בס"ד

URL Classification

By Simcha (323104562) & Sari (324832914)

Abstract

In the final assignment in this course, we chose to do Phishing URL Classification. The reason is that the topic is not heavy at all, and gives us the opportunity to play with the AI tools more, from all directions. After reading many (too many) articles, we came to the conclusion that most of them lack the element of language. What is the language of the victim and will a system optimized for one language (English for example) distinguish well for URLs from France, Russia or Turkey?

And so we decided to focus on analyzing elements in the URL that are directly related to the language in which it was written. Therefore, the system we implemented can be extended to other languages. We worked on Turkish and English, with an emphasis on extensive pre-processing for the purpose of extracting the files. The article we worked with gave us the general idea which we implemented and improved (by playing with the model).

Introduction

The phenomenon of phishing is widespread. And the scenario is simple - a victim receives a link and without thinking clicks on it. Nowadays people are (slightly) more careful, and therefore the attackers are also smarter. That is, the attackers will try to fool their victim by sending a URL that looks legitimate by spoofing techniques. For example: URL of faceb00k instead of facebook. As the builders of the protection system, we would of course like there to be no false negatives at all so as not to fall into a malicious link. But, we would like it not to increase the false positive too much, as we would not want to miss legitimate links that are sent to us. Ultimately, customers may complain that the system filters out legitimate links from them.

Related work

https://eprints.kingston.ac.uk/id/eprint/41980/1/Weedon-M-41980-AAM.pdf - Statically parses immediately accessible data, such as lengths and presence of special characters. Does not contain analysis elements by language (such as connected words, gibberish words like we have...) Of course we took some features from here as well.

https://scholar.google.co.il/scholar_url?url=https://infoscience.epfl.ch/record/136823/files/Topic_www_09.pdf&hl=iw&sa=X&ei=w2jeZbvKMpiCy9YPqICOgAg&scisig=AFWwaebolaaLoJJoSFWsHFhUBPac&oi=scholarr - Trying to identify known websites and therefore determine whether the URL is legitimate or not. The problem with this is that there are too many sites. But, it is probably also possible to take from this map the understanding that it is possible to consider parameters such as "brand names" within URLs.

https://www.sciencedirect.com/science/article/abs/pii/S0957417418306067 - The most beautiful article we found, conceptually describes how to perform a pre-processing process

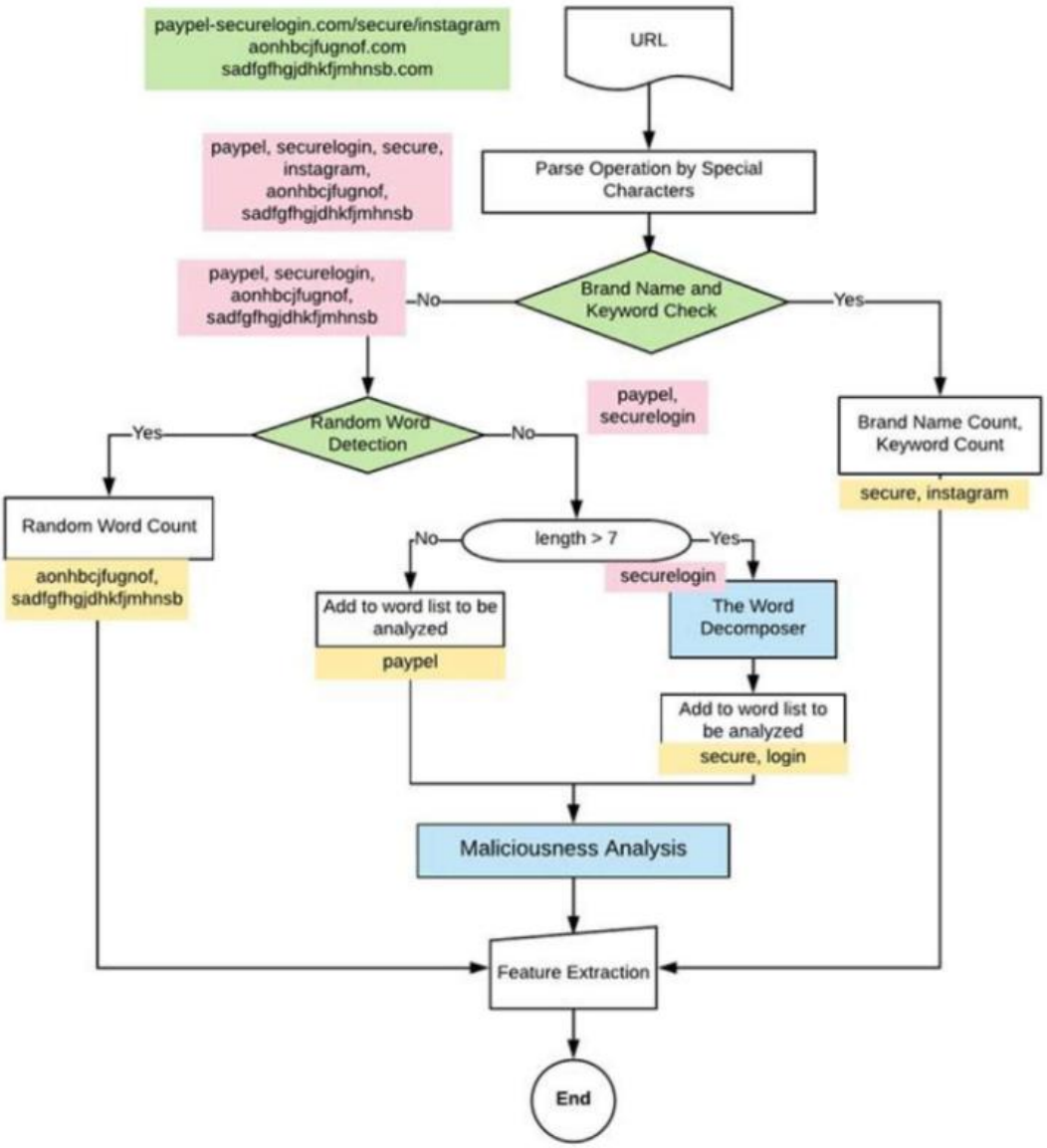
on a URL considering the language, and from there to extract NLP features. We have added a few more features besides what appears here.

The achieved contribution

We introduced a new approach to general analysis by language, this was something that was new (to us), so it was really nice to mess with it and discuss it. To be honest, humanity as a whole has not gotten any better and there are better things than we have realized. But we wanted to do it ourselves because it was really cool.

Evaluation

This is the most interesting part. Let's see the architecture of the solution:



We implemented all this in the code!!

When a URL enters the system, the first and most important step is preliminary analysis.

In this analysis, it is broken down into parts (the parts of the URL: domain, subdomain, path to the file, etc.)

and a list of words that appear in it is obtained from this. Then their classification begins - which words are gibberish (Which is a AI classification problem in itself!), which words are a combination of several other words, we will remove from the content words that were found interesting such as "brand lists" (facebook, google, etc.) and "keywords" (admin, login, secure). For each list we have dozens and hundreds of words.

And at the end we will also find the evil words. These are words that are "similar" to other words, and we did this with the edit distance algorithm. Also known as the Lewinstein algorithm. (Dynamic programming algorithm for finding similarity between strings)

We would have to pay attention to the order of operations (for example, after breaking down a word, we need to check if we got brands or keywords again. And for example, we need to find out if a word is random or a combination of several words).

From the following process, we created for each URL a dictionary in Python that contains the following: url, domain, registered_domain, tld, subdomain, class (legitimate or phishing), protocol, path_to_file, words_raw, brand_words, keyword_words, random_words, word_list, word_composed and word_malicious.

After that, we moved on to the extraction of a little more than 20 features, most of which are in the following picture:

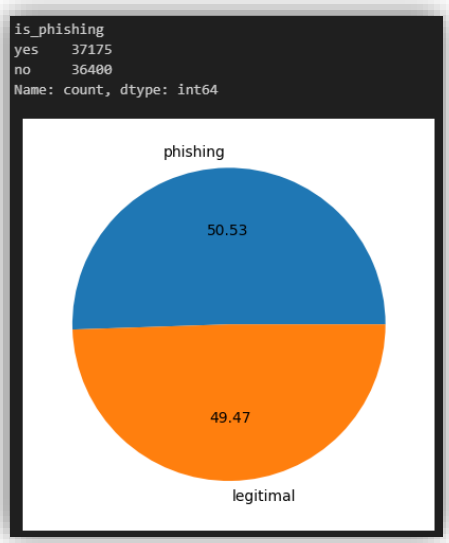
NLP based features.

Feature	Explanation
Raw Word Count	The number of words obtained after parsing the URL by special characters.
Brand Check for Domain	Is domain of the analyzed URL in the brand name list?
Average Word Length	The average length of the words in the raw word list.
Longest Word Length	The length of the longest word in the raw word list.
Shortest Word Length	The length of the shortest word in the raw word list.
Standard Deviation	Standard deviation of word lengths in the raw word list.
Adjacent Word Count	Number of adjacent words processed in the WDM module
Average Adjacent Word Length	The average length of the detected adjacent words.
Separated Word Count	The number of words obtained as a result of decomposing adjacent words.
Keyword Count	The number of keywords in the URL.
Brand Name Count	The number of the brand name in the URL.
Similar Keyword Count	The number of words in the URL that is similar to a keyword.
Similar Brand Name Count	The number of words in the URL that is similar to a brand name.
Random Word Count	The number of words in the URL, which is created with random characters.
Target Brand Name Count	The number of target brand name in the URL.
Target Keyword Count	The number of target keyword in the URL.
Other Words Count	The number of words that are not in the brand name and keyword lists but are in the English dictionary (e.g. computer, pencil, notebook etc ...).
Digit Count (3)	The number of digits in the URL. Calculation of numbers is calculated separately for domain, subdomain and file path.
Subdomain Count	The Number of subdomains in URL
Random Domain	Is the registered domain created with random characters?
Length (3)	Length is calculated separately for the domain, subdomain and path.
Known TLD	[*"com", "org", "net", "de", "edu", "gov", etc.*] are the most widely used TLDs worldwide. Is the registered TLD known one?
www, com (2)	The expression of "www" and "com" in domain or subdomain is a common occurrence for malicious URLs.
Puny Code	Puny Code is a standard that allows the browser to decode certain special characters in the address field. Attackers may use Puny Code to avoid detecting malicious URLs.
Special Character (8)	Within the URL, the components are separated from each other by dots. However, an attacker could create a malicious URL using some special characters {'-', '.', '/', '@', '?', '&', '=', '_'}.
Consecutive Character Repeat	Attackers can make small changes in brand names or keywords to deceive users. These slight changes can be in the form of using the same character more than once.
Alexa Check (2)	Alexa is the name of a service that places frequently used websites in a certain order according to their popularity. Is the domain in Alexa Top one million list?

The dataset contains 37175 phishing URLs, and 36400 legitimate URLs. There was nothing of their own and we put them all into pre-proccessing to create huge lists of dictionaries and save them in a json file, which are then loaded again for the feature extraction phase.

Dataset exploration

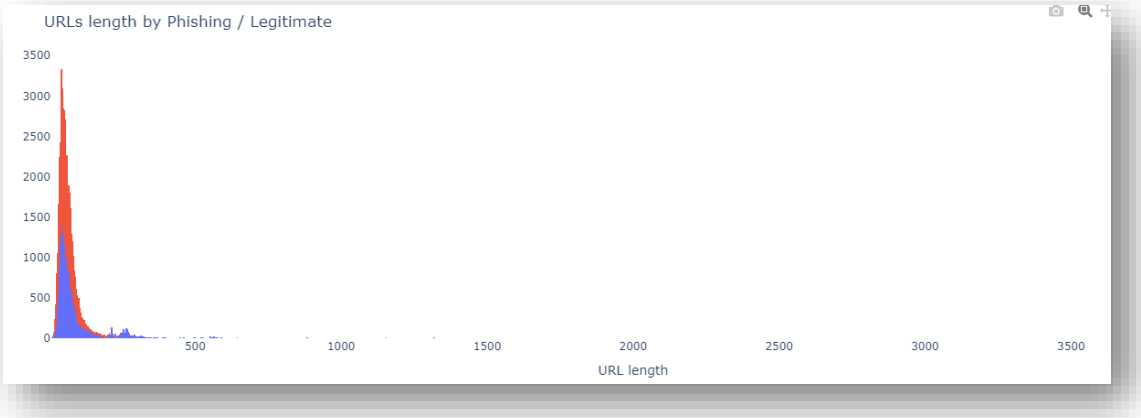
Before the pre-processing phase itself, we checked that the data set is balanced, that there are no duplicates and that there are no empty parts.




```
df.isnull().sum()

url           0
is_phishing   0
dtype: int64
```

Then just to see that "classic" features are not enough (again, this happened before the pre-processing, when we explored the data) we tested some of them. For example: URL_len

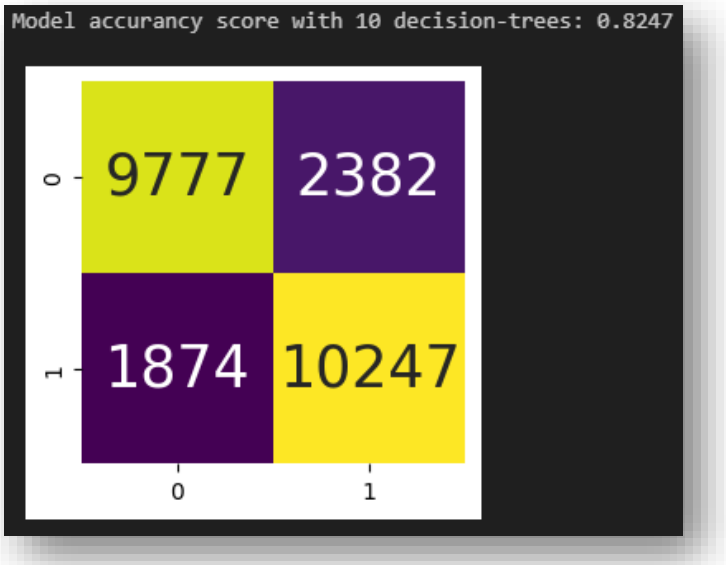


We explored a little more in this style that I will save from this document.

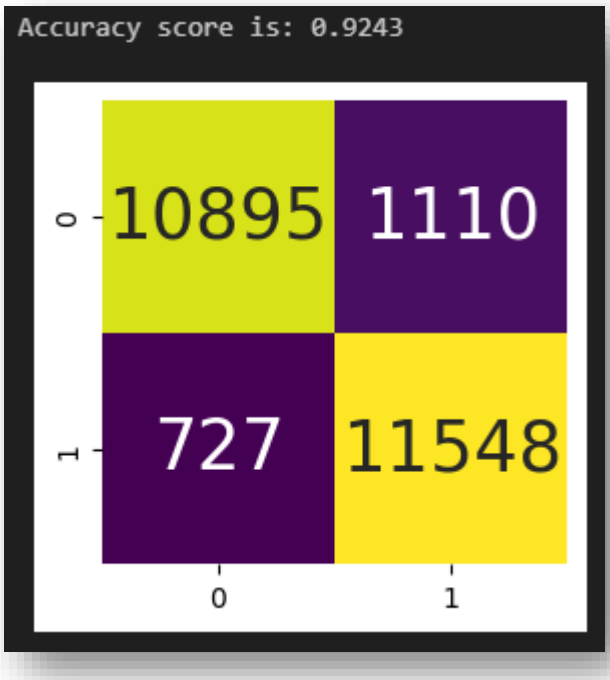
Algorithm and results

Let's start from the end.

This is what we achieved with regular features of the (The features that are also used in the articles mentioned related work):

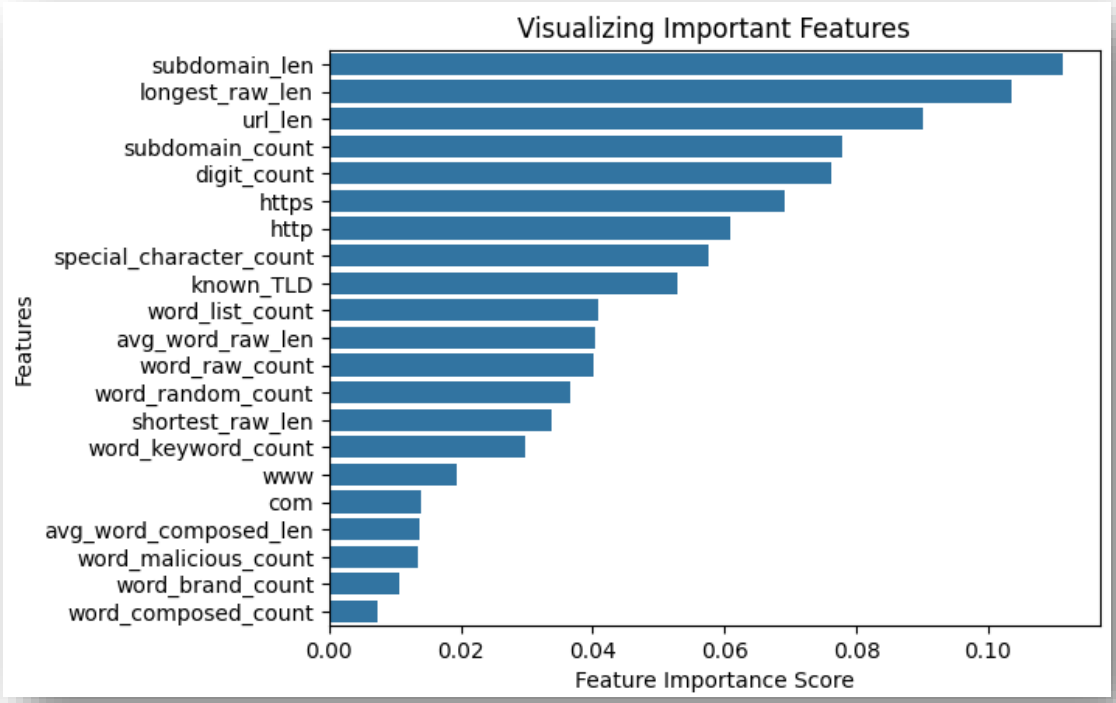


And this is what we achieved after using our NLP features, for the information extracted in the pre-processing stage:



In both the test and the train in a 3:7 ratio, with 10 decision trees in the Random Forest algorithm. After testing it performed better than the others. (and the articles also said that)

After filtering the features and leaving those that contribute more, we stayed with these features. which were received thanks to the **pre-processing process**, which is the center of gravity of our project:



Summary

Our task was to classify phishing and legitimate URLs. When we used normal features, we did not achieve much success (80 percent). After a long preliminary analysis process, after which we released "smarter" features, we went up to 92+ percent. We used algorithms for checking similarity between words, checking gibberish words, etc. We removed almost everything possible from URL. This is how we created a more powerful dataset with language-adapted widgets. The best algorithm (also among 7 that appeared in articles such as knn, Naive-base, k-star, SMO, etc.) was random forest. He also gave us better results (we saw while playing)

The success was 10 percent better than other articles.

Our system can indeed be improved! Through better (and AI-based) classification of words. Their breakdown and analysis is still in the preliminary stage.