Bauhaus-Universität Weimar
Faculty of Media
Degree Programme Computer Science and Media

A Pipeline for Scalable Text Reuse Analysis with Applications to Wikipedia and the Common Crawl

Master's Thesis

Milad Alshomary

1. Referee: Prof. Dr. Benno Stein
2. Referee: Prof. Dr. Norbert Siegmund

Submission date: June 29, 2018

Declaration

Unless otherwise indicated in the text or references, this thesis is entirely the product of my own scholarly work.

Weimar, June 29, 2018

. . . . . .
Milad Alshomary

Abstract

This work aims to understand the text reuse phenomena in Wikipedia and to quantify the amount of Wikipedia content reused on the web. In particular our research questions are; (1) What kinds of text reuse cases are within Wikipedia articles? and (2) How much of the web content is just a reuse of Wikipedia text? Following that is the question of how much ad revenue is generated by this reused content.

To answer these questions, the first contribution of this thesis is building a framework for extracting text reuse cases from big datasets. The framework is a form of a pipeline of tasks that use heuristic based algorithms that run in a distributed manner on a cluster of machines. Using this framework we extracted 100 million text reuse cases from Wikipedia articles, and 1.6 million text reuse cases between Wikipedia and a sample of the web. Besides that, this framework could be further used in future work to extract text reuse cases from other big datasets.

The second contribution of this work is analyzing the extracted text reuse cases from Wikipedia. We describe two situations in which a piece of text is reused and provide simple heuristics to automatically classify text reuse cases.

Third, we present and describe the text reuse cases that were extracted in between Wikipedia and the web. We quantify the amount of reused content in a sample of the web and present a basic approach for estimating the ad revenue generated by reusing the free content of Wikipedia.

Contents

1 Introduction **1**

2 Background And Related Work **5**
2.1 Text reuse detection 5
2.2 On Wikipedia 7
2.3 Finding similar items in big datasets 7
2.4 Cluster computing 8
2.5 Data 9
2.6 Summary 10

3 Text Reuse Extraction **11**
3.1 Text Reuse Pipeline 11
3.2 Experiments on Candidate Elimination 15
3.2.1 Ranking Function 16
3.2.2 Hashing Methods 20
3.3 Text Reuse Pipeline on Wikipedia 24
3.4 Text Reuse Pipeline on Wikipedia and the Web 29
3.5 Summary 34

4 Text Reuse Analysis **36**
4.1 Text Reuse In Wikipedia 36
4.2 Text Reuse between Wikipedia and the Web 43
4.3 Text Reuse exploratory web tool 48
4.4 Summary 49

5 Conclusion and Future work **50**
5.1 Contributions 50
5.2 Future Work 51

A Text Reuse Cases **53**

Bibliography **58**

The roots of education are bitter, but the fruit is sweet.
– *Aristotle*

Acknowledgements

A very special gratitude goes out to Micheal Völske and Dr. Martin Potthast. Without their dedication and help, this work would have never been achieved. I am also grateful to the Web Information System group staff for giving me the opportunity to be part of their continuous ambitious research.

I would like to thank Prof. Dr. Benno Stein and Prof. Dr. Norbert Siegmund for accepting my work under their supervision.

To my life-coach, my partner Sarah Alburakeh: because I owe it all to you. Many Thanks!

I also thank my colleagues for being always around for help when it was needed. Namely: Payam Adineh, Masoud Allahyari, Arefeh Bahrami, Alexander Bondarenko.

And finally, last but by no means least, I am grateful to my siblings, my mother and father Salha Hamza and Zaed Alshomary, who have provided me through moral and emotional support in my life.

اخيرا وليس اخرا , اتوجه بالشكر لأخوتي, أمي و أبي, صالحة حمزه و زيد الشومري لما قدموه لي من حب و ثقه. من غير دعمكم لما حققت شيئاً

Thanks to everyone who made a mark in my life!

Chapter 1

Introduction

Text reuse is a broad concept that refers to any kind of reuse of a text, whether it's quoting, paraphrasing, translation or summarization. Text reuse phenomena became ubiquitous particularly with the rapid growth of the Internet and digital media. Potthast [2012] describes the current state of text reuse as an arms race between technologies that ease the access to text and information on the one hand, and technologies that detect text reuse on the other.

Building frameworks for extracting and analyzing text reuse is of high importance. It helps in both uncovering information flow between entities on the Internet as well as revealing the influence they have on one another. For example, Clough and Wilks [2001] analyzed the influence of the Associated Press in the UK on the state of journalism and how various news pieces could be rewritten under different conditions like author perspective, time available for edits and other factors. Citron and Ginsparg [2015] analyzed text reuse cases in the scientific community and how text is reused between scientific papers. In the eTRAP [1] Project, studying text reuse is used for better understanding the interaction between ancient societies. Plagiarism detection is another important application of text reuse detection. Potthast et al. [2013] give an overview of the plagiarism detection tools and methods that were proposed in the PAN [2] competition in 2013.

Wikipedia is a multilingual encyclopedia that forms a collaborative environment where the reader can also be an author and edit/add content. It gained a lot of success and grew rapidly (see Figure 1.1) to become a giant public source of information. A recent study by Thompson and Hanley [2017] showed how Wikipedia has a potential influence on the scientific community. Being free to

[1] Electronic Text Reuse Acquisition Project `https://www.etrap.eu/`

[2] `https://pan.webis.de/`

use and easy to access, Wikipedia became a source for text reuse as can be seen through the existence of many web sites that are merely a replica of Wikipedia content, though with an addition of ads. These ads generate revenue to these websites while Wikipedia gains nothing. This phenomenon was studied by Viégas et al. [2004] where they analyzed the effect between Wikipedia and other big websites. Driven by our curiosity, we formulate and investigate the following research questions:

1. What kinds of text reuse occur among the articles within Wikipedia?
2. How much of the web is just a reuse of Wikipedia content? Furthermore, how much revenue is generated - due to the ads in the reusing websites- from the free content of Wikipedia?

To answer these questions, the first contribution of this work is creating a framework for extracting text reuse cases from big datasets through cluster computing. The framework is in the form of a pipeline that consists of three main subtasks. The first subtask is text preprocessing, which takes the dataset as an input and produces a list of paragraphs represented as a feature vector for each document. The second subtask is candidate elimination. It takes two datasets of feature vectors (feature vector for each paragraph) as an input and produces candidate pairs of documents. In the final subtask, we perform a detailed text alignment between document pairs to produce the exact pieces of text that have been reused between any candidate pair of documents. Further details on the pipeline can be found in Section 3.1 and 3.2. We applied the framework to extract text reuse instances (1) within Wikipedia in Section 3.3 and (2) between Wikipedia and the Common Crawl [3] corpus as representative sample of the web in Section 3.4.

Using this framework we made the text reuse extraction task feasible on big datasets. Through a set of heuristics, the framework could be configured to solve a trade off between time of processing and accuracy of the results. We were able to make the computation time needed to extract text reuse cases within Wikipedia equal to 15 days and in between Wikipedia and the web (Common Crawl) equal to 200 days.

The second contribution of this work is towards analyzing the extracted text reuse cases in two situations: within Wikipedia itself and between Wikipedia and a sample of the web. In the case of Wikipedia's internal texts, we extracted around 100 million text reuse case generated by 360k documents, which makes up around 9% of Wikipedia. We provide insights on the possible causes that

[3] `http://commoncrawl.org/`

might lead to a text reuse situation and we identify 2 classes of text reuse; content reuse and structure reuse. We describe and characterize each class and provide basic heuristics for automatic classification. In the case of text reuse between Wikipedia and the web, we randomly sampled 10% of the Common Crawl and ran the pipeline in between the sample and Wikipedia. We extracted around 1.6 million text reuse cases. Around 16k web pages from this sample have a text reuse case with a Wikipedia article. We further analyze and explore these text reuse cases and give an estimation of the amount of revenue that is generated by reusing the content of Wikipedia.

To make it possible for further exploration and presentation of the text reuse cases in future work, the final contribution is a web tool that helps in exploring different dimensions of the text reuse cases that were extracted by the pipeline framework. More details on this tool can be found in Section 4.3.

The next chapter gives an overview of text reuse detection and the popular approaches that address this task. The third chapter outlines our approach and the pipeline architecture along with the experiments performed to construct the pipeline tools and their parameters. In chapter four, we present the results of applying the pipeline on Wikipedia text internally, as well as between Wikipedia and the web. The final chapter discusses our findings and offers insight into opportunities for potential future work.

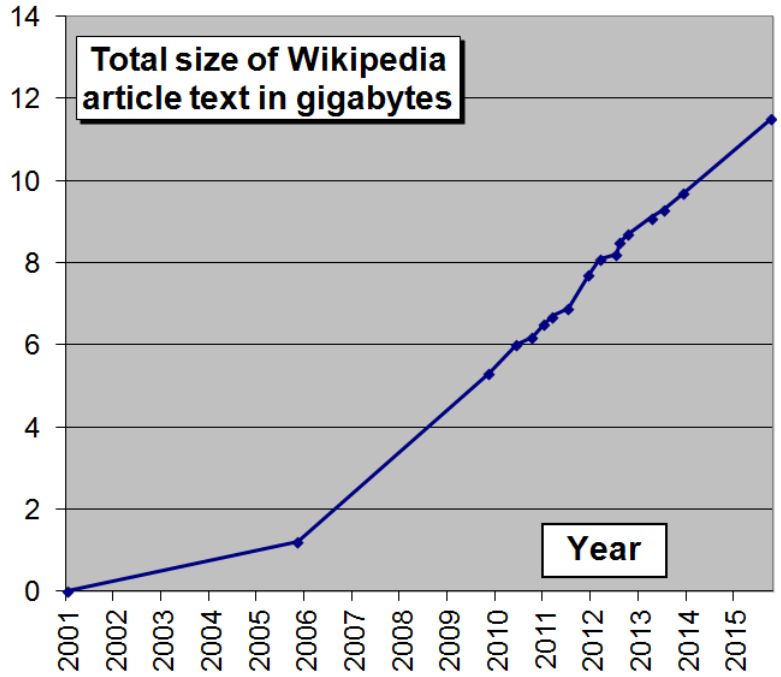


Figure 1.1: The growth of Wikipedia content over time measured in gigabytes. Commons [2018]

Chapter 2

Background And Related Work

In this chapter we will give a background and overview of related work. We start by presenting some of the work that has been done on text reuse detection and its applications. Second, we review some of the work that was attracted by the importance and controversy of Wikipedia. We conclude this chapter with a short survey of nearest neighbor search in big datasets and cluster computing.

2.1 Text reuse detection

Extracting and analyzing text reuse between entities helps in many applications. In the digital humanities, Büchler et al. [2010] used a text reuse detection framework to understand the interaction between the authors in ancient Greek literature. Clough and Wilks [2001] quantified the influence of the Associated Press in the UK on journalism by analyzing text reuse cases. Their work also helped in understanding the various ways of reusing texts and paraphrasing as subject to the perspective of the journalist, time available for edits and the space provided for the news in a journal. In the scientific community, text reuse happens very frequently between researchers due to the accumulative nature of knowledge. Citron and Ginsparg [2015] analyzed text reuse cases using the arxiv.org as a scientific corpus. They reported an expected negative correlation between the amount of text reuse in an article and its influence. Plagiarism is one form of text reuse, where the plagiarizer copies pieces of text from other sources without referencing them and tends to disguise the plagiarized text through different levels of paraphrasing and obfuscation. Potthast et al. [2013] give an overview on plagiarism detection methods that were proposed in the fifth international competition on plagiarism detection run by the PAN organization.

According to Potthast [2012], text reuse on the web happens when a user

searches the web for potential resources, copies the text from these sources and modifies it with their own words. This behavior defines the strategy on which any text reuse detection frameworks could operate. Given a suspicious document d_s and a collection D, first the framework searches the collection D for any potential resources that could be a source of the document d_s (source retrieval). Afterwards, a detailed examination (text alignment) is performed between the candidate documents and the suspicious document d_s to find any potential reuse of text.

Many researches addressed the source retrieval subtask as a form of an information retrieval task. Potthast et al. [2014] overviews the proposed methods for the sixth international competition of plagiarism detection. All methods followed a similar strategy. First the suspicious document is divided into chunks. For each chunk, key phrases are extracted using: TF-IDF, BM25 or head noun clusters. Later queries from the extracted key phrases are formulated and submitted to the search API. Finally some post processing is applied on the retrieved documents to keep only the candidate documents.
In contrast with the aforementioned retrieval approaches that depend on the semantics of a document, Stamatatos [2011] used only stopwords to represent documents. By building stopwords N-grams profiles for each document, the retrieval task of candidate documents works by computing the similarity between the profiles of documents. Bär et al. [2012] also showed that structural and syntactical similarities between texts could also be candidate features that help in detecting cases of text reuse. The Fuzzy Fingerprints method was proposed by Stein et al. [2007] as a direct method to compute binary value indicating whether two items are similar or not. Instead of computing pairwise similarity checks over every pair of items (which is rather costly), the method with linear time computes fingerprints for each item. Then collision between the fingerprints of the items is considered enough to deduce that they are similar.

As for the evaluation of the source retrieval task, the PAN competition suggests precision and recall taken from the field of Information retrieval. What is considered as a true positive is a retrieved document that has high jaccard similarity with the suspicious document.

In the text alignment task, Potthast et al. [2014] also reviewed the proposed methods. Mainly in all of them, three subtasks are performed. First is seed generation where similar substrings are to be located in each of the documents. Word-grams, stopword-grams or sentence matching were used as seeds. Second task is seed extension, which, given the matching seeds from the two documents, merges the close ones to create potential aligned passages be-

tween the source and suspicious document. Methods used in this step are rule based, clustering or dynamic programming. The final step is filtering in which further criteria are checked to eliminate for example short aligned passages or overlapping ones.

2.2 On Wikipedia

Wikipedia gained a lot of success and became a giant public source of information. The Wikipedia model and its success attracted a lot of research. In such an open environment, issues like vandalism and quality flaws are to be expected. Potthast et al. [2008] proposed a method for automatic detection of vandalism. Anderka et al. [2011] investigated and analyzed the quality flaw issue in Wikipedia. Weissman et al. [2015] analyzed duplicate and contradictory information in Wikipedia on the sentence level. They extracted sentences and clustered them into six types; Templates, Identical sentences, near duplicates with contradictory facts, Copyediting, References and others. Viégas et al. [2004] examined the interaction between Wikipedia and two big online communities (Stack Overflow and Reddite). They evaluated the amount of added value that Wikipedia contributes to those online communities by examining the posts on those communities that contain links to Wikipedia. As a result of their analysis, they point out that the posts that link to Wikipedia are exceptionally more valuable than other posts and generate a revenue value in the order of $100K per year. Another related and interesting work is done by Thompson and Hanley [2017], which shows through a controlled experiment how Wikipedia is playing a role in driving the research in the scientific community.

2.3 Finding similar items in big datasets

Searching for similar items in a dataset is the core of many applications in data mining, like finding near duplicate web pages or plagiarism detection. However when the size of the dataset gets larger or the cost of similarity computation is high (due to high dimensionality), the linear search of all possible similar pairs becomes overly time complex.
Methods of Nearest Neighbor Search (NNS) were proposed to accommodate the aforementioned problem. These methods could be of two classes.
The first class is the exact nearest neighbor search in which only the neighbor subspace of a document is searched. Bentley [1975] used a kd-tree data structure as an exact neighbor search method. This method constructs a binary tree data structure that partitions the search space leading to a search time of

O(log n). However, in case of high dimensional datasets, searching the exact neighbor turns on to be infeasible (Wang et al. [2014]).
The second class approximates the exact neighbor with a certain probability. Locality sensitive hashing (LSH) represents a family of hash functions that is able to hash similar items into the same hash code within certain probability bounds. One of the earliest implementations of LSH methods is minHash proposed by Broder [1997] which approximates the jaccard distance between two items. Andoni and Indyk [2006] proposed a random projection method that approximates the angular distance between items.
In contrast with LSH methods, learning to hash is a data dependent hashing approach. The goal is to learn from a dataset a hash function that is able to hash similar items into same hash code. Wang et al. [2018] suggest that any learning to hash method contains four main elements; (1) the hash function to be learned, (2) similarity measurement in the original space to be approximated, (3) loss function designed to minimize the gap between the approximated and original distance between pairs of items and finally (4) the optimization method which represents the method of updating the hash function's parameters.
An example of a learning to hash method is the work by Chaidaroon and Fang [2017], where they build a variational autoencoder neural network (Sonderby et al. [2016]) that learns from the data how to hash TF-IDF vectors into a binary code by optimizing the reconstruction error. In chapter 3 we present an experiment to evaluate (in terms of precision and recall of detecting text reuse cases) the two hashing schemes; random projection as a data independent hashing method and VDSH as a data dependent hashing method.

2.4 Cluster computing

Apart from working on efficient algorithms and enhancing their time complexity, another path to scale up computation performance is to distribute the work on multiple machines. However, distribution of the task comes with a cost of maintaining failures, parallelism and communication between machines.
MapReduce was first proposed by Dean and Ghemawat [2008] as a programming model and implementation that allows performing parallel computations on a cluster of machines. The framework manages on the low level all the cluster management issues while offering a high level interface that makes distribution of a task possible.
The main concept behind this paradigm is that any computational operation is split into two stages:

1. Map: the initial operation where the same function/code is performed

on each logical chunk of the distributed data to perform the intended transformation.

2. Reduce: the aggregation step where the data is grouped by a specific key and then a function is applied on the grouped data.

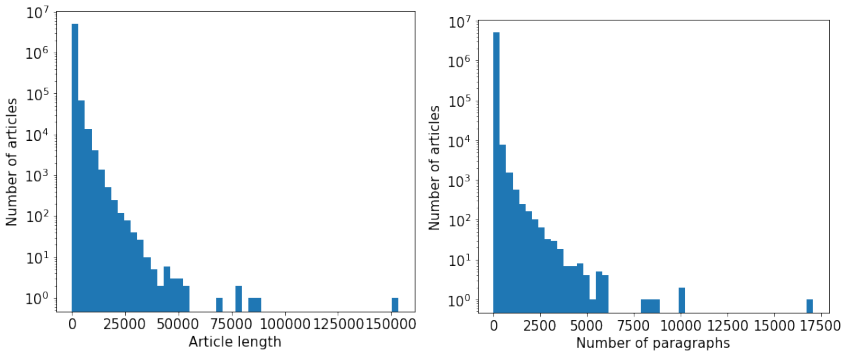
The Hadoop framework (White [2012]) is an open source project that adapts the mapReduce methodology and offers a framework for highly scalable distributed applications. Hadoop comes as two main components; mapReduce, same as Dean and Ghemawat [2008] and the HDFS (Hadoop distributed file system) component.
However, Zaharia et al. [2010] highlight some deficiencies of the Haoop framework and show classes of problems where the framework doesn't serve the intended purpose. In the same paper they propose a new concept called RDD (resilient distributed dataset) which is an abstraction of a resilient (fault tolerant that can be recomputed in case of failure) object that is partitioned on multiple machines. The data in an RDD is stored in memory (to reduce the access to the hard disk) and any operation performed on an RDD could be either map or reduce and is performed on parallel.

2.5 Data

In this work we use the English Wikipedia dataset that was released by Wikipedia foundation on May 2016 [1]. The dataset is stored as a compressed XML file that contains all Wikipedia articles. The content of each article is stored as Wikitext [2] which consists of syntax and keywords for formatting the page content. We use an open source tool [3] to extract the article content from the Wikitext. The tool outputs an xml file that contains around 5 million articles. Each article contains an Id, title and text content. In Figure 2.1, we show the distribution of article length in tokens and also the distribution of number of passages (the passages are generated from the tool based on the provided Wikitext) per article.

As a representative sample of the web we use the Common Crawl repository which contains 7 years of web content that was crawled by The Common Crawl Foundation [4]. The dataset is hosted on Amazon S3 as part of the Amazon public datasets program [5] and contains 5 billion web pages of different

[1] `https:www.wikimedia.org`
[2] `https://en.wikipedia.org/wiki/Help:Wikitext`
[3] `http://attardi.github.io/wikiextractor/`
[4] `http://commoncrawl.org`
[5] `https://registry.opendata.aws/commoncrawl/`



(a) **(b)**

Figure 2.1: Figure (a) Distribution of number of passages per article. Figure (b) Distribution of passages length.

languages. The dataset we work on was pulled on April 2017. To extract the main content as text from the HTML page we use the Boilerpipe framework based on the work by Kohlschütter et al. [2010]. In chapter 3 we present in details the process of extracting the main content and we describe the data we worked on.

2.6 Summary

In this chapter we gave an overview of related work in the field of text reuse extraction and applications in which the text reuse extraction can play a role. We presented interesting research questions that evolves around Wikipedia indicating it's importance and controversy. Later in the chapter we showed method of scaling up the computation of finding similar items either by special heuristic based algorithms (section 2.3) or by utilizing a cluster of machines to distributed computation in parallel (section 2.4).

Chapter 3

Text Reuse Extraction

In this chapter we describe the text reuse extraction pipeline. In the first section we present the proposed structure in high abstraction along with required theory and techniques. The second section contains the experiments we performed to figure out the best methods and parameters for the pipeline implementation. In the last two sections we present details on how the pipeline was used in both scenarios:

1. Within-collection text reuse extraction, in which the pipeline is applied on one dataset to extract the text reuse cases between it's documents. Found in section 3.3

2. Cross-collection text reuse extraction, in which the pipeline is applied on two datasets to extract the text reuse cases between documents from two datasets. Found in section 3.4

3.1 Text Reuse Pipeline

The main goal of the text reuse extraction task is to take two collections of documents as an input and output all possible pairs of text reuse instances between documents from these two collections. The task could be formulated as follows:
Given: two collections of documents $D1$ and $D2$, the task is to find all pairs of chunks of text: t_i x t_j where: $\exists d_1 \in D1, d_2 \in D2 : t_i \subset d_1 \wedge t_j \subset d_2$ and the two chunks t_i and t_j overlap with a minimum percentage that make them candidates to be a text reuse case. Nevertheless, $D1$ could be equal to $D2$ and consequently the task becomes within-collection text reuse extraction from a dataset.

To carry out this task we divide it into three subtasks (Figure 3.1); First, content from documents is extracted and cleaned then text is represented as

a feature vector. Second, heuristic based measurements are computed on each pair of documents to compute the likelihood of having text reuse cases. The main goal of second task is to filter out those pairs of documents that are not likely to have text reuse. Finally, detailed text alignment is performed on the candidate pairs from the previous task to output pairs of text reuse.

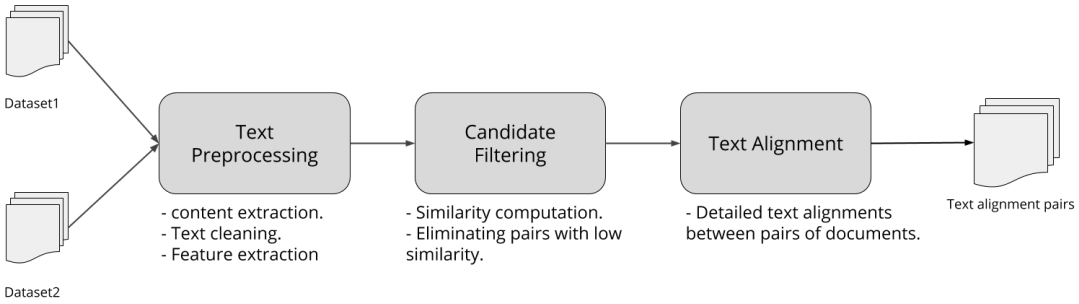


Figure 3.1: Pipeline of subtasks for text reuse extraction

The first subtask takes as an input a collection of documents D and outputs for each document a list of text chunks (passages). Each of them is represented as a feature vector. Two main steps are performed in this subtask; the main content extraction and text feature extraction.
In the first step, given a document the main useful content is extracted and split into chunks/passages of text. This step requires tools and heuristics for better identification of the main content of a document which becomes more challenging in case of web content. We explain more in details the heuristics and tools that were used for Wikipedia in Section 3.3 and the Common Crawl in section 3.4.
In feature extraction, given the text of a chunk the task is to extract text features and represent the chunks as feature vectors. In section 3.2 we present our experiment setup in which we evaluate set of text representation methods based on TF-IDF weighting scheme, word embedding [1] , and stopword N-grams. To this end, each document in the dataset is represented as a set of feature vectors (Figure 3.2).

The second subtask takes two collections of documents, in which each document is a set of feature vectors. It outputs a data structure that contains only those pairs of documents that are likely to be reusing text from each other.

[1] `https://en.wikipedia.org/wiki/Word_embedding`

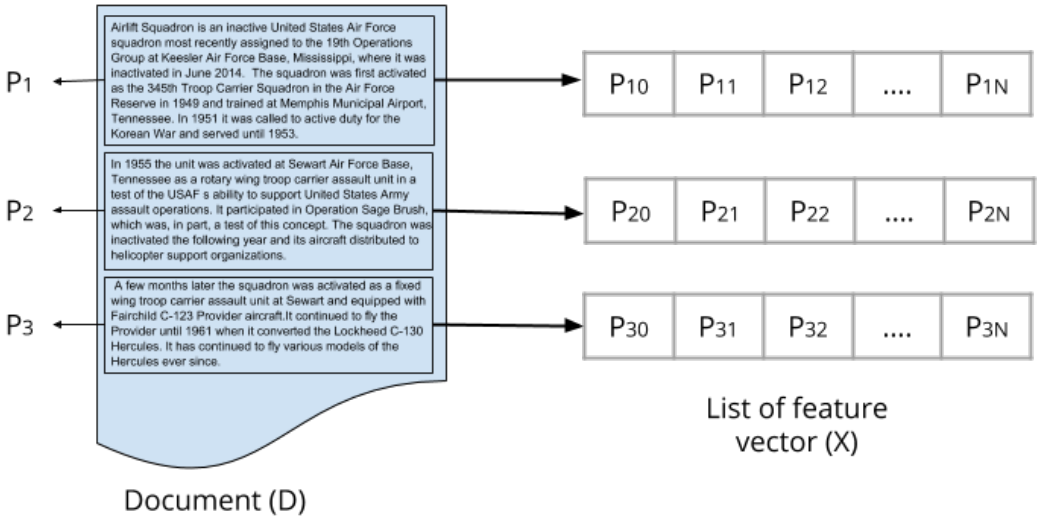


Figure 3.2: The output of the preprocess step is for each document in the collection a set of feature vectors. Each feature vector is a representation of a chunk/passage of text

The decision of whether two documents are candidates is based on the similarity between their chunks. Intuitively two documents are considered candidates if they have at least one pair of chunks whose similarity exceeds a specific threshold. Other considerations could be that the similarity of two documents is an average over the similarity of each pairs of their chunks.
The implementation of this subtask requires three main decisions to be made: (1) A similarity function between two chunks of text.(2) A formula to compute the candidacy score (that reflects the likelihood of having text reuse) between two documents from the similarity of their chunks. (3) A proper threshold of the candidacy score between two documents to be considered candidates.

However, performing pairwise similarity computation on each pair from the two collections is an expensive task especially for large datasets. To address this issue, we use the concept of hashing for near neighbor search. Instead of assessing the candidacy score for each document against all other documents, we only assess a subset that with some probability contains the candidate documents. We apply some similarity preserving hashing function on each chunk to produce a short binary code. The similar chunks would be hashed to the same hash code and following that documents who share similar chunks would intersect with at least one of the hash codes. We aim to design a hash function that guarantees all candidate pairs would at least intersect in one hash code

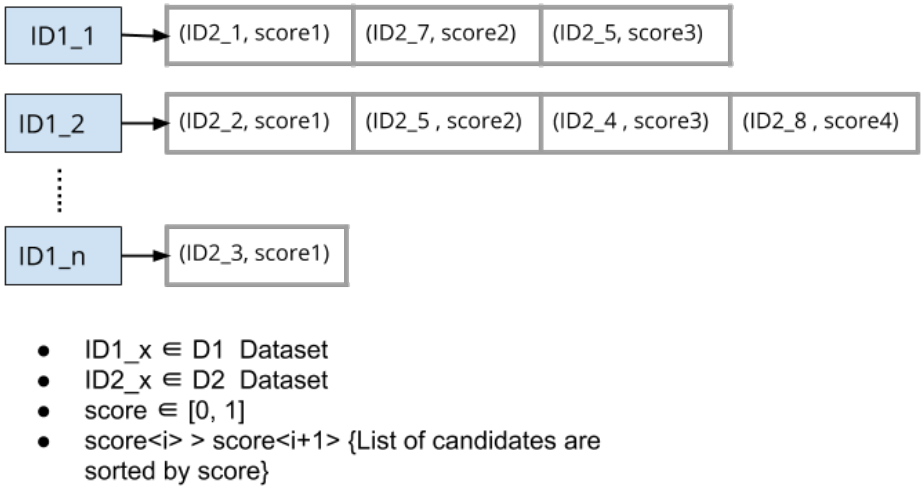


Figure 3.3: An example of the output RDD structure of the candidate elimination subtask.

while eliminating a lot of false positive cases.
We use the hashing step as the first step in the second subtask (optionally). After applying hashing on each chunk, each document has a set of hash codes. Hence, we only compute the pairwise candidacy score between document pairs that intersect in at lease one hash code. In section 3.2.3 we present an experiment in which we assess and compare between two known similarity hashing techniques; random projection as an implementation of the LSH family and variational deep semantic hashing (VDSH) by Chaidaroon and Fang [2017] which is considered as a data dependent learning to hash method.

The third subtask takes the candidate pairs as an input and produces pairs of text reuse cases. The input data structure as illustrated in Figure 3.3 is an RDD where each item contains a document id from dataset D_1 and a list of all candidate documents from the second dataset D_2. The candidates are sorted by their candidacy score which allows setting up a threshold or stopping criteria where the scan starts with the candidates with high score and continues till reaching that threshold or stopping criterion.
For each candidate pair of documents, a detailed examination of the text is performed to locate any span of text that could be a text reuse case. To locate a text reuse case, three main steps are proposed by Potthast [2012] which we summarize in the following:

- Seed Generation, in which both texts are tokenized into words, sentences or characters, N-grams of these tokens are extracted from each of the doc-

uments. Later similar N-grams from the two documents are identified.

- Seed Extension, where the identified seeds in both documents are extended to make up passages of text reuse. The extension could be performed by creating clusters of seeds and considering them as one piece of text.
- Post filtering of the extracted text reuse cases. Given two passages of texts (considered a reuse case), filters are applied to decide whether these two passages represent a text reuse case.

3.2 Experiments on Candidate Elimination

Since we are focused in this research on addressing the challenge of extracting text reuse cases from big datasets, we fixed the parameters of the third subtask as recommended by the PicaPica framework and use it as ground truth while we experimented with different heuristics and implementations for the second subtask to generate candidate pairs with high precision and recall while keeping the computation time feasible.
In this section we will present two experiments. The first experiment was performed to choose a proper candidacy function that reflects the likelihood of two documents having a text reuse case. The second experiment was performed to evaluate proposed semantic preserving hashing methods in the context of text reuse extraction.

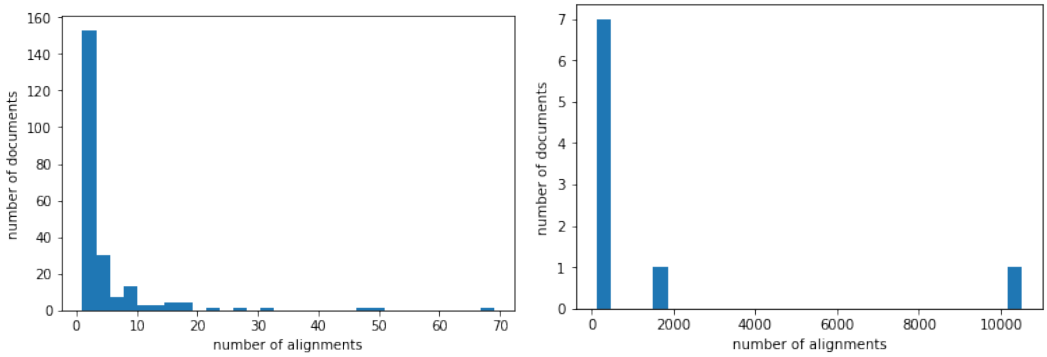
To setup the experiment framework we give the following definitions:

- D is a set of documents
- C is a set of document chunks resulted from splitting documents in D into chunks/passages.
- S is a sample drawn from D , in which the following holds:

  $\forall s_i \in S, \exists D_i^{'} \subset D \wedge D_i^{'} \neq \Phi : D_i^{'}$ is a set of documents that have text reuse with s_i

In our experiment, D is the Wikipedia dataset and S is constructed by first filtering in only Wikipeida documents that are longer than 2000 tokens (long documents are more likely to have been reused by others) and randomly sampling 10000 documents. Then using PicPica framework we compute pairwise text alignments between the sample and all the Wikipedia dataset. Out of 1000 documents 232 found to have text reuse cases at least with one document in Wikipedia dataset. Those 232 documents is our sample S. For each

document s_i in this sample there is a set $D_i^{'}$ containing all other documents from Wikipedia that have text reuse case with s_i and we refer to them as relevant documents. In Figure 3.4 we show the distribution of number of relevant documents. It is observable that over 90% of the sample S have only few relevant documents (< 10) out of the whole dataset D. That said, constructing heuristics to identify those few relevant documents is a challenging task. Using only the semantic similarity (bag of words representation) between documents as a measurement of likelihood of text reuse is not enough. While there are many documents that are semantically similar, only few have text reuse cases. Hence, only semantic similarity would lead into a lot of false positive cases.



(a) **(b)**

Figure 3.4: Figure (a) Distribution of number of relevant documents less than 100. Figure (b) Distribution of number of relevant documents more than 100.

3.2.1 Ranking Function

The goal of this experiment is to find a proper method to be used for filtering in only pairs of documents that are likely to have text reuse cases. Following the definitions presented in previous section we further define:

- $cand(d_i, d_j) \rightarrow \mathbf{R}$ as a function that takes two documents and generates a score that indicates the likelihood of finding text reuse cases between these two documents.
- $rank(s, D)$ a function that returns for each $s_i \in S$ a list R_i of all $d_i \in D$ ranked by the score returned by applying $cand(s_i, d_i) \rightarrow \mathbf{R} : di \in D \wedge s_i \in S$

To evaluate the proposed *cand* functions, We apply the *rank* function on our Sample S and Wikipedia D. As a result we get for each document s_i in

S a sorted list of all Wikipedia articles based on the likelihood of having text reuse case. To compare the performance of the proposed *cand* functions we can use the following evaluation measurements:

- Average max rank which can be defined in the following equation:

$$\frac{\Sigma_{si \in S} max(R_i | r \in D_i^{'})}{|S|} : R_i = rank(s_i, D)$$

  Which is the average rank (across the sample) of the last relevant document in the ranked list. That reflects on average how many documents in the ranking list we need to perform detailed text alignment on.

- Precision/Recall curve: for each $s_i \in S$ and R_i we compute the Precision/Recall curve by computing the precision and recall values on a set of thresholds along the ranked list R_i. At each threshold, precision is the number of retrieved (have a rank less than the considered threshold) and relevant documents divided by the threshold rank while recall is the number of retrieved and relevant documents divided by the number of relevant documents. The overall Precision/Recall curve is an average the Precision/Recall curves of all $s_i \in S$

We evaluate four *cand* functions; cosine similarity between vectors in the TFIDF space, Jaccard similarity of stopword N-grams, consine similarity in paragraph embedding space and finally weighted average of paragraph embedding and stopword N-grams. In all of the three methods, the *cand* function is applied on the document chunk level and then the candidacy score of two documents is computed as following:
$cand(d_i, d_j) = max(sim(c_i, c_j)) : c_i \in d_i, c_j \in d_j$ where sim is one of the aforementioned similarity methods.

To compute the cosine similarity in the TF-IDF space, We first represent each document chunk $c \in C$ as a TF-IDF vector. To do so, we tokenize the text into words, remove stopwords and then compute the IDF (inverse document frequency where the document here is a document chunk) for each of the tokens (t) (Equation 3.1). Later each document chunk is represented as a vector of all tokens where the weight for each token is presented in Equation 3.2. Then the cosine similarity between the two vectors V_1 and V_2 is given in Equation 3.3.

$$IDF(t, C) = log \frac{|C| + 1}{DF(t, C) + 1} \tag{3.1}$$

$$TFIDF(t, c, C) = TF(t, C) * IDF(t, C) \tag{3.2}$$

$$similarity(V_1, V_2) = \frac{\Sigma_{i=1}^{n} V_{1i} * V_{2i}}{\sqrt{\Sigma_{i=1}^{n} V_{1i}^2} * \sqrt{\Sigma_{i=1}^{n} V_{2i}^2}} \tag{3.3}$$

In Jaccard similarity of stopword N-grams, we use the implementation proposed by Stamatatos [2011]. We compute the top frequent stopwords in Wikipedia shown in Figure 3.5. Then for each document chunk we apply the following; (1) We tokenize the text, (2) keep only the top frequent stopwords, and (3) generate N-grams of stopwords which represent the chunk (called SWNG profile). Later, the similarity between two chunks is measured using Jaccard similarity (Equation 3.4) between the two N-gram profiles.

$$J(SWNG1, SWNG2) = \frac{|SWNG1 \cap SWNG2|}{min(|SWNG1|, |SWNG2|)} \tag{3.4}$$

We experimented with (N) values of (5, 8, 10) of the N-gram. We observed that some document chunks are short and increasing the value of N causes some chunks to have an empty profile (no N-grams). To address this issue, we also computed Jaccard similarity over two N-grams schemes and created a weighted sum of the two values. We experimented with different weighting schemes and the best mixed scheme is given as: $w1 * J_{8-grams} + w2 * J_{10-grams}$ where w1=0.8 and w2=0.2.
In Figure 3.6 (b), we show the Precision/Recall curves for all the proposed stopword N-grams methods. The best performance is for the mixed stopword N-grams of 8 and 10 grams.

the, of, and, in, a, to, was, is, for, as, on, with,
by, s, he, that, at, from, his, it, an, were, are, which,
this, be, also, had, or, has, first, their, its, one,
after, but, new, who, not, they, she, have, her, two, been,
when, other, during, all, into

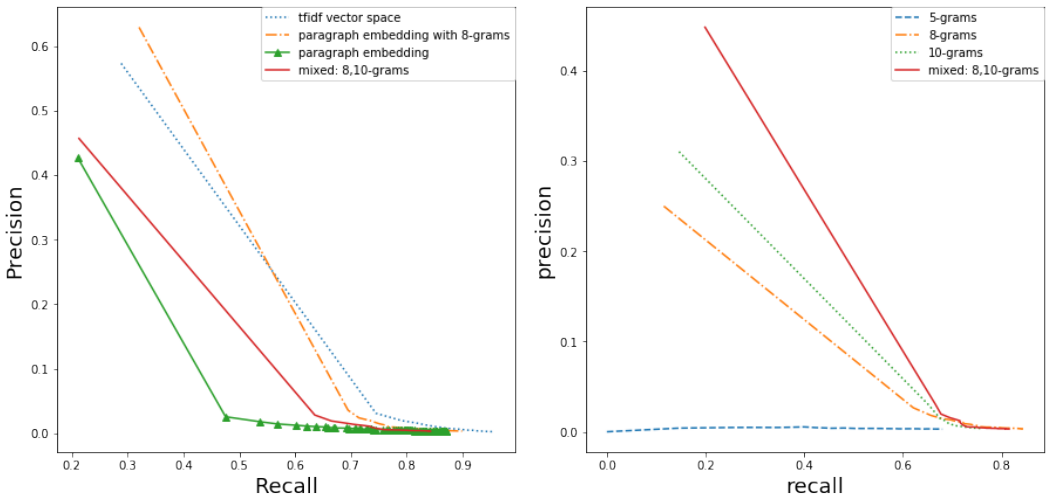
Figure 3.5: List of top frequent stopwords in Wikipedia.

The concept behind paragraph embedding is to represent each document chunk as a vector in a word embedding space. The space is generated by using a skip-gram word embedding implementation [2] where each word is mapped into a continuous vector space. We trained the model over the articles of Wikipedia.

[2] https://spark.apache.org/docs/2.2.0/mllib-feature-extraction.html#word2vec

Then, each document chunk is mapped into a vector in the embedded space as following:(1) The document chunk is tokenized.(2) Stopwords are removed. (3) An averaged vector is computed from the vector representation of each of the tokens in the chunk. This final vector is the representation of the document chunk in the embedded space. In our experiment we choose 100 as the number of dimensions of the embedded space.

In paragraph embedding with stopwords N-grams, the similarity score of two document chunks is a weighted average of two similarity scores; Cosine similarity of the paragraph embedded vectors and Jaccard similarity over stopwords 8-grams.



(a) **(b)**

Figure 3.6: Figure (a) precision/recall curve for the 4 proposed methods. Figure (b) precision/recall curve to compare various n values for stopwords ngrams. The precision/recall points are computed at thresholds of [1, 101, 201...,100k]

In Figure 3.6 (a) We show the Precision/Recall curves for the proposed methods. The poor performance of paragraph embedding method comes from the fact that it represents the similarity on a higher semantic level which generates many false positives (cases where two chunks are semantically similar but don't represent a text reuse case). By mixing the paragraph embedding score (semantic features) and the stopwords N-grams score (structural features) as a weighted sum we achieved higher precision than all other methods for recall around 0.4. However, the cosine similarity as a similarity function performs the best in terms of precision for recall higher than 0.6.

In Table 3.1 we present some statistics for each of the evaluated methods

that give insights on how they perform. The first column in the table is the average maximum rank which was defined previously as an evaluation measurement. The second column represents the minimum similarity value in the ranked list (the similarity value of the last relevant document in the ranked list) averaged across the sample. This value could be used as a threshold for similarity between two documents to consider them candidate pairs. The last column is the minimum similarity value for each sample instead of averaging. Even though the Precision/Recall curve for the cosine similarity is better than the mixed N-grams method (Figure 3.4 (a)), we observe from Table 3.1 that the average max rank for the mixed N-grams performs slightly better than the cosine similarity. That said, on recall equal to 1 (all relevant documents are detected) the two methods perform almost the same. However, for lower recall thresholds less than 1 the cosine similarity over TF-IDF space performs better.

	avg(max(rank()))	avg(min(similarity()))	min(min(similarity()))
cosine similarity	12250	0.49	0.025
mixed ngrams	11554	0.12	0.0
paragraph embedding	39873	0.89	0.63
paragraph embedding with 8-grams	33552	0.38	0.25

Table 3.1: Comparing the proposed candidate elimination methods in terms of; average maximum rank, average minimum similarity, minimum minimum similarity.

3.2.2 Hashing Methods

Given the two datasets D1 and D2 as an input for the second stage in the pipeline, the task is to apply the *cand* function on every pair d_1 x $d_2 : d_1 \in D1 \wedge d_2 \in D_2$. The time complexity of running this task is $|D1| * |D2|$ and for big dataset the time needed for running the task becomes infeasible. To address this challenge, we use H a semantic preserving hashing method. Applying H on a document d produces a set of binary hashes (binary hash for each chunk in the document). The best hashing method H satisfies the following:

- Any pair of documents from the two datasets which have text a reuse case would have at least one binary hash in common:
  $\forall d_1 \in D_1 \wedge d_2 \in D_2 : d_1 \wedge d_2$ have text reuse $\equiv H(d_1) \cap H(d_2) \neq \phi$
- The number of pairs of documents that have at least one binary hash in common is smaller than the number of all pairs from the two datasets:
  $|X| << |D_1| * |D_2| : X = \{(d_1, d2) : d_1 \in D_1 \wedge d_2 \in D_2 \wedge H(d_1) \cap H(d_2) \neq \phi\}$

As a result of using a hash method that satisfy the previously mentioned criteria, we only run the *cand* on smaller number of pairs of documents (those who intersect at least in one binary hash).

To evaluate the proposed hashing functions, we use the same experiment setup already introduced in section 3.2. Given $s_i \in S$ and D'_i applying $H(d_i)$ hashing function generates a set of binary hashes $\{h1, h2..., h_k\}$. A true positive case is any document $d'_{ij} \in D'_i$ that has at least one binary hash in common with s_i binary hashes. We compute precision and recall for every $s_i \in S$ and we average the values over all the documents in the sample S.

We experimented with two methods of hashing; (1) Random projection as one of the implementations of the locality sensitive hashing family (LSH) which is considered as a data independent hashing method. (2) Variational deep semantic hashing (VDSH) by Chaidaroon and Fang [2017] as one of the data dependent hashing technique.

Random projection works by constructing L hash functions, each of the form: $g_i = (h_{i,1}, h_{i,2}, ...h_{i,k})$ where $h_{i,j} : R^m \rightarrow \{0, 1\} = sign(V_i * X)$ is a function that takes a point in the original space X of m dimensions and returns 0 or 1 (By applying the *sign* function on the result of multiplying the point X with V_i). In other words, each of the $h_{i,j}$ represents a unit vector in the original space and to compute g_i for a chunk (point in the original space) the point is projected on the K vectors. It is proven that the close points in the original space (similar chunks) would be projected into same binary hash with certain probability. However, to amplify the chance that the similar points in space have same binary hashes, the process of generating g_i is repeated L times. For the random projection hashing function, two parameters (L and K) need to be calibrated to find the best configuration. Increasing K (length of the binary hash) leads to high precision and low recall while increasing L (number of binary hashes) we increase the recall and reduce the precision. In Table 3.2 we present the evaluation results of different K and L parameters. Its observed that choice between different configurations is a trade off between precision and recall. While we aim for high recall close to 1.0 we still need a good precision that guarantee fewer false positive cases.

The VDSH method proposed by Chaidaroon and Fang [2017] is a variational autoencoder neural network that learns to hash vectors from the original space (TF-IDF) into latent short vectors that follows roughly a normal distribution. The architecture of the neural network (NN) is shown in Figure 3.7. It consists of two components. The first component is the encoder in which the first two layers (L1, L2) have ReLu activation functions and the Layers L31

K	L	precision	recall
8	2	0.000031	0.8741
8	6	0.0000156	0.994
16	4	0.0000989	0.324

Table 3.2: Different configurations of the random projection hashing method and the performance of each in terms of precision and recall

and L32 are the encoded sigma and mean of the learned latent vectors distribution. The second component is the decoder which takes a sample s from the learned normal distribution and decodes it back into $X^{`}$ in the original space.

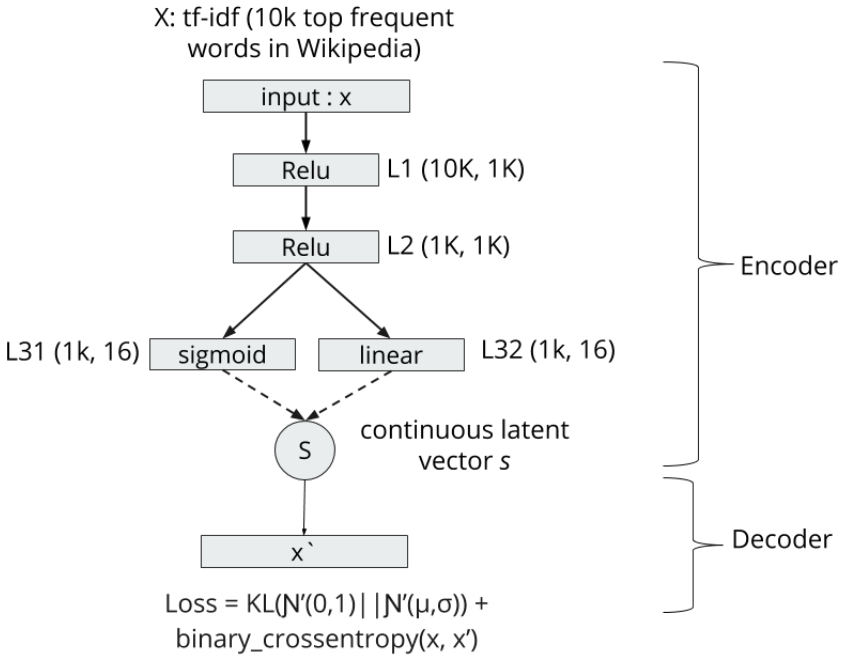


Figure 3.7: VDSH neural network structure

The loss function of the neural network is a sum of two losses. The first is the reconstruction error which is the difference between the original vector X and the reconstructed one by the decoding layer $X^{`}$. The second is measured

as the KL-divergence [3] between the latent vector represented by $N(\mu, \sigma)$ and the normal distribution $N(0, 1)$. For a vector in the original space of N dimensions, the decoder component of the model outputs a continuous latent vector of dimension $k << N$. The last step is to convert the continuous latent vectors into a binary vector. For this we use the same thresholding method used in the paper and the final binary vector is the binary hash used to represent a document chunk.

To prepare the training data for the model, we use the Wikipedia dataset. We first filter out short articles (less than 2k tokens) and randomly sample 10% of the articles. That generates 300k document chunks. We preprocessed the text of each chunk by applying the following: (1) Removing stop words and all non alphanumeric tokens. (2) Stemming all tokens using an implementation of snowball lemmatizer [4]. (3) take top 10k frequent words in the sample. (4) Fit a TF-IDF model on the sample and convert all chunks into vectors in the 10k TF-IDF space. However, to reduce the time needed to train the model, we choose randomly only 100k document chunks from the whole sample to train the model.
Since the VDSH hashing function is a neural network, many parameters could be calibrated to enhance the performance in terms of precision and recall. In our experiment we choose to fix all the parameters as recommend by Chaidaroon and Fang [2017] and we explore different values for the size of the hidden layer (L1, L2) and the size of the latent vector.

	size of hidden layer	size of latent vector	hamming distance	precision	recall
1	500	32	0	0.0007	0.13
2	500	32	1	0.000078	0.51
3	1000	32	0	0.0007	0.13
4	1000	32	1	0.0001	0.48
5	1000	16	0	0.00045	0.73
6	1000	16	1	0.000068	0.87

Table 3.3: Different configuration of the VDSH neural network and the performance of each in terms of precision/recall

To increase the recall, besides considering document chunks in the same binary hash to be similar we also consider chunks in similar binary hashes. Hamming distance is a measurement of how close two binary vectors one to each other.

[3] `https://en.wikipedia.org/wiki/Kullback-Leibler_divergence`
[4] `http://snowballstem.org/`

By increasing the Hamming distance, we could increase the recall. However, that could also cause a decrease in the precision since the likelihood of including false positives increases. In this experiment we explored two different setups; Hamming distance of 0 (exact match) and Hamming distance of 1. In Table 3.2, we present the performance of few of the trained models with different configuration.

Choosing between the proposed methods and their configuration is a problem of trade off between precision and recall. We want to use the hashing method that gives high recall close to 1. However increasing the recall by reducing the K in the random projection or increasing hamming distance in the VDSH comes with a cost of harming the precision. For the implementation of the pipeline we choose to use the VDSH with the configurations mentioned in row 5 of Table 3.3. By experimenting on the Wikipida sample the chosen model gives considerably high recall of 0.73 and still accepted precision of 0.00045. In section 3.4 we present how much computation we could save by using the VDSH model.

3.3 Text Reuse Pipeline on Wikipedia

In this section we present in detail the implementations of each of the subtasks in the pipeline that were used to extract the text reuse cases from within Wikipedia. We run the pipeline on the version of Wikipedia that was released in May 2016. All the computations were performed on the Betaweb cluster [5] using Apache Spark framework and all the data files are stored as HDFS files on the cluster.

Text preprocessing In this subtask, we first extract the main content of each article from the wikipedia syntax using an open source tool [6]. The output of the tool is a set of text chunks (passages) for each article. However, we perform post processing on this output to further eliminate short chunks by removing them or merging with other longer chunks. It was observed that the tool generates list's bullet points in Wikipedia syntax each as a standalone chunk. Most of these chunks are shorter than one sentence and represent names of cities or other entities. So as the second step we filter out all these chunks of type bullet points. Third, we run Algorithm 1 which is a heuristic base algorithm that attempts to merge short chunks with their neighboring longer ones. However, for some chunks the algorithm fails in merging them.

[5] The cluster owned by the Web information systems in Bauhaus University and consists of 130 nodes each of has 12 cores and 192GB memory

[6] `http://attardi.github.io/wikiextractor/`

Those represents headers of lists, disambiguous, and redirection pages. Hence, as a final step we remove all these short chunks that failed to be merged.

Algorithm 1 Document chunks merging

```
procedure BALANCE_PARAS_LENGTH(paras, min_no_tokens)
    output_paras ← []
    APPEND(output_paras, paras[0])
    last_para_idx ← 0
    curr_para_idx ← 1
    while curr_para_idx ≤ length(paragraphs) − 1 do
        curr_paragraph ← paragraphs[curr_para_idx]
        last_output_para ← output_paras[last_para_idx]
        if length(curr_paragraph) ≤ min_no_tokens ∨
length(next_paragraph) ≤ min_no_tokens then
            CONCATENATE(output_paras[last_para_idx], curr_paragraph)
        else
            APPEND(output_paras, curr_paragraph)
            last_para_idx ← last_para_idx + 1
        end if
        curr_para_idx ← curr_para_idx + 1
    end while
    return output_paras
end procedure
```

As a result of the above process, we have a collection of articles each of is split into a set of text chunks. Next step is to represent each chunk as a feature vector. From section 3.2.1, we found out that TF-IDF weighting scheme performs relatively better than the other methods. Hence, we use it as an implementation of the candidacy scoring function in the pipeline.
In the feature extraction step we tokenize the text of each chunk into words and

	Number of paragraphs	Paragraph average length
Wikipedia dump output	73 million	93
Filter out lists bullet points	34 million	53
Paragraph merging	12 million	137
Filter out short paragraphs	11.4 million	144

Table 3.4: Statistics on the Wikipedia corpus after performing each step of the content extraction and chunking

remove stopwords. Then, we fit a TF-IDF model over the collection same way as in the experiment and use it to transform all chunks into the equivalent TF-IDF vector. The final output of this subtask is an RDD (resilient distributed dataset) of elements each contains: $sequence\_id$ (unique auto increment identifier), $wiki\_id$ (wikipedia documetn id) and a list of $tfidf\_vector$. We persist the RDD as an HDFS file on the cluster.

Candidate elimination Given as an input one collection of documents as an RDD, the task is to compute a pairwise score of all documents using the *cand* function and only keep the pairs that have candidacy score above certain threshold. We found by experimenting in section 3.2.1 that the cosine similarity to be performing relatively better than other methods. Hence, the implementation of the *cand* function in this subtask is as following:

$$cand(d_i, d_j) = max(cosine\_similarity(p_i, p_j)) : p_i \in d_i, p_j \in d_j$$

To estimate the time needed to process all the pairs of Wikipedia dataset, we took a sample of 1k articles. We computed the pairwise candidacy score between the sample and all Wikipedia articles which is equivalent to $4.2 * 10^9$ operations. The time needed to finish the task was around 10 minutes. Hence, the time estimated to compute the pairwise candidacy between all pairs of Wikipedia ($18 * 10^{12}$ operations) is around 30 days.
The fact that the *cand* function is a symmetric ($cand(d_i, d_j) = cand(d_j, d_i)$) function allows us to skip half of number of operations needed and consequently the time needed is 15 days. Since that is still feasible, we skip the usage of hashing method in this implementation and we apply the *cand* function on all pairs. For the candidacy score threshold, we choose from table 3.1 the minimum similarity value found experimentally as our threshold which guarantees to some extent including all pairs that are likely to have text reuse.

Figure 3.8 and Algorithm 2 highlight the main steps performed in the candidate elimination subtask and how the computations are carried out in a distributed manner on the cluster. First, the two RDDs (tfidf-rdd1 and tfidf-rdd2) are loaded from the same HDFS file that contains the TFIDF representation of Wikipedia. Second, the two RDDs are partitioned to guarantee that the data is distributed evenly on all nodes (each partition is hosted on a node and processed all at once). We choose 4k partitions for the second RDD (tfidf-rdd2) to increase the parallelism. To be more fault tolerant and to reduce the memory consumption, we also split the first RDD (tfidf-rdd1) into 100 partitions to be processed separately. Third, we repeat for each of these partitions the following: (1) Collect the data of the partition into one node (the driver node) and broadcast it (copy it) into all the nodes, (3) the function $pairwise\_candidate\_scoring$ in line 9 of Algorithm 1 is applied in parallel on

all the N partitions of the tfidf-rdd1 and (4) finally the results are persisted into an HDFS file.

On each partition, the function *pairwise_candidate_scoring* computes the candidacy score only on pairs of documents: $d1, d2 : d1 \in tfidf\_rdd1 \wedge d2 \in broadcast$ if the following holds: $d1.sequence\_id > d2.sequence\_id$ Which guarantees only computing half of the pariwise similarity matrix .
As a result, the output of this subtask is an RDD in which each element contains as a key a wikipedia document id ($wiki\_id$) and as a value a list of tuples ($wiki\_id, score$) sorted by the candidacy score.

Algorithm 2 Candidate elimination distributed computation

procedure CANDIDATE_ELIMINATION($wiki\text{-}tfidf, k, n$)
    $tfidf\text{-}rdd1 \leftarrow load\_hdfs(wiki\text{-}tfidf)$
    $tfidf\text{-}rdd2 \leftarrow load\_hdfs(wiki\text{-}tfidf)$
    $tfidf\text{-}rdd1 \leftarrow partition(tfidf\text{-}rdd1, n)$
    $tfidf\text{-}rdd2 \leftarrow partition(tfidf\text{-}rdd2, k)$
    for $i \leftarrow 1, k$ **do**
        $tfidf\text{-}rdd1\text{-}part \leftarrow choose\_partition(i)$
        $broadcast \leftarrow broadcast(tfidf\text{-}rdd1\text{-}part)$
        $pairwise\_candidate\_scoring(broadcast, tfidf\text{-}rdd1)$ ▷ pairwise candidate socring will be executed on parallel on each partition of $tfidf\text{-}rdd1$
    end for
end procedure

Text alignment In this subtask we run a detailed text alignment for every pair of candidate documents. We first load the candidates RDD which is produced from the previous subtask. Second, we load the Wikipedia dataset into an RDD object. Third, we convert the RDD into a dictionary where the key is the document id ($wiki_id$) and the document text as a value. Then using the Apache spark framework we broadcast the dictionary and make it accessible to all the executors on the cluster. Finally, we partition the candidates RDD and execute on parallel on each partition a heuristic based text alignment extraction algorithm.

Even though many false positive pairs of documents were eliminated in the previous subtask through the candidacy score threshold, still running detailed text alignment over all candidate pairs could take long time. To address this issue, we implement a heuristic based text alignment algorithm to make it possible to do a trade off between completeness of the task and the computa-

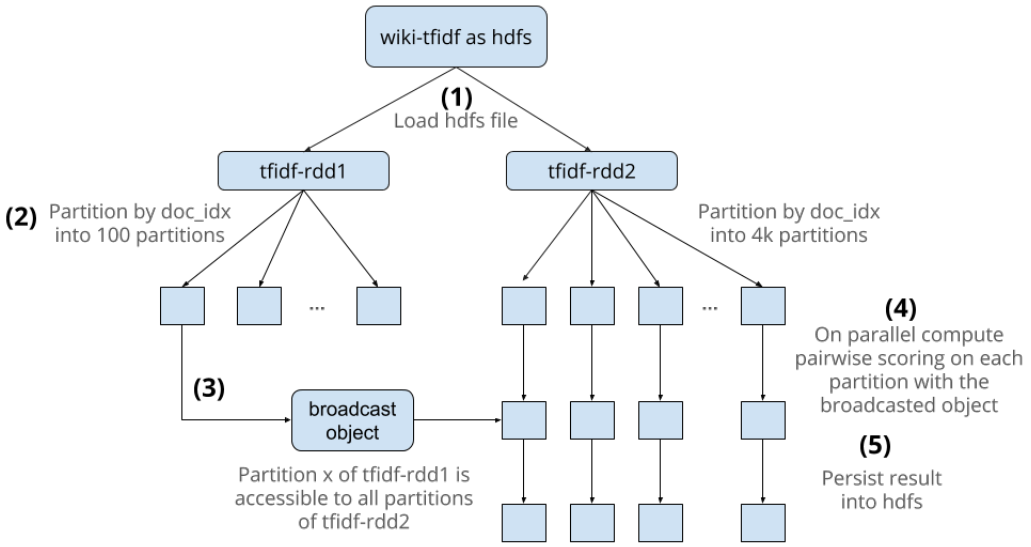


Figure 3.8: Illustration of the candidate elimination task distributed on the cluster

tional time. The algorithm takes a document and the sorted list of candidate documents and perform the following heuristics:

- **Candidacy score threshold:** By scanning further in the sorted candidacy list the likelihood of finding text reuse cases gets smaller. Hence, we could only examine candidate till reaching a specific candidacy score threshold. The threshold could be adjusted according the time available for the task.

- **Number of miss cases threshold** During the scanning of the sorted candidates list of a document, a miss case occurs when no text alignment is found. Since the candidates are sorted based on the likelihood of having text reuse, more miss cases means less likely to find further text reuse cases. By setting a threshold we could stop scanning after facing specific number of miss cases.

To extract the text alignment from within Wikipedia, we run the subtask with number of miss cases heuristic. To choose a proper threshold, we took the average first rank experimentally found in section 3.2.1 which is 250. That would on average find at least one document that have an alignment if it exists.

We apply the text alignment on document level and we use as an implementation the Picapica framework [7] with the following configuration:

[7] http://www.picapica.org/

- **Seed Generation:** For this, we use a simple word tokenizer implementation and generate the seeds from N-grams of words with length n of 3 words and overlap of 1.
- **Seed Extension:** We use the framework implementation of DBScan clustering algorithm proposed by Ester et al. [1996] with epsilon equal to 150 and minPoints 5 to cluster the seeds into longer pieces of text.
- **Seed postprocess:** For this we use the heuristics proposed by Stamatatos [2011] as a postprocess to filter out non likely text reuse cases by choosing minimum length of 200 tokens and minimum similarity of 0.5

In Algorithm 3 we show the proposed heuristic based text alignment algorithm that runs on parallel on each partition:

Line 5 in Algorithm 3 is executed on parallel over the N partitions with a threshold of number of miss cases.

3.4 Text Reuse Pipeline on Wikipedia and the Web

In the following section we present the specification of the text reuse pipeline in the case of running it on two datasets. Our two datasets are Wikipedia (same version used in the previous section) and the Common Crawl representing the web. We use the Common Crawl pulled on April 2017. Both datasets exist as HDFS files on the cluster.

Text preprocessing For Wikipedia we apply same steps presented in section 3.3 to extract the main content from Wikipedia syntax. To extract the main content for each web page in the Common Crawl, there exist many approaches. Indeed the main content extraction from web pages is a standalone research field. As a tool we used the Boilerpipe framework based on the work by Kohlschütter et al. [2010] which uses heuristics called shallow features to extract the main content from a web page. The output of the tool is a list of text chunks that represent the main content of a web page. We further apply some heuristics to remove blocks that might not represent the main content. First we filter out any text chunk of length less than 3 tokens which represents names and titles of buttons, links, etc. Second, We remove headers and footers content following simple heuristics explained in Algorithm 4. Third, We merge short chunks using the same algorithm presented in Section 3.3. Finally, we

Algorithm 3 Text alignment algorithm

Require: *partition* ▷ A list of documents with their sorted list of candidates, *wiki_broadcast* ▷ A broadcasted dictionary of Wikipedia document ids and their text compressed , *candidacy_score_threshold* The last score to be examined, *no_misses_threshold* Number of miss cases threshold

```
partition_alignments ← []
for document ∈ partition do
    no_misses ← 0
    compressed_text ← GET_VALUE(wiki_broadcast, document.id)
    document_text ← DECOMPRESS(compressed_text)
    for candidate ∈ document.candidates do
        compressed_candidate_text ← GET_VALUE(wiki_broadcast, candidate.id)
        candidate_text ← DECOMPRESS(compressed_candidate_text)
        alignments ← ALIGN(document_text, candidate_text)
        ADD(doc_alignments, alignments)
        if SIZE(alignments) = 0 then
            INCREMENT(no_misses, 1)
        end if
        if candidate.score ≥ candidacy_score_threshold then
            break
        end if
        if no_misses ≥ no_misses_threshold then
            break
        end if
    end for
    ADD(partition_alignments, doc_alignments)
end for
return partition_alignments
```

filter out all chunks that are shorter than one sentence and couldn't be merged with any other chunk.

The second step in text preprocessing is to extract features from the text. We use the TF-IDF weighting scheme as justified in Section 3.3. However in this case we have two datasets; Wikipedia and the web. The question that emerges is whether to choose the model's vocabulary and its inverse document frequency (IDF) representation jointly from the two datasets or from one of them. The answer differs depending on the use case. Since our task is to identify all the chunks on the web that are similar to Wikipedia text (chunks that represent the language model of Wikipedia), it suffices to take only the vocabulary of Wikipedia.

Algorithm 4 Remove headers from web page content
*Removing footers, work the same but by starting from the last paragraph and scan the array backwards

```
1: procedure REMOVE_HEADERS(paras, min_tokens, min_stopwords)
2:     idx ← 0
3:     reached_main_content ← False
4:     while ¬reached_main_content ∧ idx ≤ lengeth(paras) do
5:         para ← paras.get(idx)
6:         if lengeth(para) > min_tokens ∧ contains_punct(para) ∧
   number_of_stopwords(para) > min_stopwords∧ then ▷ If the criteria
   holds then we reached main content
7:             reached_main_content ← True
8:         end if
9:         idx ← idx + 1
10:    end while
       return paras[idx:]          ▷ Return the part of the paragraphs array
   starting from the idx which contains the main content
11: end procedure
```

As illustrated in Figure 3.9, the text representation step is done in two stages. The first stage is the training stage were the IDF is extracted only from Wikipedia chunks. We take the top N frequent words (N=260k) after tokenizing the text and removing the stopwords. The second stage is a transformation in which the IDF is used to compute the TFIDF weights for both Wikipedia and the web.

Since in this implementation of the pipeline we are dealing with the Common Crawl which is much bigger than Wikipedia, performing pairwise candidacy score computation on every pair of documents gets more challenging and demands long time computations. To address this, we deal with the task as a problem of near neighbor search. In section 3.2.2 we experimented and compared two hashing methods for near neighbor search; random projection and VDSH method. Following the evaluation results, we decide to use the VDSH method. Since the task is to model the language of Wikipedia, we argue that training the VDSH model only on Wikipedia content suffices. Doing so would create binary hashes (buckets) that represent the semantics of Wikipedia. Consequently any piece of text in the Common Crawl that contains Wikipedia text would theoretically be hashed into one of those binary hashes.

To prepare the training data for the VDSH model, we perform the following on the Wikipedia dataset: (1) we tokenize the text. (2) remove: stop

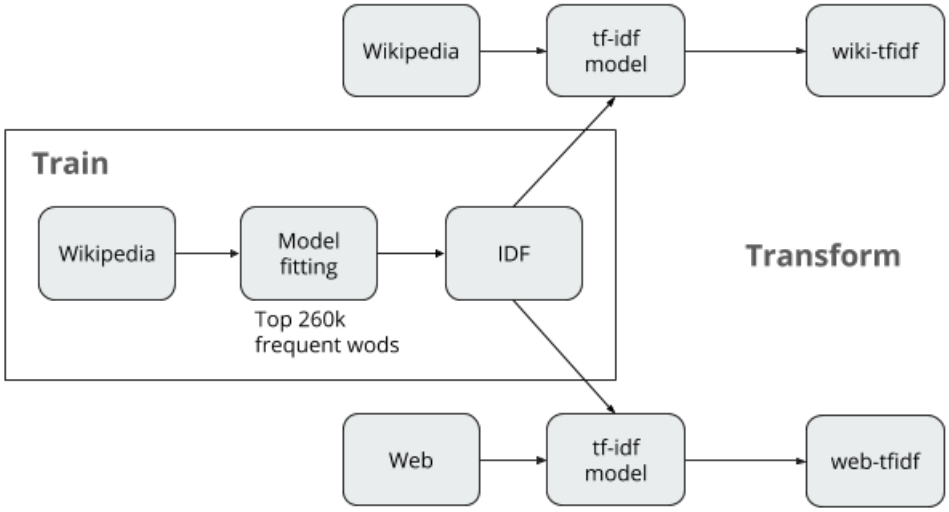


Figure 3.9: Illustration of the text representation

words, any non alphanumeric tokens, and 2-character tokens. (3) compute the inverse document frequency (IDF) for all tokens and choose the top 10k frequent ones to represent the TF-IDF space.(4) we take a sample from the articles of Wikipedia that are longer than 2k tokens and use them as training data for the VDSH model. Also, the same training data is used to learn threshold values to convert the continuous learned latent vectors into binary. After training a VDSH model, we use it to hash both Wikipedia articles and the web pages by performing the following steps: (1) Computing the TFIDF representation using exactly the same (IDF) computed in the training stage. (2) Using the encoder component of the VDSH model to encode the TFIDF vector into a continuous latent code. (3) We binarize the contentious code using the threshold learned in the training stage.

The output of this subtask is for each of the two datasets (Wikipedia, Common Crawl) two RDDs: one contains the TFIDF representation of each document and one contains the binary hashes.

Candidate elimination The task is to compute pairwise candidacy scores of each pair of documents in Wikipedia and the web. As per section 3.2.1, the chosen implementation of the *cand* is:

$$cand(d_i, d_j) = max(cosine\_similarity(p_i, p_j))$$

Where:

$$p_i \in d_i, p_j \in d_j, d_i \in Wikipedia, d_j \in web$$

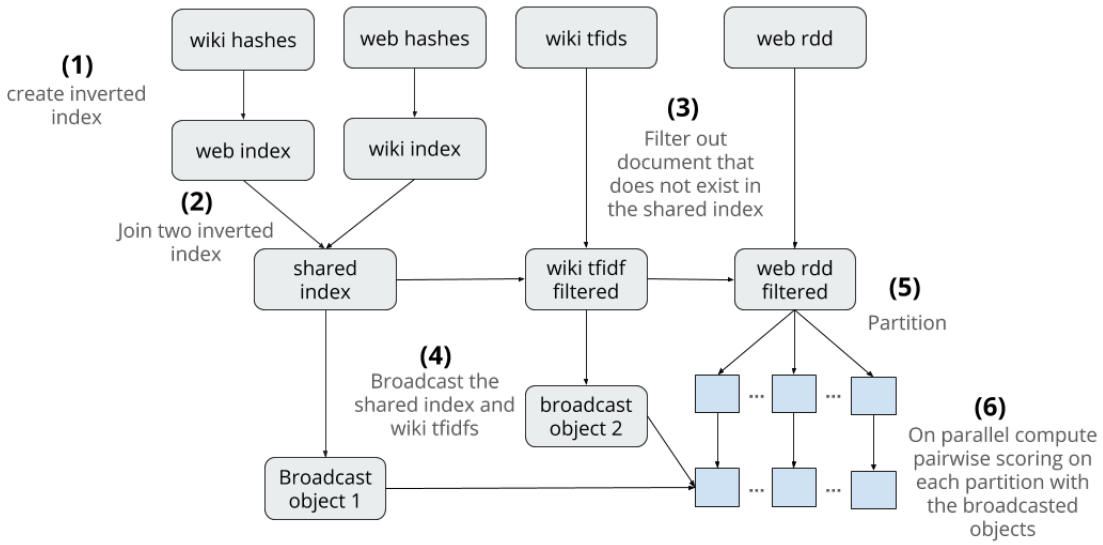


Figure 3.10: Illustration of candidate elimination task on web vs wikipedia

However, the *cand* function is applied only on documents from the web and Wikipedia that share at least one binary hash. To estimate the gain of using the VDSH hashing method in terms of performance, we randomly sampled 10% of the Common Crawl ($590 * 10^6$) and computed the number of pairs of documents from the sample and Wikipedia that intersect at least in one binary hash (hashing bucket). The number of pairs that satisfy the condition is equal to $6 * 10^{11}$ while the total number of pairs is $2.5 * 10^{14}$. That said we reduced the number of pairs to be examined by 3 orders of magnitude and at the same time we retained 73% recall according to the experiment in section 3.2.2.

In Figure 3.10 we illustrate the distributed computation performed in this subtask. After loading each dataset's TF-IDF vectors and binary hashes RDDs, we first create an inverted index for each of the two datasets (wiki index and web index). The inverted index keys are binary hash and the values are lists of document IDs. Second, an inner join between the two inverted indices is performed to keep only shared hashes (keys) that exist in both indices. In the third step, we keep only documents that contain at least one binary hash in the inverted shared index. For the fourth step, we broadcast the shared index (that has for each binary hash a list of Wikipedia document IDs) and the wiki tfidf (represents a lookup table that gives for each Wikipedia document ID a list of TF-IDF vectors). In the fifth step the web rdd is partitioned into N partitions. Finally in the last step, the pairwise candidacy scoring is performed on parallel on all the partitions of the web rdd. In Algorithm 4 we show the pseudo code which demonstrate the candidacy scoring algorithm.

Algorithm 5 Hash based candidacy scoring

```
1: procedure CANDIDACY_SCORING(web_partition, hash_index, wiki_index, threshold)
2:     results ← []
3:     for page ∈ web_partition do
4:         wiki_ids ← retrieve_wiki_ids(page.hashes, hash_index)
5:         for wiki_id ∈ wiki_ids do
6:             wiki_tfidfs ← retrieve_wiki_tfidfs(wiki_id, wiki_index)
7:             candidacy_score ← cand(page.tfidfs, wiki_tfidfs)
8:             if candidacy_score ≥ threshold then
9:                 results.add(page.id, wiki_id)
10:            end if
11:        end for
12:    end for
13:    return results
14: end procedure
```

Text Alignment In this subtask, we first load the candidates RDD from the previous subtask. Each item in this RDD contains *page_id* and *candidate_documents* which is a sorted list (based on the candidacy score of each item) of tuples. Each tuple consists of a Wikipedia document id and a candidacy score. Second we load both Wikipedia and the web prepossessed documents into two RDDs. Third, we join the web RDD with the candidates RDD. The resulting RDD has for each item web_id, web_text, list of wiki_id. Third, we convert the wikipedia RDD into a dictionary where the keys are document ids and the values are documents' text. We further compress the values of the dictionary (so it fits in the memory) and then broadcast the dictionary. In the final step, we partition the candidates RDD and in parallel for each partition we perform the heuristic based text alignment extraction mentioned in Algorithm 3.

3.5 Summary

In this chapter we presented our approach towards building a framework for extracting text reuse cases from big datasets. In the beginning of the chapter a formal overview on the main subtasks has been given. Then an experiment setup for evaluating methods of candidate elimination tasks was presented and used to evaluate our proposed methods. Later in the chapter, two situation of text reuse extraction were presented and the detailed implementation of the pipeline in each of them was shown. The first situation is to extract text reuse cases in between Wikipedia articles, and the second is to extract the text

reuse cases between Common Crawl and Wikipedia. In the following chapter we proceed to analyze the extracted text reuse cases in both situations.

Chapter 4

Text Reuse Analysis

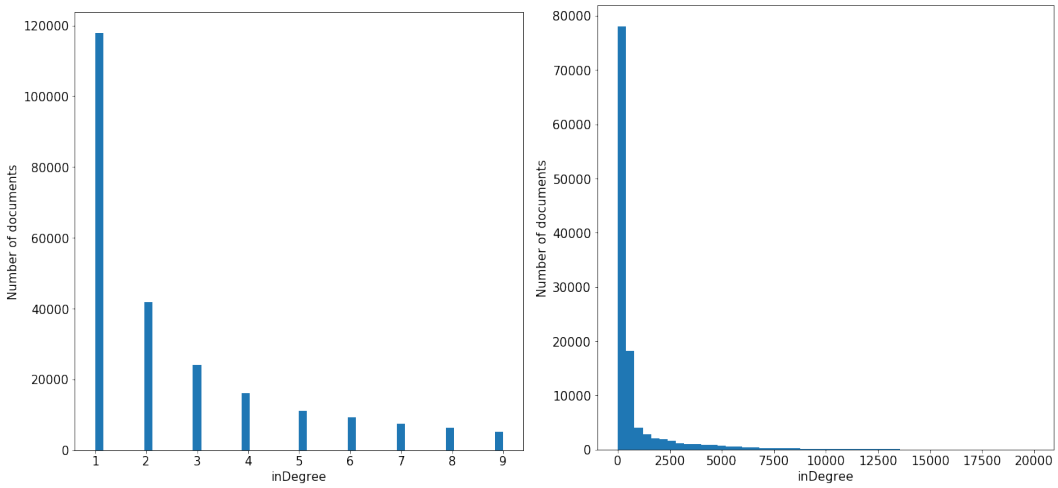
In this chapter we present the results of running the text reuse extraction pipeline in the addressed two usecases; text reuse in between Wikipedia documents and the text reuse between the Common Crawl (web) and Wikipedia. We try using the extracted data to answer our research questions about the kinds of text reuse in Wikipedia, and how much web content is a reuse of Wikipedia content.
In the first section we look at text reuse phenomena in Wikipedia. We describe the text reuse cases we extracted and we give an explanation on possible situations that might lead to these phenomena. We categorize text reuse cases into two classes and we present basic heuristics for automatic classification. The second section reviews text reuse cases that have been extracted from a sample of the web. It quantifies the amount of text reuse and give a basic approach to estimating the value of the reused Wikipedia content on the web.

4.1 Text Reuse In Wikipedia

From Wikipedia we extracted around 100 million text reuse cases. These occur in 360k Wikipedia articles which makes up around 9% of Wikipedia. To understand how the articles in Wikipedia interact with each other in terms of text reuse, one way is to look at the text reuse as a graph problem. The nodes of the graph are the documents and the presence of a text reuse case generates an edge between the two documents. In our case, we don't distinguish the source and the destination of a text reuse case. Hence, our graph is an undirected graph.

One of the graph properties that we looked at is the Degree of a node (number of edges that are connected to the node) which reflects the number of interactions (text reuse cases) that a document has with other documents. In Figure 4.1, we show the distribution of the degree value in the graph. Only

33% of the documents have a degree value bigger than 10. However those documents generate 97% of the text reuse cases extracted from Wikipedia.



(a) **(b)**

Figure 4.1: Graph Degree distribution. Figure (a) Degree $<$ 10. Figure (b) Degree $>=$ 10

A connected component in a graph is a subgraph where any two nodes are connected with each other by a path. Looking at connected components in the text reuse graph reveals the existence of clusters of documents that represent different topics and flows of text. We show in the following table few examples of connected components of size 3:

Component Articles	topic
Gibibyte, Tebibyte, Exbibyte	Measurement units of storage
Te Solte La Rienda, El Reloj Cucu, Clavado En Un Bar	Songs by same singer
Mark West (basketball), Jim Eakins, Earle Higgins	Basketball players

Table 4.1: Example of Wikipedia articles connected as a component in the text reuse graph of Wikipedia.

A more detailed analysis of the text reuse graph of Wikipedia could help in understanding the information flow in such collaborative environment like Wikipedia. Due to the restrictions in time, we leave this further analysis for future work.

Figure 4.2 shows the distribution of the length (as number of words) of the reused text in Wikipedia. The average number of tokens in the reused texts

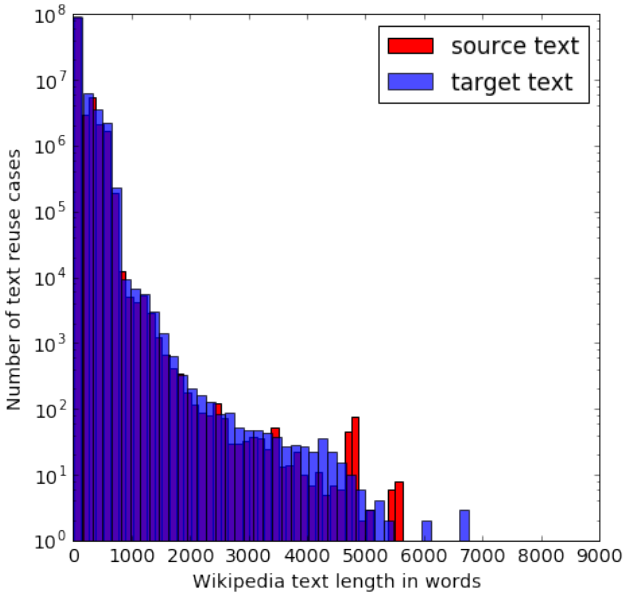


Figure 4.2: Distribution of the length (number of words) of the reused text. Log scale is used on y-axis.

(both source and target) is around 78 words which is shorter than a paragraph. Most of text reuse cases (87%) are short reused texts with length shorter than 100 words.

To understand what kinds of text reuse cases exist in Wikipedia, we first need to identify the reasons behind this phenomena. After observing few examples of text reuse, we hypothesize that two texts overlap (text reuse case) because of mainly two reasons: (1) The two texts are describing two different topics/entities that share some level of similarity, which in turn, leads to having same text spans reused for describing both entities. (2) The two texts are covering the same underlying topic.

Situations that apply to the first reason lead to text reuse cases that we call structure reuse in which the same structure of the text is reused but the main overall topic/theme remains different. The level of similarity in between the two underlying topics of the two texts varies from low to high similarity. Consequently, the two texts overlap to a different degree. Another reoccurring example of structure text reuse from the extracted data happens between geographical locations. In Figure 4.3 (lower box) the two underlying topics share the characteristics of being villages in a Voivodeship in Poland but sill in different administrative districts which resulted in text reuse of low overlap. However, in same Figure 4.3 (upper box) we show another example of two villages that share almost all the geographical characteristics but one is

located to the east and the other is located to the north-east of some land mark (two different villages but they share many geographical characteristics). The resulted text reuse is of high overlap (almost looks identical) but still the two underlying described entities are different. That poses big challenge on developing good heuristics to identify the structure reuse.

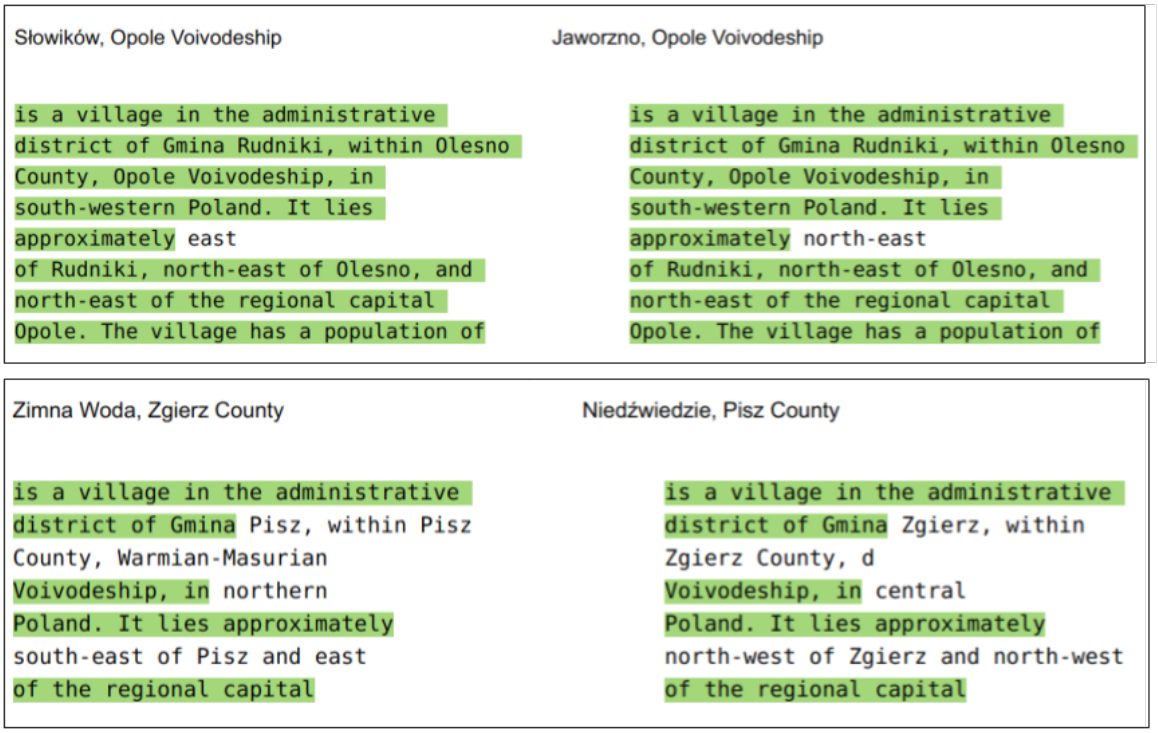


Figure 4.3: Examples of structure text reuse. The upper box represent two villages with high similarity while the bottom box two villages that share less characteristics

The second reason of text reuse in which the two underlying topics are identical, leads to text reuse cases where the text is copied from one source to another. However the two versions of texts that describe the same event/fact/entity get updated and modified differently which causes divergence between the two texts. This could end up having two versions of text that contain contradictions, or one contains more information than the other. An example of this case is shown in Figure A.1 in the Appendix, Two articles talking about the same topic (tooth eruption). However, the two texts has changed and different terminology were introduced like dentition in the first article while its called tooth eruption in the other. We notice also the different facts introduced in the two articles about the start of dentition stage. While one article states that it starts around age of six months the other says at age of eight months. The already mentioned example besides many other examples create undesirable

situation of inconsistency in Wikipedia that could be thought of as a quality flaw. The two texts in such cases should be unified.

Given a text reuse case X between two topics (documents) $DocTopic1$ *and* $DocTopic2$, we further hypothesize that by identifying the location of the two topics in the ontology tree [1], we could tell into which class the text reuse case belongs. Structure reuse usually happens between topics that are located on the same level of the ontology tree.

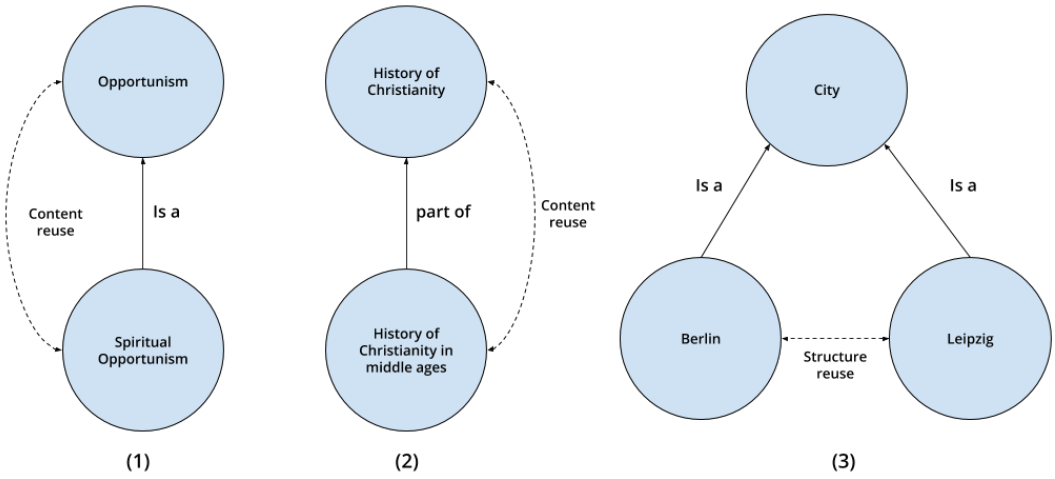


Figure 4.4: Examples of ontological relation between topics in Wikipedia and what text reuse case that might occur because of the ontological relation

Examples of this relation shown in Figure 4.4 where both topics (Berlin and Leipzig) have the relation "is a" with the concept city and have the characteristics of their parent in the ontology tree. In Figure 4.3 both cases of structure reuse show a situation where the source and the destination topics are on the same level in the ontology tree. However, when $DocTopic1$ and $DocTopic2$ are located vertically in the ontology tree (the relation between the two topics is either "is a" or "part of" as in Figure 4.4) then the text reuse case is of class content reuse. As an example in Figure A.1 in the Appendix, we can identify the relation between the two topics as one (Tooth eruption) is part of the other (Human tooth development) and consequently the text reuse case is of class content reuse. Another example of content text reuse in Figure A.2, where the relation between the two topics is "is a" (Spiritual opportunism is Opportunism).

To be able to build heuristics that automatically classify text reuse cases, we extracted the following features from each text reuse case; The first feature

[1] `https://en.wikipedia.org/wiki/Ontology_(information_science)`

is the Jaccard similarity of N-grams (text overlap) between the two texts. We chose for N values of 2 as a minimum value that captures a degree of text reuse, 5 as an estimated length of one clause, and 10 and 15 to cover the length of a sentence. The variant values of N help in distinguishing different levels of text overlap. Also using combinations of N-gram Jaccard similarity of different N values indicates different situations of text reuse cases. For example high similarity of 2-grams and low similarity of 5-grams indicates a structure reuse situation.
The second feature is Jaccard similarity of named entities between the two texts. Low values of named entities similarity could be an indicator of a structure text reuse case (two texts talking about different named entities). The last feature we extracted was the percentage of reused text from both the source article and the destination article. Having a high percentage of an article as reused text is a strong indicator of a structure reuse case.

From the aforementioned extracted features, we constructed the following set of heuristics that automatically classify a text reuse case:

$$
\begin{aligned}
H1 :& ne\_sim \in (0.5, 1.0] \quad \wedge \\
& 10grams\_sim > 0.5 \quad \wedge \\
& (s\_percent\_reused < 0.5 \vee t\_percent\_reused < 0.5) \\
& \Rightarrow \text{content reuse} \\
H2 :& 2grams\_sim > 0.5 \quad \wedge \\
& 5grams\_sim < 0.5 \quad \wedge \\
& (s\_percent\_reused > 0.5 \vee t\_percent\_reused > 0.5) \\
& \Rightarrow \text{structure reuse} \\
Where :& \\
& ne\_sim \quad \text{is named entities similarity} \\
& Ngrams\_sim \quad \text{is similarity over N grams} \\
& s\_percent\_reused \quad \text{is percentage of text reused from the source article} \\
& t\_percent\_reused \quad \text{is percentage of text reused from the target article}
\end{aligned}
\tag{4.1}
$$

To validate the performance of each of the heuristics, we applied each on the text reuse cases dataset, then randomly sampled 100 cases from the data that is labeled (by the examined heuristic) as structure reuse cases and another 100 cases from the data that is labeled as content reuse cases. We repeated the validation again but this time we only sampled from text reuse cases with a minimum of 200 tokens to account to the skew in the length distribution of

text reuse cases towards short texts. For each of the eight samples (4 samples for each heuristic) we manually computed the precision which is the number of correctly classified cases out of the whole 100 instances. We didn't compute the recall because we don't have information of the total number of cases in each of the classes. In Table 4.2 we show the precision of each of the two constructed heuristics.

Both heuristics had 100% precision in detecting structure text reuse. That is because most of the text reuse cases are structure text reuse cases. However, in detecting the content text reuse cases H1 has higher precision than H2 and the precision is higher for both heuristics in sample4. We think that is related to the fact that most of the structure text reuse cases are short ones.

	structure reuse		content reuse	
	sample1	sample2*	sample3	sample4*
H1	100%	100%	58%	73%
H2	100%	100%	21%	50%

Table 4.2: Evaluation table for H1 and H2 heuristics in term of precision. sample2 and sample4 are randomly sampled from text reuse cases longer than 200.

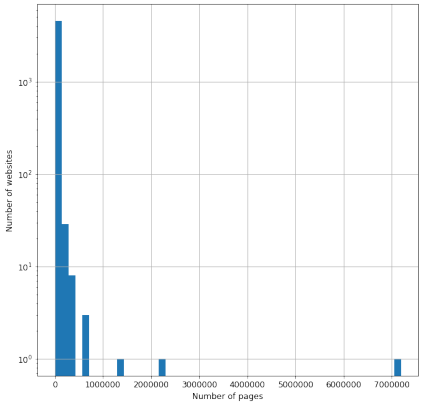
Later we examined the validity of our hypothesis on the ontological relation between topics of a text reuse case and how it correlates with text reuse class (structure/content). From the 100 randomly sampled (sample4 in Table 4.2) text reuse cases of content reuse class (collected using H1 heuristic), we manually examined 20 content text reuse cases. We found 10 text reuse cases where the two topics aligned vertically in the ontology tree. We used the text reuse cases in sample1 of heuristic H1 to look at all structure text reuse cases. All the 20 structure text reuse cases had the two topics on the same level in the ontology tree.
However, during the examination of text reuse cases, we encountered cases that need to be further examined. For example in Figure A.3 in the Appendix, the two articles (Culture of Dominica and Antillean Creole French) have no relation in the ontology tree but they still have content text reuse case where both tell in their history section the story of the french colonization in the Caribbean.

Another frequent case we encountered is two articles who have a horizontal relation in the ontology tree but have a content text reuse case. An example of this is two school districts (Susquehanna Township School District and Warrior

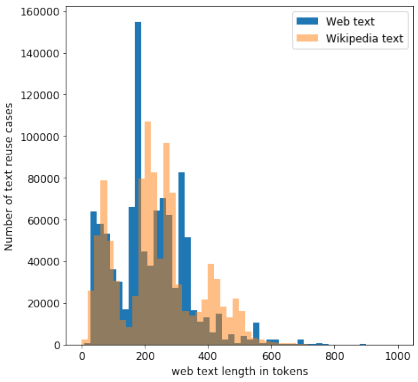
Run School District) both are in Pennsylvania and have content text reuse where they both define the same policy that deals with nutritious meals served at school. An example of dealing with such case in Wikipedia is to suggest opening a new article that gives a unified explanation of this definition (the nutritious meals policy in Pennsylvania schools).
Those cases and many others worth further investigation to understand the interaction between concepts in Wikipedia and how it affects the possibility of having text reuse cases.

4.2 Text Reuse between Wikipedia and the Web

To examine the text reuse phenomena between the web and Wikipedia, we randomly sampled 10% of the Common Crawl and we ran the pipeline in between the sample and Wikipedia. The sample contained 1.4 million domains. In Figure 4.5(a) we show the distribution of the number of web pages per domain (website) in the sample. It worth noting that around 990 thousand domains contain less than 10 pages in the sample. So the text reuse extraction we perform per website is not on all its web pages (some websites has only few web pages examined).



(a) Distribution of number of pages per website in the web sample



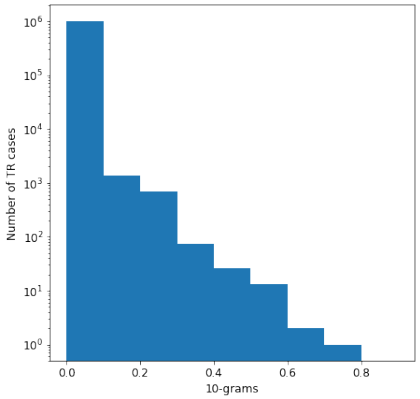
(b) Distribution of reused text length for both Wikipedia and the web. For readability of the graph we omitted text reuse cases longer than 1k tokens which only around 150 cases

Figure 4.5

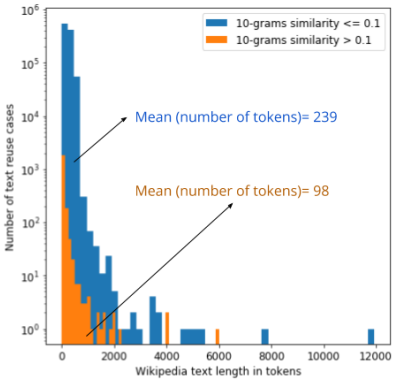
As a result of running the pipeline on this sample, we extracted around 1.6 million text reuse cases. However, only 15k web pages of the sample have at

least one text reuse case with a Wikipedia article. In Figure 4.8(b), we show the distribution of the reused text length (number of words) of both the source text (from Wikipedia) and the destination text (from a web page). We observe that most text reuse cases (75%) are less than 300 words in length.

Due to time limitation, in our analysis of text reuse cases between the web and Wikipedia we concentrated on quantifying the amount of text reuse without looking in detail on what characteristics these cases have and whether there is recognizable patterns in the data. However, we only looked at the similarity between the two texts in terms of the percentage of overlap between 10-grams (Jaccard similarity). In Figure 4.6(a), we show the distribution of 10-grams similarity. Most text reuse cases has similarity less than 0.1. Taking that value as a threshold we divide text reuse cases into two groups.



(a) Distribution of 10-grams similarity values of web text reuse cases



(b) Distribution of reused Wikipedia text for text reuse cases with 10-grams similarity $=<$ 0.1 (blue) and cases with 10-grams similarity $>$ 0.1 (orange)

Figure 4.6

The observation of the first group of text reuse cases with similarity of 10-grams greater than 0.1 shows that most of these are cases with short reused Wikipedia texts with an average of only 98 tokens (Figure 4.6(b)). These text reuse cases represent situations in which the two texts talk about the same subtopic (content text reuse). An example of these cases is shown in Figure A.4 (upper box) in the Appendix.

The second group of text reuse cases with similarity less than or equal to 0.1, are cases with relatively longer texts with an average of 239 tokens (Figure 4.6(b)). Those text reuse cases are similar to the structure reuse cases between Wikipedia articles. These cases occur when a web page copies text from a Wikipedia article (information about a city for example) that shares

similar structure with other Wikipedia articles resulting a structure text reuse cases in between the web page and those structurally similar articles. An example of this case is shown in Figure A.4 (lower box) in the Appendix.

The 15k web pages that we found to be reusing text from Wikipedia were generated by 4898 out of the 1.4 million websites. On average each website has 3 pages reusing text from Wikipedia. To put this into perspective, we computed for each website the percentage of web pages that reuse text from Wikipedia. In Figure 4.7 we show the distribution of this percentage. On average around 0.02 of the website content is just a reuse of Wikipedia content. However, There exist 87 websites who have the situation where 50% of their examined web pages are reuse of Wikipedia articles.

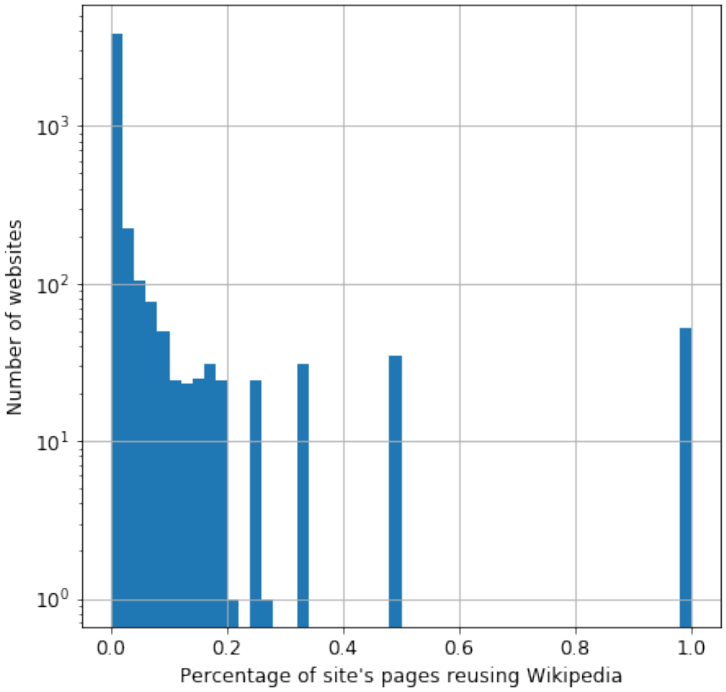


Figure 4.7: Distribution of the percentage of webpages that are reusing text from Wikipedia per website

Estimating the revenue of a website is not a straight forward task. Website revenue is usually generated in many ways like; Selling services or products, Affiliate marketing (promoting other entity's products) and more importantly advertising.

In this research we concentrate on computing the ads revenue generated by reusing Wikipedia content. Usually the reusing websites copy content from Wikipedia and add it to their own web pages along with ads. The traffic

that this content attracts in turn generates money for the reusing website. For example the Cost-Per-Mille (CPM) ads [2] is a method of advertisement in which a specific amount of money is paid (from the owner of the ad to the advertising website) for every 1000 page views on a web page that contains the Ad.

However it is hard to give an accurate estimation on how much traffic each web page gets or how much is the CPM rate for a website. Due to the time limitation we took a very basic approach to estimate the potential ads revenue made by the evaluated web sample. First, we draw a sample of 100 reusing web pages and manually check the presence of ads on these pages. All the 100 pages contained at least one Advertisement. Building upon this observation we assume all reusing web pages contain ads. Second, to estimate the ads revenue generated by Wikipedia content, we compute the following for each website: (1) Using the service Worth of web [3] we estimate how much the website worth. The tool computes the value of a website based on statistics about public traffic, visitors and page views a website gets to estimate the potential advertising revenue. (2) We compute the proportion of web pages in the website that contains Wikipedia content. (3) The ads revenue generated by Wikipedia content is computed by multiplying the website value by the proportion of web pages that contains Wikipedia content. Finally we sum up the proportional revenue of all the websites and that would be the ads revenue generated from Wikipedia content. In table 4.2 we show the estimated revenue of few web sites. The total value computed was 1.2$ million monthly revenue. This estimated revenue is what is expected to be generated by the reused free content of Wikipedia in a 10% sample of the web.

The revenue of 1.2 million per month generated by Wikipedia content seemed to be high estimation. To investigate more the websites behind this big number we observed the existence of big websites like; google.com, bbc.com, tumblr.com, etc. We understand that these websites are big enough and generate revenue without the reuse of Wikipedia pages. To address this fact we give another number that represents a lower estimation of the revenue, we only take those websites with a percentage of reused Wikipedia pages equal to or more than 50%. The number of websites that satisfy this condition equals to 87 and computing the monthly revenue on only those websites results in around $15k.

The final method we propose to estimate the monthly revenue (generated by the reusing websites) is based on the Wikipedia page views and the average

[2] `https://en.wikipedia.org/wiki/Cost_per_impression`

[3] `https://www.worthofweb.com/calculator`

CPM. According to a report by MonetizePros [4], the average estimated CPM value is equal to $2.8 (we call it avg_cpm). This average is computed again based on assumption and aggregation of publicly available information about the CPM rate averages. We extracted for each reused Wikipedia page the average of the monthly pageviews from Wikipedia (avg_page_views). We assume that the computed average reflects also the number of page views that the reusing web page would receive. Having the estimated average of CPM and information about number of page views that each web page gets, we could compute the total monthly revenue (monthly_revenue) of the reused Wikipedia content according to equation 4.2. The estimated monthly revenue according to this approach equals to $900k.

$$monthly\_revenue = \sum_{i}^{n} (avg\_page\_views/1000) * avg\_cpm \tag{4.2}$$

Interpolating this value into the whole web needs more information about the distribution of number of reused web pages per website, the distribution of websites monthly revenue, etc. We leave this task for future work. However, this approach we have proposed is built on a lot of assumptions which might not hold. We don't claim its an accurate estimation but it forms a start that further research could extend on.

website	website value	proportion of Wikipedia content	Wikipedia content value
pdxretro.com	195$	0.012	2.5$
searchquarry.com	8,850$	0.096	850$
asiatees.com	3,810$	0.017	66.19$
onefivenine.com	36,000$	0.017	613.22$
wikia.com	25,615,320$	0.0008	23,037$
...	....	...	...
Total			1.2 million

Table 4.3: The manually examined websites and their estimated value

The last statement to be highlighted is the fact that most of these reusing websites don't give any reference of Wikipedia as a source. We created a script to crawl the 15k web pages. For each web page the script extracts all the URLs in the page and searches for any URL that links to "wikipedia.org". Out of the 16k web pages the script found 760 web pages that contain at least one reference to Wikipedia. However, Wikipedia content is published under the terms of the Creative Commons Attribution Share-Alike license (CC-BY-SA)[5]. The license states that the content/text can be reused but the entity

[4] `https://monetizepros.com/cpm-rate-guide/display/?cn-reloaded=1`

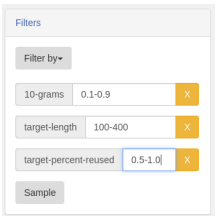
[5] `https://en.wikipedia.org/wiki/Wikipedia:Reusing_Wikipedia_content`

that reuses the content must attribute the reused work to the source where it was copied from. Clearly, these websites that reuse Wikipedia content without any reference are violating the terms of use.

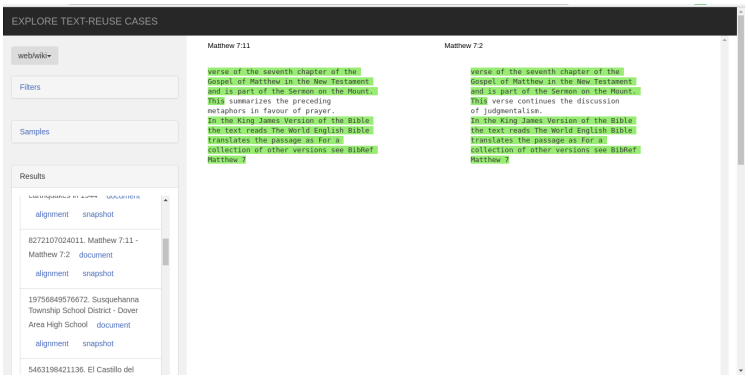
4.3 Text Reuse exploratory web tool

Data exploratory analysis is a challenging task and needs tools that support the exploration of different dimensions of the data to uncover patterns, answer questions and test hypotheses. Because we are convinced that there are yet many questions to be answered on the text reuse phenomena both in Wikipedia and between the Wikipedia and the web, we developed a web tool that enables the user to query the text reuse dataset with certain criteria and extract samples to be examined in detail (Figure 4.8(b)).

The web tool connects to the cluster and loads the text reuse dataset as an RDD distributed on the cluster (making it possible to explore big datasets). The used dataset is the output of the pipeline after performing the feature extraction step. The extracted features from text reuse cases are; N-gram similarity, name entities similarity, percentage of text reused from the total article and finally length of the reused text.



(a) An example of filters that can be chosen to sample from the text reuse dataset



(b) A snapshot of the text reuse analysis web tool

Figure 4.8

Using the tool, users can build their own criteria made of filters over the mentioned features and load a random sample from the data that satisfies the

requested criteria. For example in Figure 4.8(a), the user has selected to sample from text reuse cases that have 10-grams similarity between 0.1 and 0.9, reused target text of length between 100 and 400, and percentage of reused text of the target article is between 0.5 and 1.0. Then the sample can be browsed and each of the text reuse cases could be visualized. By choosing a text reuse case, users can visualize how the two texts in the text reuse case are overlapping. Besides that, users can also show the whole text of the two articles to inspect the overall text reuse situation between them.
Moreover, we added to the tool the sampled data we generated using the heuristics from the previous section. Currently through the tool, user can browse both Wikipedia text reuse cases and the text reuse cases between Wikipedia and the web. However any dataset contains text reuse cases generated from the pipeline could be browsed and explored using the tool.

4.4 Summary

The first section in this chapter presented and quantified the text reuse cases that were extracted from Wikipedia in the previous chapter. It gave insights on possible reasons of text reuse phenomena in Wikipedia and following that it categorized text reuse cases into two classes and presented a simple heuristic for automatic detection of each class.
Next, in the second section, the text reuse cases extracted in between Wikipedia and a web sample were presented and the amount of Wikipedia content used by this sample was quantified. A very basic method to estimate the revenue generated by this sample through advertisement over the free content was presented. However, this method doesn't give the final answer but it forms the first step toward establishing a framework that more accurately give an estimation of the revenue. Besides that in this analysis we assumed that the source of text reuse is always Wikipedia which might not hold in all cases.
In the end of this chapter, we present a web tool that could be used in exploring text reuse cases.

Chapter 5

Conclusion and Future work

In this work we constructed a pipeline for text reuse extraction from big datasets through cluster computing and heuristic based algorithms. Using the constructed pipeline we extracted text reuse cases in two scenarios; (1) In between Wikipedia articles and (2) between Wikipedia articles and a sample from Common Crawl as a representation of the web. We further analyzed the extracted text reuse cases in both scenarios and gave insights on this phenomena. In this chapter we highlight our main contributions and findings. We also present the possible areas of improvement on the current work and applications.

5.1 Contributions

We started our research by asking the following research questions; (1) What kinds of text reuse cases happen in between the articles of Wikipedia? (2) How much of the web is just a reuse of Wikipedia content? A follow-up to that is the question of how much revenue is generated from this reused content?

To answer these questions, in chapter 3 we proceed to build a framework for extracting text reuse cases from big datasets. We showed in abstraction how the framework could be in a form of a pipeline of subtasks and what the required algorithms and techniques are for each subtask. From there we move to concentrate on evaluating methods of the candidate elimination subtask where we construct an evaluation framework of these methods and used it to evaluate set of methods we proposed. Later in the chapter 3, we show two implementations of the pipeline in which they were used for extraction of text reuse cases within Wikipedia and in between Wikipedia and the web.

To finally answer our research questions, in chapter 4 we analyze the extracted text reuse cases in both scenarios and give insights to understand the text reuse phenomena in Wikipedia. We believe these insights only uncover

part of the big image and more could be done in this matter. For the text reuse cases that we extracted from the web sample, we quantify the amount of Wikipedia content reused in the sample and outline a humble approach to give a very rough estimation on the revenue that is generated by reusing content from Wikipedia. This approach doesn't give a final answer to our research question but it forms the first step towards the answer. At the end of chapter 4 we present the web tool that was developed for exploring text reuse cases for further exploration of this phenomenon.

5.2 Future Work

Our contribution towards building a pipeline for text reuse extraction from big datasets could be used in the future for further analysis of text reuse cases between different datasets. A possible future task is to look at the text reuse cases between Wikipedia and the scientific research community to complement the work done by Thompson and Hanley [2017]. This could add value in understanding the effect of Wikipedia on the scientific community.

In the experiments we performed in section 3.2 we concentrated on the evaluation of methods of the candidate elimination task leaving the final text alignment subtask as recommended by the used framework(PicaPica). However the results we analyzed and consequently our findings are subject to the restrictions/parameters of the final subtask. We do not claim to find all various kinds of text reuse cases because of this limitation. Another extension of our work is to experiment with different parameters of the text alignment subtask and examine their effect on the resulted text reuse cases. Besides that the experiment setup that we used including the sample we generated from Wikipedia text reuse cases was also restricted by the used text alignment framework. The sample doesn't contain all kinds of text reuse cases that might occur between two texts. For more accurate evaluation of candidate elimination methods, a more comprehensive representative sample should be used.

Using a variational auto-encoder neural network for semantic hashing showed potential results. Even though we kept our exploration of the NN configuration to a minimum and only tuned few parameters, the VDSH model outperformed the random projection hashing technique. A potential extension of this work is to have a detailed look into using VDSH to build a semantic hash function for candidate elimination in the text reuse extraction pipeline.
Furthermore, in section 3.2 we showed how constructing a scoring function of weighted average of two components, semantic scoring (paragraph embedding) and structure scoring (stopwords N-grams), enhanced the precision/recall curve.

Another possible candidate elimination method to be evaluated is a process of two steps. The first step is using VDSH to find those document which are semantically similar (semantic component), and the second step is to use stopwords N-grams to keep only those that have similar structure features (structure component) from the semantically similar candidates.

In this work, the insights and analysis of text reuse cases in Wikipedia is only the tip of the iceberg. We believe that more patterns and discoveries in the data can found. As presented in section 4.3, a web tool that we developed could help in further exploration and analysis of the extracted text reuse cases. In addition to that, a possible application of our work is to create a tool to evaluate the current situation of Wikipedia, and as an output, highlight the kind of quality flaws that results from having repeated versions of same text that should be unified. This tool could also notify authors (who are applying updates on pieces of texts) of the existence of duplicate versions of text and suggest updating all versions to keep Wikipedia in a coherent state.

As stated before, in our work on text reuse extraction between Wikipedia and Common Crawl we did not take into consideration the identification of the source of the reused text. The assumption was always that Wikipedia would be the source from which the text is taken. However that might not be always the case. To give more accurate results, information about the date of creating a web page or a Wikipedia page could be utilized to identify the direction of text flow.

In this work we proved the presence of websites that reuse texts from Wikipedia and generate revenue from free content. We provided a very basic and rough estimation of this revenue on a sample of the web. For greater accuracy of the estimation, further detailed examination could be performed in future works.

Appendix A

Text Reuse Cases

This chapter contains list of figures that shows text reuse cases extracted from Wikipedia and between Wikipedia and Common Crawl. We refer to these figures and discuss them in chapter 4.

Figure A.1: Example of content text reuse in Wikipedia

Figure A.2: Content text reuse. The relation between the two topics in the ontology tree is "is a"

Figure A.3: Content text reuse between two articles that don't have ontological relation.

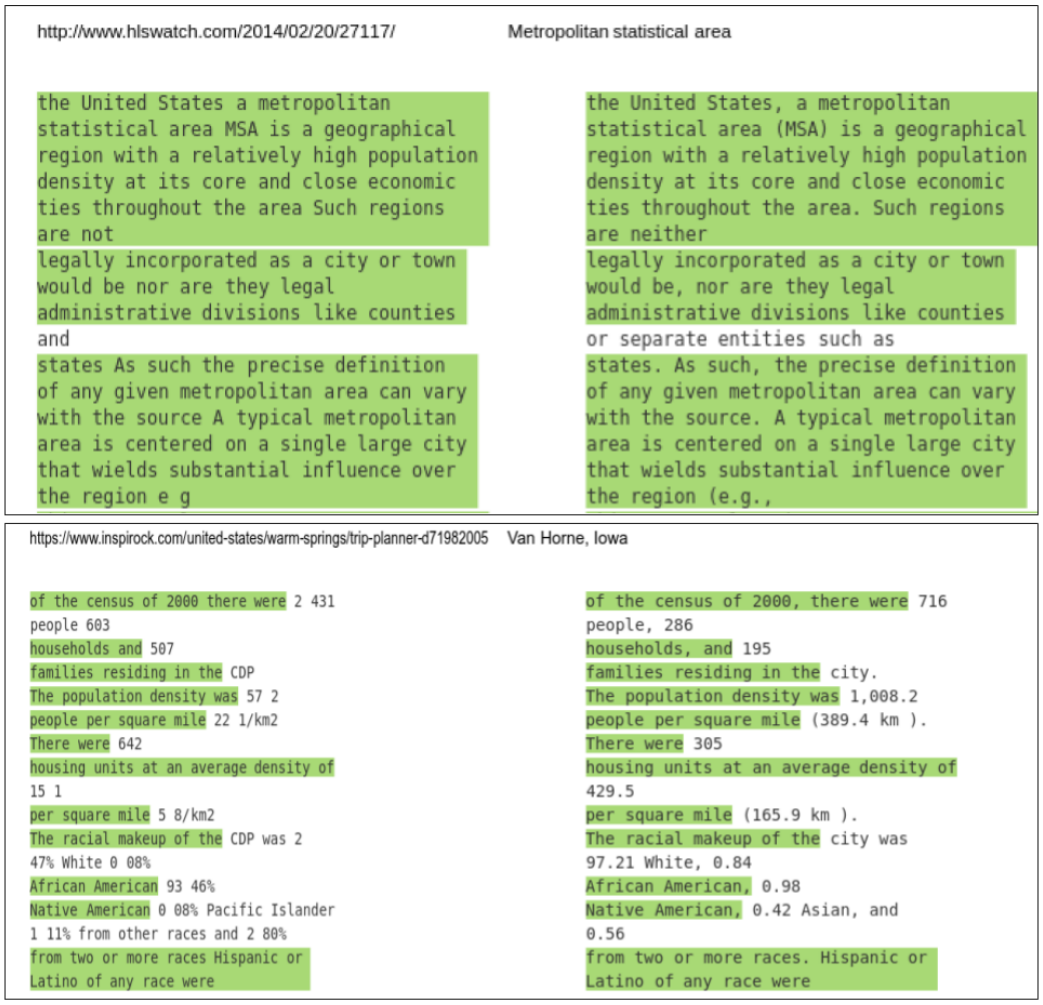


Figure A.4: Example of text reuse case between a web page and an article from Wikipedia. The top box contains text reuse case of the first group (content text reuse). Bottom box contains text reuse case of the second group (structure text reuse)

Bibliography

Maik Anderka, Benno Stein, and Nedim Lipka. Detection of text quality flaws as a one-class classification problem. In *CIKM*, 2011. 2.2

Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006. 2.3

Daniel Bär, Torsten Zesch, and Iryna Gurevych. Text reuse detection using a composition of text similarity measures. *Proceedings of COLING 2012*, pages 167–184, 2012. 2.1

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975. ISSN 0001-0782. doi: 10.1145/361002.361007. URL `http://doi.acm.org/10.1145/361002.361007`. 2.3

Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997. 2.3

Marco Büchler, Annette Geßner, Thomas Eckart, and Gerhard Heyer. Unsupervised detection and visualisation of textual reuse on ancient greek texts. 2010. 2.1

Suthee Chaidaroon and Yi Fang. Variational deep semantic hashing for text documents. *arXiv preprint arXiv:1708.03436*, 2017. 2.3, 3.1, 3.2.2, 3.2.2, 3.2.2

Daniel T Citron and Paul Ginsparg. Patterns of text reuse in a scientific corpus. *Proceedings of the National Academy of Sciences*, 112(1):25–30, 2015. 1, 2.1

Paul D. Clough and Yorick Wilks. Measuring text reuse in a journalistic domain. 2001. 1, 2.1

Wikimedia Commons. Size of wikipedia, 2018. URL `https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia`. [Wikipedia; accessed 13-June-2018]. 1.1

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. 2.4, 2.4

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996. 3.3

Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features. In *WSDM*, 2010. 2.5, 3.4

Martin Potthast. Technologies for reusing text from the web. 2012. 1, 2.1, 3.1

Martin Potthast, Benno Stein, and Robert Gerling. Automatic vandalism detection in wikipedia. In *ECIR*, 2008. 2.2

Martin Potthast, Matthias Hagen, Tim Gollub, Martin Tippmann, Johannes Kiesel, Paolo Rosso, Efstathios Stamatatos, and Benno Stein. Overview of the 5th international competition on plagiarism detection. In *CLEF Conference on Multilingual and Multimodal Information Access Evaluation*, pages 301–331. CELCT, 2013. 1, 2.1

Martin Potthast, Matthias Hagen, Anna Beyer, Matthias Busse, Martin Tippmann, Paolo Rosso, and Benno Stein. Overview of the 6th international competition on plagiarism detection. In *CLEF*, 2014. 2.1

Casper Kaae Sonderby, Tapani Raiko, Lars Maaloe, Soren Kaae Sonderby, and Ole Winther. Ladder variational autoencoders. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3738–3746. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6275-ladder-variational-autoencoders.pdf`. 2.3

Efstathios Stamatatos. Plagiarism detection using stopword n-grams. *Journal of the Association for Information Science and Technology*, 62(12):2512–2527, 2011. 2.1, 3.2.1, 3.3

Benno Stein, Sven Meyer zu Eissen, and Martin Potthast. Strategies for retrieving plagiarized documents. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 825–826, New York, NY, USA, 2007.

ACM. ISBN 978-1-59593-597-7. doi: 10.1145/1277741.1277928. URL `http://doi.acm.org/10.1145/1277741.1277928`. 2.1

Neil C. Thompson and Douglas Hanley. Science is shaped by wikipedia: Evidence from a randomized control trial*. 2017. 1, 2.2, 5.2

Fernanda B. Viégas, Martin Wattenberg, and Kushal Dave. Studying cooperation and conflict between authors with history flow visualizations. In *CHI*, 2004. 1, 2.2

Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*, 2014. 2.3

Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):769–790, 2018. 2.3

Sarah Weissman, Samet Ayhan, Joshua Bradley, and Jimmy Lin. Identifying duplicate and contradictory information in wikipedia. In *Proceedings of the 15th ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 57–60. ACM, 2015. 2.2

Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012. 2.4

Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10 (10-10):95, 2010. 2.4