Journal of Computer Security 22 (2014) 155–199
DOI 10.3233/JCS-130491
IOS Press

Web search query privacy: Evaluating query obfuscation and anonymizing networks [1]

Sai Teja Peddinti [a,*] and Nitesh Saxena [b]
[a] *Polytechnic School of Engineering, New York University, New York, USA*
E-mail: psaiteja@nyu.edu
[b] *University of Alabama, Birmingham, USA*
E-mail: saxena@cis.uab.edu

Web Search is one of the most rapidly growing applications on the internet today. However, the current practice followed by most search engines – of logging and analyzing users' queries – raises serious privacy concerns. In this paper, we concentrate on two existing solutions which are relatively easy to deploy – namely Query Obfuscation and Anonymizing Networks. In query obfuscation, a client-side software attempts to mask real user queries via injection of certain noisy queries. Anonymizing networks route the user queries through a series of relay servers, hiding the actual query source from the search engine. A fundamental problem with these solutions, however, is that user queries are still obviously revealed to the search engine, although they are "mixed" among queries generated either by a machine or by other users. We focus on TrackMeNot (TMN), a popular query obfuscation tool, and the Tor anonymizing network, and try to analyse whether these solutions can actually preserve users' privacy in practice against an adversarial search engine. We demonstrate that a search engine, equipped with only a short-term history of a user's search queries, can break the privacy guarantees of TMN and Tor by only utilizing off-the-shelf machine learning techniques.

Keywords: Search privacy, query obfuscation, anonymizing networks

1. Introduction

The popularity of Internet has foreseen many developments, a major one being the transformation of Web into a huge repository of information. Efficiently searching this vast amount of information is important, which led to the emergence of search engines. These search engines accept queries containing few words and provide related results to the users. To improve the relevance of these results, the leading search engines started logging and analysing the user queries. This prevalent practice has received considerable attention from media, public and researchers all over the world because of the possible privacy breaches.

Public awareness was raised when the U.S. Department of Justice issued a subpoena to Google for a week's worth of search query records and one million URLs

[1] This submission combines and extends two previously published papers [23,24].
[*] Corresponding author. E-mail: psaiteja@nyu.edu.

from its Web index in August 2005 [11]. Later in 2006, AOL released three months of search logs to the research community. Even though these logs were pseudonymized, the identities of a few users could be extracted based on the information embedded in their queries [3,13]. After these major incidents, growing attention has been given to how major search engines like Yahoo!, Google, AOL and MSN process and store the user queries over a long period of time.

Archiving the search queries is necessary for the search engine to improve the relevance of the search results. Also, analysing these stored records helps in generating revenue for the search engines through user specific advertising. There are policies in place to restrict the duration of storing these search records, but privacy breaches can take place however small duration these stored records correspond to.

The privacy breaches due to stored query logs, as seen in the case of AOL logs, can be broadly classified into *implicit* and *explicit* categories. Explicit privacy breach takes place because of the information embedded in the query itself. Frequently, the query content alone conveys personally sensitive information. Some examples include: a user searching for a particular disease he or his family member might be suffering from, searching for one's social security number (SSN) or phone number to check if it exists on the web, locating directions, subscribing to news items and performing "ego-surfing".[2] Implicit privacy violations, on the other hand, occur when the sensitive information cannot be learned directly from the query logs. In this case, information needs to be extracted using aggregation and profiling methods or data mining techniques. As an example, it is possible to infer the income level of a user by keeping track of the brand of products he/she often searches for [32].

Realizing the need for web search query privacy, researchers have come up with a number of techniques. These techniques can be broadly classified into three categories, based on the changes required at the server and the client sides, the requirement of any additional third-party infrastructure and the level of trust that the user needs to impose. The first class of solutions involves the use of Private Information Retrieval (PIR) protocols [20]. These protocols, however, require infrastructural changes at both the server and the client. Though practical PIR protocols guarantee computational privacy, they are not feasible to be deployed in practice due to the high communication and computation overload.

The second class of solutions requires the presence of third party infrastructure, like a proxy, e.g., Scroogle [30] or an anonymizing network, e.g., Tor [8]. These solutions are advantageous in the sense that they do not require changes at the server side, however, they force the user to impose unwanted trust onto the third party entities and have performance penalties. In the case of Scroogle, the user needs to trust a single server hosted by a third party company – which accepts the user queries, strips away any identifying information associated with the queries (like the cookies or the IP addresses), and acts as a relay between the search engine and the user. Performing web search over an anonymizing network certainly provides better protection

[2]Ego-surfing is a popular practice among users to search for their own names, just to check what results might appear.

and fault tolerance than a single proxy because it relies on multiple peers. These anonymizing networks are typically implemented using onion routing, whereby the user queries are routed through a series of nodes/relay servers before reaching the search engine; thereby hiding the actual source of the query. Because these relay nodes do not modify the queries in any way, the actual query source remains visible to the search engine if there is some identifying information associated with the query (like a cookie). To achieve better search privacy while using the anonymization networks, client side tools like Private Web Search (PWS) [28] have been proposed, which remove any identifying information associated with the query.

The third class of solutions are based on the principle of *Query Obfuscation*, whereby a client-side software injects many noisy queries into the stream of real user queries transmitted to the search engine. The basic idea is to make these noisy queries closely resemble the real queries, so that it is hard for the search engine to distinguish the real queries and profile the user. Since these methods only require changes at the client side, they can be easily adopted by privacy conscious users. These methods do not modify the user queries in any way, but just add few additional noisy queries; hence they do not prevent explicit privacy breaches, but only protect the user against implicit privacy violations.

Though each class of solutions has its own advantages and disadvantages, the third class is the easiest to deploy (since it only requires client side changes) and provides the highest level of confidence to a user, since the user has complete control and does not need to impose trust on any external entities. Since search engines generally are not motivated enough to incorporate any server side changes for user privacy, the second class of solutions, requiring third party infrastructure, are preferred over the first class.

1.1. Our contributions

In this work we try to assess and quantify the level of privacy provided by the third and second class of solutions, by evaluating one solution in each class; namely TrackMeNot (TMN) [15,34] query obfuscation tool and the Tor [8] anonymizing network.[3]

Both these problems are independent and yet related. They are independent because they are based on fundamentally different principles, yet they resemble similar binary classification problems (where one tries to separate the instances of two classes when mixed together). In the case of TMN, the problem boils down to identifying the queries of a user from the pool of queries containing user and *machine generated* queries. On the other hand, the case of anonymizing networks reduces to the problem of identifying the queries of a user from the pool of queries generated by the user and *other anonymizing network users*.

[3]We have independently evaluated these solutions and the results have been published at [23,24]. This submission combines and extends the two previously published papers.

We note that our problems are different from the problem of identifying user queries from a search log (see, e.g., [17,18]). First, an adversary in our work is the search engine itself and not a third party attempting to de-anonymize a search log. Second, unlike a third party, the search engine is already in possession of users' search history using which it can effectively train a classifier. Moreover, the goals of our study are also different; we are interested in using known classifiers to evaluate the minimum effort needed by an unsophisticated adversary to break the privacy guarantees offered by these protection services.

A higher level goal of our work is to assess and quantify the effectiveness of query obfuscation and anonymizing networks in preserving the user privacy in practice, against an adversarial search engine. In our assessment we try to model a naive adversary, with access to partial user search history and simple off-the-shelf machine learning techniques. Our results establish lower bounds on the accuracies that can be achieved, since we utilize as little information as possible. It is always possible to improve the results by utilizing more information or stronger machine learning techniques.

It has been mentioned in [28] that search queries, even though stripped off of any accompanying information like IP address and cookies, reveal some information about the user. This is associated to the linkability among queries. We make use of this query linkability to separate the machine generated TMN queries when mixed with user queries, and to separate a specific user's queries when mixed with queries of other Tor users. We try to address this open problem of linking queries to users by using machine learning techniques (called query classification) and show that queries of a user can be identified with reasonable accuracies, just by analysing the query content.

For TrackMeNot (TMN), we demonstrate that an adversarial search engine can break the privacy guarantees of TMN. More specifically, by treating a selected set of 60 users – from the publicly-available AOL search logs [1] – as users of the TMN software, we show that user queries can be identified with an average true positive rate of 48.88%, while the average TMN query misclassification rate being only 0.02%.

For an anonymizing network like Tor, we demonstrate that an adversarial search engine can extract user of interest's queries by utilizing only the query content. By treating a selected set of 60 users – from the publicly-available AOL search logs – as users of interest performing web search over Tor, we show that their queries can be identified with 25.95% average true positive rate when mixed with queries of 99 other Tor users, and with 18.95% average true positive rate when mixed with 999 other Tor users. Though the average accuracies are not so high, our results show that a few users of interest can be identified with accuracies as high as 80–98%, even when mixed with queries of 999 other Tor users.

We would like to highlight that the results for the first experiment are specific to the TrackMeNot tool, which is currently the only real-world implementation of query obfuscation. The results of the second experiment, in contrast, are generic and apply to any anonymizing network since we do not utilize any network specific information during the analysis.

1.2. Paper organization

The remainder of this paper is organized as follows. In Section 2, we present how author identification problems compare to query classification problems, and what prior work has been done so far in this area. Since we need to work with real user logs for our experiments, we describe our novel experimental methodology in Section 3. Our adversary model and metrics used to evaluate the performance of the machine learning techniques are discussed in Section 4. Section 5 explains the details of our first experiment, where we assess a practical query obfuscation tool called TrackMeNot. In Section 6, we explain the details of the second experiment, where we evaluate the effectiveness of anonymizing networks in protecting user privacy. This is followed by Section 7, where we try to analyse and interpret our results.

2. Related work

The query classification problem is very similar to authorship attribution problems, which have a long history. An early example of authorship attribution is the "Federalist Papers" [21]. The primary goal was to model unique author styles by looking at the text characteristics, such as vocabulary richness (Zipf's word frequency distribution), choice of rhymes, word length and habit of hyphenation. Reference [7] showed that it was important to select content-independent attributes, such as punctuation, usage of prepositions (e.g., "the", "if", "to"), etc. to reveal authors' individual characteristics.

With the evolution of electronic text like emails, blogs, tweets and online messages, the problem of authorship attribution has also evolved [38]. Compared to the traditional authorship attribution problem, where the texts were long and had few possible authors (as in the case of "Federalist Papers"), electronic data is neither long (e.g., tweets) nor does it have few authors. Firstly, shorter chunks of data makes it harder to apply regular text analysis techniques, such as bag-of-words. Second, style of the text may not help in identifying the author since it changes according to the recipient (like in e-mails – same author can write in different styles for different recipients). In most cases, the number of possible authors is either unlimited or very large. These issues make authorship attribution problem more challenging in the context of electronic data. However, studies like [7,19] and [2] show that sophisticated classification techniques (like Support Vector Machines (SVM) and multiclass classifiers – OVA and AVA) can give promising results even with electronic data. In this paper, we use some of these sophisticated techniques when off-the-shelf classifiers cannot help in identifying user's queries.

Query classification is even harder compared to e-mails or articles since the query length is much shorter. However, studies show that "Vanity Queries" [31] (those containing user identifying information, such as name, telephone numbers and zip codes) significantly help in identifying a user, given an anonymized search log containing queries (but not identifiers). Jones et al. [18] have worked on de-anonymizing query log bundles, which are used by the search engines to preserve the search logs and also to meet user privacy requirements. Making use of the vanity queries, they show that an attacker, having access to a query log bundle, can identify if the bundle contains a query session of a particular user or even identify the names and locations of users in that bundle. They also make use of analytical vulnerabilities and show that it is possible to un-bundle the logs and cluster queries into users. In a related work [17], by the same group of authors, it is shown that simple classifiers – those mapping queries into age, gender and location of user sending the queries – can be combined to map sequence of queries into a candidate user set which is 300–600 times smaller than random guess. They also show that it is possible for an attacker to identify the query session of a user, if he is able to guess some likely queries the user might make, and the chances greatly improve if the attacker gets to know some of the user's unique queries.

3. Query data

In order to pursue our study, we should work with real user queries. To this end, one possibility was to seek users who may volunteer to let us record their queries. However, due to the privacy concerns (which form the basis for our work), it was not feasible to recruit such volunteering users.

To address the above problem, we used a novel experimental methodology. We worked with the released AOL search data [1] and simulated the existing queries as they would have appeared to the search engine if the users were using the query obfuscation tools or the anonymizing networks. The AOL search data was well suited for this purpose because it consists of a large number of real user queries (21 million), corresponding to a large user base (650,000) and spanning over a reasonably long period of time (3 months). Though the AOL logs correspond to a different time period (year 2006), it does not affect our experiments because we concentrate on the query content alone and do not consider the associated query timestamps, as we discuss later in the paper. Since most queries do not have temporal dependence, we proceed with the use of historical AOL search logs for our experiments.

3.1. Relevance of AOL data

The behavior of a user group which uses privacy enhancing technologies is expected to be different from the group which is not aware of privacy issues or the existence of privacy enhancing tools. We do not have any information about the privacy consciousness of AOL users. However, since the AOL data was released before the concern for web search privacy grew among users (because most users were not actually aware that search engines were collecting such logs), or before the anonymizing techniques were actually proposed or used for this purpose, we believe the AOL logs most likely capture the non-privacy conscious user's search behavior.

We should note that it may not be possible to obtain data which can capture the difference in behavior of privacy conscious users in the real world. Privacy conscious volunteers may not be willing to share their web search query traces for research, and even if they share, it might not reflect their regular/normal behavior and would be corresponding to a made up personality. However, when people are knowledgeable about how a privacy enhancing technology works and if they believe in it offering protection, then there is a high chance that these privacy conscious users might open up and exhibit their normal web search behavior believing they are protected. Hence, we believe using the AOL logs might allow us to obtain a first step understanding of the protection offered by the privacy enhancing technologies.

3.2. AOL data statistics

These AOL logs span across three months: March, April and May of 2006. We use the May month's data for simulating the user and reserve the data from the first two months as the user history (which we assume is available to the search engine before the user started using TMN or Tor). Since search engines have existed for longer than a decade while the search privacy concerns grew recently in 2006, we believe it is safe to assume the search engine has a partial search history of the user – until the user started using these privacy tools. The first 2 months data will be used as a training set and the last month data will be the test set, in case of our supervised machine learning classifiers. This AOL data is a tuple of the form ⟨*AnonymousUserID, Query, QueryTime, ItemRank, ClickURL*⟩. For our experiments we do not make use of the *ItemRank* and *ClickURL* fields, since they are not available for all the queries.

Since it is hard to conduct the experiment on all the 650,000 users, we selected few users from the AOL logs for our experiments. This selection is done not at random, but based on the users' behaviour across different categories (discussed below) so that a wide variety of users are covered. The following five categories have been identified, since they were directly observable from the logs. For obtaining the statistics across each category, we considered all the 650,000 AOL users across all three months. All the graphs are plotted in logarithmic scale to closely observe the trends.

3.2.1. Number of queries

Over a period of time, different users send different number of queries. We calculated the total number of searches performed by each user and plotted the number of users across different query bands. From Fig. 1 (a power law distribution), we can see that most users lie below the 500 query mark with the bulk of them (about 98.72%) performing less than 100 searches over a three month duration. The rest are

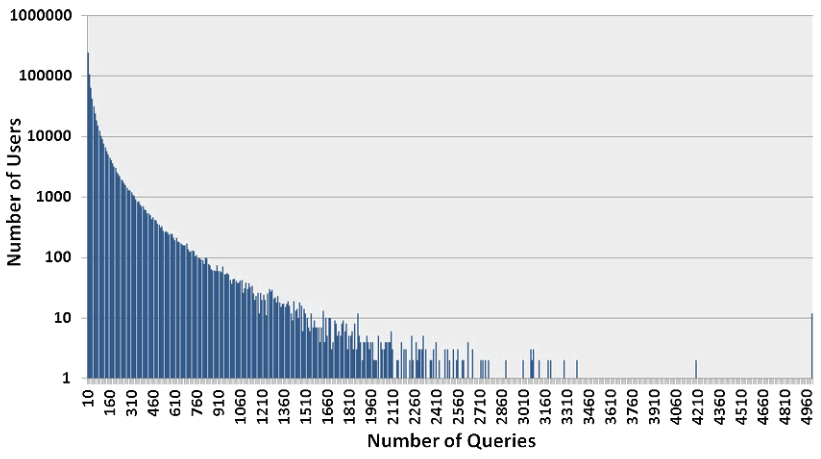

Fig. 1. Number of queries. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-130491.)

spread across the graph in small numbers. The same characteristics are also seen in the graphs plotting the maximum number of queries fired in a day, a week and one month versus the number of users in each query band. We thus combined these four into the same category (i.e. number of queries over a 3 month period).

3.2.2. Average query frequency

Users have different querying rates, which may turn out to be an identifying feature. We computed the average timing difference between successive queries for each user and plotted the number of users across different time bands – shown in Fig. 2. We can see that the largest user group has average query frequency less than 100 s; this is because most of the users send few queries in immediate succession and then remain inactive (might be because the user removed his cookies, and all his later queries were not attributed to the same user). We see that the number of users with moderate time difference (of few thousand seconds) between queries are few, because a large number of users seem to take long breaks between query sessions pushing the average time difference between queries to large values. Hence, there is a very small group of users who frequently send queries to the search engines.

3.2.3. Sensitive query content

The content of search queries obviously varies across users and we believe it might play a very important role in conveying information about the query source. We considered two broad classes for the query content: sensitive and insensitive. Sensitive queries are those, which a user may not be willing to reveal to the outside world, such as his/her medical condition, interest in weaponry (considering the alarming increase in terrorism), those related to child abuse and pornography, and so on. On the other hand, a user may not mind the public taking notice of his/her insensitive queries, such as those related to movie interests, sports and education.

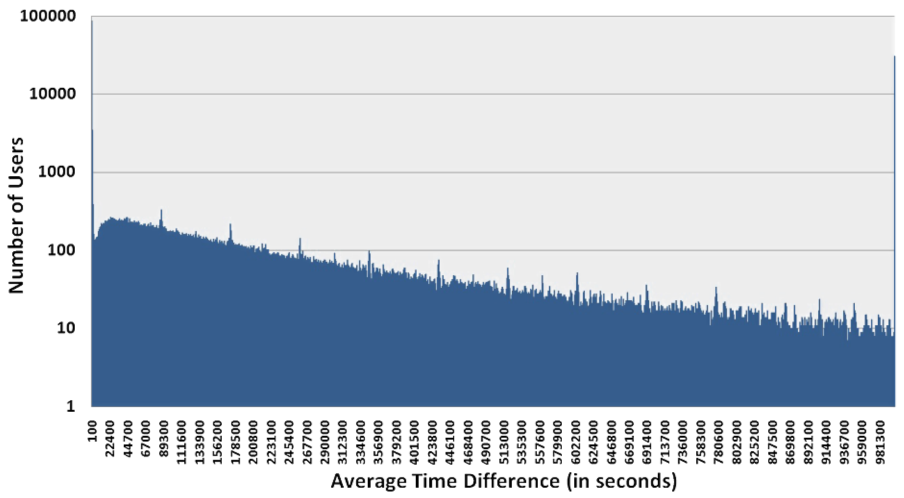

Fig. 2. Average query frequency. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-130491.)

We adopted two methods to identify the sensitive query distribution (one for each experiment), as we were unsure which might be a better way. For the first experiment, we resorted to machine learning techniques for classification of query content since it is easy compared to keyword based identification. We manually labelled a small subset of queries into sensitive and insensitive categories (we referred to various press articles discussing sensitive queries that appeared in the AOL logs [3,13]), and trained a Naive Bayes classifier with this data. We used this trained classifier to classify the rest of the user queries. The cross-validation true positive rate of this classifier on the manually labelled set was 68.0095%. Having classified each user's queries into the sensitive and insensitive categories, we plotted a graph indicating the number of users across different sensitive/insensitive percentages (see Fig. 3). The graph is alarming and contrary to what one would normally expect. A large number of users were classified to be making sensitive queries. This anomaly could be because of the way we trained the classifier; while training, we labelled the complete query in the training set to be sensitive or insensitive instead of just selecting some relevant keywords, because we did not want the filtering mechanism to miss queries – such as "how to kill your wife" – which are not necessarily keyword sensitive.

For the second experiment we decided to use the keyword based sensitive query identification, to see how the distribution might vary. We observed queries of 2000 AOL users and identified certain sensitive keywords relating to medical data, terrorism, weaponry, child abuse and pornography. Some sample keywords include steroids, marijuana, rape, porn and suicide bombers. If any one of these keywords occurred in a query, the query was labelled as sensitive. We labelled the queries of all the AOL users and found the percentage of sensitive-insensitive query distribution for each user. We plotted the user distribution across different percentages of

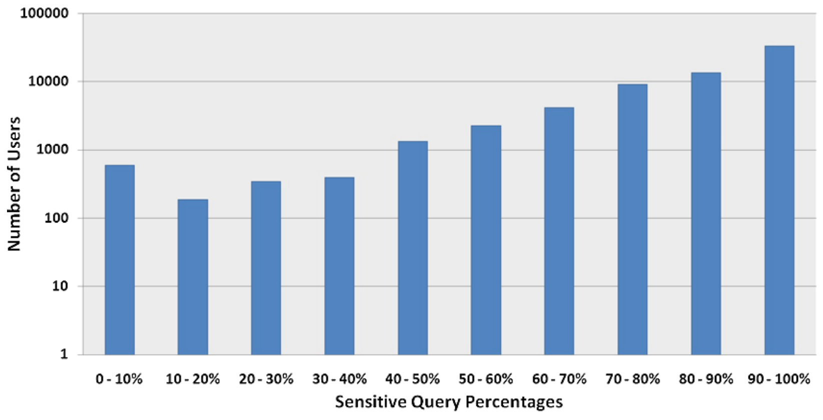

Fig. 3. Sensitive query distribution using Method1. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-130491.)

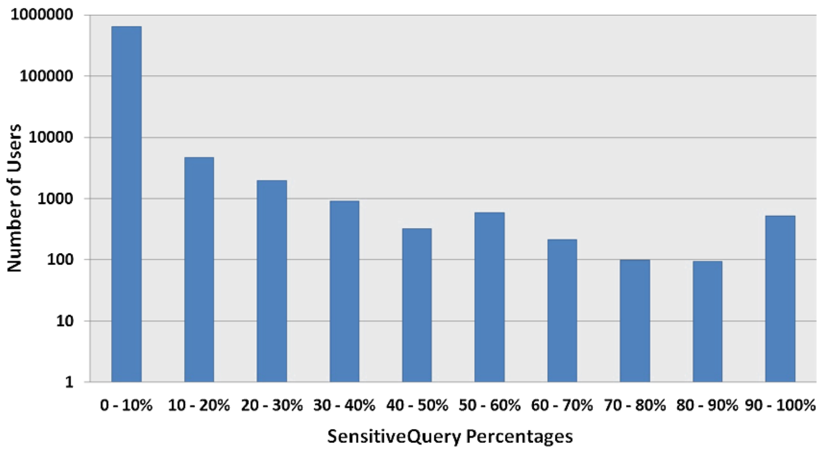

Fig. 4. Sensitive query distribution using Method2. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-130491.)

sensitive queries – shown in Fig. 4. Having a different keyword set might alter the distribution a little, but we expect the distribution to mostly remain the same.

3.2.4. Weekday/weekend distribution

From the logs, we observed that some users perform web search only during weekdays (they might be using corporate machines) and some only over the weekends. Speculating this as an important feature for user identification, we calculated the number of queries fired by each user over weekdays and weekends. We categorized users into three groups – those who search only over weekdays, those who search

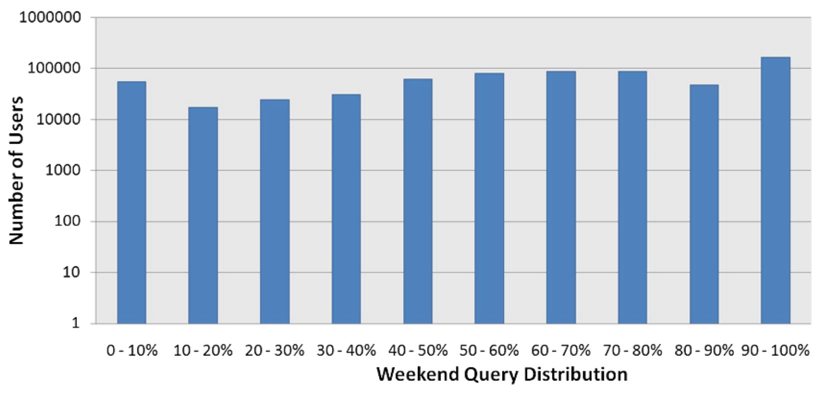

Fig. 5. Weekday/weekend distribution. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-130491.)

only over weekends and those who distribute their queries between weekdays and weekends. Figure 5 provides a graphical distribution of this data.

3.2.5. Query length

Though query lengths are implicitly attached to the queries, they may contribute towards identifying the user. As an example, if there is a user A who sends two word queries consistently, and we come across a query consisting of 5 words in length, then it is highly unlikely that user A might have generated that query. Hence considering the query length as an important attribute, we plotted the number of users across three different query length bands – Short, Medium and Long. The users in short band have average query length of less than 3 words, those in medium band have average query length lying between 4 to 6 words, and users in long band have average query length greater than 6 words. From Fig. 6, we can observe that a large number of users send short queries.

4. Our adversarial model

The adversary in our work is the search engine itself and not a third party, whose goal is to work against these privacy preserving solutions and identify the user queries for profiling and aggregation purposes. Unlike a third party, the search engine is already in possession of the users' search history which it can effectively put through use to gain an advantage. We consider the adversary to be passive and that it only analyses the logged queries. In particular, we assume it does not inject any manipulated responses to the user in an attempt to identify the user queries.

We try to model a naive and unsophisticated adversary, who makes use of off-the-shelf machine learning techniques to identify the user queries with as little effort as possible. Since the effectiveness of these machine learning techniques largely depends on choosing optimized parameters, we keep our attacks simple by either using

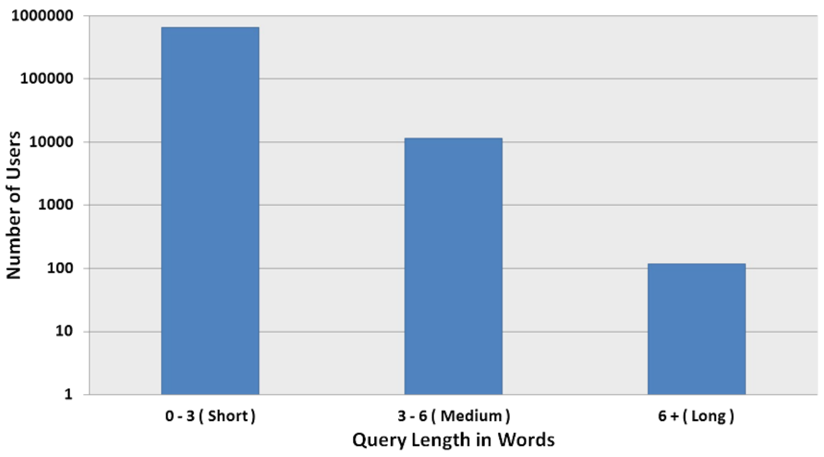

Fig. 6. Query length distribution. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-130491.)

default parameters for the techniques or making the simplest parameter optimizations. Also, we assume that the adversary does not make use of any information associated with the queries besides the queries themselves, like the cookie information associated with the search queries, the exact query timestamps or the Tor exit node IP addresses. This additional information may not be available when the user is using a sophisticated privacy solution (like PWS [28] + Tor). Therefore, our results establish the lowest accuracies that can be achieved, since we utilize the least information that is always disclosed, and it is possible to improve the results by utilizing more information or stronger machine learning techniques.

We can consider both the query obfuscation and anonymizing network problems as binary classification problems, where we try to associate query instances to either a user class or *Other* class. Here the *Other* class represents the TMN class in the query obfuscation experiment and the other Tor users in the anonymizing network experiment. For measuring the efficiency of the machine learning techniques in correctly identifying queries belonging to each class, we use two metrics: (1) percentage of *correctly identified User queries*, and (2) percentage of *Other class queries incorrectly identified* as user queries. If there are u user queries and t other class queries, recorded by the search engine, and a classifier predicted $u' + t'$ queries as user queries, where u' corresponds to correctly identified user queries and t' corresponds to incorrectly identified other queries, then our two metrics are given by u'/u and t'/t, respectively. We shall henceforth refer these measures as "True Positives" and "False Positives". (Standard Machine learning definitions use "accuracy" and "misclassification rates" as the performance metrics.) The classifier is said to be doing a good job if u'/u is close to 1 and t'/t is close to 0, i.e., percentage of correctly classified user queries is close to 100% and percentage of incorrectly classified TMN queries is close to 0%.

4.1. Relevance of classification results

Let us assume that the search engine is interested in identifying the queries corresponding to a user of interest, A. Assume there are n_u number of A's queries and n_o number of *Other* class queries (note that generally $n_u \ll n_o$). The search engine can select a query at random and can trivially identify it to be A's query with a probability $p_{\text{naive}} = \frac{n_u}{(n_u+n_o)}$. Instead, if we apply our classification approach which has a true positive rate of x (i.e., probability of correctly identifying A's queries) and a false positive rate of y (i.e., probability of incorrectly identifying others' queries as A's queries), then we obtain a (very small) subset of $x * n_u$ A's queries and $y * n_o$ other users' queries. Now, if we pick a random query from this subset, then the probability that this query is A's query is $p_{\text{class}} = \frac{x*n_u}{(x*n_u+y*n_o)}$. If our classification is doing a good job, i.e. if x is high and y is low, then p_{class} would be significantly higher than p_{naive}, which in turn would mean that we are doing a much better job of identifying user's queries than we do with a random guess.

For example, consider n_u to be 100 and n_o to be 100,000. Assume the classifier's true positive rate, x, to be 50% and the false positive rate y to be 0.1%. Based on the numbers, we can see that the classifier would label 150 queries as A's queries. The probability of identifying A's queries in this small subset of 150 queries is much higher ($p_{\text{class}} = 0.33$) when compared to the default probability ($p_{\text{naive}} = 0.001$). The adversary, i.e. the search engine, will prefer to use these less noisy 150 queries for personalization and aggregation purposes as compared to the 100,100 more noisy queries. This example clearly shows the need to use a classifier to obtain a purer sample of user queries.

In all our experiments, we try to evaluate these true positive rate x and the false positive rate y for the classifiers, which help us in validating whether the p_{class} is higher than p_{naive}; our results show it is indeed the case.

5. Evaluating privacy provided by query obfuscation

The goal of this experiment is to find how effective query obfuscation can be in preserving users' privacy in practice. We focus on a real world query obfuscation tool called TrackMeNot (TMN) [15,34], which is implemented as a Mozilla Firefox plugin. This tool programmatically generates queries, mimicking the user search behaviour, and attempts to hide the real user queries in this fake query stream. TMN has taken necessary measures to closely simulate user's search behavior and has evolved considerably over time. Currently TMN is a popular and robust query obfuscation tool, with 469,098 downloads for the plugin version 0.6.721 [35].[4]

Previously, theoretical models have been developed to bring insights into the effectiveness of query obfuscation for search privacy [37] and a brief analysis of TMN has

[4]We refer the reader to Bruce Schneier's criticism of TMN when it was introduced in 2006 [29].

recently been conducted in [6] using search logs from a single user (see Section 2.1 of [6]). The current work represents the first step, to the best of our knowledge, towards a large scale analysis of TMN.

We set out to investigate the following question: *Is it possible (and to what extent) for an adversarial search engine – equipped with users' search histories – to filter out TMN queries using off-the-shelf machine learning classifiers*?

Overview of results: We answer the above question affirmatively. We selected 60 users from the publicly-available AOL search logs and treated them as users of the TMN software. As per our metrics defined in Section 4, we are able to achieve an average true positive rate of 48.88% for identifying user queries, while the average false positive rate is only 0.02%. We also observed that for few users, the true positive rates were greater than 80% and as high as 100%, whereas for others, the true positive rate was less than 10%. In almost all cases, the p_{class} value was very close to 1 and much higher than the value of p_{naive}, which was always less than 0.1. Based on our results, we can conclude that most users are susceptible to privacy violations even while using TMN, some of them being significantly more vulnerable than others.

5.1. *Background*: *TMN query generation*

In this section, we discuss TMN query generation process. We first try to understand this process based on what was reported in [15], and then, for deeper insights, inspect TMN's source code [34]. At a high level, the goal of TMN is to hide user queries in a stream of random machine generated queries. In order to make these random queries indistinguishable from real user queries, TMN adapts the randomly generated queries to content a user is interested in. This is achieved by studying the web search responses to real user queries and extracting useful information. TMN uses this extracted information to adapt the random generated queries accordingly. To understand how TMN works and adapts to user queries, consider a small example. Assume user A installed TMN and started using it. Initial TMN queries would be generated from popular news RSS feeds, hence the fake query stream contains queries like "new apple macbook pro launch", "Obama and whitehouse", etc. When A searches for "google android versions", TMN notices this query and its accompanying search results. By learning from this, it generates newer queries like "Google android", "android phones", etc. and adds them to the query stream. Later on when user searches for "restaurants in California", TMN adds queries related to "california" and "restaurants" to the query stream. This adaptation to user queries helps TMN generate realistic looking fake queries.

5.1.1. *Understanding TMN from the literature*

TMN hides the user queries in a stream of programmatically generated search queries, which mimic or simulate the user's search behavior. In order to make the fake queries look realistic, TMN includes many features described below. TMN maintains a dynamic query list, which is instantiated with an initial seed list of queries obtained from popular RSS feeds and publicly available recent searches. Later, individual queries from this list are randomly selected and substituted with query-like words from HTTP response messages returned by the search engine for actual user queries – thereby adapting the newer queries to reflect the content a user is interested in. Over time, each TMN instance develops a unique set of queries and adapts itself to the user's interested content and mimics the user more closely.

TMN employs a "Selective Click-Through" mechanism, which simulates the user behavior of clicking on the query results returned by the search engine. It also keeps track of all the user searches by monitoring all outgoing HTTP requests from the browser using the "Real Time Search Awareness" mechanism. The "Live Header Maps" feature enables TMN to adapt dynamically to the client's browser, such as browser version and operating system details, helping TMN to use the exact set of headers that the browser uses. TMN also implements "Burst Mode" queries in order to incorporate the common user behavior of firing related queries in immediate succession as part of a query session.

With all these features, TMN is believed to be a good simulator of user's searching behavior. However, it has certain drawbacks as mentioned in [15]. TMN cannot mask a user's private information (e.g., names or phone numbers) included in the search queries, and it cannot prevent user identification based on the IP address or cookies. In order to hide one's IP address while searching, TMN developers recommend the use of anonymizing networks, such as Tor [8,15]. Reference [29] points out TMN might generate "hot-button issue" searches – those involving sensitive search query terms (e.g., "HIV", "drug-use" and "bombings"), which the user might not be willing to search. The TMN authors claim that this problem can be prevented by configuring the initial RSS input feeds and thus controlling the type of queries sent by TMN. Based on these discussions, we can say that TMN (potentially) only provides protection against aggregation and profiling of individual search queries by adversarial search engines. With and without the use of TMN, user's area of interest would be exposed to the adversary, but when using TMN, the actual search queries would be masked in a stream of related queries. The better the simulated queries resemble the actual user queries, the better are the chances for TMN to hide the actual user queries.

5.1.2. Understanding TMN from the source code

In order to obtain a deeper understanding of TMN, we analyzed the supporting code of TMN's Firefox extension. Mozilla extensions which are written in XUL and JavaScript, provide an easy way to develop new applications on top of the basic Firefox browser platform. The XUL language extends the GUI of the browser while the JavaScript helps in defining the functionality.

When TMN is installed on the Firefox browser, it creates a default query seed file and a query list. This query list is initialized with some queries extracted from the default or supplied RSS feeds and this list is padded with some queries from the query seed file. Once done, a search is scheduled immediately (delay is 0 s). For the later queries, some non-zero random timer values are used based on the query generation frequency chosen by the user (and some random offset) using the TMN control panel.

After the delay timeout, a fake search query request is initiated. A random query is selected from the query list and with some probability the query is modified – only the longest word in the query is retained or a negated word is added to the query (such as "word1 word2 – word3"), or quotation marks are added. This modified query is used as the search query. Sometimes, if "Burst Mode" is enabled, a sequence of related queries might be generated from the selected query by omitting some keywords at random. These Burst Mode queries are sent within short intervals of time, so as to form a chain of related searches – simulating user search sessions, where queries sent in immediate succession relate to the same topic.

When encoding the generated search query into a URL, some browser details are to be mentioned in the header portion of the URL request. TMN maintains a list of headers and URLs for each search engine, and an entry in these lists gets updated when TMN observes any change in the user communication with the search engine. The previously selected and modified query is encoded into the search request URL with updated headers and an XMLHttpRequest is generated. Apart from displaying the sent query on the Firefox status bar, TMN also stores this request URL in logs for later reference. When there is a state change in the XMLHttpRequest sent, i.e., when a response is received from the server, an appropriate action is taken based on the HTTP status response. If an error occurs, it is logged. If the HTTP status response is *OK*, then the HTML response is processed and keywords are extracted from the textual content on the web page. The TMN query list is updated by randomly replacing few old queries in the list with these new extracted keywords, thereby incorporating the user interests. Since, the newer queries are again selected from this same query list, the newer queries would better adapted to the content the user is interested in. Also based on some probability, TMN tries to simulate the user click-throughs. To this end, TMN identifies the links on the HTML response, processes these links, and picks one of them at random. After some delay, another XMLHttpRequest is generated with the selected link, thereby simulating the user behavior of clicking a link. TMN does not process the returned html response for this click-through link. If Burst Mode is enabled, TMN schedules the next search with the following search queries in sequence. After a timeout, TMN repeats the whole procedure again with another randomly picked query from the query list.

In this way, the TMN query list (and so the TMN query seed file) gets updated with keywords extracted from the web response returned by the search engine, both for the user queries and TMN queries. In the long run, TMN gets adapted to the query content the user is interested in and generates better queries making it (potentially) much harder for the search engine to differentiate the noisy queries from the original user queries. Because some form of randomization occurs at each and every step, it is impossible for two TMN instances to generate the same set of TMN queries.

5.2. *Experimental study of TMN: Preliminaries*

Based on our discussion in previous section, we find that TMN has taken necessary measures to simulate user's search behavior and generate noisy queries as similar as possible to user's queries. TMN has also evolved considerably over time resulting in a potentially robust and popular query obfuscation tool. In this work, we set out to investigate whether it is still possible (and to what extent) for an adversarial search engine to filter out TMN queries using off-the-shelf machine learning classifiers, and thus evaluate the privacy guarantees provided by TMN. Our *adversarial model* is discussed in Section 4. We assumed that the search engine would have access to user's search histories for a certain duration until the point the user starts using the TMN software.

In order to pursue our study, we should work with real user queries; and as discussed in Section 3 we work with AOL logs. We selected a few users from the AOL logs and simulated their behaviour of issuing queries to the search engine while TMN is installed and running on their machines. TMN is a Mozilla extension and these extensions, installed on a Firefox browser, operate only on one user profile – the one on which it was installed. Hence, we can have multiple Firefox user profiles, each with its own independent TMN instance, simulating a different user.

Due to the resource limitations on a single machine, it is difficult to run many Firefox user profiles simultaneously. To remedy this, we used the PlanetLab [25] system, a global distributed research network used by researchers to develop network applications and run network simulations. PlanetLab resources are assigned to the users as a resource slice, and these slices are instantiated by assigning nodes to it. Each of these nodes need to be configured with the experimental environment, which in our case is a working Mozilla Firefox browser with the TMN plugin installed.

5.2.1. *Selecting users*

We decided to select 15 AOL users from the following four categories, i.e., a total of 60 users.

Number of queries. From Fig. 1, we find that most users are below the 100 query mark, and of these, more than 70% perform fewer than 30 searches during the three month period. Thus, we selected eight users at random from the set of users who fire less than 30 queries, five users from the set of users who fire less than 100 queries and two users who pose more than 100 queries.

Average query frequency. The graph in Fig. 2 is smooth everywhere except for a sharp peak at 200 s. To take this into account, we randomly selected five users from the set of users with an average query frequency of less than 200 s, five users lying near the second rounded peak at 35,000 s, and the remaining five from the set with more than a million seconds average gap between successive queries.

Sensitive query content. Since there are a large number of users in the 100% sensitive band (Fig. 3), we randomly selected six users from this set. Five users are selected from the 30% insensitive – 70% sensitive band, two users from 10% sensitive – 90% insensitive band, and the remaining two from 100% insensitive query set.

Weekday/weekend distribution. Based on the distribution in Fig. 5, we equally divided the choice of users among those who fire all their queries over weekdays, those who distribute 40% on weekday and 60% on weekend, and those who search only during weekends.

5.3. Experimental set-up and implementation

After the user selection, the task ahead was to simulate the user logs while a TMN instance per user is running, and record all the resulting queries. Sixty nodes (corresponding to each selected user) were allocated to the PlanetLab resource slice, and each of these nodes maintains one Firefox user profile. Since Mozilla is a GUI application and X11 forwarding (necessary to run GUI applications over SSH connections) is not enabled on the PlanetLab machines due to security reasons, we had to install a VNC server on each of the nodes, which provides a GUI enabled remote access to these machines. Google was chosen as our (adversarial) search engine.

To simulate user's search behavior as per AOL log files, we developed a Mozilla extension which reads the user logs and fires the queries at timestamps listed in the logs. Similar to the TMN plugin, the new plugin also generates the user queries as XMLHttpRequests. The html response – from the server – to these queries is processed by TMN, since TMN does not find the corresponding request URL in its database (see Section 5.1). TMN treats the webpage to be a valid response to an actual user query and adapts itself to the new data – the exact behavior we need. Since the AOL user logs belong to a different time frame (year 2006), they were translated to the present time. The average query frequencies of TMN instances were chosen at random so as to keep them as close as possible to the real user behavior. We also ran 5 additional TMN instances with varying average TMN query frequency, for the same user, on our local machines in order to evaluate the effect of TMN query frequency on the level of privacy provided by TMN. After configuring the necessary settings on PlanetLab machines, both the user log simulator and TMN were started. These experiments were conducted for a period of one month, and backup of the logs was taken at regular intervals.

5.3.1. Classification of user and TMN queries

For our machine learning requirements, we used WEKA [36], an open source software which supports many machine learning algorithms and data preprocessing options. We used this off-the-shelf machine learning toolkit in order to estimate the true positive rate with which we (adversarial search engine) can filter user queries, from the pool of user and TMN queries we obtained as described in previous section. For this experiment we treat WEKA as a black box and do not dwell deep into the working of the algorithms. Generally performance of a machine learning technique depends on the optimal parameters chosen for the problem; to keep our attacks simple we do not make any optimizations, but use the default parameters.

Two main categories of machine learning algorithms which can be used for our application are clustering and classification algorithms. Classification is a supervised mechanism, where we need to train the classifier on some labelled training set, and assign labels to quantities in the test set. Clustering algorithms, without any prior knowledge of labelled data, try to group the data into clusters/groups, such that elements in a group share some common features. Clustering is unsupervised [36].

5.3.2. Preparing the data

The pool of simulated user and TMN query logs, collected over the one month period (as discussed in previous section), form our test data which needs to be clustered or classified. We labelled each query in the test data as a user or TMN query, since we want to test the performance of machine learning algorithms after categorizing the queries. The data includes the query, its label and the timestamp when the query was fired. For indicating the time, we used WEKA's DATE attribute in "yyyy-MM-dd HH:mm:ss" format. This string is internally converted by WEKA into a numeric representation (akin to the unix time stamp) when using it for machine learning. The queries are strings and WEKA cannot directly handle string attributes. So we used a preprocessing filter, called *StringToWordVector*, which breaks down the words in the string and converts them into numeric attributes. Each string gets converted into a word vector of 1's and 0's in these attributes, where '1' indicates the presence and '0' indicates the absence of the word in the string. For example, having just two queries – "apple macbook pro" and "google android phones" – results in 6 numeric attributes, one for each word ($\langle apple, android, google, macbook, phones, pro\rangle$). Both the queries can be represented as vectors in these attributes, like $\langle 1, 0, 0, 1, 0, 1\rangle$ and $\langle 0, 1, 1, 0, 1, 0\rangle$. These word vectors are generated from the set of words appearing in the training set. Any new unseen words appearing in the test set are neglected. So, a test query like "apple ipad" would be converted to $\langle 1, 0, 0, 0, 0, 0\rangle$ vector, after neglecting the "ipad" word. So every query would be represented as a big tuple/vector in the query words, the timestamp and the label we assign.

5.3.3. Clustering algorithms

We started with the unsupervised/clustering schemes since they are simple and potentially more powerful (as no labelled training is needed). We tested the performance of well known clustering algorithms, such as SimpleKMeans, Farthest First and EMClusterer [9] with *default parameters*, using the *Classes-to-Clusters* evaluation mode in Weka. In this testing mode, the pre-assigned labels are masked and the data gets processed using the other attributes. Once the clusters are formed, the labels are unmasked and the majority class in each cluster is determined to find the performance of the algorithm as per these labels. However, the clustering algorithms with default parameters could not distinguish user queries from those of TMN and placed both types of queries into the same (TMN) cluster, for all of our test users. We note that it is possible to achieve better user query identification results by fine tuning the parameters of the clustering algorithms or applying other procedures, such as n-grams. However, since our goal is to simulate a naive adversary trying to identify the efficiencies using simple off-the-shelf machine learning tools with no parameter optimization, we defer this task to future work, and rather concentrate on classification algorithms.

5.3.4. *Classification algorithms*

To train the classifiers we need to have sample data corresponding to both the user and TMN classes (i.e., pre-labelled data). If only one of user or TMN training data is used, all the queries would get classified into the same class since there are no identifying features available for the second class. The training set for the user queries was obtained from AOL two month user history, as discussed in Section 3. To obtain the TMN training set, we used the logs from a TMN instance which was run independently of all our simulations on a desktop machine for a period of one week.

With these training and test sets, we chose five classifier algorithms, out of the several classifiers applicable to our scenario, based on their performance in few preliminary tests. They are: Logistic (Regression), Alternating Decision Trees (ADTree), Random Forest, Random Tree and ZeroR. For now, we neglect other better classifiers like SVM, so as to estimate the lowest accuracies that can be achieved. For the sake of completeness, a brief description of each of these classifiers is provided below:

- *Logistic* (*Regression*): Regression classifier models are used to predict the probability of occurrence of an event by trying to fit the data to a logistic (linear, polynomial, etc.) curve in a multidimensional space. Based on the location of the data point in the multidimensional space (relative to the logistic curve), a label is assigned to it. For example, in the case of linear regression, all points lying on one side of line in the 2-dimensional space are given one label, and those on the other side are given another label. The positioning of the line is determined based on the training data, so as to minimize the errors and get a good separation between the class points. Logistic regression is mainly used when there are two classes of data (where the classes can be separated by a line in the point space), but multinomial versions also exist [36].
- *Alternating Decision Trees* (*ADTree*): These are similar to binary search trees, where we start from the root and traverse to the leafs by taking the path based on the value of the search variable. A decision tree algorithm contains decision and prediction nodes. These decision (non-leaf) nodes specify a condition while the prediction (leaf) nodes contain a number. Based on the attribute values in the input data vector, we travel along one path from the root node to the leaf/prediction nodes which determine the classification label for the particular

data vector. Variants of the traditional decision trees exist, where we simultaneously travel along many paths up to the leaf prediction nodes and the end result is determined by considering all the prediction node values covered [36].

- *RandomForest*: They are also based on classification trees, but unlike alternating decision trees Random Forests use a collection of classification trees. The input is made to travel across all the trees and the final decision is made based on voting, where the output of each classification tree is taken into consideration [36].
- *RandomTree*: It is based on classification tree principles, but the nature of the nodes in the tree is different. The algorithm considers K randomly chosen attributes at each node in the tree (unlike a single attribute as in alternating decision trees) and provides an estimation of class probabilities instead of specifying a single label/class for the data vector [36].
- *ZeroR:* It is the simplest classification algorithm and is based on majority. The algorithm identifies the majority class label in the training data and classifies every data element in the test set with this majority label, thereby providing the threshold accuracy that should be provided by other classifiers [36].

Query and date attributes. To check for the influence of each of the attributes (query and date) on the classification, we tested the performance of the above four classifiers (except ZeroR as its user true positive rate is 0% due to a large TMN query set) across the following three settings for a couple of test users. Our goal was to determine as to what extent these attributes might be useful for classification.

(1) Considering only date and label value attributes.
(2) Considering only query and label value attributes.
(3) Considering both query and date along with label value attributes.

The results obtained are indicated in Table 1. (The percentages indicate the fractions of user queries correctly identified by the classifiers – i.e. the true positive rates; the false positive rates, i.e. the TMN query misclassification rates, were close to 0% in most cases and so are not listed.) We can clearly see that out of the three settings, considering only query attribute along with label values provides the maximum true positive rate. Including the date attribute (internally it is converted to a numeric value by WEKA) reduces the true positive rate and considering only the date attribute yields the worst true positive rate. Therefore, for the analysis of rest of the experimental data, we neglect the date attribute and consider only query and label values as the data to be classified. There could be other possible uses of the date attribute like timing windows, but we neglect them for now to keep our attacks simple. Since Naive Bayes is a standard classifier which can be used when date attribute is not considered, we replaced ADTree with Naive Bayes classifier for the rest of our analysis.

Table 1

% of user queries correctly classified with different attributes

		Classifier true positive rates			
		Logistic (%)	ADTree (%)	Random forest (%)	Random tree (%)
User 1	Only query	92.59	82.22	92.59	89.63
	Only date	14.44	13.7	13.7	13.7
	Both query and date	92.59	13.7	89.63	46.30
User 2	Only query	85.19	85.71	86.77	86.24
	Only date	3.17	0.53	0.53	0.53
	Both query and date	10.58	0.53	68.25	0.53

Table 2

% of user queries correctly classified for different TMN query frequencies

TMN query frequency	True positive rates (%)		False positive rates (%)	
	Naive Bayes	Logistic (Regression)	Naive Bayes	Logistic (Regression)
10 per minute	6.25	56.25	0	0.06
5 per minute	0	56.25	0	0.02
1 per minute	56.25	56.25	0	0.12
30 per hour	56.25	56.25	0	0
10 per hour	56.25	56.25	0	0

TMN average query frequency. To test for the effect of TMN's average query frequency in protecting users' privacy, as mentioned earlier, we ran another 5 simulations apart from the 60 simulations considered before. Each of these 5 simulations, simulated the same user but with different TMN query frequencies – 10 per minute, 5 per minute, 1 per minute, 30 per hour and 1 per hour. After one month, these TMN logs were analyzed using the chosen classifiers. The results obtained for Naive Bayes and Logistic (Regression), which yielded the best true positive rates, are depicted in Table 2. Though the performance of Naive Bayes was varying a little, the Logistic regression classifier was found to have a constant true positive rate. This suggests that using different query frequencies would more or less provide the same level of privacy. In other words, higher TMN frequency may not help in hiding user's query better, contrary to one's intuition.

Independent user history. Since using an independent TMN log for training the classifier turned out to be helpful in identifying the user queries with good accuracies, we performed a test to validate whether any user log data other than the actual user's history would also give similar results (if this were the case, the search engine would not need access to every user's history of searches). To this end, we considered four users – $user_1$, $user_2$, $user_3$ and $user_4$, from the AOL log data. Now, instead of using a user's history to train the classifier for that user, we used the history of $user_4$ as the training data and tried to classify $user_1$, $user_2$ and $user_3$'s simulated queries from their respective TMN query pools using Logistic, RandomForest, RandomTree and Naive Bayes (after replacing ADTree, as described before) classifiers. In all the cases, none of the user queries were identified correctly – that is the true positive rate turned out to be 0%.

Our analysis above shows that an independent user log is not helpful in distinguishing between user and TMN queries, but an independent TMN log is. One reason for this could be that the independent TMN log was functional around the same time frame as other TMN instances (i.e., it was run along with other TMN instances) and it used the same default RSS feeds to populate the TMN query list. Note that an adversarial search engine can also produce such updated TMN log from time to time for training the classifiers.

We note that many users are not likely to pay attention to the RSS feeds chosen for query generation and may use the default ones. Thus, in our experiments, we used the default RSS feeds thereby generating the same initial seed list of queries. We have not closed the browsers while conducting our experiments because of the common practice among users to put their computers to sleep and re-invoke them instead of switching them off and rebooting the machines each time, and also due to their tendency to continue using the browser without restarting unless it crashes. We acknowledge that not closing the browsers might affect the efficiency of TMN, because TMN uses the RSS feeds to update the query list with new keywords only when the browser restarts.

5.4. Classification results

After collecting the query and label data from the 60 user simulations, we were ready to execute the selected classifiers. As mentioned earlier, for each of the 60 users, we randomly chose the TMN query frequency, simulating a real user behavior (as shown in Table 2, this should not impact our results much). Hence the number of TMN queries mixed with the real user queries differs in each experiment, impacting our p_{naive} and p_{class} values. We built the training set with user history log and an independent TMN log as discussed previously. The results of classifiers over the test data are depicted in Table 3. For simplicity, we have not listed the results for all the classifiers; rather we only report the performance of the standard Naive Bayes classifier and the maximum true positive rate achieved among the other three classifiers (Random Forest, Random Tree and Logistic). Also, the true positive rates shown are the *mean true positive rates* of the users belonging to different AOL categories (as defined in Section 5.2.1).

We find that for all users, the classifiers did a very good job of correctly identifying almost all TMN queries; average false positive rate was close to 0.02%. In other words, there were very few TMN queries which were wrongly classified. The true positive rates for identifying the user queries were not very high in general;

Table 3

Mean true positive rates of user queries and mean false positive rates of TMN queries for each category of users

	Users	True positive (%)		False positive (%)		p_{naive}	p_{class}
		Naive Bayes	Max.	Naive Bayes	Max.		
No. of queries							
0–10	8	6.15	11.52	0	0.07	0.002	0.97
11–100	5	7.08	33.14	0	0.25	0.011	0.79
100+	2	18.71	33.86	0.06	0.29	0.08	0.94
Average query freq. (s)							
0–100	5	28.16	40.41	0.03	0.01	0.007	0.99
35,000	5	30.83	71.86	0.01	0.01	0.006	0.87
$>10^6$	5	9.23	36.28	0	0	0.003	1
Sensitive query content (%)							
0	2	60	60	0	0	0.003	1
10	2	61.46	64.79	0	0	0.002	1
70	5	45.53	63.96	0.02	0.14	0.007	0.84
100	6	23.97	39.02	0	0.16	0.017	0.84
Weekday/weekend distribution							
Only weekdays	5	12.28	12.28	0	0	0.002	1
Only weekends	5	23.2	99.99	57.26	0.08	0.004	0.57
Distributed	5	1.22	99.92	86.94	0.08	0.032	0.99

average true positive rates over all users was 48.88%. In most cases, the classifier was able to identify a reasonable fraction of user queries correctly. However, there were indeed some cases (e.g., one in Sensitive Query Content and one in Average Query Frequency categories) where 100% true positive rate was achieved in identifying the user queries. There were 4 other user instances for which more than 80% true positive rates were achieved. The p_{naive} and p_{class} values are reported for the best performing classifier in each AOL category. In all cases, the values of p_{class} are much higher than p_{naive}, which reinforces that classifiers are useful for obtaining a purer sample of user queries.

5.5. *Discussion of results*

In this section, we discuss and attempt to interpret the results obtained in Section 5.4. The first key insight from our results is that the classifiers were very accurate in identifying the TMN queries (mean false positive rate over all users was only 0.02%). This is perhaps because the TMN query log – using which the classifiers were trained – consisted of a reasonably large number (42334) of TMN queries (although only corresponding to a week's period) which was likely sufficient to extract features for identifying TMN queries. Recall that this log was generated around the same time frame as our test user instances, which might have been helpful in correct classification of TMN queries. Note that an adversarial search engine can also produce such updated TMN log from time to time for training the classifiers. A very low false positive rate of TMN queries implies that any query classified as a user query, is indeed a user query with significantly high probability.

The true positive rates for user queries, on the other hand, were not as good as they were for TMN queries (we obtained a mean user query true positive rate of 48.88% over all users). One possible reason for relatively low rate in this case is that we were only able to leverage users' two-month history for training purposes. Since a large number of users only fired less than 100 queries (as seen from Fig. 1) over 3 months, the classifiers did not have a large number of user queries to work with. Due to this reason, perhaps it was not possible to derive identifying characteristics for user queries in a number of cases. We believe that, in practice, the search engines can utilize long-term search histories available to them prior to a user starts using the TMN software, resulting in much better true positive rates. Even with our current average identification rates, the value of p_{class} is almost close to 1 and is much higher than p_{naive}, which is always lower than 0.1. The search engine can identify nearly 50% of user queries and still use them for profiling and aggregation purposes (since almost no TMN queries were incorrectly classified, as discussed above). Remember, the amount of information we used for the classification was very little (just the query content) and we believe the results would improve if we made a better use of the time stamps and use the *ClickURL* and *ItemRank* features. Note also that our true positive rates were found to vary significantly across different users. We observed that queries corresponding to some of the users could be identified with greater than 80% and as high as 100% true positive rates, whereas for others, the identification rate was less than 10%. Based on our current experiments, we can conclude that most users are susceptible to privacy violations even while using TMN, and some of these users are significantly more vulnerable than others (as we discuss below).

Looking at Table 3, we can make inferences regarding which users are possibly more vulnerable based on our different categories: number of queries, average query frequency, sensitive query content and weekday/weekend distributions. User query identification true positive rates seem to be slightly improving with the number of queries posed by the users. Although the false positive rates are increasing very slightly, we can ignore them considering a good improvement in user query classification rate. These results are justifiable because the more the number of queries sent by a user, more are the chances to identify user query patterns and hence better are the true positive rates. Users with very fast (less than 100 s) and very slow (more than 1 million seconds) average querying frequencies seem significantly less vulnerable compared to those with moderate (35,000 s) frequencies. The very fast and very slow category users are those who send very few queries in immediate succession and then remain idle or spread their few queries across 3 months duration. Since the queries available for analysis are few, the true positive rates are bound to be less for these users compared to the ones belonging to the moderate category.

We do not notice any significant effect of the sensitivity of query content on classification accuracies. However, for users who did not pose any insensitive queries (based on our categorization in Section 3.2.3), true positive rates were found to be relatively lower. Therefore, based on our sensitive query classification, the users who fire a larger fraction of sensitive queries were better camouflaged by TMN than those who fire a larger fraction of insensitive queries. This might be because of the presence of many sensitive queries in the initial query set generated from the default RSS feeds.

Users who engage in web search only during weekdays turned out to be much better protected compared to those who pose queries only over weekends (queries of such users can be identified with almost 100% success). This might be because the nature of queries posed during the weekends make them more attributable to the user – as they are based on the user interests. The queries posed during weekdays (highly likely to be from a work place) may not exactly reflect the same, mostly due to the self imposed restrictions on searching for queries based on user interests from a work place. Hence if users send queries only during weekends, then these queries have higher chances of being related to the user. Finally, from Table 2, we also observed that using different TMN average query frequencies would more or less provide the same level of privacy. In other words, higher TMN frequency may not help in hiding user's query, contrary to one's intuition.

In summary, our results indicate that TMN is very susceptible to machine learning attacks. In fact, TMN could be weaker than what our attacks imply. This is because we only used some simple off-the-shelf classifiers with default parameters and this itself resulted in considerable true positive rates. Use of better and stronger machine learning algorithms, with optimized parameters, is very likely to further increase the accuracies.

5.6. Experimental summary

We focused on TrackMeNot (TMN), a real-world search privacy tool based on query obfuscation, and demonstrated that a search engine, equipped with only a short-term history of user's search queries, can break the privacy guarantees of TMN by only utilizing off-the-shelf machine learning classifiers. More specifically, by treating a selected set of 60 users – from the publicly-available AOL search logs – as users of the TMN software, we showed that user queries can be identified with an average true positive rate of 48.88%, while the average TMN query false positive rate was only 0.02%.

6. Evaluating privacy provided by anonymizing networks

The first experiment shows that query obfuscation is not very reliable and does not provide complete privacy to the user as expected. The next best privacy option is to use additional third party infrastructure with minimum expansion of trust beyond the user. This leads us to anonymizing networks which certainly provide better protection and fault-tolerance than the use of a single third party proxy server. As discussed before, anonymizing network routes user queries over a path consisting of a series of nodes (called relay servers) distributed all over the internet, effectively hiding the actual source of the query. We have considered the Tor [8] anonymizing network for studying the effect of anonymizing networks in protecting the user's search privacy.

In this experiment we try to analyse how effective an anonymizing network can be – in preserving users' privacy in practice – against an adversarial search engine. (From here on, we will call the anonymizing network as Tor, for simplicity of presentation and without loss of generality.) We observe that the web search over Tor network has one fundamental drawback: the search query has to reach the search engine in clear text format, for the search engine to be able to process the query and return the response back to the user. In other words, a user's search queries are not hidden from the search engine. However, these queries are indeed "mixed" among the queries issued by other users of the same anonymization service.

We ask the following question: *Is it possible for an adversarial search engine to associate queries coming out of Tor exit nodes to Tor users who issued these queries?*

Our adversarial model is discussed in Section 4. In an attempt to keep our attacks more generic and not limited to only one particular type of anonymizing network, we assume that the adversary does not make use of any information specific to Tor anonymizing network – like the Tor exit node IP addresses. We base our work on an important observation, made by other researchers [5], that although a potentially large number of users might be accessing web search over the Tor network, only a small fraction of these users really remain anonymous to the search engine in practice. The reason is that a significant number of users, even while using Tor, remain logged in to their accounts with search engines (e.g., Gmail accounts with Google) and may not disable cookies and other identifying information [5]. It is possible to use TOR clients like Private Web Search (PWS) [28] and remove any identifying information accompanying the query, but users are not generally aware of such tools. This implies that a user's queries might be getting mixed among queries of only a small number of other (anonymous) Tor users, potentially making these queries more identifiable.

Overview of results. We answer the aforementioned question affirmatively. More specifically, by treating a selected set of 60 users – from the publicly-available AOL search logs – as "users of interest" performing web search over an anonymizing network, we show that their queries can be identified with high true positive rates of 80–98%, even when queries of up to 999 "other users" (other Tor users) are mixed together. Our results indicate that we can identify a user of interest's queries with 25.95% average true positive rate, when queries of up to 99 other Tor users are mixed together, and this average true positive rate drops to 18.95% when queries of 999 other Tor users are mixed together. Our experiments indicate that users who pose a larger number of queries, those whose queries are longer and those with higher fraction of sensitive queries are more vulnerable. Our results cast serious doubt on the effectiveness of anonymizing web search queries by means of anonymizing networks. Since our attacks only exploit a minimal amount of information (just the query content) for the query association task and short-term search histories, and only use existing classification tools, stronger attacks resulting in improved query identification rates are very much plausible.

6.1. Problem formulation and study methodology

We base our work on the query information alone, because the search query must reach the search engine in clear text format even while using any anonymizing network (the query passes through many in-between relay nodes in encrypted format when using TOR). In an attempt to keep our attacks more generic and not limited to only one particular type of anonymizing network, we assume that the search engine does not make use of any other information associated with the queries, such as the query timing information, the exit node IP address, unblocked cookies accompanying the queries or the Click-through patterns followed by the user. This allows us to completely avoid the need to simulate the re-routing of queries through Tor.

An important point to notice, in the context of working with old logs, is that we should prevent combining query logs belonging to different time periods so as to prevent identification of queries based on the temporal information. For example, if user A's query log belongs to year 2005 and user B's query log belongs to year 2010, then separating their queries when mixed together is a lot easier using the temporal information embedded in the queries, such as the current news topics, etc. This problem does not arise in our case since all the selected users belong to the released AOL logs. In addition, the timestamps of the queries cannot be used directly. If the queries are channelled through Tor, the queries are going to experience considerable (random) delay because of the Tor's re-routing. Hence, we do not make use of the exact value of the timestamps, but we experiment only with timing windows and consider the queries within those windows.

For our study, we assume that the adversarial search engine has access to the list of possible Tor users performing web search over Tor ($\mathbb{U}$). This is a realistic assumption because the search engine has access to a user's search history and it can determine whether a particular user has possibly started to use Tor for issuing queries. This can be done, for example, by identifying the query patterns (such as query frequency) for a user or for an IP address with the help of cookies. If for a reasonable duration, e.g., a week, the search engine does not receive any queries from a user or an IP address, violating the user's typical querying pattern, it can mark that user as a potential Tor user. Even when the users delete their cookies, there are chances that the search engine might mistake these users to be possible Tor users. However, since the user querying patterns do not change when using new cookies, the search engine might be able to map these mistaken Tor users to the new cookies if the querying patterns match, and continue profiling the users. Such mapping techniques or anti-aliasing techniques have been studied before in [22]. Though the content they deal with in [22] are large texts like bulletin boards and web pages, we believe similar techniques could be developed for mapping web search users associated with different cookies. Thus, we assume throughout our study that the search engine can generate a possible Tor user list and keep updating it. A determined search engine may also run Tor relays and collect the IP addresses of all Tor machines which connect to these relays.

Over a period of time, the search engine logs a set of queries (denoted $\mathbb{Q}$) that it receives from the Tor exit nodes (list of exit nodes is publicly available). These queries are issued by the users appearing in the list $\mathbb{U}$ and are all mixed together. Our (i.e. the adversarial search engine) goal is to identify the queries in $\mathbb{Q}$ that correspond to some or all users in $\mathbb{U}$. We model this identification problem as a classification problem in machine learning, whereby we train a classifier with the prior search history of the Tor users (collected prior to the time they started using Tor) and ask the classifier to classify the queries in $\mathbb{Q}$ to the respective users in $\mathbb{U}$. Since we might want to associate the queries to all users, we will need one class per user. Therefore, the problem reduces to a multiclass classification [27] problem. In the rest of this paper, we denote the AOL users, whose queries are to be separated from the mixed query set received at Tor exit nodes as 'users of interest' and the total number of users using web search over Tor as $\mathbb{U}$. The users present in $\mathbb{U}$, apart from the users of interest, will be referred as 'other users'.

The size of $\mathbb{U}$ (denoted N) is an important parameter for our study and for the level of privacy that can be provided to the users performing web search over Tor. Intuitively speaking, the larger N is, more difficult it would be to correctly classify queries. However, we argue that in practice N may not be very large. As discussed in Section 6, recent research [5] shows that although there are, on an average, 1893 Google users at one Tor exit node over one week, about 872 of these users access the services by signing into Google, making themselves identifiable even while using Tor. Of the rest 1000 who did not sign into Google, a significantly large fraction of users may not have disabled or deleted their cookies while using Tor. Cookies and other identifying information can be blocked by using tools such as PWS [28], as mentioned before. However, as discussed in [5], a significant number of users are not aware of these options. In summary, even if a large number of users might be using Tor for private web search, only about few hundreds ($\sim$500) of unidentified users exist at each Tor node and with 1500 Tor exit nodes in total, only about few hundred thousands ($\sim$750,000) of users actually remain anonymous to the search engine. In addition, the search engine might not want to track each and every one in this anonymous user set, but instead it might want to concentrate on few users – selected based on the kind of sensitive queries they send or based on their real world identities (like suspected terrorists). In light of these important observations, we consider a maximum of $N = 1000$ anonymous web search users, and try to associate the queries in $\mathbb{Q}$ to these users. We believe that this number 1000 is reasonable for experimental purposes.

As discussed in Section 4, we aim to identify the true positive rate x and false positive rate y for each user of interest. We want x to be high and y to be low for p_{class} to be significantly higher than p_{naive}. As a concrete example, our classification attacks for AOL user #67910 yield $p_{\text{class}} = 0.73$ ($x = 45/192$ and $y = 16/46{,}062$), which is about 183 times more than the probability of a random guess $p_{\text{naive}} = 0.004$, when $N = 100$. Through our classification experiments in the rest of this paper, we aim to find out the values of x and y for a diverse set of users firing different type of queries, and for different values of N (100, 200, 300, 500 and 1000).

6.2. *Selecting users*

As mentioned earlier, we do not intend to identify queries belonging to all anonymous Tor web search users present in $\mathbb{U}$, but only concentrate on a specific few (60) users of interest. Note that since we are concentrating on the query content alone, we have not considered any categories related to the query timing information, such as the average timing difference between queries and weekday–weekend distribution. We had selected 20 users of interest from each category as follows:

Number of queries: We selected 14 users of interest at random from set of users who fire less than 100 queries, 4 users of interest at random from set of users who fire 101–500 queries and 2 users of interest at random from the set of users who fire more than 500 queries over a period of three months.

Query length: Most users send queries of average length less than 3 words, as seen from Fig. 6. Following these statistics, we have chosen 15 users of interest randomly from the set of users sending short queries, 3 users of interest were selected at random from the set sending medium length (3–6 words) queries and 2 users of interest were selected at random from the set sending long (more than 6 words) queries.

Sensitive queries: From Fig. 4, we observed that a large number of users belong to the 0–10% sensitive query band and the rest are spread over other percentages in small numbers (a few hundreds). Hence, 10 users of interest are selected at random from the set of users sending 0–10% sensitive queries, 2 users of interest are selected at random from 10–20% sensitive query band, 2 users of interest are selected at random from 20–30% sensitive query band, 3 are selected from 50–60% sensitive query band and another 3 from 90–100% sensitive query band in proportion to the actual user distribution.

6.3. *Selecting classifiers*

Differentiating user queries from the machine generated queries is relatively easy compared to differentiating one user's queries from another user's. During the first experiment we observed that clustering algorithms with default parameters do not help much, hence we did not consider them. Since the current problem is harder, compared to the first experiment, we dropped the classifiers discussed in Section 5.3.4 and have chosen Support Vector Machine (SVM). This selection is based on strong recommendations, such as [14], to use SVMs for textual classification or categorization, and its wide spread application in similar projects [7] and [19].

There is more than one implementation of SVM algorithm in WEKA, and we have selected WLSVM – which is the WEKA integrated version of LIBSVM (the popular software for Support Vector Classification, Regression and Distribution Estimation). WLSVM in WEKA implements five classification algorithms and of these three were not suitable for our scenario. The other two, named C-SVC and nu-SVC – where SVC stands for Support Vector Classification, are suitable for our problem. Considering our main goal of keeping the attacks simple enough for a naive adversary, we preferred C-SVC over nu-SVC because of the simplicity in choosing the algorithm parameters. The positive and negative training data for our classifier is obtained from the user search history – positive examples for a user include all the queries sent by the user so far and negative examples include the queries sent by the other Tor users in the past.

C-SVC is a binary classifier, that separates instances of two classes when mixed together. Multiclass classification can be solved by converting multi-class problem into multiple binary classification problems. These are popularly called as One-vs-All (OVA) and All-vs-All (AVA), as described in [26]. Assume there are N users/classes, with each user assigned an integer in the range $[1, N]$. All the queries in the training data belonging to user i (where $i \in [1, N]$) are labelled i. In OVA model, we build one separate classifier for each user/class in the dataset. For ith classifier, the positive examples will be the training data with label i and the negative examples include all the data with a label different from i. We classify the test data with the class label depending on which corresponding classifier outputs greater/larger value. In AVA model, we build $N(N-1)$ classifiers, one for each pair of classes i and j. Each of these classifiers (i, j) gets trained on only the data belonging to classes i and j. At the end, the label is predicted by following a voting mechanism (see [26] for details).

Both OVA and AVA are applicable to our problem and can yield good accuracies. C-SVC directly performs multiclass classification by implementing AVA when the number of classes are more than 2; thus, we used this feature directly. For OVA, we used a meta classifier in WEKA, which helps us to implement the OVA pipeline using C-SVC as the base classifier.

Classifier parameters play an important role in correctly identifying the queries and increasing the performance. The performance of C-SVC classifier is determined by three parameters – kernel type, cost parameter-C and an Epsilon (ξ) value. By following the SVM parameter guide [16], we chose the simplest linear kernel for our problem. The optimal values for other two parameters, C and ξ, have been chosen using another meta classifier in WEKA called *CVParameterSelection*. Given a sample dataset and the parameter to be optimized, this meta classifier performs classification on the sample dataset using all the parameter values within a specified range and identifies the best value based on the classification performance.

Table 4
Comparison of true positive rates for attribute selection

Classifier	True positives					
	No additional attributes (%)	Including query length (%)	Including timing window (%)			
			3 h	4 h	6 h	12 h
AVA	16.26	14.58	13.16	14.08	13.62	14.41
OVA	13.65	14.41	13.98	12.99	12.63	14.15

6.4. Selecting attributes

As discussed previously, we decided to concentrate on the query content alone, without using any additional information. However, we wanted to test the influence of time feature on the achievable accuracies and so performed a small experiment. Since time feature cannot be used directly because of the inherent delay when queries are sent over Tor, we considered timing windows of considerable duration. Since it is hard to predict what size of the timing window might provide better results, we divided 24 h in a day into different non-overlapping windows of sizes of 3, 4, 6 and 12 h and compared the accuracies with each timing window size. Also, we considered the Query Length feature, though it is implicit in the Query information. User's anonymous ID and Query are the necessary attributes. In order to determine the impact of each additional attribute on the classification results, we tried to identify the average true positive rates of user query identification for all the users of interest when $N = 100$, by including one additional attribute at a time. The average true positive rates are indicated in Table 4. We can see that by including the Query Length feature reasonable performance is achieved both in the case of OVA and AVA. Addition of timing windows did not provide much improvement over the existing true positive rates, both in the case of OVA and AVA. There could be other possible and better uses of these query times, but we neglect them for now. Hence for all the following experiments we included Query Length as an additional attribute along with Query and the user's anonymous ID.

6.5. Experiment results

In our experiments, we tried to estimate the true positive rate of the classifiers in correctly identifying queries of 60 users of interest, who are chosen at random based on the AOL query statistics described in Section 6.2. For each user of interest, we measure the true positive rate and false positive rate (along with p_{naive} and p_{class}) across five datasets, where in each dataset, we vary the number of 'other users' whose queries are mixed with that of the current user of interest. The five datasets containing randomly selected 99, 199, 299, 499 and 999 other users were generated. In order to be consistent across all 60 users of interest, we used the same 'other user' datasets.

Thus, when the user of interest's query set is mixed with the queries of these 'other users', we form datasets with N as 100, 200, 300, 500 and 1000 users.

For each user of interest, we performed both OVA and AVA classification with C-SVC as the base classifier. As outlined in Section 6.3, the kernel option in C-SVC is chosen as Linear Kernel. The best C and ξ parameters chosen by the meta classifier (CVParameterSelection) are $C = 336$ and $\xi = 0.001$.

For each of the three categories, we summarized the OVA results indicating the average values of *True Positive* and *False Positive* for all the users of interest in specific sub-categories. The summary of OVA results for Number of Queries is given in Table 5, summary of OVA results for Query Length is given in Table 6 and the summary of OVA results for Sensitive Queries is depicted in Table 7. The values in the table indicate the true/false positive rates in percentages followed by the fraction which gave those values. These fractions were included to indicate the actual number of queries correctly classified or misclassified – helping us to measure the p_{naive} and p_{class}. The results for AVA classification, for each category, came out to be very similar to that of OVA classification, and are thus not reported in the paper.

The average true positive rates shown in the tables are reasonable. Across all the 60 users of interest, the average true positive rate was 25.95%, when $N = 100$, and this rate drops to 18.95% when $N = 1000$. More importantly, though the average true positive rates are not very high, our results show that few users of interest could be identified with true positive rates as high as 80–98%, even when $N = 1000$. This can be seen from Table 8, which lists the top five users of interest in each category with high end true positive rates (several of them have rates as high as 80–100%). Each row in the figure corresponds to a user of interest, and these users within each category, are arranged in the decreasing order of their true positive rates (when $N = 100$). Also, across all 60 users for $N = 1000$, the p_{naive} values range between 0–0.015, while the p_{class} values vary between 0.13–0.8.

6.6. *Interpretation and discussion of results*

In this section, we attempt to discuss and interpret the results of our study, and draw some useful conclusions.

The first observation looking at Tables 5–7 is that the average true positive rates (i.e., the fraction of correctly classified queries) indicate that in most cases some fraction (at least a quarter) of users' queries can always be correctly classified. The false positive rates are also very low in almost all cases, which can be credited to our choice of optimal parameters for the classifier.

Looking at Table 5 across the rows, we find that the true positive rates are likely to increase with the number of queries posed by the user – the more active a user is, the more identifiable his/her queries become. We can explain this based on the nature of machine learning techniques. When more data is available about a user, the machine learning techniques can extract better information about the user interests, and so can make better predictions. Lack of data results in more noise and therefore

Table 5

OVA results summary for number of queries

Number of queries fired on average (Z)	Total number of users	100 user		200 user		300 user		500 user		1000 user	
		True positive	False positive	True positive	False positive	True positive	False positive	True positive	False positive	True positive	False positive
$Z < 100$	14	4.35% (1/23)	0.02% (1/4497)	4.35% (1/23)	0.06% (5/8775)	4.35% (1/23)	0.06% (7/11,800)	4.35% (1/23)	0.01% (3/20,144)	0.00% (0/23)	0.02% (7/39,481)
$100 < Z < 500$	4	8.86% (7/79)	0.38% (20/5247)	8.86% (7/79)	0.11% (11/10,239)	8.86% (7/79)	0.13% (18/13,766)	7.59% (6/79)	0.06% (15/23,501)	6.33% (5/79)	0.07% (33/46,061)
$500 < Z$	2	36.71% (76/207)	0.36% (19/5248)	19.32% (40/207)	0.22% (23/10,238)	13.53% (28/207)	0.11% (15/13,766)	14.01% (29/207)	0.06% (15/23,501)	14.93% (31/207)	0.05% (22/46,061)

Table 6

OVA results summary for query length

Query length	Total number of users	100 user		200 user		300 user		500 user		1000 user	
		True positive	False positive	True positive	False positive	True positive	False positive	True positive	False positive	True positive	False positive
Short	15	19.40% (13/67)	0.025 (1/4497)	17.91% (12/67)	0.20% (19/9555)	16.42% (11/67)	0.15% (19/12,848)	16.42% (11/67)	0.10% (21/21,934)	15.15% (10/66)	0.06% (26/42,991)
Medium	3	20.0% (4/20)	0.15% (8/5248)	20.0% (4/20)	0.01% (1/10,238)	20.00% (4/20)	0.01% (1/13,766)	20.00% (4/20)	0.00% (1/23,501)	20.00% (4/20)	0.00% (11/46,061)
Long	2	93.33% (28/30)	0.02% (1/5247)	90.00% (27/30)	0.00% (0/10,238)	90.00% (27/30)	0.01% (1/13,767)	90.00% (27/30)	0.035% (6/23,501)	90.00% (27/30)	0.03% (14/46,061)

Table 7

OVA results summary for sensitive queries

Sensitive percentage bands	Total number of users	100 user		200 user		300 user		500 user		1000 user	
		True positive	False positive	True positive	False positive	True positive	False positive	True positive	False positive	True positive	False positive
0–10%	10	39.2% (245/625)	0.83% (39/4723)	38.24% (239/625)	0.53% (49/9215)	38.24% (239/625)	0.38% (47/12,390)	37.76% (236/625)	0.22% (46/21,151)	35.68% (223/625)	0.16% (65/41,455)
10–20%	2	14.78% (34/230)	0.06% (3/5248)	13.04% (30/230)	0.05% (5/10,239)	13.04% (30/230)	0.03% (4/13,767)	12.61% (29/230)	0.02% (5/23,502)	13.04% (30/230)	0.06% (29/46,062)
20–30%	2	36.36% (24/66)	0.13% (7/5248)	37.88% (25/66)	0.12% (12/10,239)	37.88% (25/66)	0.12% (16/13,767)	37.88% (25/66)	0.14% (34/23,502)	37.88% (25/66)	0.09% (40/46,062)
50–60%	3	22.22% (4/18)	0.08% (4/5248)	22.22% (4/18)	0.03% (3/10,239)	22.22% (4/18)	0.03% (4/13,767)	22.22% (4/18)	0.02% (4/23,502)	22.22% (4/18)	0% (1/46,062)
90–100%	3	96.3% (78/81)	0.1% (5/5248)	96.3% (78/81)	0.06% (6/10,239)	96.3% (78/81)	0.06% (8/13,767)	74.07% (60/81)	0.03% (7/23,502)	29.63% (24/81)	0.03% (13/46,062)

Table 8

OVA users with best accuracies

OVA Top5 users	User	100 user		200 user		300 user		500 user		1000 user	
		True positive	False positive	True positive	False positive	True positive	False positive	True positive	False positive	True positive	False positive
Number of queries	1	50% (4/8)	0.019% (1/5248)	50% (4/8)	0.19% (19/10,239)	50% (4/8)	0.15% (21/13,767)	50% (4/8)	0.00% (1/23,502)	50% (4/8)	0% (0/46,062)
	2	25% (5/20)	0.019% (1/5248)	25% (5/20)	0.01% (1/10,239)	25% (5/20)	0.01% (2/13,767)	25% (5/20)	0.01% (2/23,502)	25% (5/20)	0% (0/46,062)
	3	58.33% (112/192)	0.30% (16/5248)	32.29% (62/192)	0.26% (27/10,239)	20.31% (39/192)	0.13% (18/13,767)	18.75% (36/192)	0.06% (14/23,502)	23.4375% (45/192)	0.03% (16/46,062)
	4	17.69% (23/130)	0.78% (41/5248)	17.69% (23/130)	0.32% (33/10,239)	17.69% (23/130)	0.33% (46/13,767)	17.69% (23/130)	0.1% (23/23,502)	14.62% (19/130)	0.12% (56/46,062)
	5	11.11% (2/18)	0.08% (4/5248)	11.11% (2/18)	0.04% (4/10,239)	11.11% (2/18)	0.007% (1/13,767)	11.11% (2/18)	0% (0/23,502)	11.11% (2/18)	0.017% (8/46,062)
Query length	1	94.74% (54/57)	0% (0/5248)	94.74% (54/57)	0% (0/10,239)	94.74 % (54/57)	0% (0/13,767)	94.74% (54/57)	0% (0/23,502)	94.74% (54/57)	0.01% (5/46,062)
	2	84.88% (73/86)	0% (0/5248)	84.88% (73/86)	0% (0/10,239)	84.88% (73/86)	0% (0/13,767)	84.88% (73/86)	0.012% (3/23,502)	84.88% (73/86)	0.00% (1/46,062)
	3	59.68% (37/62)	0% (0/5248)	59.68% (37/62)	0.01% (1/10,239)	59.68% (37/62)	0.05% (7/13767)	59.68% (37/62)	0.034% (8/23,502)	59.68% (37/62)	0.033% (15/46,062)
	4	57.14% (4/7)	0.06% (3/5248)	57.14% 4/7	0.19% (19/10,239)	57.14% (4/7)	0.14% (19/13,767)	57.14% (4/7)	0.10% (24/23,502)	57.14% (4/7)	0.028% (13/46,062)
	5	50% (12/24)	0% (0/5248)	50% (12/24)	0% (0/10,239)	50% (12/24)	0.007% (1/13,767)	50% (12/24)	0.01% (2/23,502)	50% (12/24)	0.00% (2/46,062)

Table 8

(*Continued*)

OVA Top5 users	User	100 user		200 user		300 user		500 user		1000 user	
		True positive	False positive	True positive	False positive	True positive	False positive	True positive	False positive	True positive	False positive
Sensitive queries	1	100% (107/107)	0.095% (5/5248)	100% (107/107)	0.048% (5/10,239)	100% (107/107)	0.0508% (7/13,767)	100% (107/107)	0.034% (8/23,502)	0% (0/107)	0% (0/46,062)
	2	97.82% (45/46)	0.095% (5/5248)	97.82% (45/46)	0.048% (5/10,239)	97.82% (45/46)	0.0508% (7/13,767)	97.82% (45/46)	0.034% (8/23,502)	97.82% (45/46)	0.0824% (38/46,062)
	3	96.59% (1417/1467)	0.57% (30/5248)	96.59% (1417/1467)	0.097% (10/10,239)	96.59% (1417/1467)	0.036% (5/13,767)	96.52% (1416/1467)	0.021% (5/23,502)	96.52% (1416/1467)	0.045% (21/46,062)
	4	89.13% (82/92)	0.114% (6/5248)	89.13% (82/92)	0.087% (9/10,239)	89.13% (82/92)	0.08% (11/13,767)	30.43% (28/92)	0.025% (6/23,502)	30.43% (28/92)	0.006% (3/46,062)
	5	75% (9/12)	0.15% (8/5248)	75% (9/12)	0.039% (4/10,239)	75% (9/12)	0.036% (5/13,667)	75% (9/12)	0.0297% (7/23,502)	75% (9/12)	0.009% (4/46,062)

poor results. Table 6 shows that longer queries are likely more identifiable. This is because only a very small fraction of users issue longer queries (more than 6 words), as seen from Fig. 6, and thus their queries are distinctive among the pool of a large number of shorter queries. Following the same reasoning, we observe, from Table 7, that true positive rates are expected to increase as the sensitivity of the query content increases (recall the sensitive/insensitive query distribution in Fig. 4). This is an important insight demonstrating that users who pose more sensitive queries are likely easily identifiable. Unfortunately, this contradicts the fundamental goal of using an anonymizing network in the first place – to hide the fact that one is issuing sensitive queries. In all cases, the p_{naive} values (ranging between 0–0.015) are much smaller than the p_{class} values (ranging between 0.13–0.8), showing that the classifiers are generating a small but good sub-set of queries, containing more user queries and fewer noisy/other queries. Though the results might seem intuitive and follow the trend that users who stand out are easily identifiable, they provide us a very good estimate of what percentage of these user queries can actually be identified because of the query properties.

6.7. Experimental summary

In this section, we studied the problem of identifying a user's queries from a pool of queries received by a search engine over an anonymizing network. We demonstrated that an adversarial search engine, equipped with only a short-term search history, can extract user of interest's queries by utilizing only the query content and off-the-shelf machine learning classifiers. More specifically, by treating a selected set of 60 users – from the publicly-available AOL search logs – as users of interest performing web search over an anonymizing network, we showed that their queries can be identified with 25.95% average true positive rate when $N = 100$, and with 18.95% average true positive rate when $N = 1000$. Though the average true positive rates are not very high, our results show that few users of interest can be identified with rates as high as 80–98%, even when the value of $N = 1000$.

7. Analysis

In this section, we attempt to provide an explanation, as to why we obtained the presented true positive rates in both the experiments and how it was possible to identify the user queries.

7.1. Reasons behind the true positive rates

To understand our results, we try to find the reasons behind how and why a query gets identified as a user query. Analysing the results of the machine learning classification and its behaviour change as the experimental parameters vary is not a trivial

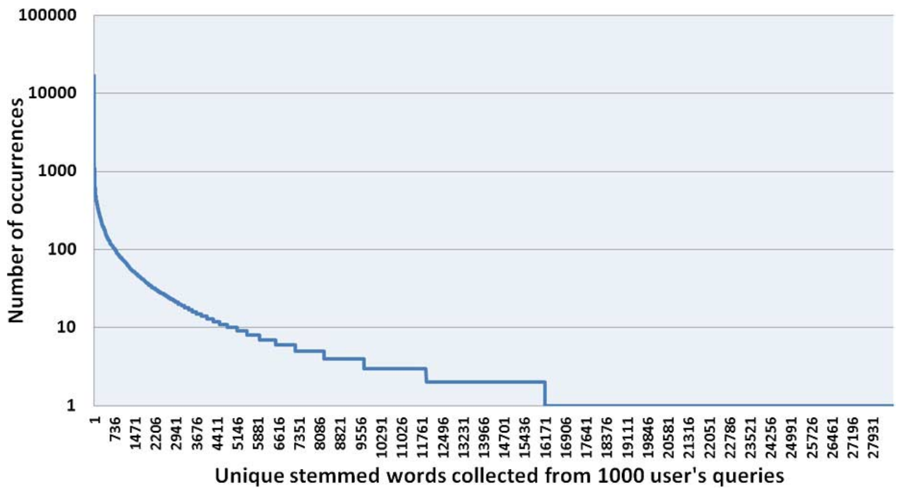

Fig. 7. Root keyword distribution. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-130491.)

task. This is something that cannot be done in a straight-forward known way. However, we still make an attempt to study the factors behind the classification.

Since the data fed to the machine learning algorithm contained queries broken down into word vectors, we tried to identify the word usage distribution among 1000 users. By trimming down all the query words, using stemming algorithms present in WEKA, we identified the "root" keywords appearing in the queries and sorted them in the decreasing order of occurrence. The distribution with the vertical axis, on log scale, indicating the number of occurrences and the horizontal axis indicating the number of unique root keywords can be seen in Fig. 7. There were 28,659 root keywords (i.e. stemmed words) in total and of these only 4797 root keywords had more than 10 occurrences. This distribution mimics the "Long Tail" behaviour of web search queries, as discussed in [10], where each user is considered a bit eccentric and is expected to send both common queries (all root keywords with more than 10 occurrences in the distribution) and a few unique/unusual queries (at least one keyword with less than 10 occurrences in the distribution). These unique/unusual keywords are what we think might be contributing for query identification.

Let us consider the SVM classifier in the second experiment and assess how these unique keywords influence the classifier decision. By converting the query strings to word vector, we are mapping the search queries to points in the multi dimensional space and the SVM classifier is trying to find a maximum margin hyperplane that helps in separating the user query points from the rest. The test queries, which are closer to the user training queries in the multidimensional space are labelled as user queries. If the user has unique/unusual keywords in his training data, the presence of these unique/unusual keywords in the test queries brings them closer to the user query points in the multidimensional space, thereby getting the query identified as a user query. In this way having unique/unusual query words helps in easy query identification. The more the number of such unique/unusual queries, the better are the chances for obtaining good accuracies – but this alone does not guarantee it. Even if there are unique/unusual queries in the user training set, there could be more occurrences of such query words in the 'other user' training set, there by preventing the query from getting classified as a user query. For example, lets say "culinary" is a unique keyword occurring in the training set of user of interest A. This word being unique only guarantees that it does not occur more than 10 times in the 1000 user query set. However, this word can occur 7 times in the other users' training set and 2 times in A's training set, resulting in the query not being classified as A's query owing to the majority.

7.2. *Rules governing the classifier*

After manually comparing the predictions made by the classifier, we came up with three possible explanations of how the classifier might be predicting these user of interest's queries. There could be other better explanations than the ones we propose. Let us call these explanations, as the three rules determining the label of a test query:

(1) If the exact test query is repeated in the user of interest's training set, then the query is identified as the user of interest's query. Say we came across the test query "how to kill a spider" and the exact query repeated in the user of interest A's training set, then we can be sure that this test query was sent by user A. This is trivial.
(2) If a subset of keywords in the test query occur as a full fledged query in the user of interest's train set, then the query is identified as the user of interest's query. For example, if our test query is "bake a fruit cake" and if we have "bake a cake" as a query in the A's training set, then the test query shall be identified as A's query.
(3) If the test query can be obtained by combining terms from queries in the user of interest's train set, then the query is classified as a user of interest's query. Say our test query is "California bike tours" and if there are "famous bike tours" and "New York to California" queries in the A's training set, then the test query is classified as A's query.

Similar rules also apply to the identification of non user queries based on terms in the non user (or 'Other') training set. If a query satisfies one rule with user of interest's training set, and one rule with non user training set, then based on the parameters that we chose for the classifier and the number of times these query terms appear in each of the training sets, the query will be classified. The presence of unique query words in the training set and test set, would help in easing this decision process.

7.3. Influence of a time gap

The web content that a user is interested in varies with time. The search queries, which are closer in time are better related than the ones sent long ago. In our scenario, we had the test data set immediately follow the user training set. This might be one reason why we were able to achieve considerable true positive rates. A time gap between the test and the training data sets might make it harder to de-anonymize the data, because the common content between the test and the train sets decreases with time. However, prior research [33] shows that users tend to pose exact same queries over and over, as it is easier compared to remembering the url of the search result, or more efficient than performing an internal search within the website. This behaviour is described as "bookmarking" [33]. For a considerable long period, these queries do not change and this helps the machine learning approach in identifying at least a small fraction of the user queries. We tried identifying the percentage of same queries repeated in the test and train sets (assumed to be bookmark queries), for two AOL users with IDs 20894930 and 67910. The percentage of bookmark queries were less than 6%, but the machine learning true positive rates were higher than 58%, in fact higher than 90% for user ID 20894930. Thus at least a small fraction of user queries could still be identified, even when there is time gap between the test and training data sets.

7.4. Reasons behind decrease in true positive rate

We understood how the accuracies are obtained, but the question that is remaining is – *"In the second experiment, why do the accuracies decrease when we increase the value of N, the number of users using web search over anonymizing network?"*. Following the three rules mentioned above, we determine the label of a test query based on the number of occurrences of similar query words in the training sets. The more the occurrences of such words in the user of interest's training set, the more are the chances for it to be identified as a user of interest's query and vice versa. With the increase in the value of N, the size of the other users training set greatly increases, improving the chances of occurrence of these test query words within. This decreases the probability of classifying the query as a user of interest's query. Here is one example that we came across for one AOL user. "j c penney catalog" was a test query and there was no exact looking query, or a query formed by subset of its keywords in the particular user's training set. However, the words "j c penney" and "catalog" had occurred before in the particular user's train set, and hence it was labelled as the particular user's query when $N = 100$ (as per explanation 3 above). However as N increased to 300, this query was not labelled as the user of interest's query, since there were more occurrences of "j c penney" and "catalog" terms in the other users' query training set. In this way, depending on the occurrence of query terms in the training sets, the accuracies decrease as the value of N increases.

However, due to the “bookmarking” behavior discussed before, the classifiers will be able to identify at least a few of the user’s queries when we increase the number of other/noisy queries in the dataset. Adding more number of anonymizing network users would definitely degrade the classifier’s true positive rates, but due to user’s repeated queries, these true positive rates would still be larger than zero.

8. Conclusions

In this paper, we studied and tried to quantify the levels of privacy provided by query obfuscation tools and anonymizing networks. As a representative example of tools based on Query Obfuscation principle, we focussed on TrackMeNot and demonstrated that a search engine, equipped with only a short-term history of user’s search queries, can break the privacy guarantees of TMN by only utilizing off-the-shelf machine learning classifiers. More specifically, by treating a selected set of 60 users – from the publicly-available AOL search logs – as users of the TMN software, we showed that user queries can be identified with an average true positive rate of 48.88%, while the average TMN query false positive rate was only 0.02%.

We tried identifying a user’s queries from a pool of queries received by a search engine over an anonymizing network like Tor. We demonstrated that an adversarial search engine, equipped with only a short-term search history, can extract user of interest’s queries by utilizing only the query content and off-the-shelf machine learning classifiers. By treating a selected set of 60 users – from the publicly-available AOL search logs – as users of interest performing web search over an anonymizing network, we showed that their queries can be identified with 25.95% average true positive rate when $N = 100$, and with 18.95% average true positive rate when $N = 1000$ (where N is the size of user set performing web search over Tor). Though the average true positive rates are not very high, our results show that few users of interest can be identified with true positive rates as high as 80–98%, even when the value of $N = 1000$. We even tried to identify the reasons behind how and why a query gets classified as a user query, and answered why the true positive rates tend to decrease as the number of users using the anonymization service increase. Our results, therefore, cast serious doubt on the effectiveness of anonymizing web search queries by means of anonymizing networks and Query obfuscation tools.

One of the strengths of our attacks is that they only use minimal information (query content) for identification of users’ queries, and use off-the-shelf classification techniques, while still being reasonably successful. Under realistic conditions, it would certainly be possible to improve our attacks by taking into account other information that would be available to the search engine under normal circumstances. For instance, exact query timestamps may very well be a useful attribute. Similarly in the case of Tor, exit node IP addresses are also likely to improve the accuracies. Say, the exit nodes vary for every t minutes and a query coming from an exit node was identified to be coming from user A, then another related query coming from the same exit node within t minute time frame, has more chances to be associated to the same user A. In addition, novel classification mechanisms can be designed specifically tailored to this query identification problem. Finally, a search engine can build better classifiers by training them on long-term (longer than 2 months) search histories of the users. We believe these additions might provide significant improvements in correct classification of the queries and further reduce false positive rates. By utilizing the geographical locality information accompanying the queries and the users [12] (place names and details pertaining to certain localities) and using contextual information for query classification [4], we plan to further improve the results. We defer these items as an interesting avenue for future research.

Acknowledgments

We are grateful to PETS'10 and ASIACCS'11 anonymous reviewers for their insightful feedback. We also thank Lisa Hellerstein for discussion on machine learning classifiers and her helpful comments on our work, and the developers of TMN – Helen Nissenbaum and Vincent Toubiana – for their useful suggestions on a previous draft of the paper.

References

[1] AOL search log mirrors, http://www.gregsadetsky.com/aol-data/.

[2] S. Argamon, M. Šarić and S.S. Stein, Style mining of electronic messages for multiple authorship discrimination: first results, in: *KDD'03: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

[3] M. Barbaro and T.J. Zeller, A face is exposed for AOL searcher No. 4417749, *The New York Times*, August 09, 2006.

[4] H. Cao, D.H. Hu, D. Shen, D. Jiang, J.-T. Sun, E. Chen and Q. Yang, Context-aware query classification, in *SIGIR'09: Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, New York, NY, USA, 2009, pp. 3–10.

[5] C. Castelluccia, E. De Cristofaro and D. Perito, Private information disclosure from web searches, in: *Proceedings of the 10th International Conference on Privacy Enhancing Technologies, PETS'10*, Springer-Verlag, Berlin/Heidelberg, 2010, pp. 38–55.

[6] R. Chow and P. Golle, Faking contextual data for fun, profit, and privacy, in: *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2009.

[7] O. de Vel, A. Anderson, M. Corney and G. Mohay, Mining e-mail content for author identification forensics, *SIGMOD Rec.* **30**(4) (2001), 55–64.

[8] R. Dingledine, N. Mathewson and P. Syverson, Tor: the second-generation onion router, in: *Proceedings of the 13th Conference on USENIX Security Symposium, SSYM'04*, Vol. 13, USENIX Association, Berkeley, CA, USA, 2004, p. 21.

[9] R. Evans, Clustering for classification, Master's thesis, Computer Science, University of Waikato, 2007, available at: http://adt.waikato.ac.nz/uploads/approved/adt-uow20070730.091151/public/02whole.pdf.

[10] S. Goel, A. Broder, E. Gabrilovich and B. Pang, Anatomy of the long tail: ordinary people with extraordinary tastes, in: *WSDM'10: Proceedings of the Third ACM International Conference on Web Search and Data Mining*, ACM, New York, NY, USA, 2010, pp. 201–210.

[11] Google resists U.S. subpoena of search data, 2006, available at: http://www.nytimes.com/2006/01/20/technology/20google.html?_r=1.

[12] L. Gravano, V. Hatzivassiloglou and R. Lichtenstein, Categorizing web queries according to geographical locality, in: *CIKM'03: Proceedings of the Twelfth International Conference on Information and Knowledge Management*, ACM, New York, NY, USA, 2003, pp. 325–333.

[13] S. Hansell, Marketers trace paths users leave on internet, *The New York Times*, September 15 2006, available at: http://www.nytimes.com/2006/08/15/technology/15search.html?pagewanted=all&_r=0.

[14] M. Hearst, B. Schvlkopf, S. Dumais, E. Osuna and J. Platt, Trends and controversies – support vector machines, *IEEE Intelligent Systems* **13**(4) (1998), 18–28.

[15] D. Howe and H. Nissenbaum, TrackMeNot: Resisting surveillance in web search, in: *On the Identity Trail: Privacy, Anonymity and Identity in a Networked Society*, I. Kerr, C. Lucock and V. Steeves, eds, 2008.

[16] C.-W. Hsu, C.-C. Chang and C.-J. Lin, A practical guide to support vector classification, available at: http://www.csie.ntu.edu.tw/∼cjlin/papers/guide/guide.pdf.

[17] R. Jones, R. Kumar, B. Pang and A. Tomkins, "i know what you did last summer": query logs and user privacy, in: *CIKM'07: Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, 2007.

[18] R. Jones, R. Kumar, B. Pang and A. Tomkins, Vanity fair: privacy in querylog bundles, in: *CIKM'08: Proceeding of the 17th ACM Conference on Information and Knowledge Management*, 2008.

[19] M. Koppel, J. Schler, S. Argamon and E. Messeri, Authorship attribution with thousands of candidate authors, in: *SIGIR'06: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2006.

[20] E. Kushilevitz and R. Ostrovsky, Replication is not needed: single database, computationally-private information retrieval, in: *FOCS'97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science, FOCS'97*, 1997.

[21] F. Mosteller and D.L. Wallace, *Inference and Disputed Authorship: The Federalist*, Addison-Wesley, Reading, MA, 1964.

[22] J. Novak, P. Raghavan and A. Tomkins, Anti-aliasing on the web, in: *WWW'04: Proceedings of the 13th International Conference on World Wide Web*, ACM, New York, NY, USA, 2004, pp. 30–39.

[23] S.T. Peddinti and N. Saxena, On the privacy of web search based on query obfuscation: a case study of TrackMeNot, in: *Proceedings of the 10th International Conference on Privacy Enhancing Technologies, PETS'10*, Springer-Verlag, Berlin/Heidelberg, 2010, pp. 19–37.

[24] S.T. Peddinti and N. Saxena, On the effectiveness of anonymizing networks for web search privacy, in: *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ASIACCS'11, ACM*, New York, NY, USA, 2011, pp. 483–489.

[25] PlanetLab: An open platform for developing, deploying, and accessing planetary-scale services, available at: http://www.planet-lab.org/.

[26] R. Rifkin, Multiclass classification, 06 March 2006, 9.520 Class 08, available at: http://ocw.mit.edu/NR/rdonlyres/2CA61B6A-9C0D-4E8A-AEE3-F4B1BFD96AF0/0/lec8.pdf.

[27] R. Rifkin and A. Klautau, In defense of one-vs-all classification, *J. Mach. Learn. Res.* **5** (2004), 101–141.

[28] F. Saint-Jean, A. Johnson, D. Boneh and J. Feigenbaum, Private web search, in: *WPES'07: Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*, 2007.

[29] B. Schneier, Schneier on Security: TrackMeNot, available at: http://www.schneier.com/blog/archives/2006/08/trackmenot_1.html, 2006.

[30] Scroogle, http://scroogle.org/.

[31] C. Soghoian, The problem of anonymous vanity searches, *SSRN eLibrary*, 2007, available at: http://ssrn.com/abstract=953673.

[32] B. Tancer, *Click: What Millions of People Are Doing Online and Why It Matters*, Hyperion Publishers, 2008.

[33] J. Teevan and E. Adar, Information re-retrieval: repeat queries in Yahoo's logs, in: *SIGIR'07: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, New York, NY, USA, 2007, pp. 151–158.

[34] TrackMeNot: Browser Plugin, available at: http://www.mrl.nyu.edu/∼dhowe/trackmenot/.

[35] TrackMeNot 0.6.721, 2010, available at: https://addons.mozilla.org/en-US/firefox/addon/trackmenot/.

[36] I.H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd edn, Morgan Kaufmann Series in Data Management Systems, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2005.

[37] S. Ye, S.F. Wu, R. Pandey and H. Chen, Noise injection for search privacy protection, in: *International Conference on Computational Science and Engineering (CSE)*, (2009), pp. 1–8.

[38] R. Zheng, J. Li, H. Chen and Z. Huang, A framework for authorship identification of online messages: Writing-style features and classification techniques, *J. Am. Soc. Inf. Sci. Technol.* **57**(3) (2006), 378–393.