOREST MODEL TRAINED WITH VOLUNTEERS' DATA, ALL FEATURES. TESTING ON VOLUNTEERS' AND SYNTHETIC DATASETS.

Testing Data	Accuracy	F-measure	Precision	Recall
Volunteers	0.9963	0.916	0.891	0.943
Synthetic	0.9891	0.792	0.685	0.938

of features representing inter-timing of requests, `referer`, as well as properties of responses and URLs provides very good classification, despite the complexity of the problem.

Last rows in the tables show how the performance varies when all features in Table I that have relation to `referer` are excluded. We see that performance decreases when compared to previous experiments. However, even without attributes related to `referer`, the J48 decision tree reaches a precision of 89% and recall of 80% – i.e., F-Measure of 84.6%. BN and MLP classifiers show lower-quality results.

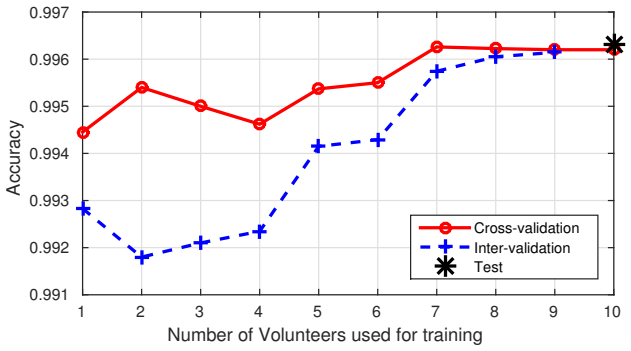
B. Testing on different datasets

Given these considerations, the Random Forest model trained with all the features of the Volunteer's dataset is selected as the best classifier. We now test its performance on data never seen during training. The first row of Table III reports results considering the volunteers' test set (15% of the dataset, not previously used), that are very similar to the cross-validation ones of Table II. Second row of Table III reports results against synthetic traces, that was never used during the training. In this latter case, results are slightly worse, with a significant decrease of precision (68.5% instead of 89.1%). This can be explained by factors such as: (i) while volunteers have visited thousand of pages during data collections, they are members of a homogeneous community (academy). Thus, the training set misses types of pages seen in more generic top-500 Alexa domains; (ii) synthetic datasets does not realistically represent navigation paths (thus, `referer` relations) and inter-request times. This somehow questions the results obtained in previous works when only synthetic traces are used for validation. Despite this, numbers are in-line with previous works, suggesting that the machine learning approach reaches performance similar to or better than ad-hoc heuristics.

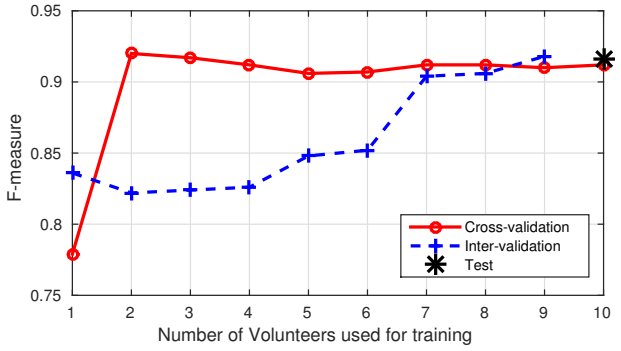
C. Training with Different Numbers of Volunteers

Next, we perform different experiments by varying the number of users in the training set to assess its impact on results. We consider Random Forest and all features. On each experiment round, we increase the diversity and size of the training set by considering traces from an increased number of volunteers. Then, we compute performance metrics (i) using 10-fold cross validation on the volunteers used for training; (ii) validating the trained model using the other volunteers' traces not included in the training. We refer to this second case as *inter-validation* case.

Results are in Fig. 2, where both accuracy (Fig. 2(a)) and F-Measure (Fig. 2(b)) are plotted for cross-validation and inter-validation datasets. Notice that not all volunteers have the same number of user-actions, and the number of volunteers in the



(a) Accuracy



(b) F-Measure

Fig. 2. Effects of varying the number of volunteers for training.

inter-validation set decreases as we move volunteers to the learning set. The training with 10 volunteers corresponds to the trained Random Forest model of the previous section, and its testing results of Table III are reported in Fig. 2 (*Test* marker).

Considering the cross-validation estimates, both accuracy and F-Measure improve until they reach a plateau. When two or more volunteers form the learning set, results do not change significantly in the cross-validation. More interesting and more important, the inter-validation with independent users shows consistent results, with numbers approaching those of the cross-validation when seven or more volunteers are in the learning set. In a nutshell, the behavior of these independent volunteers have already been learned from others in the learning set. Since our data is focused on a close community, we cannot verify whether performance of models remains the same in larger populations. Such evaluation requires further datasets, and we plan to tackled that in future works.

D. Training on Synthetic Dataset

In previous sections we used for training the traces collected from actual users, since these data is more realistic and capture web browsing behavior of actual users. Now we investigate what happens when considering synthetic datasets for training and testing, as normally done in the literature. Furtehrmore, we also check performance when the classifier is trained with synthetic data, and then tested with the Volunteers' dataset. This answers the question "what happens in practice" when actual users' traces have to be classified.

TABLE IV
RANDOM FOREST MODEL TRAINED WITH SYNTHETIC DATA, ALL FEATURES. TESTING ON VOLUNTEERS' AND SYNTHETIC DATASET.

Testing Data	Accuracy	F-measure	Precision	Recall
Synthetic	0.9989	0.975	0.981	0.970
Volunteers	0.9764	0.529	0.482	0.582

As for the dataset with volunteers' activity, classes in the synthetic data are unbalanced, with 97% of requests marked as *automatic*. Thus, a naive classifier could reach accuracy of 0.97 by labeling all requests as *automatic*.

As before, we consider only the Random Forest classifier using all features. Again, performance metrics are extracted using 85% of synthetic data for training. Then, we validate results on either 15% of remaining synthetic data, or on the whole volunteers' dataset.

Table IV reports test results. We can see that the machine learning approach seems to present an impressive performance (much better than any heuristics presented in related works) when looking at synthetic test set in the first row. Precision, recall and F-Measure for the *user-action* class are above 97% in this experiment. Compare these results to Table III, where similar results are presented for volunteers' learning set. This means that the browsing behavior in synthetic datasets is simpler to learn with respect to volunteers' dataset.

However, a completely different picture emerges when the classifier is called to work on volunteers' data, last row in Table IV. Performance drops considerably, with the F-Measure becoming as low as 52.9%. This happens because volunteers' behavior is much more complex, and clearly the dataset contains nuances that cannot be learned from synthetic or simulated data. Such results illustrate the importance of real data for training the machine learning algorithms, and question also testing on pure synthetic traces.

E. Comparison with Methodology in [3]

We apply the heuristic presented in [3] to our datasets, to contrast its performance with the machine learning approach we propose. Results are in Table V.

TABLE V
PERFORMANCE OF HEURISTIC PRESENTED IN [3] TESTED WITH VOLUNTEERS AND SYNTHETIC DATASETS.

Dataset	Accuracy	F-measure	Precision	Recall
Volunteers	0.9880	0.784	0.711	0.870
Synthetic	0.9680	0.782	0.628	0.944

Numbers in the table are compatible with what is reported in [3] – i.e., precision is on the 66%–80% range, whereas recall is close to previously reported results (91%–97% range). More interesting, we can contrast these numbers to those in Table III to compare an ad-hoc heuristic vs machine learning classifiers. Consider the testing on volunteers' dataset. It confirms that the machine learning approach delivers better performance than previous work, thanks to the ability of exploiting all features when building the model and during classification. Overall, machine learning offers thus a solid alternative for classifying user actions.

V. SUMMARY AND FUTURE WORK

This paper discussed the use of machine learning classifiers for the identification of explicit user actions in passive HTTP traces. In particular, we have (i) evaluated different subsets of features and machine learning models; (ii) analyzed the importance of dataset choices for training and testing the algorithms, and how such choices affect classification results; and (iii) studied how the performance of algorithms varies when different numbers of users are used for training.

Our work is a first step toward an automatic methodology for the analysis of HTTP traces. Some of our analyses have limitations which we plan to tackle in future work. First, we acknowledge that our data of volunteers' activity is not representative of the complete Internet population, nor it covers all types of client browsers. Yet, it is at least similar to previous works, and allowed us to show that the machine learning approach can reach comparable performance. Second, the machine learning approach may also require periodical updates of the classification model to keep pace with the web evolution. This will require mechanisms to obtain ground truth data from users with different profiles. Finally, although we compared three distinct models, many other algorithms and configurations of parameters could be tested, as well as mechanisms to evaluate and avoid over-fitting during training.

ACKNOWLEDGMENT

The research leading to these results has been partly funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-129, BigDAMA.

REFERENCES

[1] A. Finamore, S. Saha, G. Modelo-Howard, S.-J. Lee, E. Bocchi, L. Grimaudo, M. Mellia, and E. Baralis, "Macroscopic View of Malware in Home Networks," in *Proceedings of the CCNC*, 2015, pp. 262–266.
[2] R. Kumar and A. Tomkins, "A Characterization of Online Browsing Behavior," in *Proceedings of the WWW*, 2010, pp. 561–570.
[3] Z. B. Houidi, G. Scavo, S. Ghamri-Doudane, A. Finamore, S. Traverso, and M. Mellia, "Gold Mining in a River of Internet Content Traffic," in *Proceedings of the TMA*, 2014, pp. 91–103.
[4] G. Xie, M. Iliofotou, T. Karagiannis, M. Faloutsos, and Y. Jin, "Resurf: Reconstructing Web-Surfing Activity from Network Traffic," in *Proceedings of the Networking*, 2013, pp. 1–9.
[5] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger, "Understanding Online Social Network Usage from a Network Perspective," in *Proceedings of the IMC*, 2009, pp. 35–48.
[6] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX," *Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 2037–2064, 2014.
[7] S. Ihm and V. S. Pai, "Towards Understanding Modern Web Traffic," in *Proceedings of the IMC*, 2011, pp. 295–312.
[8] A. Finamore, M. Mellia, M. Meo, M. M. Munafò, and D. Rossi, "Experiences of Internet Traffic Monitoring with Tstat," *IEEE Netw.*, vol. 25, no. 3, pp. 8–14, 2011.
[9] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.
[10] L. Breiman, "Random Forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[11] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian Network Classifiers," *Mach. Learn.*, vol. 29, no. 2-3, pp. 131–163, 1997.
[12] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 1st ed. Upper Saddle River, NJ: Prentice Hall PTR, 1994.
[13] H. Kim, K. C. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices," in *Proceedings of the CoNEXT*, 2008, pp. 1–12.