Collapsing Near-Duplicate Web Pages Into Authoritative Versions

CSCI 2580 - Web Search Engines - Final Project
Spring 2016
Purnima Padmanabhan
Erdem Sahin

Summary

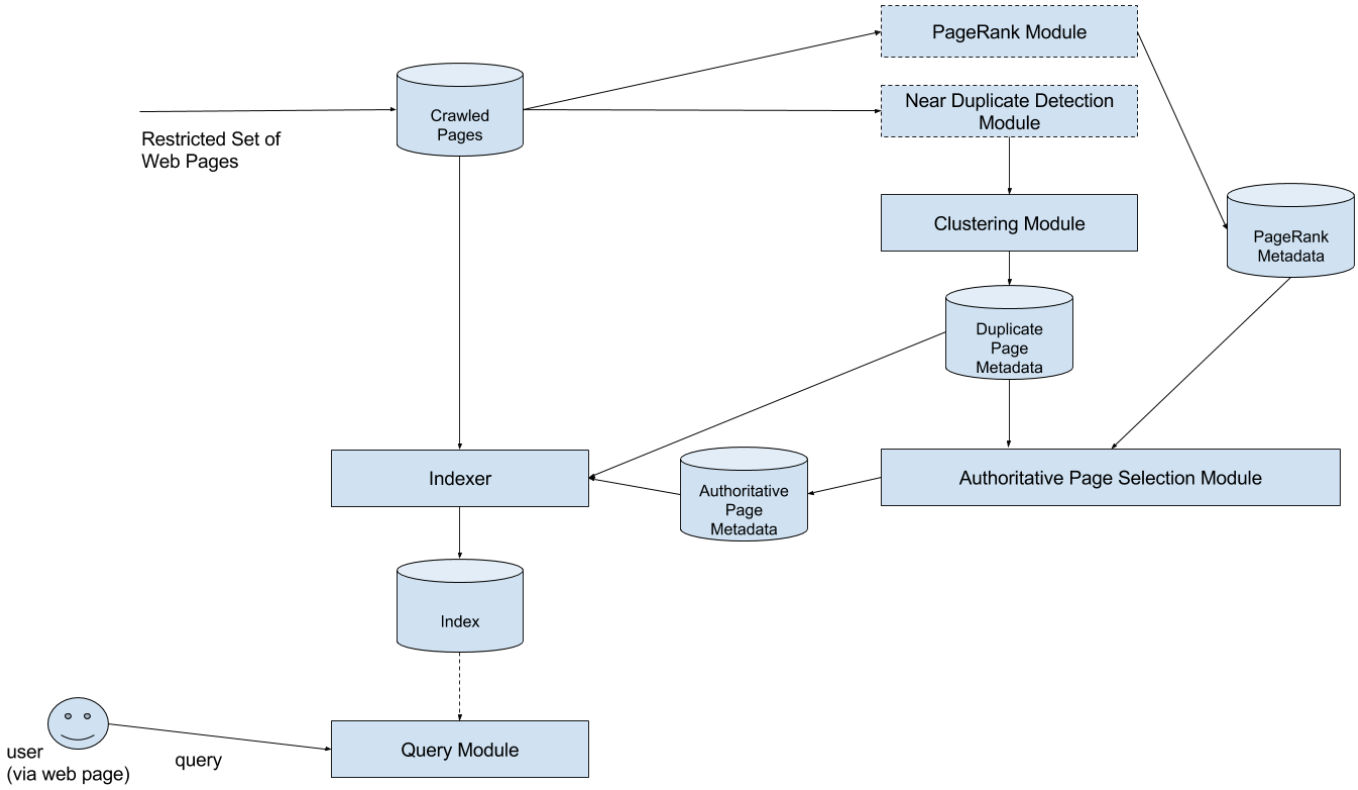
We improve a search engine such that it offers near-duplicate detection. Our search engine finds near-duplicate web pages (using the Shingles algorithm), clusters near-duplicate pages (using Union-Find), and determines the “authoritative” version among near-duplicates (using PageRank). The search engine then only indexes the most authoritative web page among a cluster of near-duplicates. In the search result, we highlight the most authoritative web page, along with displaying links to the less authoritative versions.

This offers several benefits to the search engine. From a user perspective, it ensures that the search results page is not filled up by nearly identical pages and increases the likelihood of presenting a more diverse result set to the user. From the search engine architecture perspective, it decreases storage requirements as only the most authoritative versions of the web page need to be indexed. It can further contribute to reducing the computation and bandwidth resources if the near-duplicate information is provided to the crawler so that the crawler can skip less authoritative duplicates or crawl them with very low frequency.

Many pairs of documents on the web are not exactly identical, but may still contain the same textual content and the satisfy the same underlying information need. The content differences between the pages may arise from different images embedded in the documents, ads being generated dynamically, and so forth. However, for the purpose of satisfying a user’s query that is not focussed on images/ads, these documents have identical information, and must be treated as duplicates[6]. Hence, the focus of our project is limited to text-only duplicate detection, where the text between the HTML tags of <title> and <body> is extracted and passed as input to the shingles generator.

In the sections below, we describe the steps that we undertook to implement near-duplicate detection, clustering, and authoritative document selection. We evaluate the performance of our implementation and discuss the issues encountered during our implementation process. Lastly, we mention future directions based on this work.

Architecture



Explanation of system architecture

A real search engine will first use a web crawler to obtain the set of web pages to use as the information source. This collection may contain duplication of information. The pages in this collection are saved to a directory. We skipped this crawling phase and start with the assumption that we have the required pages made available to us.

Subsequently, these pages are fed to the PageRank (PR) module, which calculates the PageRank scores of the individual pages [8]. These pages are also fed to the near duplicate detection (NDD) module, which is based on the Shingling algorithm.[1] The NDD and PR computations are independent processes. The NDD outputs pairs of duplicate documents, as (doc1, doc2), (doc1, doc3), etc. Since a number of documents in the directory may be near-duplicates, clustering of the document collection needs to be carried out to find the clusters of duplicate documents. Hence, in order to do that, we use the Union-Find algorithm to compute clusters or components of near-duplicate documents [5]. The cluster size varies from 1 to N, where N is the total number of pages. In case a page does not have any near-duplicates, it is in a cluster by itself, and is automatically (trivially) the most authoritative document of the cluster.

In order to select the most authoritative document to represent a cluster of duplicate documents (cluster size > 1), we use PageRank scores of the documents in the cluster as the selection criteria. We retrieve the pre-computed scores for the documents from the PR metadata. We select and retain the document with the highest PR score to "represent" the cluster. This is the most authoritative document, which will be indexed by the indexer of the web search engine. The other duplicate documents in the cluster are retained as metadata, but are not indexed, as the purpose of near-duplicate detection is to eliminate the storage and processing overhead associated with indexing and searching for duplicate documents when a user fires a query. This is represented by the Authoritative Page Selection module. After indexing all the authoritative documents across every cluster, the indexer builds the index using the TF-IDF weighting approach. For the indexer and retriever, we have used the Apache Lucene library [9].

When a user submits a query to the system, the retriever is run to find a list of documents that best answer the user's query. As mentioned, only the most authoritative documents are searched - and not their duplicates - so that the user does not view the same content.

Evaluation

We evaluated our near-duplicate detection module on 2 datasets that we generated. We have an *Article Dataset* that originates from a collection of scientific articles [1]. These articles are in text file format, but since the shingles algorithm is designed to detect text similarity, this dataset

[1] We used the shingling algorithm as given on the course website:
http://cs.nyu.edu/courses/spring16/CSCI-GA.2580-001/Duplicates.html

is appropriate for us to test our implementation. We also have a *Wikipedia Dataset* that is similar to the one we were provided with for Programming Assignment 3, but expanded to contain more pages. The pages in the Wikipedia Dataset contains links between each other so they are also used for PageRank calculations.

We first inspected the original datasets for duplicates or near-duplicates. The *Article Dataset* had some duplicates, which we eliminated by keeping only one version. Then, we introduced near-duplicates by following a randomized process, similar to the one introduced in [4]. We used the following parameters:

- `PAGE_DUP_RATIO:` The fraction of pages for which duplicate(s) should be generated.
    - As a default, we set this to 0.3 (i.e. we generated duplicate(s) for a page with probability 0.3).

- `MAX_DUPS_PER_PAGE:` If duplicates are being generated for a page, the maximum number of duplicates to generate (picked randomly from [1, MAX_DUPS_PER_PAGE].)
    - As a default, we set this to 3 (i.e. a page with duplicates had between 1 and 3 duplicate versions).

- `DIF_SEGMENTS_RATIO:` If duplicates are being generated for a page, the fraction of locations for which a random segment should be inserted.
    - A “segment” is defined as a sequence of words that are different from the original text. After each word in the original text, we draw a random number to determine whether we should insert a random segment. As a default, we set this to 0.05 (i.e. we will insert a random segment after roughly 5% of original words).

- `MAX_SEGMENT_LENGTH:` If a random segment is being generated, the maximum length of the segment.
    - As a default, we set this to 3 (i.e. a random segment has length between 1 to 3). We pick nouns at random from the WordNet English dictionary [2].[2] Since some noun entities in WordNet consist of multiple words (e.g. eye_doctor, which we

[2] We use the JWI library to access WordNet [3].

convert to “eye doctor”), we split each noun to component words and add the words one-by-one until we reach the MAX_SEGMENT_LENGTH.

Since we construct the near-duplicates ourselves with the logic above, we trivially obtain the labels for pairs of near-duplicate documents to evaluate our Shingles algorithm on. If we pick document *abc* and generate its near duplicate *abc_1*, we expect the pair (*abc, abc_1*) to be identified as near-duplicates by our algorithm.

Expanding on the example above, let us assume that we decided to generate 2 duplicates for *abc*. That is, we generate *abc_1* and *abc_2.* In this case, we expect the pairs (*abc, abc_1*), (*abc, abc_2*) and (*abc_1, abc_2*) to all be identified as near-duplicates. This is because upon manual inspection of the generated texts (i.e. based on “human review”), the contents of *abc_1* and *abc_2* were also identified as near-duplicates of each other. Thus, we expect all near-duplicates generated from a source document to be identified as near-duplicates of each other.

Continuing from the same example, we observed that our algorithm tended to do well in identifying (*abc, abc_1*) and (*abc, abc_2*) as duplicates, but it tended to miss (*abc_1, abc_2*). To address this, we employed the Union-Find data structure [5] to cluster the near-duplicates together. Each document started off in its own cluster, and as new pairs of near-duplicates were identified, the clusters were joined. For instance, if (*abc, abc_1*) and (*abc, abc_2*) were identified as duplicates by the algorithm, after Union-Find, *abc, abc_1,* and *abc_2* were labeled as near-duplicates of each other. This was in line with our project approach of clustering near-duplicates. Furthermore, this approach notably improved our recall with respect to the setup outlined above.

Upon running our algorithm, we calculated the Precision, Recall and F_1-score by comparing the near-duplicate labels produced by the algorithm with the labels expected by our near-duplicate generation process. These metrics are employed widely in the literature (e.g. in [4]) to evaluate near-duplicate detection algorithms. Intuitively, a high precision is desired to make sure that the algorithm does not mistakenly identify unrelated documents as near-duplicates, and a high

recall is desired to ensure that the algorithm identifies most pairs of near-duplicates correctly. The F_1 score "combines precision and recall into a single number".[3]

Below we present the results of our experiments with a few different parameters.

`Using DIF_SEGMENTS_RATIO = 0.05`

Dataset	Without Union-Find			With Union-Find		
	Precision	Recall	F	Precision	Recall	F
Article	1.0	0.643	0.783	1.0	0.857	0.923
Wikipedia	1.0	0.481	0.649	1.0	0.614	0.76

`Using DIF_SEGMENTS_RATIO = 0.06`

Dataset	Without Union-Find			With Union-Find		
	Precision	Recall	F	Precision	Recall	F
Article	1.0	0.488	0.656	1.0	0.791	0.883
Wikipedia	1.0	0.384	0.553	1.0	0.546	0.705

`Using DIF_SEGMENTS_RATIO = 0.07`

Dataset	Without Union-Find			With Union-Find		
	Precision	Recall	F	Precision	Recall	F
Article	1.0	0.344	0.512	1.0	0.469	0.638
Wikipedia	1.0	0.342	0.509	1.0	0.443	0.614

We make the following observations:

- The algorithm demonstrates good precision and recall for reasonable `DIF_SEGMENTS_RATIO` values (e.g. `DIF_SEGMENTS_RATIO` < 0.06). Therefore, for near-identical documents, our algorithm performs well.

[3] The source for this quote is the course website.
http://cs.nyu.edu/courses/spring16/CSCI-GA.2580-001/Evaluation.html

- The Shingles algorithm performs solidly with respect to precision across the various scenarios. That is, in most cases it does not output false positives.
- The algorithm's recall and F_1 score improve notably when Shingles is combined with Union-Find.
- We observed that for increasing `DIF_SEGMENTS_RATIO` values (e.g. `DIF_SEGMENTS_RATIO` >= 0.07), our algorithm gives an increasing number of false negatives. This results in the recall and thus the F_1 score of our implementation to fall somewhat sharply. However, we accept this as a desired property because as the variants start differing more and more from each other, they would likely cease to become near-duplicates based on human evaluation. Therefore, as desired, our algorithm categorizes fewer documents as near-duplicates as the documents are increasingly dissimilar.

In most experiments we observed lower scores for the Wikipedia dataset compared to the Articles dataset. We attributed this to the fact that the texts in the Wikipedia dataset were much shorter compared to the Articles dataset.

Issues Encountered

The biggest issue we faced was obtaining a reliable, gold-standard dataset that we could test the performance of our algorithm on. Our near-duplicates and expected labels were machine-generated; therefore they may differ slightly from what a human would label as near-duplicates. Further, the shingles approach slows down considerably after a document collection size of over 50 documents. For a full collection of 200 long scientific articles, the algorithm was still executing 2 hours after it was run. Hence, we had to limit the number of files that we indexed. The Wikipedia dataset contains about 30 documents.

It was also observed that our duplicate detector usually produced more false negatives. A false negative, in our case, occurs when two near-duplicate documents are not marked as near-duplicates by the algorithm. This ties back to the difficulty of not being able to find a gold-standard dataset. Part of the problem is that there is no concrete boundary that allows a document to be considered a near-duplicate or not. So, the definition of 'near-duplicateness' is not very precise even to begin with, while generating the data. Beyond that, the shingles heuristic is not guaranteed to provide accurate results every time. For example, consider a situation where two documents are near-duplicates:

Doc1: "... the quick brown fox jumped over the wall..."
Doc2: "...the lazy brown fox climbed up the wall…"
Shingle with minimum permutation value for Doc1: quick brown fox
Shingle with minimum permutation value for Doc2: lazy brown fox

These shingles map to different hash values and consequently (most likely), different permutation values. As these shingles have the minimum permutation values for their documents, and these permutation values are not equal for the round in which they were generated, these documents will not be considered as duplicates. If this occurs a number of times throughout the different rounds of the algorithm, the documents will be deemed as non-duplicates, even though they clearly are near-duplicates. Hence, whether two documents are determined to be duplicates or not greatly depends on the shingle boundaries, which is highly variable between document pairs.

Given the lack of clarity while defining our own test datasets combined with the approximation nature of the shingles algorithm, there is room for future improvement of the evaluation process.

Future Scope

We have considered the following as possible extensions to the system:

- Try out other algorithms in place of the shingles generator. Some examples of algorithms include SimHash[12] and SpotSigs[6]. As mentioned, the shingles approach is extremely slow for > 50 documents.
- To address the slowness, we could also try parallelizing the pairwise near-duplicate detection process. A sketch process to do this in the MapReduce paradigm is given in [7].
- Make the list of near-duplicate documents and URLs available to the web crawler. The crawler can then use this information to decide the crawl rate for following links to and downloading non-authoritative near-duplicates in the future.
- Use a near-duplicate dataset with crowdsourced labels. Such a dataset would reflect human users' judgement criteria more closely. The combination of [10] and [11] would give a human-labeled gold standard. We attempted to get access to these but were not able to obtain access to [11] due to the lengthy and complicated licensing process.

Sources

[1] Science and Scientific Discussion Articles. Accessed Online: http://textfiles.com/science/

[2] WordNet: A Lexical Database For English. Accessed Online: http://wordnet.princeton.edu/wordnet/

[3] JWI: The MIT Java Wordnet Interface. Accessed Online: http://projects.csail.mit.edu/jwi/

[4] Jun Fan, Tiejun Huang. A Fusion of Algorithms in Near Duplicate Document Detection. Accessed Online: https://conferences.telecom-bretagne.eu/data/qimie2011/fan_huang-informal_QIMIE_2011.pdf

[5] Robert Sedgewick and Kevin Wayne. Algorithms. Accessed Online: http://algs4.cs.princeton.edu/15uf/UF.java.html

[6] Adaptive Near-Duplicate Detection via Similarity Learning. Accessed Online: http://research.microsoft.com/pubs/130851/ANDD.pdf

[7] Kira Radinsky. Duplicate Detection. Accessed Online: http://webcourse.cs.technion.ac.il/236621/Winter2010-2011/ho/WCFiles/tutorial%204%20(Duplicate%20Detection).pdf

[8] Lawrence Page and Sergey Brin. The PageRank Citation Ranking: Bringing Order to the Web. Accessed Online: http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf

[9] Apache Lucene. Accessed Online: https://lucene.apache.org/

[10] Omar Alonso, Dennis Fetterly, and Mark Manasse. ClueWeb 09 Labeled Near-Duplicate News Articles. Accessed Online: http://research.microsoft.com/en-us/downloads/a0200ed2-b0d9-4a9d-9170-bf1b448dcb67/

[11] Clueweb 09 Dataset. Accessed Online: http://www.lemurproject.org/clueweb09.php/

[12] Caitlin Sadowski and Greg Levin. SimHash: Hash-based Similarity Detection. Accessed Online: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.473.7179&rep=rep1&type=pdf