Semantic Document Distance Measures and Unsupervised Document Revision Detection

Xiaofeng Zhu, Diego Klabjan
Northwestern University, USA
xiaofengzhu2013@u.northwestern.edu
d-klabjan@northwestern.edu

Patrick N Bless
Intel Corporation,
Chandler, AZ, USA
patrick.n.bless@intel.com

Abstract

In this paper, we model the document revision detection problem as a minimum cost branching problem that relies on computing document distances. Furthermore, we propose two new document distance measures, word vector-based Dynamic Time Warping (wDTW) and word vector-based Tree Edit Distance (wTED). Our revision detection system is designed for a large scale corpus and implemented in Apache Spark. We demonstrate that our system can more precisely detect revisions than state-of-the-art methods by utilizing the Wikipedia revision dumps [1] and simulated data sets.

1 Introduction

It is a common habit for people to keep several versions of documents, which creates duplicate data. A scholarly article is normally revised several times before being published. An academic paper may be listed on personal websites, digital conference libraries, Google Scholar, etc. In major corporations, a document typically goes through several revisions involving multiple editors and authors. Users would benefit from visualizing the entire history of a document. Currently, there are no unsupervised methodologies for detecting document revisions, and the increasing number of revised or duplicate digital documents poses challenges for retrieving the most relevant information. Hence, it is worthwhile to develop a system that is able to intelligently identify, manage and represent revisions. Given a collection of text documents, our study identifies revision relationships in a completely unsupervised way. For each document in a corpus we only use its content and the last modified timestamp. We assume that a document can be revised by many users, but that the documents are not merged together. We consider collaborative editing as revising documents one by one.

The two research problems that are most relevant to document revision detection are plagiarism detection and revision provenance. In a plagiarism detection system, every incoming document is compared with all registered non-plagiarized documents (Si et al., 1997; Oberreuter and VeláSquez, 2013; Hagen et al., 2015; Abdi et al., 2015). The system returns true if an original copy is found in the database; otherwise, the system returns false and adds the document to the database. Thus, it is a 1-to-n problem. Revision provenance is a 1-to-1 problem as it keeps track of detailed updates of one document (Buneman et al., 2001; Zhang and Jagadish, 2013). Real-world applications include GitHub, version control in Microsoft Word and Wikipedia version trees (Sabel, 2007). In contrast, our system solves an n-to-n problem on a large scale. Our potential target data sources, such as the entire web or internal corpora in corporations, contain numerous original documents and their revisions. The aim is to find all revision document pairs within a reasonable time.

Document revision detection, plagiarism detection and revision provenance all rely on comparing the content of two documents and assessing a distance/similarity score. The classic document similarity measure, especially for plagiarism detection, is fingerprinting (Hoad and Zobel, 2003; Schleimer et al., 2003). For every document, a fingerprint is created using hash functions to represent the content and is applied to measure document similarities. However, it is difficult to design a unique hash code for every document as the corpus size increases. Another widely used approach is computing the sentence-to-sentence

[1] https://snap.stanford.edu/data/wiki-meta.html

Levenshtein distance and assigning an overall score for every document pair (Gustafson et al., 2008). Nevertheless, due to the large number of existing documents, as well as the large number of sentences in each document, the Levenshtein distance is not computation-friendly. Although alternatives such as the vector space model (VSM) can largely reduce the computation time, their effectiveness is low. More importantly, none of the above approaches can capture semantic meanings of words, which heavily limits the performances of these approaches. For instance, from a semantic perspective, "*I went to the bank*" is expected to be similar to "*I withdrew some money*" rather than "*I went hiking.*" Our document distance measures are inspired by the weaknesses of current document distance/similarity measures and recently proposed models for word representations such as word2vec (Mikolov et al., 2013) and Paragraph Vector (PV) (Le and Mikolov, 2014). Replacing words with distributed vector embeddings makes it feasible to measure semantic distances using advanced algorithms, e.g., Dynamic Time Warping (DTW) (Sakurai et al., 2005; Müller, 2007; Matuschek et al., 2008) and Tree Edit Distance (TED) (Tai, 1979; Zhang and Shasha, 1989; Klein, 1998; Demaine et al., 2007; Pawlik and Augsten, 2011, 2014, 2015, 2016). Although calculating text distance using DTW (Liu et al., 2007), TED (Sidorov et al., 2015) or Word Mover's Distance (WMV) (Kusner et al., 2015) has been attempted in the past, these measures are not ideal for large-scale document distance calculation. The first two algorithms were designed for sentence distances instead of document distances. The third measure computes the distance of two documents by solving a transshipment problem between words in the two documents and uses word2vec embeddings to calculate semantic distances of words. The biggest limitation of WMV is its long computation time. We show in Section 5.3 that our wDTW and wTED measures yield more precise distance scores with much shorter running time than WMV.

We recast the problem of detecting document revisions as a network optimization problem (see Section 2) and consequently as a set of document distance problems (see Section 4). We use trained word vectors to represent words, concatenate the word vectors to represent documents and combine word2vec with DTW or TED. Meanwhile, in order to guarantee reasonable computation time in large data sets, we calculate document distances at the paragraph level with Apache Spark. A distance score is computed by feeding paragraph representations to DTW or TED.

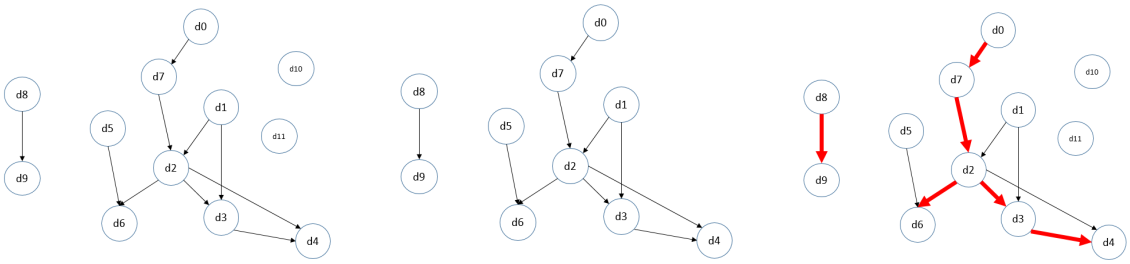
The primary contributions of this work are as follows.

- We specify a model and algorithm to find the optimal document revision network from a large corpus.
- We propose two algorithms, wDTW and wTED, for measuring semantic document distances based on distributed representations of words. The wDTW algorithm calculates document distances based on DTW by sequentially comparing any two paragraphs of two documents. The wTED method represents the section and subsection structures of a document in a tree with paragraphs being leaves. Both algorithms hinge on the distance between two paragraphs.

2 Revision Network

The two requirements for a document $\bar{D}$ being a revision of another document $\tilde{D}$ are that $\bar{D}$ has been created later than $\tilde{D}$ and that the content of $\bar{D}$ is similar to (has been modified from) that of $\tilde{D}$. More specifically, given corpus $\mathscr{D}$, for any two documents $\bar{D}, \tilde{D} \in \mathscr{D}$, we want to find out the yes/no revision relationship of $\bar{D}$ and $\tilde{D}$, and then output all such revision pairs.

We assume that each document has a creation date (the last modified timestamp) which is readily available from the meta data of the document. In this section we also assume that we have a $Dist$ method and a cut-off threshold τ. We represent a corpus as network $N^0 = (V^0, A)$, for example Figure 1a, in which a vertex corresponds to a document. There is an arc $a = (\bar{D}, \tilde{D})$ if and only if $Dist(\bar{D}, \tilde{D}) \leq \tau$ and the creation date of $\bar{D}$ is before the creation date of $\tilde{D}$. In other words, $\tilde{D}$ is a revision candidate for $\bar{D}$. By construction, N^0 is acyclic. For instance, d_2 is a revision candidate for d_7 and d_1. Note that we allow one document to be the original document of several revised documents. As we only need to focus on revision candidates, we reduce N^0 to $N = (V, A)$, shown in Figure 1b, by removing isolated vertices. We define the weight of an arc as the distance score between the two vertices. Recall the assumption that a document can be a revision of at most one



(a) Revision network N^0 (b) Cleaned revision network N (c) Possible solution R

Figure 1: Revision network visualization

document. In other words, documents cannot be merged. Due to this assumption, all revision pairs form a branching in N. (A branching is a subgraph where each vertex has an in-degree of at most 1.) The document revision problem is to find a minimum cost branching in N (see Fig 1c).

The minimum branching problem was first solved by Edmonds (1967). The details of his algorithm are as follows.

- For each node select the smallest weighted incoming arc. This yields a subgraph.
- If cycles are present in the selected subgraph, then recursively find the replacing arc that has the minimum weight among previously non-selected arcs to eliminate cycles.

In our case, N is acyclic and, therefore, the second step never occurs. For this reason, Algorithm 1 solves the document revision problem.

Algorithm 1 Find minimum branching R for network $N = (V, A)$

1: **Input:** N
2: $R = \emptyset$
3: **for** every vertex $v \in V$ **do**
4: Set $\delta(u)$ to correspond to all arcs with head u.
5: Select $a = (v, u) \in \delta(u)$ such that $Dist(v, u)$ is minimum
6: $R = R \cup a$
7: **end for**
8: **Output:** R

The essential part of determining the minimum branching R is extracting arcs with the lowest distance scores. This is equivalent to finding the most similar document from the revision candidates for every original document.

3 distance/similarity Measures

In this section, we first introduce the classic VSM model, the word2vec model, DTW and TED. We next demonstrate how to combine the above components to construct our semantic document distance measures: wDTW and wTED. We also discuss the implementation of our revision detection system.

3.1 Background

3.1.1 Vector Space Model

VSM represents a set of documents as vectors of identifiers. The identifier of a word used in this work is the tf-idf weight. We represent documents as tf-idf vectors, and thus the similarity of two documents can be described by the cosine distance between their vectors. VSM has low algorithm complexity but cannot represent the semantics of words since it is based on the bag-of-words assumption.

3.1.2 Word2vec

Word2vec produces semantic embeddings for words using a two-layer neural network. Specifically, word2vec relies on a skip-gram model that uses the current word to predict context words in a surrounding window to maximize the average log probability. Words with similar meanings tend to have similar embeddings.

3.1.3 Dynamic Time Warping

DTW was developed originally for speech recognition in time series analysis and has been widely used to measure the distance between two sequences of vectors.

Given two sequences of feature vectors: $A = a_1, a_2, ..., a_i, ..., a_m$ and $B = b_1, b_2, ..., b_j, ..., b_n$, DTW finds the optimal alignment for A and B by first constructing an $(m + 1) \times (n + 1)$ matrix in which the $(i, j)^{th}$ element is the alignment cost

of $a_1...a_i$ and $b_1...b_j$, and then retrieving the path from one corner to the diagonal one through the matrix that has the minimal cumulative distance. This algorithm is described by the following formula.

$$\begin{aligned} DTW(i,j) = Dist(a_i, b_j) + min(\\ & DTW(i-1,j), \quad //insertion \\ & DTW(i,j-1), \quad //deletion \\ & DTW(i-1,j-1)) \; //substitution \end{aligned}$$

3.1.4 Tree Edit Distance

TED was initially defined to calculate the minimal cost of node edit operations for transforming one labeled tree into another. The node edit operations are defined as follows.

- **Deletion** Delete a node and connect its children to its parent maintaining the order.
- **Insertion** Insert a node between an existing node and a subsequence of consecutive children of this node.
- **Substitution** Rename the label of a node.

Let L_1 and L_2 be two labeled trees, and $L_k[i]$ be the i^{th} node in $L_k (k = 1, 2)$. M corresponds to a mapping from L_1 to L_2. TED finds mapping M with the minimal edit cost based on

$$\begin{aligned} c(M) = min\{ & \sum_{(i,j)\in M} cost(L_1[i] \to L_2[j]) \\ & + \sum_{i \in I} cost(L_1[i] \to \wedge) \\ & + \sum_{j \in J} cost(\wedge \to L_2[j])\} \end{aligned}$$

where $L_1[i] \to L_2[j]$ means transferring $L_1[i]$ to $L_2[j]$ based on M, and $\wedge$ represents an empty node.

3.2 Semantic Distance between Paragraphs

According to the description of DTW in Section 3.1.3, the distance between two documents can be calculated using DTW by replacing each element in the feature vectors A and B with a word vector. However, computing the DTW distance between two documents at the word level is basically as expensive as calculating the Levenshtein distance. Thus in this section we propose an improved algorithm that is more appropriate for document distance calculation.

In order to receive semantic representations for documents and maintain a reasonable algorithm complexity, we use word2vec to train word vectors and represent each paragraph as a sequence of vectors. Note that in both wDTW and wTED we take document titles and section titles as paragraphs. Although a more recently proposed model PV can directly train vector representations for short texts such as movie reviews (Le and Mikolov, 2014), our experiments in Section 5.3 show that PV is not appropriate for standard paragraphs in general documents. Therefore, we use word2vec in our work. Algorithm 2 describes how we compute the distance between two paragraphs based on DTW and word vectors. The distance between one paragraph in a document and one paragraph in another document can be pre-calculated in parallel using Spark to provide faster computation for wDTW and wTED.

Algorithm 2 DistPara

Replace the words in paragraphs p and q with word2vec embeddings: $\{v_i\}_{i=1}^{e}$ and $\{w_j\}_{j=1}^{f}$
Input: $p = [v_1, .., v_e]$ and $q = [w_1, .., w_f]$
Initialize the first row and the first column of $(e+1) \times (f+1)$ matrix DTW_{para} $+\infty$
$DTW_{para}(0,0) = 0$
for i in range $(1, e+1)$ **do**
  for j in range $(1, f+1)$ **do**
    $Dist(v_i, w_j) = ||v_i - w_j||$
    calculate $DTW_{para}(i,j)$
  end for
end for
Return: $DTW_{para}(e,f)$

4 wDTW and wTED Measures

4.1 Word Vector-based Dynamic Time Warping

As a document can be considered as a sequence of paragraphs, wDTW returns the distance between two documents by applying another DTW on top of paragraphs. The cost function is exactly the DistPara distance of two paragraphs given in Algorithm 2. wDTW observes semantic information from word vectors, which is fundamentally different from the word distance calculated from hierarchies among words in the algorithm proposed by Liu et al. (2007). The shortcomings of their work are that it is difficult to learn semantic taxonomy of all words and that their DTW algorithm can only

be applied to sentences not documents.

Algorithm 3 wDTW

Represent documents d_1 and d_2 with vectors of paragraphs: $\{p_i\}_{i=1}^{m}$ and $\{q_j\}_{j=1}^{n}$
Input: $d_1 = [p_1, .., p_m]$ and $d_2 = [q_1, .., q_n]$
Initialize the first row and the first column of $(m+1) \times (n+1)$ matrix DTW_{doc} $+\infty$
$DTW_{doc}(0,0) = 0$
for i in range $(1, m+1)$ **do**
  for j in range $(1, n+1)$ **do**
    $Dist(p_i, q_j) =$ DistPara(p_i, q_j)
    calculate $DTW_{doc}(i,j)$
  end for
end for
Return: $DTW_{doc}(m,n)$

4.2 Word Vector-based Tree Edit Distance

TED is reasonable for measuring document distances as documents can be easily transformed to tree structures. A document can be viewed at multiple abstraction levels that include the document title, its sections, subsections, etc. Thus for each document we can build a tree-like structure with title $\rightarrow$ sections $\rightarrow$ subsections $\rightarrow$...$\rightarrow$ paragraphs being paths from the root to leaves. Child nodes are ordered from left to right as they appear in the document.

We represent labels in a document tree as the vector sequences of titles, sections, subsections and paragraphs with word2vec embeddings. wTED converts documents to tree structures and then uses DistPara distances. More formally, the distance between two nodes is computed as follows.

- The cost of substitution is the DistPara value of the two nodes.
- The cost of insertion is the DistPara value of an empty sequence and the label of the inserted node. This essentially means that the cost is the sum of the L2-norms of the word vectors in that node.
- The cost of deletion is the same as the cost of insertion.

Compared to the algorithm proposed by Sidorov et al. (2015), wTED provides different edit cost functions and uses document tree structures instead of syntactic n-grams, and thus wTED yields more meaningful distance scores for long documents. Algorithm 4 describes how we calculate the edit cost between two document trees.

Algorithm 4 wTED

1: Convert documents d_1 and d_2 to trees T_1 and T_2
2: **Input:** T_1 and T_2
3: Initialize tree edit distance $c = +\infty$
4: **for** node label $p \in T_1$ **do**
5: **for** node label $q \in T_2$ **do**
6: Update TED mapping cost c using
7: $cost(p \rightarrow q) =$ DistPara(p, q)
8: $cost(p \rightarrow \wedge) =$ DistPara$(p, \wedge)$
9: $cost(\wedge \rightarrow q) =$ DistPara$(\wedge, q)$
10: **end for**
11: **end for**
12: **Return:** c

4.3 Process Flow

Our system is a boosting learner that is composed of four modules: weak filter, strong filter, revision network and optimal subnetwork. First of all, we sort all documents by timestamps and pair up documents so that we only compare each document with documents that have been created earlier. In the first module, we calculate the VSM similarity scores for all pairs and eliminate those with scores that are lower than an empirical threshold ($\tilde{\tau} = 0.5$). This is what we call the weak filter. After that, we apply the strong filter wDTW or wTED on the available pairs and filter out document pairs having distances higher than a threshold τ. For VSM in Section 5.1, we directly filter out document pairs having similarity scores lower than a threshold τ. The cut-off threshold estimation is explained in Section 4.4. The remaining document pairs from the strong filter are then sent to the revision network module. In the end, we output the optimal revision pairs following the minimum branching strategy.

4.4 Estimating the Cut-off Threshold

Hyperprameter τ is calibrated by finding the absolute extreme based on an initial set of documents, i.e., all processed documents since the moment the system was put in use. Based on this set, we calculate all distance/similarity scores and create a histogram, see Figure 2.

The figure shows the correlation between the number of document pairs and the similarity

scores within one optimization process for VSM. The optimal τ in this example is between 0.6 and 0.7 where the slope of the graph noticeably changes from negative to positive.

As the system continues running, new documents become available and τ can be periodically updated by using the same method.

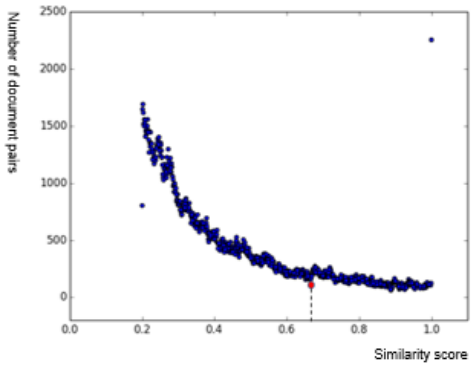


Figure 2: Setting τ

5 Numerical Experiments

This section reports the results of the experiments conducted on two data sets for evaluating the performances of wDTW and wTED against other baseline methods.

5.1 Distance/Similarity Measures

We denote the following distance/similarity measures.

- wDTW: Our semantic distance measure explained in Section 4.1.
- wTED: Our semantic distance measure explained in Section 4.2.
- WMD: The Word Mover's Distance introduced in Section 1. WMD adapts the earth mover's distance to the space of documents.
- VSM: The similarity measure introduced in Section 3.1.1.
- PV-DTW: PV-DTW is the same as Algorithm 3 except that the distance between two paragraphs is not based on Algorithm 2 but rather computed as $||PV(p_1) - PV(p_2)||$ where $PV(p)$ is the PV embedding of paragraph p.
- PV-TED: PV-TED is the same as Algorithm 4 except that the distance between two paragraphs is not based on Algorithm 2 but rather computed as $||PV(p_1) - PV(p_2)||$.

Our experiments were conducted on an Apache Spark cluster with 32 cores and 320 GB total memory. We implemented wDTW, wTED, WMD, VSM, PV-DTW and PV-TED in Java Spark. The paragraph vectors for PV-DTW and PV-TED were trained by gensim. [2]

5.2 Data Sets

In this section, we introduce the two data sets we used for our revision detection experiments: Wikipedia revision dumps and a document revision data set generated by a computer simulation. The two data sets differ in that the Wikipedia revision dumps only contain linear revision chains, while the simulated data sets also contains tree-structured revision chains, which can be very common in real-world data.

5.2.1 Wikipedia Revision Dumps

The Wikipedia revision dumps that were previously introduced by Leskovec et al. (2010) contain eight GB (compressed size) revision edits with meta data.

We pre-processed the Wikipedia revision dumps using the JWPL Revision Machine (Ferschke et al., 2011) and produced a data set that contains 62,234 documents with 46,354 revisions. As we noticed that short documents just contributed to noise (wrong edits) in the data, we eliminated documents that have fewer than three paragraphs and fewer than 300 words. We removed empty lines in the documents and trained word2vec embeddings on the entire corpus. We used the documents occurring in the first 80% of the revision period for τ calibration, and the remaining documents for test.

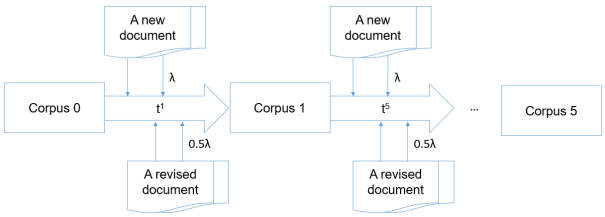


Figure 3: Corpora simulation

5.2.2 Simulated Data Sets

The generation process of the simulated data sets is designed to mimic the real world. Users open some existing documents in a file system, make

[2] https://radimrehurek.com/gensim/models/doc2vec.html

some changes (e.g. *addition*, *deletion* or *replacement*), and save them as separate documents. These documents become revisions of the original documents. We started from an initial corpus that did not have revisions, and kept adding new documents and revising existing documents. Similar to a file system, at any moment new documents could be added and/or some of the current documents could be revised. The revision operations we used were *deletion*, *addition* and *replacement* of words, sentences, paragraphs, section names and document titles. The *addition* of words, ..., section names, and new documents were pulled from the Wikipedia abstracts. This corpus generation process had five time periods $\{t^1, t^2, ..., t^5\}$. Figure 3 illustrates this simulation. We set a Poisson distribution with rate $\lambda = 550$ (the number of documents in the initial corpus) to control the number of new documents added in each time period, and a Poisson distribution with rate 0.5λ to control the number of documents revised in each time period.

We generated four data sets using different random seeds, and repeated the following process. Table 1 summarizes the first data set. We name the initial corpus Corpus 0, and define T_0 as the timestamp when we started this simulation process. We set $T_j = T_{j-1} + t^j$, $j \in [1, 5]$. Corpus j corresponds to documents generated before timestamp T_j. We extracted document revisions from Corpus $k \in [2, 5]$ and compared the revisions generated in (Corpus k - Corpus $(k-1)$) with the ground truths in Table 1. Hence, we ran four experiments on this data set in total. In every experiment, τ^k is calibrated based on Corpus $(k-1)$. For instance, the training set of the first experiment was Corpus 1. We trained τ^1 from Corpus 1. We extracted all revisions in Corpus 2, and compared revisions generated in the test set (Corpus 2 - Corpus 1) with the ground truth: 258 revised documents. The word2vec model shared in the four experiments was trained on Corpus 5.

Table 1: A simulated data set

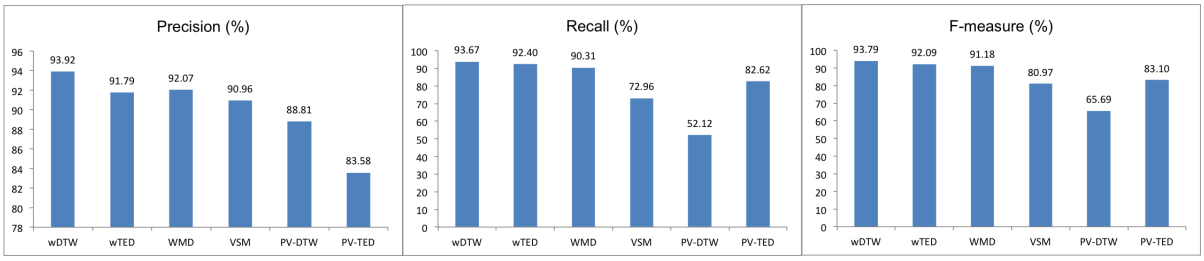
Corpus	Number of documents	Number of new documents	Number of revision pairs
0	550	0	0
1	1347	542	255
2	2125	520	258
3	2912	528	259
4	3777	580	285
5	4582	547	258

5.3 Results

We use precision, recall and F-measure to evaluate the detected revisions. A true positive case is a correctly identified revision. A false positive case is an incorrectly identified revision. A false negative case is a missed revision record.

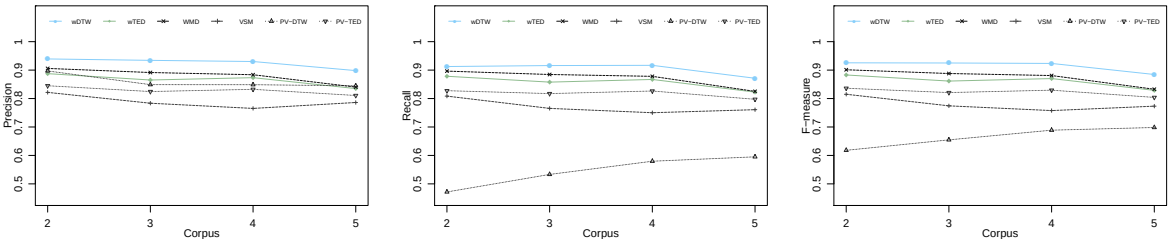
We illustrate the performances of wDTW, wTED, WMD, VSM, PV-DTW and PV-TED on the Wikipedia revision dumps in Figure 4. wDTW and wTED have the highest F-measure scores compared to the rest of four measures, and wDTW also have the highest precision and recall scores. Figure 5 shows the average evaluation results on the simulated data sets. From left to right, the corpus size increases and the revision chains become longer, thus it becomes more challenging to detect document revisions. Overall, wDTW consistently performs the best. WMD is slightly better than wTED. In particular, when the corpus size increases, the performances of WMD, VSM, PV-DTW and PV-TED drop faster than wDTW and wTED. Due to the fact that the revision operations were randomly selected in each corpus, it is possible that there are non-monotone points in the series.

wDTW and wTED perform better than WMD especially when the corpus is large, because they use dynamic programming to find the global opti-



(a) Precision (b) Recall (c) F-measure

Figure 4: Precision, recall and F-measure on the Wikipedia revision dumps



(a) Precision (b) Recall (c) F-measure

Figure 5: Average precision, recall and F-measure on the simulated data sets

Table 2: Running time of VSM, PV-TED, PV-DTW, wTED, wDTW and WMD

	VSM	PV-TED	PV-DTW	wTED	wDTW	WMD
Wikipedia revision dumps	1h 38min	3h 2min	3h 18min	5h 13min	13h 27min	515h 9min
corpus 2	2 min	3 min	3 min	7 min	8 min	8 h 53 min
corpus 3	3 min	4 min	5 min	9 min	11 min	12 h 45 min
corpus 4	4 min	6 min	6 min	11 min	12 min	14 h 34 min
corpus 5	7 min	9 min	9 min	14 min	16 min	17 h 31 min

mal alignment for documents. In contrast, WMD relies on a greedy algorithm that sums up the minimal cost for every word. wDTW and wTED perform better than PV-DTW and PV-TED, which indicates that our DistPara distance in Algorithm 2 is more accurate than the Euclidian distance between paragraph vectors trained by PV.

We show in Table 2 the average running time of the six distance/similarity measures. In all the experiments, VSM is the fastest, wTED is faster than wDTW, and WMD is the slowest. Running WMD is extremely expensive because WMD needs to solve an x^2 sequential transshipment problem for every two documents where x is the average number of words in a document. In contrast, by splitting this heavy computation into several smaller problems (finding the distance between any two paragraphs), which can be run in parallel, wDTW and wTED scale much better.

Combining Figure 4, Figure 5 and Table 2 we conclude that wDTW yields the most accurate results using marginally more time than VSM, PV-TED and PV-DTW, but much less running time than WMD. wTED returns satisfactory results using shorter time than wDTW.

6 Conclusion

This paper has explored how DTW and TED can be extended with word2vec to construct semantic document distance measures: wDTW and wTED. By representing paragraphs with concatenations of word vectors, wDTW and wTED are able to capture the semantics of the words and thus give more accurate distance scores. In order to detect revisions, we have used minimum branching on an appropriately developed network with document distance scores serving as arc weights. We have also assessed the efficiency of the method of retrieving an optimal revision subnetwork by finding the minimum branching.

Furthermore, we have compared wDTW and wTED with several distance measures for revision detection tasks. Our results demonstrate the effectiveness and robustness of wDTW and wTED in the Wikipedia revision dumps and our simulated data sets. In order to reduce the computation time, we have computed document distances at the paragraph level and implemented a boosting learning system using Apache Spark. Although we have demonstrated the superiority of our semantic measures only in the revision detection experiments, wDTW and wTED can also be used as semantic distance measures in many clustering, classification tasks.

Our revision detection system can be enhanced with richer features such as author information and writing styles. Another interesting aspect we would like to explore in the future is reducing the complexities of calculating the distance between two paragraphs.

Acknowledgments

This work was supported in part by Intel Corporation, Semiconductor Research Corporation (SRC).

References

Asad Abdi, Norisma Idris, Rasim M Alguliyev, and Ramiz M Aliguliyev. 2015. Pdlk: Plagiarism detection using linguistic knowledge. *Expert Systems with Applications* 42(22):8936–8946.

Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. 2001. Why and where: A characterization of data provenance. In *Database Theory ICDT 2001*, Springer, pages 316–330.

Erik D Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. 2007. An optimal decomposition algorithm for tree edit distance. In *Automata, Languages and Programming*, Springer, pages 146–157.

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards B* 71(4):233–240.

Oliver Ferschke, Torsten Zesch, and Iryna Gurevych. 2011. Wikipedia revision toolkit: Efficiently accessing wikipedias edit history. In *Proceedings of the ACL-HLT 2011 System Demonstrations*. ACL, pages 97–102.

Nathaniel Gustafson, Maria Soledad Pera, and Yiu-Kai Ng. 2008. Nowhere to hide: Finding plagiarized documents based on sentence similarity. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*. IEEE, pages 690–696. https://doi.org/10.1109/WIIAT.2008.16.

Matthias Hagen, Martin Potthast, and Benno Stein. 2015. Source retrieval for plagiarism detection from large web corpora: recent approaches. *Working Notes Papers of the CLEF* pages 1613–0073.

Timothy C. Hoad and Justin Zobel. 2003. Methods for identifying versioned and plagiarized documents. *Journal of the American Society for Information Science Technology* 54(3):203–215. https://doi.org/10.1002/asi.10170.

Philip N Klein. 1998. Computing the edit-distance between unrooted ordered trees. In *European Symposium on Algorithms*, Springer, pages 91–102.

Matt J Kusner, Yu Sun, Nicholas I Kolkin, and Kilian Q Weinberger. 2015. From word embeddings to document distances. In *Proceedings of the 32nd International Conference on Machine Learning*. pages 957–966.

Quoc V Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053* .

Jure Leskovec, Daniel P Huttenlocher, and Jon M Kleinberg. 2010. Governance in social media: A case study of the wikipedia promotion process. In *International Conference on Weblogs and Social Media*.

Xiaoying Liu, Yiming Zhou, and Ruoshi Zheng. 2007. Sentence similarity based on dynamic time warping. In *International Conference on Semantic Computing*. IEEE, pages 250–256.

Michael Matuschek, Tim Schlüter, and Stefan Conrad. 2008. Measuring text similarity with dynamic time warping. In *Proceedings of the 2008 international symposium on Database engineering & applications*. ACM, pages 263–267.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*. pages 3111–3119.

Meinard Müller. 2007. Dynamic time warping. *Information Retrieval for Music and Motion* pages 69–84.

Gabriel Oberreuter and Juan D VeláSquez. 2013. Text mining applied to plagiarism detection: The use of words for detecting deviations in the writing style. *Expert Systems with Applications* 40(9):3756–3763.

Mateusz Pawlik and Nikolaus Augsten. 2011. Rted: a robust algorithm for the tree edit distance. *Proceedings of the VLDB Endowment* 5(4):334–345.

Mateusz Pawlik and Nikolaus Augsten. 2014. A memory-efficient tree edit distance algorithm. In *Database and Expert Systems Applications*. Springer, pages 196–210.

Mateusz Pawlik and Nikolaus Augsten. 2015. Efficient computation of the tree edit distance. *ACM Transactions on Database Systems* 40(1):3.

Mateusz Pawlik and Nikolaus Augsten. 2016. Tree edit distance: Robust and memory-efficient. *Information Systems* 56:157–173.

Mikalai Sabel. 2007. Structuring wiki revision history. In *Proceedings of the 2007 International Symposium on Wikis*. ACM, pages 125–130.

Yasushi Sakurai, Masatoshi Yoshikawa, and Christos Faloutsos. 2005. Ftw: fast similarity search under the time warping distance. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM, pages 326–337.

Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. 2003. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. ACM, pages 76–85.

Antonio Si, Hong Va Leong, and Rynson WH Lau. 1997. Check: a document plagiarism detection system. In *Proceedings of the 1997 ACM Symposium on Applied Computing*. ACM, pages 70–77.

Grigori Sidorov, Helena Gómez-Adorno, Ilia Markov, David Pinto, and Nahun Loya. 2015. Computing text similarity using tree edit distance. In *Fuzzy Information Processing Society (NAFIPS) held jointly with 2015 5th World Conference on Soft Computing, 2015 Annual Conference of the North American*. IEEE, pages 1–4.

Kuo-Chung Tai. 1979. The tree-to-tree correction problem. *Journal of the ACM* 26(3):422–433.

Jing Zhang and HV Jagadish. 2013. Revision provenance in text documents of asynchronous collaboration. In *2013 IEEE 29th International Conference on Data Engineering*. IEEE, pages 889–900.

Kaizhong Zhang and Dennis Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing* 18(6):1245–1262.