Detecting Document Versions and Their Ordering In a Collection

Natwar Modani[1], Anurag Maurya[2], Gaurav Verma[3], Inderjeet Nair[1], Vaidehi Patil[2], and Anirudh Kanfade[4]

[1] Adobe Research, India {nmodani,inair}@adobe.com
[2] IIT Bombay, India anurag16111999@gmail.com, vaidehipatil16@gmail.com
[3] Georgia Tech, USA gverma@gatech.edu
[4] NITK, India anikanfade99@gmail.com

Abstract. Given the iterative and collaborative nature of authoring and the need to adapt the documents for different audience, people end up with a large number of versions of their documents. These additional versions of documents increase the required cognitive effort for various tasks for humans (such as finding the latest version of a document, or organizing documents), and may degrade the performance of machine tasks such as clustering or recommendation of documents. To the best of our knowledge, the task of identifying and ordering the versions of documents from a collection of documents has not been addressed in prior literature. We propose a three-stage approach for the task of identifying versions and ordering them correctly in this paper. We also create a novel dataset for this purpose from Wikipedia, which we are releasing to the research community[5]. We show that our proposed approach significantly outperforms state-of-the-art approach adapted for this task from the closest previously known task of Near Duplicate Detection, which justifies defining this problem as a novel challenge.

Keywords: Version Detection · Near Duplicate Detection · FCN · Wikipedia based Dataset

1 Introduction

Creation of documents is an iterative process. For instance, legal contracts need iterative editing to incorporate comments from stakeholders (this process is typically called red-lining), research papers go through several revisions based on feedback from reviewers and collaborators, etc. This leads to many versions of documents getting created. When one needs to find the latest version of a document or organize their documents, these additional versions of documents present added cognitive load for the user. In absence of any tool, they may have to open and skim through the documents one-by-one, which leads to inefficiency and fatigue. Using the names and timestamps may not always be a reliable mechanism, as names are given by people and most people do not strictly adhere to

[5] https://github.com/natwar-modani/versions

any naming convention. Also, the timestamps get altered by even minor and unintentional edits and saves (and sometimes due to moving/copying the files), and hence, they also do not render themselves as viable solution to the problem. Besides the human-centric motivation, several automated methods that operate on a collection of documents, like systems that cluster similar documents [11, 16] and recommend related ones [18, 19, 14, 13, 20] may perform sub-optimally in situations where the collection comprises of unnecessary documents, say, by recommending redundant and/or outdated documents.

While there have been works on somewhat related problems, such as near-duplicate document detection [4], document similarity [6], clustering [16], and plagiarism detection [8], etc., we believe that version identification is a different problem for two reasons. First, the notion of similarity of documents need to be defined differently for identifying versions as we will discuss later. Second, we also need to determine the adjacency of versions and their directionality (from older to newer version). To the best of our knowledge, our work is the first attempt to define this problem formally, provide a dataset and an approach for identifying and ordering versions of documents from a collection of documents.

Our problem statement is as follows. Given a *collection* of documents as input, identify the sets that are *versions* of each other and provides an *order of documents* within each set. Please note that the collection may comprise more than one set of versions along with some documents that have only one version.

We propose a three-stage approach for solving this problem. The first stage of our approach divides the document collection into sets of documents that can potentially be versions of each other for efficient computation. We use a two-step process, where we first find very similar paragraphs across documents using MinHash-LSH [2]. We then use the number of such (very similar) paragraphs common to a pair of documents (relative to the number of paragraphs in the shorter of the two documents) to determine if the pair of documents can be potentially versions of each other. The second stage of our approach classifies the candidate pairs of documents as versions-or-not using a fully convolutional neural network (FCN) [12] based binary classifier. We use pairwise lexical and entity-based similarities to create heatmap images, which are used by our proposed FCN-based classifier that identifies structures that indicate local similarity patterns. Once we determine which documents are versions of each other, in the third stage we find the order of creation among the versions. We create a graph with nodes as document instances and edges between a pair of document instances if the previous stage determined these document instances as potentially versions of each other. The edges have a weight based on the extent of alignment (based on minimum transformations required to convert one document instance into the other) between two document instances. We look for a directed chain which has minimum weight to find the order of versions in this graph.

We create a suitable dataset from Wikipedia articles which we are releasing to enable further research in this direction. We show experimentally that our proposed approach significantly outperforms the appropriate baselines adapted from closest task of Near Duplicate Detection on this task. This indicates that the problem of version detection is sufficiently different from the closest problem addressed in prior work.

In summary, our contributions in this paper are:

- We propose the problem of identifying and ordering versions of documents from a given collection of documents.
- We propose a three-stage approach for efficiently solving the problem.
- We create a novel real world data set from Wikipedia suitable for this task, which we are releasing to the research community.
- We show experimentally that our proposed approach significantly outperforms the appropriate baselines adapted from closest task of Near Duplicate Detection on this task.

The paper is organized as follows. We briefly review the relevant prior work in Section 2. We then describe our proposed methodology in Section 3. We describe the dataset and its construction in Section 4 and report the experimental results in Section 5. Finally, we conclude in Section 6.

2 Related Work

Although the specific problem of identifying versions in a collection of documents has not been addressed in previous literature, there has been prior work in related areas of near-duplicate document detection, paraphrase detection and plagiarism detection. While none of these related tasks require the ordering part, the detection task can be thought of as finding document 'similarity', where 'similarity' definition depends on the specific task. For Near Duplicate Detection (NDD), the similarity is defined at lexical level, i.e., documents that are near duplicates of each other would have nearly the same set of words. MinHash-LSH [2] is document fingerprinting method from the family of Locality Sensitive Hashing (LSH) functions for estimating the lexical Jaccard similarity between documents. [9] introduced sectional MinHash, a modification that divides the document in sections and uses the matching for both the characteristics (words occurring in the document) and in which section they occur. This enhancement to MinHashLSH with sectional information achieves state-of-the-art performance on the task of near-duplicate document detection. Most of the literature on NDD [1, 4, 9, 7] attempts to find a good hash function hypothesizing that a better hash function will improve the detection accuracy.

Paraphrase detection is a similar task to NDD where similarity is defined at semantic level instead of lexical level. Several methods that use deep learning have been proposed to compute a representation in the embedding space that can capture semantic similarity more effectively irrespective of the lexical differences [17, 3]. However, even though consecutive versions go through some paraphrasing of already existing content, the common content across versions is largely preserved lexically. However, different versions of same document are not always near-duplicates or paraphrase of each other and may involve considerable addition/deletion of content, which NDD/paraphrasing methods fail to consider.

For plagiarism detection, similarity can be thought of as having overlapping segments between the documents. Accordingly, [15, 8, 5] focus on finding overlapping segments across documents along with the extent of overlap. [15] used sentence-level tf-idf to quantify similarity at sentence level and proposed an algorithm to find maximal length overlapping segments. [5] used a Vector Space Model (VSM) with Parts-Of-Speech (POS) tagging, Named Entity Recognition (NER), etc., to generate potential candidates for plagiarism detection. [8] used a similar approach but only with POS tags and n-grams. While these approaches can be useful in detecting versions in a pair-wise manner, they do not solve the task of finding appropriate *sets* of documents and do not attempt to identify the ordering. Also, these methods rely largely on tf-idf frequencies and do not capture the patterns inside matching versions.

Given the motivation, the need for being able to address the problem of identifying versions from a collection is validated, and therefore, both an approach and dataset for this task are missing links we attempt to address in this paper.

3 Proposed Methodology

The key intuition we use to detect the versions of a document is that typically, across versions, the local lexical structure is preserved to a degree, even if at global level, it changes significantly. Typical edits to evolve the document from a version to the next version are insertion of new content, deletion of some of content, and replacing small parts of content. Sometimes, parts of content are also moved around with small degree of edits. In most cases, the edits do not replace very large parts of documents across versions. Hence, versions of a documents are not only semantically similar, even the words, sentences and paragraphs are largely preserved across versions. Further, even in case of paraphrasing, typically the entities are preserved even if the other words are changed.

Leveraging this insight, we design a three-stage approach. In the first stage, we want to *e ciently* find candidate set of documents that can *possibly* have a version relation among them, while retaining a high recall. Following that, we want to do more precise classification on a pairwise basis to determine if two documents are versions of each other. Finally, once we have pairwise version relation determined, we order all the versions correctly together. The reason why this task is required is due to the fact that not all predictions on the document pair level are going to be correct. For example, some additional pairs may be deemed to be versions of each other, which makes it difficult to know what is the correct order of these versions of the document.

3.1 Candidate Selection

Given that for a collection with n documents, the number of pairs of document are $O(n^2)$, which can be computationally expensive if we naively compare every pair of documents. So we use MinHash-LSH [2] based approach to find these candidate sets efficiently. However, instead of a direct application of MinHash-LSH on document level, we propose a two-step approach for better accuracy. We take the paragraphs as unit and compute the MinHash for each of the paragraphs. We use Locality Sensitive Hashing (LSH) to put the paragraphs in the same bucket if they are lexically very similar (defined as having a Jaccard similarity higher than a threshold when considering the paragraphs as sets of words). LSH can be tuned to put similar items in the same bucket and putting dissimilar items in different buckets with certain level of probability [2].

We then generate the candidate sets in the following manner. We scan the buckets one at a time. For each pair of paragraphs in one bucket, we increment count of votes for the pair of documents from which these paragraphs come from. Now, after scanning all the buckets, we check the number of votes for each pair of documents and compare that with the length of the documents in terms of paragraphs. We deem the document pairs as a candidate if the number of votes for this pair are larger than a certain fraction of the number of paragraphs in the smaller of the two documents.

The idea of dividing the document into smaller units (sections) was used in [9] also. However, their approach was to divide the document into a fixed (and typically, small) number of *sections* and look for match not only for words, but also for in which section of document are they found. By using this additional information (about location of the words), they improve the error metric compared to a vanilla MinHash-LSH based Near Duplicate Detection system. However, for our setting, there can be significant changes between versions of the document, we want to not only look for existence of similar paragraphs but want to allow for addition/deletion of content (in addition to small changes to content within preserved paragraphs).

3.2 FCN-based Binary Classification

The next step is more precise binary classification for version relation of candidate pairs of documents. While our eventual task is to identify document instances that are versions of a document, and arrange them in an order, we believe that trying to solve a simpler version of this problem and then using post-processing to recover the order (as discussed in next subsection) may perform better.

Hence, we take three different definitions of a pair of documents being related by version relation. Consider a set of v documents that are versions of a document, say $D_1, D_2, \ldots, D_v$. Here, D_1 is the first version of a document, D_2 is second version, and so on, till D_v which is the latest version. The three definitions are as follows:

Definition 1. ***Undirected (and not necessarily adjacent):*** *Any two of these documents D_i and D_j, $1 <= i, j <= v, i \neq j$ are considered as related.*

Definition 2. ***Directed (but not necessarily adjacent):*** *Two of these documents D_i and D_j are considered as related if $1 <= i < j <= v$.*

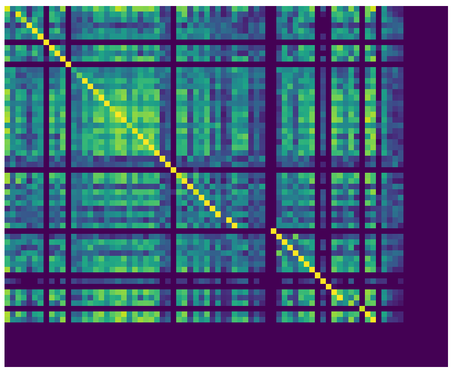
Definition 3. ***Adjacent (and directed):*** *Two of these documents D_i and D_j are considered as related if $1 <= i < j <= v$ and $i + 1 = j$.*

In addition to the local lexical structure preservation, we believe that typically, the entities present in the document tend to be preserved across versions. Based on these two hypotheses, we design our feature set for the classifiers for a candidate pair of documents. We construct a matrix of dimensions $m \quad n$, where m and n are the lengths of the first and second document respectively in the pair in terms of number of sentences. Please note that while for candidate generation, we operated at paragraph level, for the binary classification task, we are operating at a sentence level.

We compute similarity between every sentence i of first document with every sentence j of the second document based on lexical structure, denoted by S^L_{ij} and based on entities involved, denoted by S^E_{ij}. Lexical similarity S^L_{ij} is computed by representing the sentences in terms of sentence level TF-IDF (used earlier in [15] for plagiarism detection, essentially treating each sentence as a document and computing TF-IDF based on this treatment) vector representation and taking the cosine similarity. Entity based similarity S^E_{ij} is taken as the Jaccard similarity between the entities present in the two sentences.

While it is not obvious if two documents are versions of each other by analyzing the individual values within these matrices, it is fairly straightforward to observe emergent patterns in the 2D heatmaps constructed using these raw values. We can clearly see presence of local diagonals like patterns in the heatmap in Figure 1a. On the other hand, Figure 1b shows example of document pair which may have overlapping words, but the clear local diagonal patterns are missing. We train FCN-based binary classifier on these heatmaps.

Fully Convolutional Networks [12] have proved to be capable of modeling complex patterns in visual data by extracting meaningful features from images


(a) Prominent local diagonals

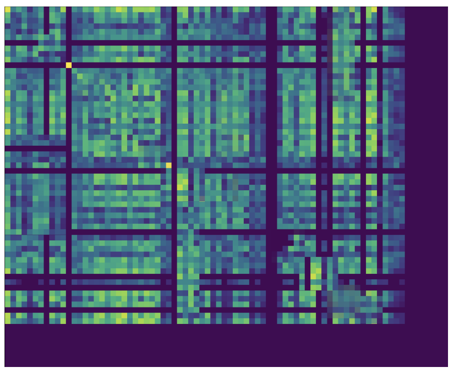

(b) Local diagonals are absent

Fig. 1: Examples of pair of documents that are (a) versions of each other, and (b) that are not

given supervised data. This motivated us to leverage FCNs for our usage for identifying the version relatedness between two documents. We specifically choose to go with FCN (as opposed to other convolutional networks) as they are invariant to the size of the heatmap and they can provide equally rich representation as the standard CNN architecture. They have a clear advantage while working with images which have a skewed aspect ratio. Our heatmap is clearly not square in shape, hence, if we use interpolation technique over it to fit to certain size (as required by conventional CNNs), the image has a high chance of getting distorted, i.e., the patterns of local diagonal in the heatmap can get diluted due to bi-linear interpolation. However, with FCN the exact size does not matter if it is less than the max_size. Even if the image dimensions do exceed a certain maximum, we can easily pool it back to the required size. In the case of max-pool it only changes the granularity of sentence level heatmap, i.e., in that case we can think of the heatmap as sentence similarity matrix with a window and stride of 2 sentences. We train the FCN directly on two channel heatmap (one channel corresponding to lexical similarity and another for entity-based similarity) to predict if the two documents are related by the version relation. This allows the model to automatically extract the useful patterns.

As originally proposed [12], the FCN is composed of both encoders and decoder parts. On the other hand, for our formulation we use just the encoder for compressing the similarity heatmap to a feature vector which is then used for binary classification. This is a little different to what FCNs are used for in general, that is semantic segmentation. But we do borrow and preserve an interesting property of convolution networks which is the fact that they respond to specific local patterns in an image. The heatmap constructed by us may have patterns like "local diagonals" which may be useful for classification. Another motivation for using FCN instead of a normal CNN is that FCNs are robust to the dimensions of the image. Since the documents have variable sizes, this means that the heatmaps themselves are of different aspect ratios (often becoming really extreme). FCN helps address the above problem. In practice the encoder also needs a fixed size image (200 200 in our case), so we pad the rest of the image by zeros. If the input has a larger length than 200 in any dimension, we max pool it to bring it down inside the 200 200 box (We can also apply max pool across 2D spatial-dimension after applying FCN for images of larger dimensions). Note that we also need to pad in case of CNNs but it's ability to detect patterns can be hampered by various aspect ratios which is not the case for FCNs. The only difference in our case when compared to a standard CNN is that we are using 1 1 convolution in place of a fully connected layer.

3.3 Finding the Order Among Versions

If we think of each instance of every document as a node and adjacent directed version relation as a directed edge, then the versions of a document will form a directed chain, if there is only one evolution path of the document. If the document is repurposed, we may see multiple outgoing edge from some nodes, and hence may get a tree. Our objective is to partition the set of all nodes as the trees/directed chains of versions of documents. The pairwise labelling may not provide sufficient information to construct this tree/chain if we use *Undirected* (definition 1) or *Directed* (definition 2) definitions of version related pairs. *Adjacent* (definition 3) definition provides sufficient information to construct the tree/directed chains in theory. However, in practice, the accuracy of models for this definition is lower (as the task is harder). Hence, we have many more false positives (which may create a graph more complex than trees/chains, including cycles) and false negatives (which may break the group of versions of a document into multiple chains/subgraphs) as we see in our experimental results.

Hence, we introduce a third part of the processing, where we take the binary classifier output and construct the graph. We then find all connected components within this graph and create fully connected subgraph (clique) for each connected component by adding all the missing edges. This step helps us in further **increasing our recall** by providing additional candidates for our final objective. Then the edges are assigned weight by computing a 'version score' between the pair. Finally, we compute the maximum weighted spanning tree on this graph.

We propose two ways of calculating the version scores. First is directly taking the FCN output as a number in range $[0; 1]$ instead of the binary output, which we call as FCN-Predict.

Another way to define the version score for the edge weight is as follows (called as alignment version score):

$$V = S \quad {}_1 \quad I \quad {}_2 \quad D \tag{1}$$

where, S is the alignment score (explained later), I denotes the number of sentence insertions, and D denotes the number of sentence deletions, and ${}_1$ and ${}_2$ are hyperparameters. Version Score can be understood to be a measure of alignment (or inverse of 'distance', in terms of changes required to convert one document to the other, between them) between two documents. Once we compute the version scores, for each connected component, we make the smallest document as the root of the tree and then do a topological ordering of the above tree to use that as a chain. This chain provides us with the final ordering among documents that are versions of each other.

To compute S, I, and D, we construct a heatmap M of sentence-level similarity by taking a linear combination of heatmaps that capture lexical (S^L) and entity-based (S^E) overlap. Alignment Score S is computed using Dynamic Programming algorithm over the matrix M with elements $S_{ij} = (S^L_{ij} + {}_3 S^E_{ij})$ (representing the overall similarity between sentence i of document 1 with sentence j of document 2, with ${}_3$ being a hyperparameter) to find maximum possible reward to go from the index $(0; 0)$ to index $(m; n)$ where m and n are number of sentences in documents i and j respectively.

At each element of the matrix M, we have 3 options, move diagonally with reward $R_d = S_{ij}$ (corresponding to the sentence being modified from sentence i of document 1 to sentence j of document 2), move horizontally (corresponding to deleting a sentence from document 1, counted in D) or move vertically (corresponding to inserting a sentence into document 2, counted in I). The goal of this approach is to quantify the extent of alignment between two documents based on minimum transformations required to convert document 1 to document 2 by finding the highest reward traversed path from the index $(0, 0)$ to (m, n). If the documents were exactly same, the path would lie along the diagonal with all S_{ij} values as 1. However, if a new sentence j is added to document 2, the traversal would involve a horizontal movement. Similarly, if a sentence has been deleted from document i, it would involve a vertical movement. Consequently, we quantify the number of sentence insertions I and sentence deletions D as the number of horizontal moves and vertical moves in the traversed path, respectively, and subtract it from the alignment score S to get a version score V.

The intuition behind these two definitions of version score is as follows. The FCN based score is going to capture the level of preservation of local lexical structures and entities. However, since it is trained for a binary classification task, it is not clear if the score would be appropriate for the task we are using it for. On the other hand, the direct alignment based score is specifically designed for this task. However, it would not capture the similarity present between documents if some segments of the document are moved within it. Hence, we use both of these and determine the appropriate method empirically.

4 Wikipedia Versions Dataset

Due to lack of an existing large-scale dataset suitable for our problem, we curated a new dataset using revision history of Wikipedia articles. We took Wikipedia revision dumps that contain all revisions of Wikipedia pages on a certain topic. Each revision is accompanied by some metadata (like timestamp, parent_id, etc.). To ensure that we consider versions that are sufficiently different from each other, we use timestamps in an automated filtering process to create the dataset.

Each edit made by the user to a Wikipedia page can be considered a separate version of that Wikipedia page (these are referred to as revisions in the Wikipedia terminology). However, often there are some artifacts in the revisions – for example, troll edits are insincere edits made by users. We observe that these edits do not last, and the page is reset to the last relevant version by the moderators within a short duration. Based on this observation, we remove all edits that do not last more than AVE=10 seconds, where AVE is defined as the average time between successive edits for a given Wikipedia page. Instead of setting a global threshold, this threshold varies for every page because some pages are edited more frequently than others. We use timestamps obtained along with metadata for the purpose of filtering. Next, we filter out pages that are rarely edited. This is done by setting a hard threshold – i.e., pages should consist of at least 10 versions. After this we stochastically sample $\lfloor U(min, width) \rfloor$ revisions from all versions, U is the uniform distribution, min is the minimum number of samples, $min + width + 1$ is the maximum number of samples, and $\lfloor . \rfloor$ is the floor function (greatest integer no larger than the number). We take min to be 7 and $width$ as 5. However, since it is still possible that the set of revisions obtained via uniform sampling don't have considerable differences between them, we stochastically filter out more revisions to encourage differences between them. This is done by having higher probability of sampling versions which have more difference in terms of length of the documents with the previously selected versions. This increases the probability of having the set of versions that differ more in content.

Once we obtain the list pages and the versions being considered for each of them, we scrape the Wikipedia article corresponding to each version. We remove all additional information in sections like References, Category, etc. from the scraped articles. This is done to ensure that the model doesn't utilize this information to learn undesirable patterns, and to ensure that the identification of relations is solely based on the content.

Our final dataset comprises 1;755 unique Wikipedia articles with a total of 10;267 versions; each article having 5:85 versions on average. We split the dataset into train, validation, and test sets in the ratio of 0:7 : 0:1 : 0:2, respectively. As mentioned in the introduction, we are releasing a dataset generated in this manner to the research community.

5 Evaluation and Results

In this section, we present our experimental results. First, we discuss the baseline we design to compare its performance against that of our approach.

Baselines: We use the Sectional MinHashLSH that achieves state-of-the-art results on near-duplicate detection task [9] as the baseline for binary classification. For ordering, we use the content length as baseline, i.e., a longer document is deemed as more recent version compared to a shorter ones.

Parameter Choices: All the hyperparameters were assigned via a grid search.

Candidate generation parameters: The threshold in MinHash-LSH for deeming two paragraphs as near duplicate was 0:5, and number of permutation functions used was 1024. The threshold on fraction of paragraphs (of the shorter document) that need be near-duplicate for including the pair of documents are candidate for being a version related pair was taken as 0:3.

FCN Architecture: There are 7 (2D-)Convolution layers followed by a Sigmoid layer in our implementation of FCN. The stride for all the convolution layers is 1 1 and activation function is *ReLU*. All convolution layers are followed by (2D-)batch normalization [10] (except last convolution layer) and (2D-)max pooling (with kernel size 2 2) except last two convolution layers. The kernel size is 5 5 for first, second and fifth convolution layers, 7 7 for third, 6 6 for fourth, 3 3 for sixth and 1 1 for the seventh convolution layer (and therefore, it is like a fully connected layer). A padding of 2 2 is used in first two layers and 3 3 for third layer. Other layers do not use padding.

Version ordering related parameters: $_1$ and $_2$ (weight of insertions and deletions in Equation 1, respectively) were taken as 0:05 and 0:15, respectively, and $_3$ (weight of entity-based similarity with respect to lexical similarity) was taken as 1:4.

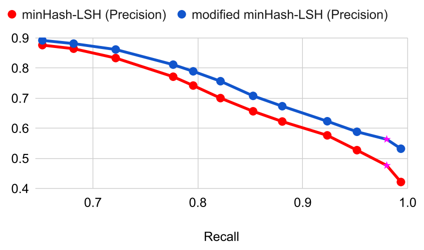


Fig. 2: Comparison of Sectional MinHash-LSH and modified MinHash-LSH

Candidate Selection: We first evaluate the effectiveness of MinHash-LSH for candidate selection. Given the size of our dataset ($2;068$ versions of 351 documents in the test set), an all-pair comparison would lead to more than $4:2$ million (for **Adjacent**; for **Undirected** and **Directed**, it is half as many) pairwise checks. The ground truth number of version pairs are $10;426$ for definition 1 (**Undirected**), $5;213$ for definition 2 (**Directed**) and $2;053$ for definition 3 (**Adjacent**). Figure 2 shows the precision and recall trade-off curves for the Sectional MinHash-LSH based approach and our proposed approach for finding the candidate pairs. Based on the operating point we choose (shown by $?$ in Figure), we end up with $19;307$ pairs for **Undirected**, $9;667$ for **Directed** and $3;876$ for **Adjacent**, which are significantly lower than the naïve quadratic number of pairs. Also, Figure 2 shows that our proposed two step method achieves up to 10% higher precision for a given recall value compared to Sectional MinHash-LSH method. Also, the gap in performance of Sectional MinHash-LSH and our proposed method increases as we go towards the higher recall side, which are the preferred operating point, as we do not want to discard too many actual version pairs in this stage.

Binary Classification: Now we compare the performance of our proposed approaches for binary classification task. We also train a Logistic Regression (LR) classifier using hand-crafted features obtained from the similarity heatmaps for comparison purpose. The features were obtained by quantifying the correlation ($corr(:)$) between the two indices of the lexical (S^L) and entity-based (S^E) similarity matrices discussed above. We use $corr(S^L)$, $corr(S^E)$, $corr(S^E \quad S^L)$, $corr((S^L)^{16})$, $corr((S^E)^{16})$, $corr((S^L + S^E)^4)$, and $corr((S^L \quad S^E)^4)$; denoting element-wise multiplication. We also experiment with several other higher powers in the range of $[2;16]$ and we find that the mentioned set of features perform the best. While training the logistic regression model with all the higher powers, we note that the learned -coefficients for powers which are not noted above were insignificant and could be dropped without a noticeable change in the final precision or recall scores.

To evaluate the performance of models, we consider the three settings defined earlier: *(a)* undirected and not necessarily adjacent (*Undirected*), *(b)* directed but not necessarily adjacent (*Directed*), and *(c)* directed and adjacent (*Adjacent*). As it is evident from the discussion earlier, these three settings progressively become

Table 1: Performance after the binary classification stage (average of 5 runs). Columns in the left indicate the performance of various models on the end-to-end task, whereas the ones in the right indicate the performance of binary classifiers alone on the tasks they were trained for, respectively

		Candidate Gen + Classifier wrt GT			Classifier Only for Respective Tasks		
Type	Model	F_1	Precision	Recall	F_1	Precision	Recall
Undirected	Baseline	0:3361	0:3210	0:3527	0:4163	0:4367	0:3977
	LR-LS	0:3718	0:4089	0:3410	0:7216	0:7342	0:7094
	LR-Full	0:3817	0:4169	0:3520	0:7813	0:7914	0:7714
	FCN-LS	0:6012	0:6373	0:5690	0:7862	0:8006	0:7723
	FCN-Full	**0:6339**	**0:7087**	**0:5734**	**0:8743**	**0:9031**	**0:8472**
Directed	Baseline	0:3361	0:3210	0:3527	0:3642	0:3743	0:3546
	LR-LS	0:3519	0:4063	0:3101	0:4179	0:4332	0:4036
	LR-Full	0:3853	0:4371	0:3461	0:4354	0:4679	0:4071
	FCN-LS	0:6191	0:6371	0:6020	0:6672	0:6817	0:6533
	FCN-Full	**0:6547**	**0:7216**	**0:5991**	**0:7246**	**0:8312**	**0:6422**
Adjacent	Baseline	0:3361	0:3210	0:3527	0:3361	0:3210	0:3527
	LR-LS	0:3819	0:4197	0:3504	0:3976	0:4420	0:3613
	LR-Full	0:4011	0:4562	0:3579	0:4139	0:4837	0:3617
	FCN-LS	0:6376	0:6551	0:6210	0:6306	0:6306	0:6317
	FCN-Full	**0:6987**	**0:7832**	**0:6306**	**0:7036**	**0:7583**	**0:6562**

more challenging, but also get closer to the actual task at hand. For instance, detecting adjacent relations is harder than detecting directed relations, as it involves not only identifying that document A comes before B, but also whether it comes *immediately* before B. For each of the three settings, we construct negative and positive examples according to the above formulation and quantify the performance of our models. Since the number of negative samples are much more in number, we sample three times the number of positive samples from the set of all negative samples to get a reasonably balanced training data set.

We run the Logistic Regression and FCN networks on only the lexical similarity heat map (LR-LS and FCN-LS, respectively), as well as, on both the lexical and entity-based similarity (LR-Full and FCN-Full, respectively), in addition to the baseline. Table 1 shows the performance comparison. The left-hand side columns give result for the eventual task of identifying consecutive versions in correct order. Please note these results compare the performance with respect to the ground truth, so the loss of recall due to candidate selection is also reflected in the loss of recall in these columns. Of course, the baseline operates directly on the full set and does not have corresponding loss of recall due to any pre-processing. In all the three settings FCN outperforms LR, and they both outperform the baseline significantly. Further, one can also see that by including the entity-based similarity (in FCN-Full and LR-Full), the performance for the classifier improves significantly.

The right-hand side columns of Table 1 give results for the tasks the classifiers were trained on. Here, the performance is with respect to the candidate pairs (and not ground truth). As one can see, our proposed FCN-Full classifier performs considerably better than other models, consistently on all metrics and tasks. Since the performance of LR depends on features that are symmetric with respect to document order, it performs a lot worse in identifying directed and adjacent version relations. We also note again that including entity-based simi-

Table 2: Performance of models trained on different task on the end-to-end task of detecting adjacent versions (average of 5 runs).

Type	Model	F_1	Precision	Recall
Baseline		0:4231	0:3997	0:4494
Undirected	FCN-Predict	0:6643	0:7012	0:6310
	FCN-Full	**0:6761**	**0:7128**	**0:6430**
Directed	FCN-Predict	**0:7013**	**0:7281**	**0:6764**
	FCN-Full	0:6942	0:7273	0:6640
Adjacent	FCN-Predict	0:7116	**0:7560**	0:6721
	FCN-Full	**0:7142**	0:7333	**0:6860**

Table 3: Performance of models trained on different task on task of detecting undirected/directed/adjacent versions, as applicable (average of 5 runs)

Type	Model	F_1	Precision	Recall
Undirected	Baseline	0:7731	0:7518	0:7956
	FCN-Predict	0:8933	0:9110	0:8763
	FCN-Full	**0:8961**	**0:9122**	**0:8805**
Directed	Baseline	0:7374	0:7140	0:7623
	FCN-Predict	**0:8291**	**0:8648**	0:7962
	FCN-Full	0:8261	0:8431	**0:8012**
Adjacent	Baseline	0:4231	0:3997	0:4494
	FCN-Predict	0:7116	**0:7560**	0:6721
	FCN-Full	**0:7142**	0:7333	**0:6860**

larity also helps in detecting version, as the performance of FCN-Full (LR-Full) is better than FCN-LS (LR-LS), which only uses lexical similarity heatmaps. We also note that baseline performs significantly below the proposed approaches, even though, these tasks (particularly, Undirected setting) are closer to what the baseline was designed for.

Version Ordering: We now discuss the result of arranging the versions in full ordered sequence. Since for this dataset, the versions form a chain instead of general directed acyclic graph, we restrict our algorithm to find directed chains with maximum weights. Comparing the results given in Table 1 (for stage 1 and 2) with Table 2 (for stage 1, 2 and 3) shows that our proposed step helps improve the results significantly for Undirected and Directed settings (6%), whereas for Adjacent, the improvement is modest (2%), as expected. While using Adjacent setting still leads to best performance on the overall task, the gap in performance is reduced significantly. Also, please note that the two methods that we use to compute the version scores – using the prediction scores of FCN (FCN-Predict) and using the edge weights based on alignment based version score as described in Section 3.3 Equation 1 (FCN-Full), yield similar results consistently across different tasks and metrics.

While our objective was to find directed ordered chains of versions of documents, we also present the result for a task based on *Undirected, Directed* and *Adjacent*. In Table 3, we see that the performance of our proposed approach is consistently better than the baseline approach across all the three settings and the three metrics. This shows that even though the baseline methods were closest to *Undirected*, our method still outperforms it significantly. Of course, for the objective we set out to achieve, the ratio of performance is even more impressive in favour of our proposed approach. As expected, we also observe a consistent drop across the three metrics as we move from the *Undirected* setting to the *Directed* setting, and then to the *Adjacent* setting. We attribute this drop to the progressive increase in task's difficult, as discussed earlier.

6 Conclusion

In this work we introduce a novel problem, namely, detecting and ordering versions of documents (in which they were created) from a document repository. We created a dataset, which we share with the research community, and also provided an end-to-end approach for the task. The dataset will help further the research in this domain. Our proposed approach involves three parts where we start by finding candidate documents using MinHash-LSH and then classify the selected candidates using an FCN-based classifier. The last step of our approach involves identifying the order of versions by finding the maximum spanning tree in the graph where documents are considered to be nodes and the edge-weights denote the version score between them. Our empirical results show that the individual parts of our proposed approach perform better than standard approaches and the end-to-end approach outperforms the state-of-the-art approach for near-duplicate document detection (NDD). This also indicates that the problem is sufficiently different than NDD, and therefore worthy of further research.

References

1. Alsulami, B.S., Abulkhair, M.F., Eassa, F.E.: Near duplicate document detection survey. International Journal of Computer Science and Communications Networks **2**(2), 147–151 (2012)
2. Broder, A.Z.: Identifying and filtering near-duplicate documents. In: Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching. p. 1–10. COM '00, Springer-Verlag, Berlin, Heidelberg (2000)
3. Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder–decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 1724–1734 (2014)
4. Chowdhury, A., Frieder, O., Grossman, D., McCabe, M.C.: Collection statistics for fast duplicate document detection. ACM Transactions on Information Systems (TOIS) **20**(2), 171–191 (2002)
5. Ekbal, A., Saha, S., Choudhary, G.: Plagiarism detection in text using vector space model. In: 2012 12th International Conference on Hybrid Intelligent Systems (HIS). pp. 366–371. IEEE (2012)
6. Elsayed, T., Lin, J., Oard, D.W.: Pairwise document similarity in large collections with mapreduce. In: Proceedings of ACL-08: HLT, Short Papers. pp. 265–268 (2008)
7. Ertl, O.: SuperMinHash - A New Minwise Hashing Algorithm for Jaccard Similarity Estimation. arXiv e-prints arXiv:1706.05698 (Jun 2017)

8. Gupta, D., Vani, K., Leema, L.: Plagiarism detection in text documents using sentence bounded stop word n-grams. Journal of Engineering Science and Technology **11**(10), 1403–1420 (2016)
9. Hassanian-esfahani, R., Kargar, M.j.: Sectional minhash for near-duplicate detection. Expert Systems with Applications **99**, 203–212 (2018)
10. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning. pp. 448–456. PMLR (2015)
11. Liu, X., Gong, Y., Xu, W., Zhu, S.: Document clustering with cluster refinement and model selection capabilities. In: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval. pp. 191–198 (2002)
12. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3431–3440 (2015)
13. Lv, Y., Moon, T., Kolari, P., Zheng, Z., Wang, X., Chang, Y.: Learning to model relatedness for news recommendation. In: Proceedings of the 20th international conference on World wide web. pp. 57–66 (2011)
14. Park, K., Lee, J., Choi, J.: Deep neural networks for news recommendations. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. pp. 2255–2258 (2017)
15. Sanchez-Perez, M.A., Sidorov, G., Gelbukh, A.F.: A winning approach to text alignment for text reuse detection at pan 2014. In: CLEF (Working Notes). pp. 1004–1011 (2014)
16. Sherkat, E., Nourashrafeddin, S., Milios, E.E., Minghim, R.: Interactive document clustering revisited: A visual analytics approach. In: 23rd International Conference on Intelligent User Interfaces. pp. 281–292 (2018)
17. Socher, R., Huang, E.H., Pennin, J., Manning, C.D., Ng, A.Y.: Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In: Advances in neural information processing systems. pp. 801–809 (2011)
18. Weng, S.S., Chang, H.L.: Using ontology network analysis for research document recommendation. Expert Systems with Applications **34**(3), 1857–1869 (2008). https://doi.org/https://doi.org/10.1016/j.eswa.2007.02.023, https://www.sciencedirect.com/science/article/pii/S0957417407000620
19. Xu, X., Hassan Awadallah, A., T. Dumais, S., Omar, F., Popp, B., Rounthwaite, R., Jahanbakhsh, F.: Understanding user behavior for document recommendation. In: Proceedings of The Web Conference 2020. p. 3012–3018. WWW '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3366423.3380071, https://doi.org/10.1145/3366423.3380071
20. Yang, X., Lo, D., Xia, X., Bao, L., Sun, J.: Combining word embedding with information retrieval to recommend similar bug reports. In: 2016 IEEE 27Th international symposium on software reliability engineering (ISSRE). pp. 127–137. IEEE (2016)