Machine Learning Engineer Nanodegree
Capstone Proposal

Bassim Lazem
September 5th, 2018

Detecting Duplicate URLs Using Deep Recurrent Networks

Domain Background

Many websites on the Internet have multiple Uniform Resource Locators (URLs) for the same page content. This happens for various reasons, for example, to ease user navigation, many sites provide shortcut paths for the same page. Another frequent reason (especially on the deep web) is adding parameters to the URLs that does not change the page content. In addition, some sites use duplicate urls for load balancing and ensure fault tolerance. This creates a problem for web crawlers, as they collect a large number of duplicate content; they waste time and resources downloading the same content over and over again.

My motivation for this project stems from my work in data mining and web crawling. I had faced this DUST (Duplicate URLs with Similar Text) issue in one of the past projects I was working on. The implemented crawler suffered from this problem, showing the same retrieved page multiple times, and it resulted in poor user experience and wasted resources. I have developed a solution for it; however, it was in post-crawling stage (after downloading the pages, an analysis is executed to detect similar pages). I'm interested in solving this issue using Machine Learning to provide online detection during crawling without needing to store the duplicate pages.

Problem Statement

The goal of this project can be stated as follows: While a crawler is visiting a URL in some site, determine which hyper-links it should follow next, discarding the identical ones to the current page. More specifically, given two URLs, predict if they direct to the same page or different ones. The end product is a ML middleware filter that is trained to detect these DUST links, resulting in better resource utilization and overall enhanced information retrieval.

Datasets and Inputs

The datasets will be collected from different open sites, crawling a sizable sample from each to train the algorithm. The following sites are candidates:

- https://ubuntuforums.org
- https://digitalspy.com
- http://www.teamliquid.net

- https://bodybuilding.com

Each site is dynamic, has large number of pages backed by millions of users. More importantly, each one of them exhibits the duplicate URLs issue. For example, the following urls are the same:
https://forums.digitalspy.com/discussion/2294848/the-jeremy-vine-show-weekdays-channel-5
https://forums.digitalspy.com/discussion/comment/91010367#Comment_91010367
https://forums.digitalspy.com/discussion/comment/91009893#Comment_91009893
https://forums.digitalspy.com/discussion/comment/91076484#Comment_91076484
https://forums.digitalspy.com/discussion/comment/91108848#Comment_91108848
https://forums.digitalspy.com/discussion/comment/91108848#Comment_91108848

Plus 21 other of the comment links in the page.

A crawler will run over part of the site, collecting the training samples (while respecting the robots.txt rules of the site). The crawled fields are:

- **id** – The id of the training page
- **url** – The URL of the page
- **referrer** – The URL of the page the originated request
- **content** – The page HTML text.

I already created a crawler for ubuntuforums.org, attached a SQLite database containing a crawled sample from it with the fields above. Notably this site exhibits another interesting issue, for example this URL:
https://ubuntuforums.org/showthread.php?t=2397178&p=13786883&mode=threaded
Is part of this URL:
https://ubuntuforums.org/showthread.php?t=2397178&p=13786883&mode=linear#post13786883
The model should be able to infer this too.
I plan to crawl around 20k pages from each site.

Solution Statement

Several approaches have been proposed to mitigate this issue, with varying degrees of success. Broadly classified into two categories, detecting duplicate URLs by analyzing the URL addresses information, or by fetching and comparing the pages content using syntax and semantic analysis. The first approach tries extracting normalization and tokenization properties of the URLs to detect duplication rules. The second approach uses similarity techniques between documents and determine duplicate pages using a threshold.

My proposal here is to utilize a mixture of both these approached to create a convenient customized solution. The idea is to create a machine-learning model to discover these duplicate URLs rules by itself. The model will be given only the URLs and a score between each pair. This score will come from the content of the pages. Therefore, the data is going to be sampled from a target site only during training, whereas the deployed model will predict on the URL strings alone. Using machine-learning especially deep learning is powerful in its ability in manually extract features, learning a representation from the URLs sequence of characters. Using LSTM in URL analysis has already shown great promise in Phishing Detection, and domaingenerated algorithms. And in fact, this [5] is the inspiration for me to apply the same technique on the DUST problem.

Benchmark Model

For this problem, the benchmark model can be simply a follow-all spider, crawling all links without identifying duplicates. This is the basic approach used by crawlers, and it is obviously going to suffer from the duplicate issue.
Another viable benchmark is to use Random Forest algorithm on the same classification problem, so it can be compared to the LSTM model. The developed solution should provide better crawling strategy for the test sites, and should crawl only different URLs and avoid same content pages.

Evaluation Metrics

Prediction results are going to be evaluated on the log loss between the predicted values and ground truth ones. The ground truth is the similarity score between two HTML documents, whereas the predicted values are coming from these documents URLs only. If the model is able to infer the duplication rules, it should have high prediction accuracy. The model also should not predict that two page are the same while they are not, since that would prevent the crawler from visiting it. Therefore, I believe a high f-score is good metric for the model.

Project Design

Data Collection

A crawler will be created to collect the URLs and their pages content from the aforementioned sites. Each page will be a record in a SQL database table with the parameters outlined in the dataset section.

Similarity Scoring

In this stage, each pair of collected URLs will be graded with a similarity score between the pages content. Both structural and semantic similarity will be attempted to provide an accurate result for duplicate detection. The outcome of this will be another table with the following fields:

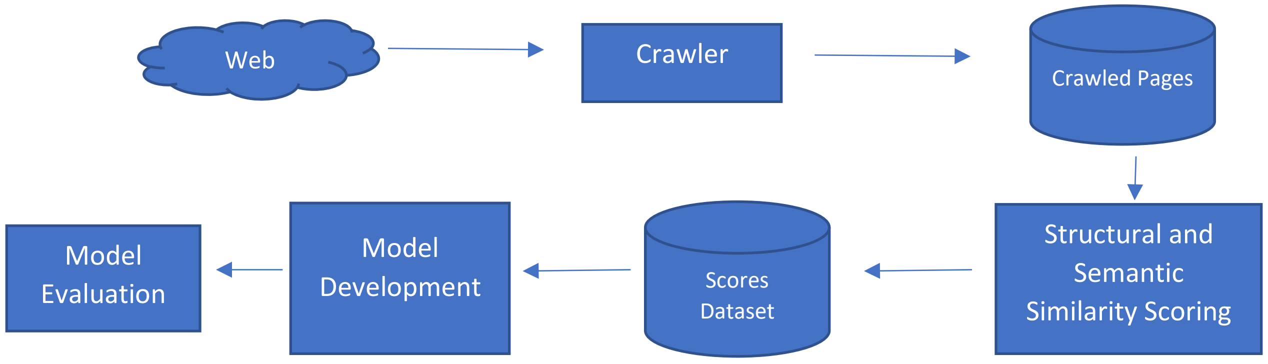
- **url1** – first URL to compare
- **url2** – second URL to compare
- **score** – similarity score between the two URL pages

Model Development

The output dataset of the second stage will be used to train the ML model. It will take the two URL strings as input and the score as the output. A Random Forest and LSTM models will be created and trained on the data.

Evaluation and Testing

In this final stage, the model will be evaluated against test set from the crawled data.



References

- [1] Detection of Distinct URL and Removing DUST Using Multiple Alignments of Sequences https://www.irjet.net/archives/V3/i1/IRJET-V3I1162.pdf
- [2] Detection and Removal of DUST: Duplicate URLs with Similar Text Using DUSTER http://www.ijircce.com/upload/2017/january/112_Jyoti%20Paper.pdf
- [3] A Graph Based Approach for Eliminating DUST Using Normalization Rules http://www.ijircce.com/upload/2016/april/181_A%20Graph.pdf
- [4] ELIMINATE DUPLICATE URLs USING MULTIPLE ALIGNMENT OF SEQUENCES http://www.mjret.in/V2I4/M59-2-4-10-2015.pdf
- [5] Classifying Phishing URLs Using Recurrent Neural Networks https://albahnsen.com/wp-content/uploads/2018/05/classifying-phishing-urls-using-recurrent-neural-networks_cameraready.pdf