Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), pages 1242–1249
Marseille, 11–16 May 2020

Online Near-Duplicate Detection of News Articles

Simon Rodier and Dave Carter
Digital Technologies, National Research Council Canada
{simon.rodier, david.carter}@cnrc-nrc.gc.ca

Abstract

Near-duplicate documents are particularly common in news media corpora. Editors often update wirefeed articles to address space constraints in print editions or to add local context; journalists often lightly modify previous articles with new information or minor corrections. Near-duplicate documents have potentially significant costs, including bloating corpora with redundant information (biasing techniques built upon such corpora) and requiring additional human and computational analytic resources for marginal benefit. Filtering near-duplicates out of a collection is thus important, and is particularly challenging in applications that require them to be filtered out in real-time with high precision. Previous near-duplicate detection methods typically work offline to identify all near-duplicate pairs in a set of documents. We propose an online system which flags a near-duplicate document by finding its most likely original. This system adapts the *shingling* algorithm proposed by Broder (1997), and we test it on a challenging dataset of web-based news articles. Our online system presents state-of-the-art F_1-scores, and can be tuned to trade precision for recall and vice-versa. Given its performance and online nature, our method can be used in many real-world applications. We present one such application, filtering near-duplicates to improve productivity of human analysts in a situational awareness tool.

Keywords: duplicate detection, near-duplicate detection, news corpora, natural language processing, situational awareness

1. Introduction

Near-duplicates are multi-sentence spans of text in the same language that convey the same content using largely similar vocabulary; parts may be identical, paraphrased or lightly modified, and limited portions may be entirely unique to one document but not its near-duplicates. Near-duplicates can be thought of as documents with a non-trivial but bounded edit distance. This paper presents a near-duplicate detection system that is online, rapid, and has been validated in an enterprise software tool.

Near-duplicate documents are prevalent in some corpora and come at a significant cost. There are computational costs in processing documents that provide no new information; and significant costs to end users, who are forced to sift through redundant documents when trying to accomplish a task. Machine learning algorithms may also be susceptible to error due to near-duplicate documents. For example, Chatterjee (2019) describes how a sudden spike in news about Brexit has thrown off machine-based trading systems; a near-duplicate detection system could be used to filter out fundamentally similar documents to help these systems cope with bursts of news articles.

We build upon the *shingling* method proposed by Broder (1997) to create a real-time near-duplicate detection system that offers state-of-the-art results, and which can be tuned to provide better precision or recall as needed.

1.1. Near-duplicate detection

There are different definitions of near-duplicate documents. For web-based news documents, Theobald et al. (2008) describe one type as consisting of news articles from different sites where the article content is the same, but the content of the surrounding site is different. The second type is essentially an inversion of the first: the page's core content is different, but the surrounding site is the same. In this work, we are concerned with identifying the first type of near-duplicate document. In particular, we require a system that is capable of receiving documents in a stream, and for each document, can answer two questions: is this document a near-duplicate of one previously seen, and if so, which? While previous systems have presented effective offline methods for filtering near-duplicate documents, we propose an online system that can answer both questions.

1.2. Related work

1.2.1. Background

The previous work on near-duplicate detection has focussed on judging the similarity of two documents without performing a computationally-expensive bit-wise comparison of entire documents. Detection of *identical* documents can be done by hashing the documents and comparing the hash values: any documents with matching hashes are then considered to be duplicates. This method is not applicable to *near-duplicate* detection however, as it provides no information about how similar any two candidates are to one another. For near-duplicate detection, most methods create a compact representation of a document and use these for comparison, though they vary in how representations are formed and compared.

The *shingling* method (Broder, 1997; Broder, 2000) views a document as a set of overlapping n-grams (or *shingles*, short sequences of words in the text). The similarity of two documents can then be measured by calculating their set similarity. When this value is above a given threshold, documents can be considered near-duplicates of one another. This method is costly however, as the number of shingles generated can be quite large. To deal with this, Broder proposes generating a document *sketch* either by finding the set of minimum hash values of a random sample of shingles within a document with a set of permutations, or eliminating all of the shingles where, for a given shingle S and number m, $S \bmod m \neq 0$.

Another method of determining similarity between documents is *cosine similarity* (Salton et al., 1975). This method translates documents into vectors, where each element of the vector relates a term's importance in the corpus (generally, the elements are calculated with term frequency – ? inverse document frequency (TF-IDF) scores). Similarity between two documents is then expressed as the cosine distance between their vectors.

Charikar (2002) introduced locality-sensitive hashing to estimate similarity between documents without the memory overhead needed to keep full vector representations of each document in memory. To do this, Charikar proposes the use of hashing functions where the similarity between two documents is the probability that their hash values are equal. Each document is then hashed with some number t of these functions, and the resulting hash values are placed in a vector which represents the document. If two documents share a number of matching vector elements that surpasses a given threshold, they are considered near-duplicates.

Chowdhury et al. (2002) introduced *I-Match*, an algorithm that generates lightweight document fingerprints by hashing all of the significant tokens present in a document. Then, whenever two documents share a fingerprint, they can be classified as near-duplicates. This method is sensitive to very slight changes in document content. Kołcz et al. (2004) presented a variation where n variants of the lexicon are used. In each variant, some portion of the lexicon is dropped, and a fingerprint is generated by hashing the document once for each variant of the lexicon. This results in each document being represented by a vector of hash values. When testing two documents for near-duplicate status, if any of the matching pairs of elements of their vectors collide, the documents can be considered near-duplicates.

Henzinger (2006) proposed detecting near-duplicates by combining shingling (Broder, 1997) and locality sensitive hashing (Charikar, 2002). The method begins by using shingling to identify near-duplicate candidates, and then filters these results using locality sensitive hashing to identify near-duplicate documents. The author obtained promising results on a very large dataset, both for near-duplicate documents on the same website and near-duplicate documents on different websites.

Other methods exist that are useful in the related task of determining similarity between documents. Blei and Lafferty (2009) propose a document similarity metric based on topic models, where documents sharing an underlying topic distribution are rated as more similar to one another. Kusner et al. (2015) present a method for judging document distances based on word embeddings. Such methods are particularly relevant for judging documents with similar underlying semantic similarity. While this task is related, it is not quite the same as near-duplicate detection, where we seek to filter out documents that are not only semantically similar, but nearly identical in content.

1.2.2. Near-duplicate detection of web-based news content

Previous research has been done on near-duplicate detection of web pages in Hypertext Markup Language (HTML). Several methods attempt to leverage the Document Object Model (DOM) – the hierarchical structure of layout elements of the web page – to aid in the task. Mathew et al. (2011) present a three step algorithm centred around the use of a Term-Document-Weight (TDW) matrix. Each element of the matrix represents the frequency of a term in a document multiplied by a weight. Judging that the importance of a term varies according to where it is found on a page, the authors compile a predefined list of weights based on the HTML tags within which a term appears, and assign weights from this list. They apply filtering to the resulting matrix to reduce the number of comparisons, and then perform a final verification step to return the set of near duplicates to an input document. Arun and Sumesh (2015) also use the TDW matrix to perform near-duplicate detection. They then filter the documents to consider by limiting candidates to those that contain a number of sentences within a given range from the input document. These candidates are then hashed and compared bit-wise. Documents with a similarity over a given threshold are considered near-duplicates. Ling et al. (2010) present a three-step method that analyzes a web page's DOM tree. They prune the tree of any extraneous data (site navigation, advertisements, etc.) and keep the document's core content. They then examine the different fields present in the document and assign them a priority based on their importance in determining near-duplicate status. They finally compute the similarity of documents using the minimum edit distance between the comparable fields of different documents and compare the distance to a given threshold to determine near-duplicate status. Web page structure can change over time, however, as sites update their templates and as web technologies change, or even as a given document page gets served differently on a laptop versus a mobile device, or as a single document gets served with different branding for various newspapers owned by the same conglomerate. Natural language processing approaches can avoid such problems.

Theobald et al. (2008) identify near-duplicate web pages by finding stopwords (common words in a language that are likely to appear in most articles) in an HTML document and chains of non-stopwords that follow them. They refer to the chains that make up this set as the "spot signatures" of the document. This method is based on the likelihood that common words such as *a*, *an* or *the* are quite likely to appear in the text of an article, but less likely to appear in the structure of a web page or its advertisements. They then consider the Jaccard similarity of the spot signatures of two documents to determine if they are near-duplicates. They report good results for their method but note that it cannot handle certain classes of web documents. As the authors report, two web pages which simply report lists of data (e.g. stock prices) may not contain any stopwords whatsoever: their system would thus conservatively reject any pairs of such documents as non-near-duplicates, even if their content is exactly the same.

Hajishirzi et al. (2010) proposed an algorithm that learns vector representations of documents using labelled training data from the target domain and uses these to build a near-duplicate classifier. Each document is then represented by the resulting vector where each element relates the importance of an n-gram in the known vocabulary to the document. Comparing these uncompressed vectors, the authors are able to obtain high precision and recall scores for the near-duplicate detection task. Trading some precision and recall for computational efficiency, the authors also present a version of their method where they generate a signature of the vector with locality sensitive hashing.

Zhang et al. (2016) introduced a method called SigNCD for near-duplicate detection. It uses normalized compression distance (NCD), which has a prohibitive time complexity when dealing with full documents, and can also be skewed by longer documents. To deal with these issues, they apply NCD to a shorter signature of the document. They eliminate the content within specific tags from the body of the HTML document (e.g.: *style*, *a*, *iframe*) and replace them with blank spaces. This is a fairly simple approach to isolating the meaningful text in documents. They then generate document signatures using a method similar to Theobald et al. (2008), identifying punctuation or stopwords in text and extracting the chains of words that surround them. One benefit of Zhang et al.'s method is that their algorithm is parameter-free, meaning that there is no tuning process to adapt it to any given domain or application.

To the best of our knowledge, the previous work which most closely resembles our proposed method is by Gibson et al. (2008). They developed a system to identify near-duplicate news articles in a collection of web pages. Their approach was largely centered around a two-step process. First, they build a classifier that processes an HTML file in order to identify the main article content within the file and separate it from other parts of the file. Second, they apply a shingling method to the article content to identify which pairs of documents are near-duplicates. Their results boast very high precision and recall. Our method is similar insofar as we share the same two high-level steps. We also use a content extraction tool to extract article content from an HTML file, and apply a shingling-based method to identify near-duplicate articles. However, our systems differ in two major ways. First, our implementation makes use of a database, eliminating the need to keep information about all articles in memory, making it scalable to many millions of documents and making it more convenient to reconcile duplicates-of-duplicates back to a canonical original document. Secondly, our method is designed to be used in an online setting, allowing us to use it as a fast, real-time near-duplicate detector in a piece of enterprise software.

1.3. Contribution

Shingling + n-grams (Database architecture)

Our system, based on Broder (2000), presents several improvements for the task of identifying near-duplicate documents from a document input stream. Our system introduces a database-backed architecture, which allows us to evaluate incoming documents against a collection of millions of previously-processed documents. While our algorithm largely implements Broder's clustering algorithm to identify a candidate original document for a potential near-duplicate, we adapt it to function online and add a final step of calculating n-gram overlaps between two potential near-duplicate documents. This final comparison is enabled by our database-backed architecture, and offers good explainability of the algorithm's decision-making, as two documents with significant vocabulary overlap are intuitively more likely to be near-duplicates than otherwise. The ratio of n-gram overlap is also a key parameter in our algorithm, allowing users to tune the system for better precision or recall.

2. Method

In this section, we will examine how our system processes documents to identify potential near-duplicates. As our dataset (see section 3.1) uses HTML files, we describe how these files are converted into raw text, how we generate compact representations of this text, and finally, how to use these representations to judge if a new document is a near-duplicate of one previously seen.

2.1. Preprocessing

The fact that our dataset uses HTML files renders the problem more challenging than simply trying to identify whether two news articles are near-duplicates: documents from a similar website may share much extraneous information in common, which could nudge a near-duplicate detection algorithm towards detecting more false positives. In contrast, near-duplicate news articles from different websites will not share this extraneous HTML structure and will have less underlying content in common, potentially raising the number of false negatives. As extraneous web page content can lead to a higher rate of false positives and negatives, our first step is to remove as much of this potentially misleading information as possible. We use a commercial tool called Diffbot[1] to separate the title and body of a document from the HTML content surrounding it. Diffbot provides an API that extracts the main text content from most web pages, returning a JSON (JavaScript Object Notation) structure that includes the body of the page in plain text. Our dataset files include relative links to other content (images, advertisements, etc.); however, these links are not useful for our task, and, due to the age of the data, they tended to generate resolution errors. To account for this, we provided Diffbot's link resolver with a URL back to a site serving empty error content. In effect, this removed linked data from the HTML data. If Diffbot returns an error code for a given request, we add the document to a queue for resubmission to the service. While this does not eliminate all errors (it occasionally returns no usable results for a document), it does generally eliminate network-related errors. From Diffbot's output, we select the article's title and body, trim any leading or trailing white space, and concatenate the fields to create the document we will process.

2.2. Near-duplicate identification

Our system's aims are twofold. First, to determine if a new document is a near-duplicate of a previously processed document. Second, if it is a near-duplicate, to return the original document to which it is linked (e.g., linking a duplicate of a duplicate back to a shared parent). In this section, we present a method to generate representations of documents and describe how to use those representations to achieve both goals.

In essence, we assign evenly-distributed random numbers as identifiers to unique n-grams in a document. From each document, we then sample a random set of n-grams, sort

[1] Accessible at `https://www.diffbot.com`.

them by their unique n-gram ID, and take the first twenty IDs from this sorted list as the document's "sketch", which can then be compared to the sketches of previously processed documents. Sorting by randomly-distributed IDs is the key to ensuring that the amount of overlap of the ? sketches of two documents is pronounced when they are similar, and small or non-existent (i.e. overlapping only by chance) when dissimilar.

2.2.1. Generating a sketch of a document

Our near-duplicate detection algorithm is a shingling method largely inspired by Broder's work (Broder, 2000; Broder, 1997). In this section we refer to the concatenated string containing the trimmed title and text of the article as the document.

Our goal is to represent each document as a *sketch* consisting of a small set of 8-byte numbers, such that two similar documents will tend to generate sets of 8-byte numbers that overlap proportionally to their similarity. The pseudo-code for generating the sketch is given in algorithm 1.[2]

We begin by translating the document into a list of n-grams ("shingles" in Broder's terminology) of length n. We then randomly select a sample of k shingles from the collection (if the document has fewer than k shingles, we consider its full list of shingles), typically with $k = 200$. As in Broder (2000), we have generated a list of p randomly-generated numbers which we will call $permutations$. For each $permutation$ in $permutations$, we find the minimum hash value of that permutation and the numeric fingerprints of the shingles in the document[3]. The lowest hash value is then added to an array. This array of length p is then our sketch for the document[4], which we subsequently use to evaluate near-duplicate document candidates.

2.2.2. Using sketches to determine if a document is a near-duplicate

The above method boils down each document to a "sketch" consisting of p numbers. When processing a new document, d_{new}, we generate its sketch and compare it to sketches of previously processed documents. Whenever this comparison yields a number of collisions greater than a threshold number $t_{collisions}$, we consider the previous document as a candidate document for which the new one may be a near-duplicate. From this list of candidates, we identify the document with the highest number of collisions as our proposed original document, $d_{candidate}$.

We still need to confirm the candidate document's status as an original of the current document. We thus make a final set of comparisons: we obtain the percentage of overlap between the n-grams of d_{new} and $d_{candidate}$, considering only the first f percent of each document – based on the observation that news data is often structured with the most relevant information towards the beginning of the document, whereas context, background, and template information tends to be placed towards the end. If the overlap is greater than a threshold, $r_{overlap}$, we conclude that d_{new} is a near-duplicate of $d_{candidate}$.

Algorithm 1 Generating sketches for documents

```
function GENERATESKETCH($document$)
Parameters
    $n \leftarrow$ length of shingles (in words).
    $k \leftarrow$ number of shingles to select for sample.
    $permutations \leftarrow$ set of $p$ static,
        randomly-generated numbers.
Algorithm
    $shingleList \leftarrow$ Sample $k$ shingles from all
        shingles of length $n$ in $document$.
    $sketch \leftarrow$ empty list.
    for all $permutation$ in $permutations$ do
        $minHash \leftarrow \infty$
        for all $shingle$ in $shingleList$ do
            $hash \leftarrow$ HASH(FINGERPRINT($shingle$),
                $permutation$)
            $minHash \leftarrow$ MIN($hash$, $minHash$)
        end for
        add $minHash$ to $sketch$
    end for
    return $sketch$
end function
```

Documents which yield no list of viable candidates through this process are considered originals.

Table 1 shows how the system evaluates five short sample documents that share various levels of similarity.

Deterministic : Returns the same results if given the same input values

2.2.3. Implementation considerations

In 2.2.1 we described a set called $permutations$ with a list of p static, randomly-generated numbers. This makes the comparisons deterministic in nature when all shingles are sampled from a document.

In 2.2.2, we describe comparing the sketch of a new document with the sketches of previously processed documents. As our algorithm is designed to detect near-duplicates on the fly in an online setting with a large number of documents, we want to avoid storing the document sketches in memory. We therefore store them in a database, linking a document's unique identifier to each of the hashes that make up its sketch. When we process a new document, we can quickly obtain the unique identifiers of any previous documents where there is a collision between the hashes, and verify if the number of hash collisions is above the required threshold to be declared a duplicate. While this does require the additional overhead of a database, it allows us to do rapid lookups of hash collisions (typically ≪ 1 sec.) without storing each document's sketch in memory.

Broder et al. (1997) point out that shingling does not always work well for very short documents, as the error in estimating document resemblance is greater than with longer documents. To counteract this, we use a higher value of f when considering small documents, which improves our

[2] We discuss the parameters for the algorithm in section 3.3.

[3] In our implementation, we define a fingerprint using the function $\sum_{i=0}^{n-1} s[i] * 31^{n-i-1}$ where s is a string of length n and $s[i]$ represents the ith character of the string. Our hash function is a bitwise exclusive-or operation between the fingerprint and permutation.

[4] We store sketches in a database to manage our large number of documents and to avoid a large memory overhead. We discuss this further in 2.2.3.

ID	Text	Original Document Hypothesis	Hash Collisions	N-Gram Overlap	Decision
1	A couple of capricious capybaras chatted coolly by the cactus, curiously considering another capy capably chewing on cantaloupe	N/A	N/A	N/A	Original
2	A pair of capricious capybaras chatted coolly by the cactus, curiously considering another capy capably chewing	1	5	0.79	Near-duplicate of 1
3	Yesterday, a pair of capricious pigeons prattled placidly by the cactus, curiously considering another pigeon capably pecking at cantaloupe	1	1	N/A - Insufficient collisions	Original
4	The pair of capricious capybaras chatted placidly by the cactus, curiously pondering another capy capably chewing on cantaloupe	1	4	0.33	Near-duplicate of 1
		3	1	N/A - Insufficient collisions	
5	The lazy llama lightly limped through the lilacs, laboriously longing for a lozenge	1,3	0	N/A - Insufficient collisions	Original

Table 1: Example of document processing with 4 documents, processed in the presented order. Parameters for this run are $n = 3$, $k = 1600$, $p = 5$, $t_{collision} = 2$, $f = 1.0$, $r_{overlap} = 0.2$.

results in real-world operation. None of the documents in the SpotSigs dataset were short enough to trigger this comparison, however, so we omit it from the algorithm above.

3. Analysis

We now measure how effective our system is at detecting near-duplicate web articles on-the-fly. We describe the dataset we used, propose our scoring metric, discuss the parameters that affect our algorithm's performance, and finally compare the results we obtained against others' similar systems.

3.1. Dataset

We evaluated our technique using the SpotSigs dataset provided by Theobald et al. (2008). The collection contains 2167 news articles divided into 68 uneven sets of near-duplicate documents, each of which contains one document marked as an original and some number of near-duplicates. Each document is an HTML file containing the text of the news article and any additional HTML related to formatting, layout, or non-article-related page content, such as headers, footers, navigation bars and advertisements. We ran our algorithm on a random permutation of the files in the dataset, where the only constraint on the random order was that the first document processed from each topic was the original.

3.2. Scoring

To score the effectiveness of our algorithm, we used a scoring metric that considered a classification correct so long as it linked the proposed document to the correct cluster. More specifically, we considered the classification of a document a true positive if it was correctly identified as being the duplicate of any other previously processed document in its cluster. A classification was considered a false positive if an original document was mistakenly classified as the near-duplicate of any other document, or if a document was classified as the near-duplicate of a document in the wrong cluster. A true negative was any original document correctly identified as such, and a false negative was any near-duplicate document incorrectly identified as an original. Furthermore, any document that could not be successfully processed was conservatively considered an original, which increased our number of false negatives.

Given the nature of our algorithm, the very first document processed is necessarily considered an original. Thus, our scoring schema also ignores this first true negative from the total score.

It should be noted that our system identifies if a given document is a near-duplicate as it is processed, and not after all documents have been considered. Our score is thus obtained in an online setting.

Finally, we should note that while the SpotSigs dataset is generally of very high quality, there are instances of separate clusters within the dataset which contain documents with near-identical article content. In order to compare ourselves fairly against other methods, we left these uncorrected within the dataset. This does however have the effect of creating a few cross-cluster false positives which a human might be inclined to judge as true positives instead.

3.3. Parameters

Table 2 summarizes our system's tunable parameters, and presents the list of values we explored for each. We tested 3-grams and 4-grams, observing that smaller n-grams tend

Parameter	**Description**	**Values tested**
n	The number of tokens in a shingle (section 2.2.1).	3, 4
k	The size of the random sample of shingles taken from the document from which we will generate the sketch (section 2.2.1).	1600
p	The number of static, randomly-generated numbers in $permutations$, which we use to generate a sketch (of size p) of a document (section 2.2.1).	20
$t_{collision}$	A threshold number of collisions among the values of the sketches of two documents. Two documents having more than this many collisions are considered near-duplicate candidates (section 2.2.2).	2
f	The portion (starting at the beginning) of two potential near-duplicate documents to compare directly (section 2.2.2).	0.4, 0.5, 0.65, 0.8, 1.0
$r_{overlap}$	The threshold minimum ratio of overlap between the n-grams of two potential near-duplicate documents; if the ratio is above this value, the documents are considered to be potential near-duplicates of one-another (section 2.2.2).	0.2, 0.3, 0.4, 0.5

Table 2: List of parameters involved in our near-duplicate detection system.

Value of n	**Value of f**	**Value of $r_{overlap}$**	**Precision**	**Recall**	**F$_1$-measure**
3	0.8	0.5	**0.991**	0.803	0.887
3	1.0	0.2	0.971	**0.940**	**0.955**

Table 3: Best results (in **bold**) generated by our system for precision, recall and F$_1$-measure, along with the parameters used to obtain them.

to produce more overlap by chance and larger n-grams tend to be more likely to be unique even when pulled from similar documents (e.g. the larger an n that is considered, the more n-grams are affected by a trivial single-word change in a document); 3-grams and 4-grams tend to be an ideal compromise for most language modelling tasks. We chose a value of k such that, for roughly 90% of our news articles, we are able to sample the entire article, but are intentionally trimming the small number of very long articles down to a more manageable size to increase processing speed. Our choice of p permutations was somewhat arbitrary, but the goal was to find the smallest number such that inspecting p n-grams for any random article would give us the gist of what the article was about. We chose $t_{collision}$ by repeatedly ingesting a set of documents and evaluating false positive rates – observing that e.g. two documents sharing only one out of twenty randomly-selected n-grams would happen by chance but be a poor predictor of similarity. We grid-searched values of f by observing that, at one extreme, some news articles (particularly short ones) are composed entirely of "new" news content, while, at the other extreme, some news articles start with a short paragraph of novel news content but end with roughly 60% background or context boilerplate text. We did not have a reasonable linguistically-motivated intuition for how to choose a value for $r_{overlap}$, so we chose equally-distributed values from 20% to 50% overlap, reasoning that any less than 20% was probably too dissimilar to consider. Finally, a note that as our system uses a heuristic-based model for near-duplicate detection (as opposed to a supervised machine learning model), setting these parameters does not induce bias or over-fitting; accordingly, we would expect these to generalize well for other data sets.

3.4. Results

In Table 3, we present selected experimental results with the parameter combinations that generated them. Our best F$_1$-measure (0.955) used $n = 3$, $f = 1.0$ and $r_{overlap} = 0.2$.[5] By performing a grid search over the three varied parameters however, we were able to empirically observe their impact on results. Depending on the application and the importance placed on precision and/or recall, one might be inclined to adjust the parameters to trade a little bit of one for the other. In particular, the threshold ratio of common n-grams ($r_{overlap}$) is a key parameter for both precision and recall. We observed that increasing this value generally lowered recall, but improved precision. This seems natural: the more of the document we consider in a direct comparison, the surer we can be of our prediction. This same comparison however, by being more stringent, reduces the number of correct near-duplicates identified. The next most impactful parameter is the portion of documents to compare directly (f), which, holding $r_{overlap}$ constant, has a substantial impact on recall. Increasing the value of f generally increases recall, whereas its effect on precision is less predictable, and interacts with the value of n. Finally, the value of n made relatively little difference; in general, we would expect higher values of n to have higher precision and lower recall, but the effect seems slight in practice. It should be noted that while we obtained a fairly constrained range of precision scores in our grid search, recall scores varied significantly. Our lowest observed precision score was 0.968 ($n = 4$, $r_{overlap} = 0.2$, $f = 0.65$) and highest

[5] As an easy-to-replicate evaluation, we also performed this grid search when processing the documents in depth-first alphanumeric order: for each topic in the dataset, we processed the original first and then all subsequent duplicates. This yielded comparable results: the best F$_1$ score was 0.950, with $n = 4$, $f = 0.8$ and $r_{overlap} = 0.2$.

Method	Precision	Recall	F_1-measure	Online
NRC-GPHIN	0.971	0.940	0.955	Yes
SpotSigs	0.96	0.92	0.94	No
SigNCD	0.95	0.89	0.92	No
ANDD-Raw	-	-	0.956	No
ANDD-LSH-Cos	0.965	0.923	0.943	No

Table 4: The results of our system (NRC-GPHIN) compared to the results reported by different near-duplicate detection systems on the SpotSigs dataset.

was 0.991, with a median of 0.983. Recall ranged from a low of 0.724 ($n = 4$, $r_{overlap} = 0.5$, $f = 0.4$) to a high of 0.940, with a median of 0.870.

4. Discussion

4.1. Method comparisons

Table 4 presents our best results compared to other systems that also evaluated their methods on the SpotSigs dataset. Note that because our system is online and selects only the most likely duplicate candidate, we do not perform pairwise evaluation in exactly the same manner (our evaluation metrics are presented in 3.2). This difference in comparison penalizes our performance results to a small extent, so we feel the comparison is still a fair one.

Our method outperforms SpotSigs (Theobald et al., 2008) and SigNCD (Zhang et al., 2016) on measures of precision, recall and accuracy. ANDD-Raw (Hajishirzi et al., 2010) slightly outperforms or is tied with our system in terms of F_1-score, however, this method is inapplicable in our context. The method requires keeping a large, uncompressed vector representation of documents in memory, which is infeasible for a real-time web application. When Hajishirzi et al. (2010) use a more computationally efficient version of their algorithm with hashed values (ANDD-LSH-Cos) instead, our proposed system outperforms it in terms of precision, recall and F_1-score.

Other algorithms have also attempted to tackle this problem, but have applied their work to different datasets, which they have generally generated automatically or semi-automatically for the purposes of their experiments. These include Gibson et al. (2008), who reported a very high F_1-score of 0.985. However, their method is offline, requires annotated data to train a supervised content classifier, and is also implemented completely in memory, all factors that make it impractical in our context – retraining a classifier each time a new document is added to the set isn't viable at scale. Mathew et al. (2011) and Arun and Sumesh (2015) both present very good F_1-scores for their algorithms (0.941 and 0.95, respectively). However, both methods rely on an HTML page's DOM to detect near-duplicates; our system presents slightly higher scores while also being independent of HTML page structure, making our method more suitable for real-world use over time. Finally, Ling et al. (2010) presents two algorithms for near-duplicate detection. One presents very high results (F_1-score of 0.967) but is dependent on regular fields found in documents (for instance, a "company name" field in a series of employment advertisements). As our documents do not necessarily contain such regular fields, the method is not directly comparable. An alternate version of their algorithm which does not assign priorities to these fields yields an F_1-score of 0.877.

4.2. Applications

We developed this near-duplicate detection system for use in the Global Public Health Intelligence Network (GPHIN) (Carter et al., 2018), a public health monitoring tool. GPHIN currently houses approximately seven million public health news articles, and continues to collect new articles in real time, which are then displayed to users. A near-duplicate detection system is necessary so as not to overwhelm users with bursts of redundant information. However, our particular domain requires that users be able to see all original documents – the domain is more tolerant of false negatives than of false positives, for fear of missing an early signal of an outbreak or natural disaster. Given this, we place a high importance on limiting our system's rates of false positives, so we tune our system as discussed in 3.4 to favour precision.

Because our system is put to use in the public health domain, we also leverage domain-specific factors to identify true near-duplicates and filter out false positives. Many news stories in the public health domain, particularly as it pertains to outbreaks of infectious diseases, evolve over time. News outlets may re-issue articles that are virtually identical on the surface, but with updated details, such as number of infected persons, or number of casualties. While these documents could often be classified as near-duplicates given their nearly identical content, they do provide new and useful information in the public health domain. As an extra step in our near-duplicate detection system, we use information extraction tools to obtain these counts from the text. Then, when documents are flagged as potential duplicates, we perform this additional comparison to ensure new information is considered original. While this is not reflected in our analysis, it is worth noting that the very definition of a near-duplicate may be domain-dependent and may require information extraction techniques in addition to pure near-duplicate detection.

4.3. Conclusion

We have presented a real-world implementation of a near-duplicate detection system that is fast, online, boasts state-of-the-art results and does not require a high memory overhead. As reported in 3.4, the method boasts very high precision scores, which can be traded for increased recall and F_1 scores, depending on the needs of the application. Our method of identifying near-duplicate news documents also functions independently from an HTML file's DOM, and can handle both HTML files or plain text as input.

5. Bibliographical References

Arun, P. and Sumesh, M. (2015). Near-duplicate web page detection by enhanced TDW and simHash technique. In *2015 International Conference on Computing and Network Communications (CoCoNet)*, pages 765–770. IEEE.

Blei, D. M. and Lafferty, J. D. (2009). Topic models. In *Text mining*, pages 101–124. Chapman and Hall/CRC.

Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G. (1997). Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8-13):1157–1166.

Broder, A. Z. (1997). On the resemblance and containment of documents. In *Compression and complexity of sequences 1997. proceedings*, pages 21–29. IEEE.

Broder, A. Z. (2000). Identifying and filtering near-duplicate documents. In *Annual Symposium on Combinatorial Pattern Matching*, pages 1–10. Springer.

Carter, D., Stojanovic, M., and de Bruijn, B. (2018). Revitalizing the Global Public Health Intelligence Network (GPHIN). *Online Journal of Public Health Informatics*, 10(1).

Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM.

Chatterjee, S. (2019). Rage within the machine: Brexit headline blizzard overloads FX algos. *Reuters*, Apr.

Chowdhury, A., Frieder, O., Grossman, D., and McCabe, M. C. (2002). Collection statistics for fast duplicate document detection. *ACM Transactions on Information Systems (TOIS)*, 20(2):171–191.

Gibson, J., Wellner, B., and Lubar, S. (2008). Identification of duplicate news stories in web pages. In *Proceedings of the 4th Web as Corpus Workshop*.

Hajishirzi, H., Yih, W.-t., and Kolcz, A. (2010). Adaptive near-duplicate detection via similarity learning. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 419–426. ACM.

Henzinger, M. (2006). Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 284–291. ACM.

Kołcz, A., Chowdhury, A., and Alspector, J. (2004). Improved robustness of signature-based near-replica detection via lexicon randomization. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 605–610. ACM.

Kusner, M., Sun, Y., Kolkin, N., and Weinberger, K. (2015). From word embeddings to document distances. In *International conference on machine learning*, pages 957–966.

Ling, Y., Tao, X., and Lv, H. (2010). A priority-based method of near-duplicated text information of web pages deletion. In *2010 IEEE International Conference on Software Engineering and Service Sciences*, pages 495–499. IEEE.

Mathew, M., Das, S. N., and Vijayaraghavan, P. K. (2011). A novel approach for near-duplicate detection of web pages using TDW matrix. *International Journal of Computer Applications*, 19(7):16–21.

Salton, G., Wong, A., and Yang, C.-S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

Theobald, M., Siddharth, J., and Paepcke, A. (2008). SpotSigs: robust and efficient near duplicate detection in large web collections. In *Proceedings of the 31st annual international ACM SIGIR conference on research and development in information retrieval*, pages 563–570. ACM.

Zhang, X., Yao, Y., Ji, Y., and Fang, B. (2016). Effective and fast near duplicate detection via signature-based compression metrics. *Mathematical Problems in Engineering*, 2016.