Vector representation of Internet Domain Names using a Word Embedding technique

Waldemar López*, Jorge Merlino* and Pablo Rodríguez-Bocca*
*Instituto de Computación
Facultad de Ingeniería, Universidad de la República.
Julio Herrera y Reissig 565, 11300, Montevideo, Uruguay.
Email: walopez,jmerlino,prbocca@fing.edu.uy

***Abstract*—Word embeddings is a well known set of techniques widely used in natural language processing (NLP), and `word2vec` is a computationally-efficient predictive model to learn such embeddings. This paper explores the use of word embeddings in a new scenario. We create a vector representation of Internet Domain Names (DNS) by taking the core ideas from NLP techniques and applying them to real anonymized DNS log queries from a large Internet Service Provider (ISP). Our main objective is to find semantically similar domains only using information of DNS queries without any other previous knowledge about the content of those domains. We use the `word2vec` unsupervised learning algorithm with a Skip-Gram model to create the embeddings. And we validate the quality of our results by expert visual inspection of similarities, and by comparing them with a third party source, namely, similar sites service offered by Alexa Internet, Inc.**

***Index Terms*—DNS, Word embeddings, word2vec, Tensorflow, Semantic Similarity, Natural Language Processing.**

I. Introduction

The amount of time that people spend online has systematically increased in recent years [1]. To understand the behavior of users in online content consumption is the focus of several research. It has large implications to network design, online business, and media industry [2]. Many studies apply machine learning to historical patterns of network resource consumption in order to extract knowledge about online customer behavior [3], [4]. Due to the inaccessibility of the information, few of these studies use the traces of DNS queries for this purpose. The few exceptions are [5], [6], [7], [8], [9], where none of them has as main objective to extract knowledge about the semantic nature of the queried domains.

There are several Web tools that try to estimate the semantic similarity between sites[1]. For example to provide web site owners the possibility to find competitors for the same target audience, and to advice end-users on alternative providers for the same content. As a novel application, in this work we apply word embeddings to Internet Domain Names traces in order to find semantically similar domains without extra knowledge about domains than its usage.

[1]http://www.alexa.com/find-similar-sites/, https://www.similarweb.com/, http://www.similarsitesearch.com/, Google Similar Pages, etc.

Word embeddings are a set of techniques that map word or phrases of a vocabulary to vectors of real numbers. The idea is that semantically similar words will be assigned nearby vectors so that the model can leverage information learned about some words to other similar words. This is equivalent to transform a discrete space of atomic symbols with one dimension per word to a continuous vector space with lower dimension. This is a much more useful and tractable representation of text. Word embeddings are typically applied to texts in the context of natural language processing in tasks such as syntactic parsing, language modeling, and predicting semantically related words [10], [11]. In natural language the context of a word is determined by the words used right after and before it in a phrase, in our work we consider the domain names queried by the same IP address after and before some domain name as the context for this domain (i.e. the trace of DNS queries).

For this work we obtained DNS recursive server logs from a large Internet Service Provider (ISP) with anonymized IP addresses. These logs contain each query resolved by a farm of servers. Each line of log indicates the time, the anonymized IP address of the client, the queried domain, and the type of DNS query (A, AAAA, MX, etc.). To create the word embedding for the domains we use the `word2vec` [12] model implemented using the Python Tensorflow library [13]. In order to evaluate the quality of our results, we explore two alternatives: to have an expert visual inspection of similarities, and to use the mean average precision (MAP) metric to measure the mismatch of similarities between our results and the results obtained from a third party source, namely, similar sites service offered by Alexa Internet, Inc. Using this technique we show that the created embedding effectively embeds semantically similar domains nearby each other and therefore it could be used to build a recommender system, to predict the domains that will be queried in the near future in order to, for example, detect traffic anomalies or apply some cache mechanism.

The rest of the paper is organized as follows: Section II introduces to current techniques to find semantically similar domains; Section III explains the Domain Name System on the Internet, its particularities for our problem, and the data used in this work. Section IV introduces the basis of the `word2vec` algorithm, its applications to other contexts, and how we use it to solve our problem. Section V presents experiments and results of applying word embedding to find semantically similar domains in our dataset. Finally, Section VI discusses future work and present main conclusions.

II. Semantic Similarity Between Internet Domain Names

The term *semantic similarity* is usually employed over sets of words (texts) to measure the likeness of their meaning. In the context of this paper we will define the semantic similarity between domain names, as the distance between their semantic content. For example, two news providers are expected to be semantically similar, as well as two retail stores, and if they offer the same category of products should be closer.

As previously mentioned, there are several web tools that try to estimate this similarity. One of the most important in this area is *Alexa*[2]. This site obtains traffic estimates based on data from multiple web browser extensions and also from sites that install a script that harvests traffic information for them. As we will explain later, in this work we use the Alexa API service in order to evaluate the quality of our similarity estimation. Other example is *SimilarWeb*[3], which shows many statistics about web sites including similarity. They claim that obtain data from four sources: a) a pool of monitored user devices, b) data obtained directly from ISPs, c) web crawlers that scan websites, and d) direct measurement from websites and mobile apps connected to them. With respect to the similarity they claim their system is based on website structure, link analysis, user surfing behavior and user rankings. Also there is a Google Chrome extension called *Google Similar Pages*, provided by Google Inc., that also shows a few semantically similar pages to the one being visited. Google does not disclose the data sources but it is reasonable to think that they can use a mix like *SimilarWeb* to provide this functionality.

A. Alexa Similarity Service

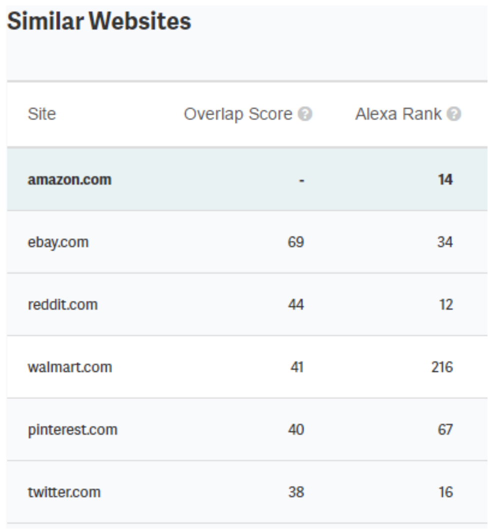
One of the main problems that we address is to evaluate the quality of the semantic relations that we discover between domains in our unsupervised process. In order to achieve this objective, in this work we use the API service called *find similar sites*, a feature offered by Alexa in its *Audience Overlap Tool*. This service allows to query for a specific domain name in order to know possible competitors based on considering similar sites. In its paid version, this service retrieves the top 100 most similar sites for a specific input domain name.

For our research we built a tool that given the list of the top used domains in our vocabulary, it retrieves the similar sites from Alexa. More precisely, we queried for 5092 top domains in our vocabulary, for which we were able to find a matching in Alexa for 2085 of those domains (55%). The reason why not all of our domains were found in Alexa could be one of many causes, for instance that not all of them are Internet Web sites or because our dataset contains domains that do not exist any more, among others reasons.

[2]http://www.alexa.com/find-similar-sites/

[3]http://www.similarsitesearch.com/

Figure 1 shows a web view of the service that displays the top most 5 similar sites for `amazon.com`. The result list has a default order by the *Overlap Score* value. The meaning of this value according to Alexa is: *the relative level of visitor (audience) overlap between any site and the target site. A site with a higher score shows higher audience overlap than a site with a lower score*. This order is important when considering the evaluation mechanism. This mechanism is described in more detail in Section V.


Similar Websites

Site	Overlap Score	Alexa Rank
amazon.com	**-**	**14**
ebay.com	69	34
reddit.com	44	12
walmart.com	41	216
pinterest.com	40	67
twitter.com	38	16

Fig. 1. Similar Sites for Amazon.com according to Alexa. Retrieval date: March, 2017.

Our approach to find the semantic similarity between domain names is to build a mathematical vector representation of domains based on DNS traces data, and compute the similarity of two domains as the cosine distance between the two vectors that represent them. In next sections we will explain why DNS traces include this valuable information, and how to obtain it.

III. Domain Names on Internet (DNS)

The DNS (Domain Name System) [14], [15] is a decentralized service for naming computers and other resources in a network. Each of these resources is assigned a domain name which is a hierarchical string defining a node in a tree structure. The domain name is formed by the labels of the tree nodes traversed from the node to the root separated by points. For example the domain name *example.com* has *com* as its top level domain, and *www.example.com* as a sub-domain. The DNS system is decentralized as the responsibility for resolving each component of the domain name is delegated to a different name server thus avoiding a single central database and a single point of failure. Also there could be several domain servers to resolve the same domain providing thus a fault tolerant configuration. This system has been in use in the Internet since 1985 and is one of the most essential services in the network. In the Internet the most fundamental service provided by DNS is to translate easily memorized domain names to IP addresses. Each domain can have different sub-domains with different type. The most common types are: A and AAAA that correspond to IPv4 and IPv6 address translation respectively, MX that define SMTP mail exchangers, NS that define other name servers, CNAME that define domain name aliases and PTR that are used for reverse DNS queries (for example to query the domain name for a given IP address).

Each domain has at least one *authoritative* name server that contains the original information about the domain and its subdomains. An authoritative name server only gives answers to DNS queries from data that has been configured on that server. Potentially, an authoritative name server could delegate a subdomain to other authoritative servers building a hierarchical tree of authorities. On the top of the hierarchy are the root DNS servers.

On the other hand, there are *recursive* DNS servers that are capable of resolving queries about domain names, by means of recursive queries to possibly several authoritative name servers starting from the root servers. These servers usually cache the results obtained to increase efficiency. The duration of the cached data depends on the TTL (time to live) configuration of each domain at the authoritative servers.

The client components of the DNS system are called DNS *resolvers*. Resolvers usually query recursive servers to find an answer, and also cache the result during the corresponding domain TTL.

DNS is critical for Internet operation. It is a large and complex system. Here we include a minimum description with the aim of offering a self-contained reading. You can read [16] for a deep explanation.

A. Our Data: DNS traces

The data used for this research was provided by a large Internet Service Provider (ISP), with millions of Internet users, who use the ISP's recursive name servers. For this study we analyze the queries log in these name servers, at 11 different days collected from December 2012 to March 2013. Each line of log data shows the date and time of the query, the anonymized IP address of the client, the queried domain and the type of DNS query requested by the client. Figure 2 shows an example of some DNS queries from an anonymized IP address.

Note that the amount of data collected in just one day is extremely large. With more than 3.5 billions of queries, 58 millions of unique domain names, and 550 thousands of unique IPs. Data across different days are pretty similar, with a little increment with time, as shown in Figure 3(a) for 3 different days. A and AAAA record types correspond to more than 90% of the queries in an average day (Figure 3(b)), and other types are less relevant when studying user Web navigation habits, therefore we filtered the data to keep only these kind of queries (A and AAAA). For these record types we can see a minimum between 5 and 6 a.m. and maximum between 8 and 10 p.m. approx as it is shown in Figures 4(a) and Figure 4(b) for the same three days.

Formally, the DNS log is a sequence of $< \mathrm{IP}_i, d_j, t_k >$, where the client IP_i queried the domain d_j at time t_k. In the log there is a set of unique (anonymized) IPs representing each client, $c \in C$, for each client we have a DNS trace, $t_c \in T$, that is a sequence $t_c = \{< d_1, t_1 >, < d_2, t_2 >, \ldots\}$. Our problem is to learn a similarity function $sim(d_i, d_j)$ between any two domains d_i and d_j, using only the set of traces T. This problem is very similar to find semantically similar words, where the trace of words are the phrases in texts. Following previous results in natural language processing, word embedding techniques like `word2vec` are shown effective for these kind of problems.

But DNS traces are not exactly a sequence of sites visited by clients, and this should have an impact in the expected result. There are a few limitations with the data obtained from the DNS *recursive* servers in this way. To take into account:

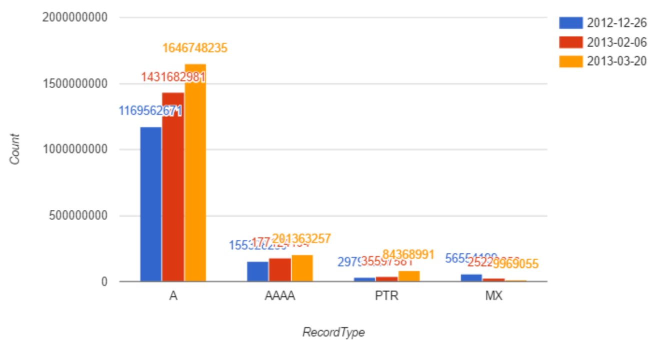
- First of all, DNS *resolvers* clients cache the requests so we do not have information about how often a domain is visited. We only have one request after the domain cache times out and then it is cached again. Therefore, the DNS traces are not a good source to measure the period of usage of a domain, just the first access.
- Also, NAT enabled gateways hide the activity of many users behind a single public IP address. This is very common in business and residential connections, but not in mobile services. It means that a trace can mix multiple clients, in some cases thousands of them. Usually the ISP assigns disjoint range of IPs to each kind of service, therefore it is possible to separate business, residential and mobile traces.
- It is a good practice of ISPs to assign a dynamic IP address to each service, where the client needs to be reconnected after a fixed period (for example 12 hours) and a new IP is assigned. Therefore, an IP identifies a service for a period, and a trace can be the concatenation of session periods from several services.
- It is a good practice of Content Providers to use several sub-domains in order to provide their content, moreover they usually use external services to provide part of their content (for example they use Content Delivery Networks to provide static and video content). Therefore, when a client access to a service, it usually adds several domains to his trace (sub-domains of the provider and external sources). This is very consistent between different clients that access to the same service, but it is not a simple task to extract the knowledge of the service used by the client from this trace.
- In a similar way, there are applications in the client device that generates traffic, and therefore queries in the trace, without an action of the user (for example antivirus, email clients, etc.). These queries are mixed in the traces and can help to find similarity between traces, hence the similarity between domains.

In order to mitigate the impact of these limitations in our procedure, we preprocess the DNS traces in the following way:

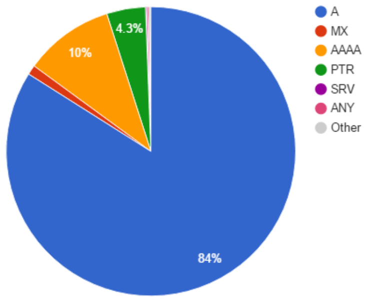
- We use a subset of the data, with queries done by IPs that belong to some known ranges corresponding to

```
21-Mar-2013 06:06:47.216 client <anonymized ip>: query: apps.facebook.com IN A
21-Mar-2013 06:06:48.149 client <anonymized ip>: query: profile.ak.fbcdn.net IN A
21-Mar-2013 06:06:53.215 client <anonymized ip>: query: pixel.facebook.com IN A
21-Mar-2013 06:08:10.728 client <anonymized ip>: query: jacaranda.ceibal.edu.uy IN A
```

Fig. 2. Example of some DNS queries from an anonymized IP address one day.

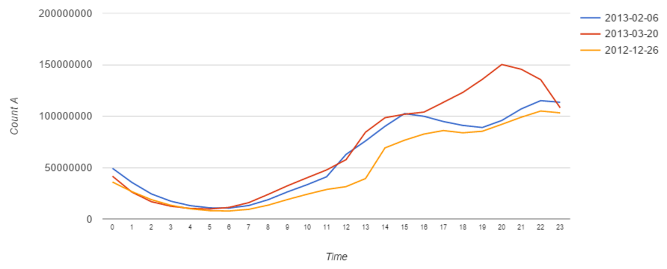


(a) Number of DNS queries for main records types on 3 different days.

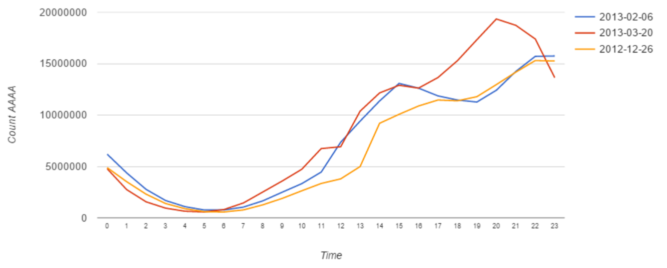


(b) Distribution of DNS queries types on March 20 of 2013.

Fig. 3. Main statistics for DNS record types.



(a) Number of type A DNS queries per hour on 3 different days.



(b) Number of type AAAA DNS queries per hour on 3 different days.

Fig. 4. DNS queries per hour.

residential or mobile services[4]. This is a simplification in order to minimize the amount of devices connected with the same IP and in order to study true user's usage patterns.

- Other simplification in our study has been the reduction of heterogeneity coming from the sub-domains. We truncate the last labels of a domain in the following way: if the top label is a country code (ccTLD) then the domain is truncated to the first 3 levels, else (non ccTLD) the domain is truncated to the first 2 levels.
- We identified top domains like `google`, `facebook`, `youtube`, `whatsapp`, `skype`, `bing`, `yahoo`, `root-servers`, `akamaid`, `verisign` (among others) that are queried all the time in any context, and do not give us any relevant value. These domains are included in a fixed *black-list*, and excluded when building the input document for the vector representation training. The idea behind this is similar to the one used when processing natural language data text, that there are a set of words (common or stop words) like *the*, *is*, *at*, *which* (among others) that are filtered previously because are not really important.
- A second set of domains are added to the *black-list* and are those that despite not being top domains are domains that most of the time are requested automatically (the list of resources after accessing a web page for instance) and not by a user. For example external sources used by a Content Provider. In order to identify these domains we apply the following empirical rule: domains that 90% of the time or more are queried immediately after a previous domain (3 seconds window) are added to the *black-list* for this reason.

[4]Even if the client IPs are anonymized, we know the type of service associated with each address.

After preprocessing the data we obtain a depurated set of DNS traces, that we will use as input in word embedding training process in order to obtain our vector representation of domains.

IV. Word Embedding Techniques

In this section, we present our method to create a vector representation of domains, but previously we will introduce the basis of word embedding techniques and its applications.

A. `Word2vec` Basis

`Word2vec` is a very computationally-efficient predictive model for learning word embeddings [12], [17], [18]. There are two main variants known as Continuous Bag-of-Words

(CBOW) and Skip-Gram. These models are similar but CBOW is used to predict target words from source context words, while the Skip-Gram model does the inverse and predicts source context words from the target words. Both have shown to be useful for extracting similarity of words in a text. In our case we used the Skip-Gram model exclusively so we will concentrate on that and skip the CBOW model.

As mentioned, the objective of the Skip-Gram model is to create an embedding useful to predict context words from source words. Thus, given a sequence of training words as $w_1, w_2, \ldots, w_T$, the Skip-Gram model objective is to maximize the average log-likelihood:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

where c is the size of the context training window. The number of possible training samples increases with c. The $p(w_{t+j}|w_t)$ probability function is defined as the softmax function:

$$p(w_i|w_j) = \frac{\exp\left[\text{score}(w_i, w_j)\right]}{\sum_{w=1}^{W}\exp\left[\text{score}(w, w_j)\right]} \tag{1}$$

where W is the size of the vocabulary. The score is a measure of the compatibility of w_i and w_j. Usually the inner product of their vector representation is used. This calculation is usually impractical as the cost of calculating $\nabla \log p(w_i|w_j)$ is proportional to W which is usually large (order 10^4 in our case). To overcome this problem we use Noise Contrastive Estimation (NCE) which approximately maximizes the log of Eq.(1). The objective of NCE is:

$$\log \sigma({v'_{w_i}}^T v_{w_j}) + \sum_{w=1}^{k}\mathbb{E}_{w\sim P_n(w)}\left(\log \sigma({v'_w}^T v_{w_j})\right)$$

where v_i and v'_i are the embedding vectors of i after and before the maximization step, ${v'_i}^T v_j$ is their inner product and σ is the binary logistic regression function:

$$\sigma(x) = \frac{1}{1+e^{-x}}.$$

The idea is to differentiate the word w_i from draws from the noise distribution $P_n(w)$ using logistic regression, where there are k noise (negative) samples for each data sample.

Several variants of this simple algorithm were been developed in the last years. Please read [12] for a deep explanation of the algorithm. And read [19] as an example of extension to a paragraph vectorization method (instead of word vectorization). Extensions beyond natural language processing are presented below.

B. *Applications of* `word2vec` *Beyond Words*

`Word2vec` inspired the application of neural network models in other contexts. For example, in [20] the authors propose a method called `med2vec` to represent medical codes and visits into a vector space. In [21] is presented the `emoji2vec` method to create a vector representation of emojis. A closer work to our paper is present in [22], where the authors study how to create a vector model of mobile applications using the usage pattern of several clients. They extend the Continuous Bag-of-Words (CBOW) in order to take into account the time interval between the usage of successive apps. They call their procedure `app2vec`. To consider a weight about time into the context of a DNS trace could help to map the relative relevance of similar domains. We expect to try this variant in future research.

C. *Vector Representation of Domain Names*

As initial work, in this paper we apply the original Skip-Gram model. We focus our work in understanding DNS traces, and how to preprocess them in order to extract their value. In future work we expect to compare different models and different quality measures in order to improve the current result.

V. Experiments and Results

Considering the 11 days of available logs, we have a dataset of 40 billions of queries. In this work, we only use the logs of a day (March 21th, 2013) as input of the training process, with 3.5 billions of queries. From these queries, we obtain the DNS traces (grouping by IP), and we preprocess them (as explained in Section III-A) in order to generate the final input. The final input is a sequence of 53 millions of domains, where the unique domains are 1.4 million. As expected, there is a large variation of popularity between domains. Figure 5 and Figure 6 show the cumulative percentages of requests per domain, where domains are represented by popularity position (from left to right) on the x-axis. We can see that top 5 thousand domains accumulate 75% of the total traffic, top 798 domains accumulate the 60% and top 10 domains accumulate the 10%. Due to the logarithmic characteristic of the function, we are able to work with a reduced vocabulary but very representative of the total traffic. Therefore, to compute similarities, we worked with a vocabulary built from the top 40 thousand most popular domains, which represent 88% approximately of the total amount of the requests under study.

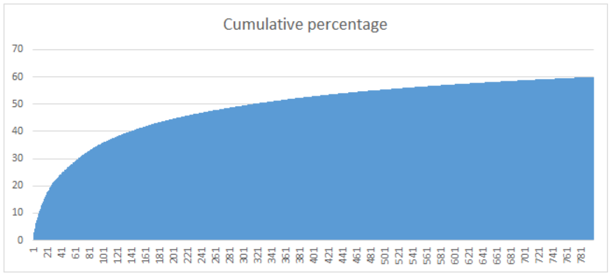


Fig. 5. 60% cumulative percentage of requests per domain.

The input is used in the training phase of the original Skip-Gram variant of `word2vec` model. We used an implementation of the standard `word2vec` algorithm in Tensorflow [13]. The core of the algorithm was not changed, we only modified the batch generation procedure. Instead of processing

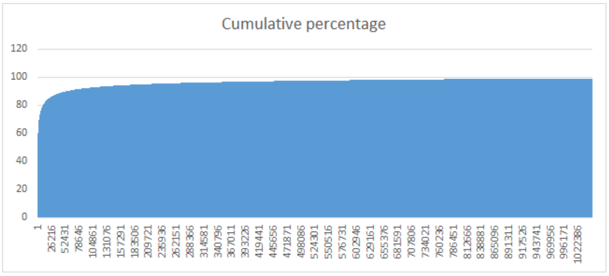


Fig. 6. 100% cumulative percentage of requests per domain.

a continuous flow of text, as in the standard case, we have logs consisting of a source IP address and a list of requested domain names. In this case the context of a domain can only be constructed by the domains requested by the same user in the same time slot. So the batch generation algorithm takes this into consideration to avoid mixing domains requested by different users. Main parameters in training are set as: a vocabulary size of 40000, an embedding size of 128 (dimension of the embedding vector), a learning rate of 0.5, and a skip window of 3 (words to consider left and right in the context). The server used is a Intel(R) Xeon(R) 4860@2.27GHz with 16 cores and 20 GiB RAM. The training takes 7 hours, and the reduction in loss of the process is shown in Figure 7.

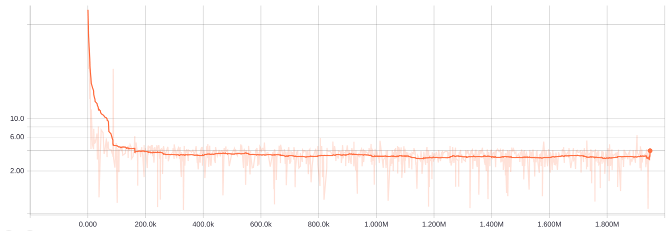


Fig. 7. Error in Skip-Gram training phase.

The output of the procedure is a vector representation in $\mathbb{R}^{128}$ of the top 40000 domains. From this vector representation we compute the cosine distance between domains, and we extract the top ten nearest domains for each one. We hope that this list of nearest domains will show semantic similarity properties. One of the main problems we need to address is to know how good are the semantic relations between domains that we discover. Namely, given the calculated set of nearest domains for a specific domain, we need to validate whether domains in this set are really semantically similar domains or not. In order to achieve this objective, in this paper we explore two alternatives: to have an expert visual inspection of similarities, and to measure the mismatch of similarities with a third party source.

A. *Visual Inspection: Analogies Through Vector Operations*

In the original `word2vec` paper [17], it was mentioned that the linear structure of the Skip-Gram model allows analogical reasoning using simple vector operations. For example the addition of vectors works as an AND logical function: the words near to the addition of two vectors will be words that are close to both original words. In our case, domains close to the addition of the vectors that represent two particular domains will be the ones that are accessed in conjunction with the added domains. Also it was shown that other analogies were possible where for example the vectors for words $king + man - woman = queen$. We show in Table I some simple addition analogies between domains, and in Table II some more complex analogies using addition and subtraction. In every case we show one of the 10 domains nearest to the resulting vector. Visual inspection verifies that our vector representation embed semantic relationship between domains.

B. *Mismatch of Similarities with a Third Party: Alexa*

As explained in Section II-A, we use the service called *find similar sites* by Alexa in order to retrieve the ordered top 100 most similar sites for a specific input domain name. From visual inspection on these lists, it is clear that only the first similar domains are relevant, followed by popular or generic domains. Therefore we will consider only the first k_{actual} domains as true similar domains for each domain (varying k_{actual}). We have this information for 2085 domains of our vocabulary (of 40000 domains). During the training phase, our evaluation procedure uses this external, but well trusted information from Alexa, to give some measure of quality. To be more accurate, periodically, every 10000 optimization steps and for all the 2085 domains that we were able to get information from Alexa's service, we compare the similarities between the actual Alexa's response and the predicted similarities found using our implementation.

The metric used in order to perform this comparison is the Mean Average Precision (MAP) [23]. This metric is chosen mainly because it takes into account the order in the similarity list. Some of the characteristics we were looking and that this metric has are:

- It assigns values between 0 and 1.
- If both similarities lists are identical, then its result is 1.
- If both similarities lists are disjoint, that is, if none of the domains in our calculated similarity list are present in Alexa's list, then its result is 0.
- Order in predicted list matters. Suppose we have computed a list of k similar domains, and the top half of this list also appears in the Alexa' service. Now, suppose we have a second similarity list where the matching with Alexa appears only for last half of the list. Despite for both lists we have matched $k/2$ domains with Alexa, we want that the metric when evaluating the first list to be higher than for the second list. That is because the $k/2$ matched domains for the first list are located above in the ordered list, that indicates that similarities for those domains are more accurate than the similarities for the $k/2$ matches in the second list.

A lack of the MAP metric is that the order of the actual list does not matter. The metric uses the Alexa's list as a set without order, where it is the same to have a coincidence with first or last domains in the set. We will mitigate this problem

TABLE I
LOGICAL ANALOGIES USING ADDITION.

v_1	v_2	$v_1 + v_2$
Ministry of tourism of Uruguay (turismo.gub.uy)	Ministry of tourism of Argentina (turismo.gov.ar)	Argentina's migration office (migraciones.gov.ar)
Ministry of tourism of Uruguay (turismo.gub.uy)	Ministry of tourism of Argentina (turismo.gov.ar)	Uruguayan travel site (pasaporteuruguay.com)
City government (montevideo.gub.uy)	Bus routes finder (montevideobus.com.uy)	City maps (mapred.com)
Airline (lan.com)	Hotel booking (booking.com)	Travel information (tripadvisor.com)

TABLE II
LOGICAL ANALOGIES USING ADDITION AND SUBTRACTION.

v_1	v_2	v_3	$v_1 + v_2 - v_3$
City government (montevideo.gub.uy)	Bus terminal (trescruces.com.uy)	Other city government (rocha.gub.uy)	Bus line connecting cities (copsa.com.uy)
City government (montevideo.gub.uy)	Bus terminal (trescruces.com.uy)	Other city government (rocha.gub.uy)	Post site (correo.com.uy)
Shopping mall in city A (puntashopping.com.uy)	City B government (montevideo.gub.uy)	City A government (maldonado.gub.uy)	Bus terminal in city B (trescruces.com.uy)
Soccer site (tenfield.com)	Soccer club A fan page (campeondelsiglo.com)	Other soccer site (ovacion.com.uy)	Soccer club B fan page (lavozdenacional.com)

by varying the k_{actual} value and evaluating the impact into the $MAP@k$ value.

Said this, we present the formula for the MAP metric. As explained, we have a set of domains, $d \in D$, for which we known the actual ordered list of top k_{actual} similar domains from Alexa, and an ordered list of predicted top k similar domains in our vector representation space. In order to compare these two lists of (actual and predicted) similar domains for a specific domain d, we use the Average Precision (AP) over all possible recall values:

$$AP@k|_d = \sum_{n=1}^{k} P(n)\Delta r(n),$$

where n is a position in the predicted list of similar domains of d, k is the size of the predicted list, $P(n)$ is the precision considering only the first n domains in the list, and $\Delta r(n)$ is the change in recall from domains $n-1$ to n. Precision and Recall metrics come from information retrieval theory, see their definition in [23].

With the Mean Average Precision (MAP), we summarize the comparison of actual and predicted lists for all available domains. Mean average precision for a set of similar domains is the mean of the average precision scores for each domain:

$$MAP@k = \frac{\sum_{d \in D} AP@k|_d}{D},$$

where $d \in D$ is a domain for which we known the top k_{actual} actual similar domains from Alexa, and an ordered list of predicted top k similar domains in our vector representation space. You can refer to [23] to see detailed explanation about mean average precision and the formula above.

We evaluate the $MAP@k$ metric for several values of k and k_{actual}, considering the first k_{actual} similarities from Alexa as a true, and the k similarities from our final vector representation as a prediction. Figure 8 shows the result. For example, the most similar domain in Alexa and the most similar domain in our vector representation are the same in 67.5% of the evaluated cases (i.e. $MAP@1 = 0.675$ with $k_{actual} = 1$). Interpretation of larger values of k and k_{actual} is not so trivial, for example $MAP@5 = 0.515$ with $k_{actual} = 3$ means that an a half of the 5 predicted similar domains are in the actual list of 3 domains, but this is weighted by the position in the list, being more important the coincidences in first places. From these results, it is clear that there are several coincidences between the two similarities sets, and again it verifies that our vector representation embed semantic relationship between domains.

VI. CONCLUSIONS

In this work we show that the use of the `word2vec` unsupervised learning algorithm is a viable alternative to find semantically similar domains using only information about DNS queries. To implement this, we use the `word2vec` algorithm with a Skip-Gram model using the Tensorflow library. To evaluate the quality of our results, we explore two alternatives: to have an expert visual inspection of similarities, and to use the mean average precision (MAP) metric to measure the mismatch of similarities between our predicted similarities and the similarities obtained from a third party source, the *find similar sites* service provided by Alexa. In both cases we find acceptable results that substantiate our approach. For example, the most similar domain in Alexa and the most

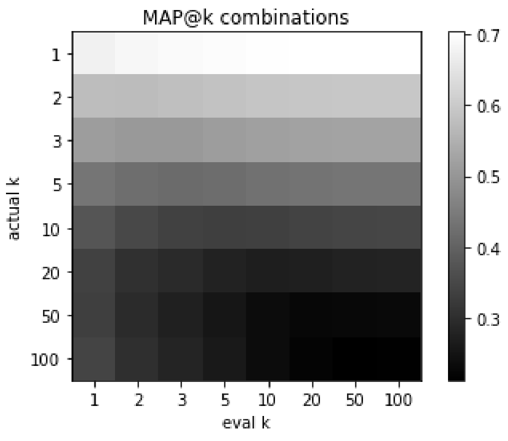


Fig. 8. MAP@k metric for several values of k and k_{actual}.

similar domain in our vector representation are the same in 67.5% of the evaluated cases.

This approach has several advantages over other prior art options, if DNS log data from the ISP can be obtained. For example, it does not require to obtain client level data via browser extensions or mobile apps reducing both development and deployment efforts. Moreover, due to the use of the `word2vec` algorithm and the Tensorflow library, we can scale to a very large volume of data points thus improving the algorithm accuracy. In our work, we use a dataset with 3.5 billions of DNS queries. Lastly the learning process is mostly unsupervised only possibly needing intervention to validate the obtained results.

We see as future work the possibility to modify the `word2vec` basic algorithm to include the time between DNS queries as a weight factor that could help to map the relative relevance of similar domains. This is the approach taken by the `app2vec` algorithm [22] for example. We also consider trying the CBOW model of `word2vec` to compare the obtained results. Lastly we see as important to try other quality measures of the obtained results to get a clearer view of the accuracy of the algorithm.

References

[1] Ofcom (communications regulator in the UK), "Adults' media use and attitudes. report 2016," 2016, Report available from www.ofcom.org.uk.

[2] Jun Yan, Ning Liu, Gang Wang, Wen Zhang, Yun Jiang, and Zheng Chen, "How much can behavioral targeting help online advertising?," in *Proceedings of the 18th International Conference on World Wide Web*, New York, NY, USA, 2009, WWW '09, pp. 261–270, ACM.

[3] Amr Ahmed, Yucheng Low, Mohamed Aly, Vanja Josifovski, and Alexander J. Smola, "Scalable distributed inference of dynamic user interests for behavioral targeting," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2011, KDD '11, pp. 114–122, ACM.

[4] H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica, "Ad click prediction: A view from the trenches," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2013, KDD '13, pp. 1222–1230, ACM.

[5] Peter B. Danzig, Katia Obraczka, and Anant Kumar, "An analysis of wide-area name server traffic: A study of the internet domain name system," *SIGCOMM Comput. Commun. Rev.*, vol. 22, no. 4, pp. 281–292, Oct. 1992.

[6] Hongyuan Cui, Jiajun Yang, Ying Liu, Zheng Zheng, and Kaichao Wu, "Data mining-based dns log analysis," *Annals of Data Science*, vol. 1, no. 3, pp. 311–323, 2014.

[7] Weizhang Ruan, Ying Liu, and Renliang Zhao, "Pattern discovery in dns query traffic," *Procedia Computer Science*, vol. 17, pp. 80 – 87, 2013.

[8] Qingnan Lai, Changling Zhou, Hao Ma, Zhen Wu, and Shiyang Chen, "Visualizing and characterizing dns lookup behaviors via log-mining.," *Neurocomputing*, vol. 169, no. Learning for Visual Semantic Understanding in Big Data, pp. 100 – 109, 2015.

[9] Mark E. Snyder, Ravi Sundaram, and Mayur Thakur, "Preprocessing dns log data for effective data mining," in *Proceedings of the 2009 IEEE International Conference on Communications*, Piscataway, NJ, USA, 2009, ICC'09, pp. 1366–1370, IEEE Press.

[10] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa, "Natural language processing (almost) from scratch," *CoRR*, vol. abs/1103.0398, 2011.

[11] Ronan Collobert and Jason Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th International Conference on Machine Learning*, New York, NY, USA, 2008, ICML '08, pp. 160–167, ACM.

[12] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.

[13] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, Software available from tensorflow.org.

[14] IETF Network Working Group, "Internet Domain Name System Standard: Domain names - concepts and facilities (RFC 1034)," 1987.

[15] IETF Network Working Group, "Internet Domain Name System Standard: Domain names - implementation and specification (RFC 1035)," 1987.

[16] Albitz P. and Liu C., *DNS and BIND, 4th Edition.*, O'Reilly, New York, Apr. 2001.

[17] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean, "Distributed representations of words and phrases and their compositionality," *CoRR*, vol. abs/1310.4546, 2013.

[18] Yoav Goldberg and Omer Levy, "word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method," *CoRR*, vol. abs/1402.3722, 2014.

[19] Quoc V. Le and Tomas Mikolov, "Distributed representations of sentences and documents," *CoRR*, vol. abs/1405.4053, 2014.

[20] Edward Choi, Mohammad Taha Bahadori, Elizabeth Searles, Catherine Coffey, and Jimeng Sun, "Multi-layer representation learning for medical concepts," *CoRR*, vol. abs/1602.05568, 2016.

[21] Ben Eisner, Tim Rocktäschel, Isabelle Augenstein, Matko Bosnjak, and Sebastian Riedel, "emoji2vec: Learning emoji representations from their description," *CoRR*, vol. abs/1609.08359, 2016.

[22] Q. Ma, S. Muthukrishnan, and W. Simpson, "App2vec: Vector modeling of mobile apps and applications," in *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, Aug 2016, pp. 599–606.

[23] Stephen Robertson, "A new interpretation of average precision," in *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, NY, USA, 2008, SIGIR'08, pp. 689–690, ACM.