TEXT RELATEDNESS USING WORD AND PHRASE RELATEDNESS

by

Md Rashadul Hasan Rakib

Submitted in partial fulfillment of the
requirements for the degree of
Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
July 2014

© Copyright by Md Rashadul Hasan Rakib, 2014

Table of Contents

List of Tables

List of Figures

Abstract

Text is composed of words and phrases. In bag-of-word model (BoW), phrases in texts are split into words that might lose the inner semantics of the phrases, can give inconsistent relatedness score between two texts. Our objective is to apply phrase relatedness in conjunction with word relatedness on the text relatedness task to improve the result. To measure the phrase relatedness we propose an unsupervised function, f using Sum-Ratio (SR) technique. The experimental result from f exemplifies the improvement over existing phrase relatedness methods on two standard datasets of 216 phrase-pairs. We compare our text relatedness approach (henceforth, $TrWP$) against two unsupervised text relatedness methods which are Latent semantic analysis (LSA) and Google tri-gram based model (henceforth, GTM); and on seven datasets out of eleven, the results from our approach are statistically significant with them at 0.05 level. In addition, those results are comparable to the results of the top ranked supervised text relatedness systems of SemEval-2012 and SemEval-2013.

List of Abbreviations Used

BoW	Bag of Word
BoWP	Bag of Word and Phrase
CDS	Compositional Distributional Semantics
CWO	Common Word Order
GTM	Google Tri-gram Model
LDA	Latent Dirichlet Allocation
LSA	Latent Semantic Analysis
NGD	Normalized Google Distance
NLCS	Normalized Longest Common Subsequence
NLP	Natural Language Processing
PMI	Point wise Mutual Information
SR	Sum-Ratio
STD	Standard Deviation
STS	Semantic Text similarity
SVD	Singular Value Decomposition
UPD	Unsupervised Phrase Detection
VSM	Vector Space Model
WSD	Word Sense Disambiguation

Acknowledgements

Thanks to all the little people who make me look tall.

Chapter 1

Introduction

Generally, a phrase is an ordered sequence of multiple words [60] that all together refer to a particular meaning. Phrases are treated as more informative feature terms [18] than words within documents. Phrase relatedness quantifies how two phrases relate to each other. It plays an important role in different Natural Language Processing (NLP) tasks; for instance, document similarity [1], classification, clustering are performed on the documents composed of phrases. Several document clustering methods [5, 18, 27, 28, 45, 50] used phrase similarity to determine the similarity between documents so as to improve the clustering result. SpamED [46] used the bi-gram and tri-gram phrase similarity between an incoming e-mail message and a previously marked spam, in order to enhance the accuracy of spam detection.

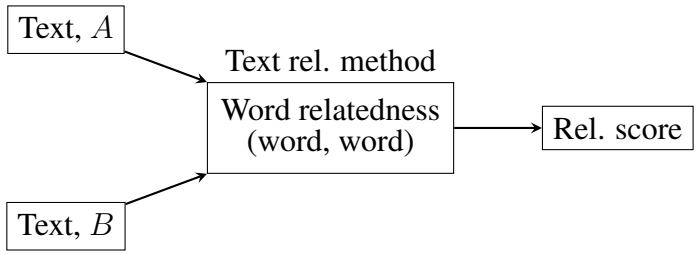
Most works on text relatedness can be abstracted as a function of word relatedness [31]. To the best of our knowledge, there is no explicit evaluation of text relatedness using phrase relatedness. This motivates us to investigate whether using phrase relatedness together with word relatedness on text relatedness task may improve the result. The high level overview of the classical BoW text relatedness approaches (e.g., GTM [36] and LSA [37]) and our proposed approach ($TrWP$) is shown in Figure 1.1.

The classical BoW text relatedness methods split phrases into words; then compute relatedness between two texts by word-pair relatedness. $TrWP$ considers text as Bag-of-Word-and-Phrase (BoWP) comprising both words and phrases. It considers a (word, bi-gram) or (bi-gram, bi-gram) pair as a phrase-pair [2] and computes relatedness between texts using both word and phrase relatedness.

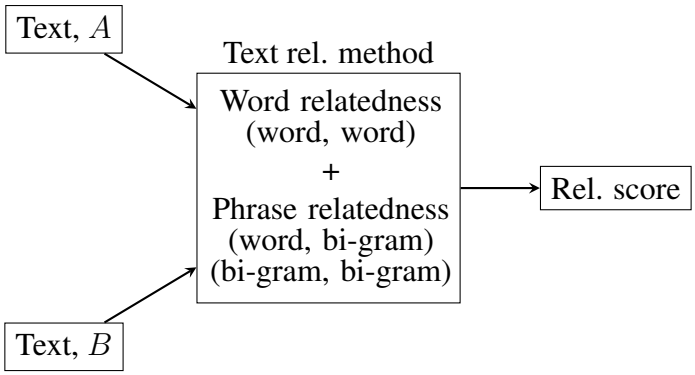
Some methods [30, 42] to compute phrase relatedness adopt compositional distributional semantics (CDS) [7], some use syntactic structure [4] of the phrases in CDS, some

[1] We use 'relatedness' and 'similarity' interchangeably in our thesis, albeit 'similarity' is a special case or a subset of 'relatedness'.

[2] We consider the bi-grams in texts as phrases. A word is also considered as a phrase [60] when relatedness is computed between word and bi-gram.



(a) Classical approach



(b) Proposed approach

Figure 1.1: High level overview of the classical and our proposed text relatedness approach ($TrWP$).

use different tools and knowledge-based resources [29, 53] and some [13, 19, 40, 49, 51, 54] use the co-occurrence statistics (count) from the web corpus. Our method computes relatedness strength between two phrases by applying simple Sum-Ratio (SR) technique on the counts of Google-(n=3,4)-grams [15] that contain two target phrases and overlapping bi-gram contexts. The relatedness strength is normalized by the normalization technique used in the Normalized Google Distance (NGD) [19].

Splitting phrases into words ignores the word order that might change the meaning [55] of phrases leading to inconsistent phrase relatedness score. For example, if we split the phrases "boat house" and "house boat" into words, we get the relatedness score one, nonetheless, as a whole unit, these two phrases do not refer to exactly the same meaning [55]. "boat house" means a house for sheltering boats whereas "house boat" means a boat that serves as a house. To the best of our knowledge, there is no state-of-the-art phrase relatedness measure that considers the whole phrase as a single unit. Therefore, we propose a phrase relatedness function f, that computes relatedness strength between two phrases by considering them as a single unit.

The relatedness measures such as Jaccard [49], Simpson [13], Dice [39, 51], PMI [54] and NGD [19] can be applied on the phrase relatedness task using the web corpus where the phrases are considered as single units. We treat these measures as web-based phrase relatedness methods. Nonetheless these web-based methods are infeasible to use because we need a locally computable and efficient relatedness method that is suitable for using in the inner loop of our text relatedness system, without any limitations in terms of number of queries to a search engine or network communication overhead. In addition, the co-occurrence based Jaccard, Simpson, Dice, PMI and NGD using the Google-n-gram corpus are not useful because the co-occurrences (e.g., "large number and vast amount") of two phrases (e.g., "large number", "vast amount") in the Google-n-gram corpus are too rare to be usable.

The reason for using the Google-n-gram [15] corpus is that it covers all the topics, words and phrases with their statistics (counts) that are used in the real world. Moreover, to generate n-grams (e.g., 3-grams, 4-grams, 5-grams) and their corresponding frequencies from a corpus is computationally expensive.

We summarize our contributions as follows:

- We propose an unsupervised function f to compute relatedness strength between two phrases where each of them may be a single word or a word bi-gram. f uses the Sum-Ratio technique which is very simple and easy to implement. It is independent of the syntactic structure of the phrases. It requires neither any human annotated resource (e.g., WordNet) nor external NLP tools (e.g., lemmatizer, POS tagger).

- We show that using phrase relatedness along with word relatedness improves the text relatedness result.

The rest of the thesis is organized as follows: a brief overview of the related work is presented in Chapter 2. The proposed approach for computing text relatedness is described in Chapter 3. Evaluation and experimental results of phrase and text relatedness are discussed in Chapter 4. We summarize contributions and future related work in Chapter 5.

Chapter 2

Related work

We discuss the related works into two aspects: phrase relatedness and text relatedness.

2.1 Related work on Phrase relatedness

Most of the phrase relatedness tasks [4, 8, 9, 10, 20, 30, 42, 47, 58] use compositional distributional semantic (CDS) models [7] where the phrases are split into words and each word is represented as a vector in Vector Space Model (VSM), constituted by the co-occurrences of the word with different contexts (attributes) [7]. The phrase meaning is obtained by combining the vectors of its individual words using different composition functions (e.g., additive, multiplicative) that produces a resultant vector ($y_1, y_2...y_j...y_m$), depicted in Figure. 2.1. For each word, w_i of phrase P, there is a vector of the co-occurrence counts denoted by $x_{i1}, x_{i2}...x_{ij}...x_{im}$ where $i = 1...n$, $j = 1...m$. n is the number of words of phrase P and m is the maximum number among the number of attributes of each word w_i. The similarity between two phrases is computed by the cosine similarity between their corresponding resultant vectors. The value of each y_j of a resultant vector is calculated by applying a composition function on the corresponding j^{th} values of each x_{ij}.

Consider the phrases "horses run" and "car race"; the statistics of the co-occurred context words (e.g., field, legs) with individual phrase-word (e.g., horses) are shown in Table 2.1 and Table 2.2 respectively. The results after applying different composition functions (e.g., additive, multiplicative) on the vector of each phrase-word are given in the last two rows of each table. If we consider the resultant vectors from additive composition function for these two phrases, the similarity (e.g., cosine similarity) between them is 0.22. Albeit the basic principle of CDS is simple, the word order [55] within the phrases does not persist that may lead to inappropriate similarity score between the phrases.

Mitchell and Lapata [42] represented a two-word phrase by two vectors with attributes selected from the BNC corpus using two different techniques: semantic space [41] and

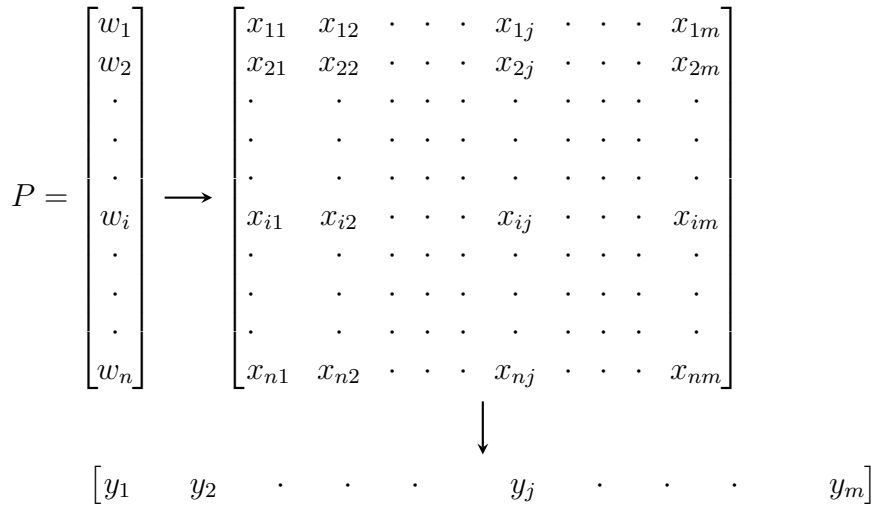


Figure 2.1: Basic overview of the compositional distributional semantics (CDS). $w_1, w_2...w_i...w_n$ are the words of the phrase P. $x_{i1}, x_{i2}...x_{ij}...x_{im}$ are the co-occurrence counts of the context words (attributes) with a phrase-word w_i. $y_1, y_2...y_j...y_m$ is the resultant vector of P.

Individual phrase-words	Context words			
	field	legs	wild	gasoline
horses	3	7	12	0
run	5	4	10	5
↓				
Composition function	Resultant vector			
horses + run	8	11	22	5
horses $\times$ run	15	28	120	0

Table 2.1: The co-occurrence counts of the context words (attributes) with individual phrase-words and the results of different composition functions, for the phrase "horses run".

LDA topic model [11]. Attributes selected by the above methods were optimized by word-based semantic similarity [16] and correlation on WordSim353 dataset [21], respectively. Several composition functions (e.g., additive, multiplicative) were applied on the two vectors for a particular phrase in order to produce a resultant vector; then the similarity between two phrases was computed using the two resultant vectors by widely used cosine measure. Hartung and Frank [30] also represented phrases by vectors to compute similarity between them. They applied Controlled LDA [30] to select the attributes of a word within a phrase.

Individual phrase-words	Context words			
	field	wheel	games	gasoline
car	10	3	15	13
race	9	0	18	4
↓				
Composition function	Resultant vector			
car + race	19	3	33	17
car $\times$ **race**	90	0	270	52

Table 2.2: The co-occurrence counts of the context words (attributes) with individual phrase-words and the results of different composition functions, for the phrase "car race".

Both Mitchell and Lapata [42] as well as Hartung and Frank [30] need to train LDA topic model in order to extract attributes which is more computationally expensive and time consuming than a generic unsupervised approach.

Annesi et al. [4] considered each of the given phrases has same syntactic structure, for instance, first and second word of each phrase are adjective and noun respectively e.g., $phrase1$=$adjective1$-$noun1$, $phrase2$=$adjective2$-$noun2$. The similarity between two phrases is calculated by aggregating the similarity between $adjective1$ and $adjective2$, and the similarity between $noun1$ and $noun2$. Annesi et al. [4] used a POS tagger to acquire the prior knowledge about the word types within the phrases. Since they used the syntactic equivalent word similarity (e.g., adjective with adjective, noun with noun), their method is unable to compute similarity between the phrases having different number of words.

UMBC [29] and OMIOTIS [53] have phrase relatedness web services built on human annotated knowledge base (e.g., WordNet). They use word-pair relatedness from WordNet to compute the phrase-pair relatedness irrespective of the word order within phrases. The UMBC and OMIOTIS do not perform well for the named entities since proper nouns are not well captured in WordNet whereas the proposed phrase relatedness method can compute relatedness scores for the named entities which are in the Google-n-gram corpus. Consider two named entities "Barack Obama" and "George Bush"; the proposed method produces relatedness score 0.65 for this pair, while UMBC gives zero.

The web-based phrase relatedness methods [35]: Jaccard [49], Simpson [13], Dice [39, 51], PMI [54] and NGD [19] use the co-occurrence counts of two phrases $P1$ and $P2$ from the web corpus, defined in Eq. 2.1, Eq. 2.2, Eq. 2.3, Eq. 2.4 and Eq. 2.5 respectively. The co-occurrence is measured based on the number of web documents in which

$P1$ and $P2$ appear together denoted by $D(P_1, P_2)$. $D(P_1, P_2)$ is normalized with respect to the independent occurrences of P_1 and P_2 represented as $D(P_1)$ and $D(P_2)$ respectively. $D(P_1)$ = number of web documents that contain P_1. $D(P_2)$ is computed similarly. We submit the queries: **"P1"**, **"P2"** and **"P1" and "P2"** to the web Search Engine (`https://www.google.com/`) in order to obtain the number of independent occurrences and co-occurrences of the phrases $P1$ and $P2$ respectively. T is the total number of web documents. The value of NGD is unbounded, ranging from 0 to ∞. Therefore, the value is bounded in between 0 and 1 using the variation of NGD defined in [24].

$$Jaccard(P_1, P_2) = \frac{D(P_1, P_2)}{D(P_1) + D(P_2) - D(P_1, P_2)} \tag{2.1}$$

$$Simpson(P_1, P_2) = \frac{D(P_1, P_2)}{min(D(P_1), D(P_2))} \tag{2.2}$$

$$Dice(P_1, P_2) = \frac{2 \times D(P_1, P_2)}{D(P_1) + D(P_2)} \tag{2.3}$$

$$PMI(P_1, P_2) = log_2 \left(\frac{\frac{D(P_1, P_2)}{T}}{\frac{D(P_1)}{T} \ \frac{D(P_2)}{T}} \right) \tag{2.4}$$

$$NGD(P_1, P_2) = e^{-2 \times \frac{max(log\ D(P_1), log\ D(P_2)) - log\ D(P_1, P_2)}{log\ T - min(log\ D(P_1), log\ D(P_2))}} \tag{2.5}$$

A dynamic programming based algorithm [56] measured phrase similarity through integrating both edit distance between the parse trees of the phrases and word-pair similarity. IRIT [17] computed similarity between the phrases using the longest common n-gram based similarity in conjunction with concept similarity [44].

2.2 Related work on Text relatedness

Existing work on determining text relatedness is broadly categorized into three major groups: corpus-based (e.g., [36, 37]), knowledge-based (e.g., [52]) and hybrid measure (e.g., [25]). A detailed survey of text relatedness can be found in [23].

In general, BoW methods fall into corpus-based category. GTM uses the Google tri-gram [15] to compute word-pair relatedness of representative words from each text to ultimately compute text-pair relatedness as described in [36] and available on the GTM Web site (`http://ares.research.cs.dal.ca/gtm/texttextform.html`). To compute word-pair relatedness, the frequencies of the Google tri-grams that start and

end with the given pair and the uni-gram frequencies of the pair are taken into account. The text relatedness method first separates common words between the texts before computing the word relatedness. The count of the separated words and the relatedness scores between the representative words from two texts are then normalized using the texts' lengths [1].

LSA [37] computes the text-pair similarity by using word-pair similarity. The word-pair similarity is computed based on the assumption that words that are similar to each other appear in the similar contexts (e.g., paragraphs, sentences) inside a corpus. At first, the text is represented as a matrix where the rows refer to individual words of the text and the columns stand for the contexts. Each cell contains the frequency of a word within a particular context. Then a matrix decomposition technique called Singular Value Decomposition (SVD) is applied on the large word-by-context matrix to reduce the number of columns, but still high, typically 50-500 dimensional semantic space. The similarity between two words is measured by the cosine similarity between their corresponding row vectors. The semantic space is a mathematical representation of a large corpus of text and every term, text, or novel combination of terms having a high dimensional vector representation. The goal of creating a semantic space for a specific domain is to use a representative set of texts as input so that LSA can make semantic similarity judgements comparable to human [59]. In our experiment section, we have used the general knowledge space (college level) for the LSA semantic space, available on the web site (http://lsa.colorado.edu/).

In [34], a corpus-based semantic text similarity (STS) measure is proposed as a function of string similarity, word similarity, and common-word-order (CWO) similarity. To compute string similarity between two words, they adopt the Normalized Longest Common Subsequence (NLCS) [3] measure. To determine word similarity, they focus on corpus-based measures. CWO similarity [34] is calculated based on the similarity of the orders of common words (e.g., common words appear in the same order, or almost the same order, or very different order) in two texts.

In [52], the text-pair relatedness is computed using both the lexical similarity and semantic relatedness between the words of two texts. For each word-pair, the semantic relatedness is measured using the semantic links between individual words from WordNet. However, there may be some proper nouns within the texts that do not exist in the WordNet. This is because, they also compute the lexical similarity for the word-pairs using the

[1] $length(text)$ = number of words within the text.

estimation of the words importance weights determined by the standard $tf.idf$ weighting scheme [48]. Then, for each word in the first text, the most relevant word from the second text is selected subject to maximizing the product of the lexical similarity and semantic relatedness between them. After that, the sum of the products for the word-pairs is normalized by the number of words in the first text. The same process is followed for each word in the second text. Finally the text-pair relatedness is computed by averaging the two sums.

Another WordNet based text relatedness method [31] follows almost the same approach as in [34]. The principal difference between them is that, [31] adopts knowledge-based (WordNet) measure instead of corpus-based to compute word-pair relatedness. The intuition behind using knowledge-based method is that corpus-based methods are closer to the syntactic representation than to semantic representation since they use the statistical information (e.g., how many times two words co-occur within a corpus). They use WorNet not only to compute word-pair relatedness but also to assign the most suitable sense to a polysemous word through applying Word Sense Disambiguation (WSD). Polysemous word is a word that has multiple meanings. WSD is the task of determining the sense of a polysemous word within a specific context [57]. In order to disambiguate the sense of a word in the first text, they choose a word from the second text having the highest relatedness between them.

In [25], a hybrid measure using three dictionaries (e.g., WordNet, OntoNotes, and Wiktionary) along with the Brown corpus is proposed. The intuition of using multiple dictionaries is that two definitions in different dictionaries referring to the same concept should be assigned large similarity. Another hybrid measure [26] combines the corpus-based and knowledge-based information from the Brown Corpus and WordNet respectively to compute the relatedness between sentences. They tune their model using the training dataset and evaluate the model on the test dataset.

Most methods in SemEval-2012 [1] and SemEval-2013 [2] shared task on semantic textual similarity (STS) are hybrid where different tools and resources have been used. UKP [6], the top-ranked system for SemEval STS 2012 task [1], uses a trained log-linear regression model and combines multiple text similarity methods based on lexical-semantic resources. UMBC [29] is the top-ranked system for SemEval STS 2013 core task [2], uses term alignment algorithm augmented with penalty terms. Term alignment [29] is the mapping of a term of a sentence to a the counterpart of another sentence. The mapping can

be obtained in different ways, for instance exact matching of two terms, finding that one term is the acronym of the other term, and matching two consecutive terms in a sentence with a single term in the other sentence (e.g. “long term” and “long-term”). If two terms are aligned, the term alignment score is one otherwise zero. UMBC also uses a word similarity feature that combines LSA word similarity and WordNet. The tools and resources used by this system are monolingual corpora, WordNet, KB Similarity, lemmatizer, POS tagger, and time and date resolution [2].

The above mentioned text relatedness approaches do not consider the word order of the phrases in texts. That is why the relatedness scores computed by these methods are inconsistent for the text pairs when any of the text within the pair contains phrases.

Chapter 3

Proposed approach for text relatedness

In this chapter we delineate the proposed text relatedness approach. At first we discuss how the word relatedness is computed. Then the terminologies used in measuring phrase relatedness are depicted. After that we outline the phrase detection mechanism. Later on the detailed phrase relatedness method has been described. Finally we combine the word and phrase relatedness to compute the relatedness between two texts.

3.1 Computing word relatedness

In order to compute the relatedness between two words, $w1$ and $w2$, $TrWP$ adopts the word relatedness measure mentioned in GTM [36], a corpus-based approach using Google tri-gram. The reason of adopting this measure is that $TrWP$ needs to compute word relatedness and to the best of our knowledge this is the state-of-the-art unsupervised word relatedness method; moreover both $TrWP$ and GTM use the same Google-n-gram corpus. The steps of computing word relatedness are given in Fig. 3.1. The main idea of the tri-gram relatedness model is to consider all the tri-grams that start and end with the given pair of words and then normalize their mean frequency using uni-gram frequency of each of the words as well as the most frequent uni-gram in the corpus used.

3.1.1 Extracting Google tri-grams

We extract two sets of Google tri-grams beginning and ending with any of the words w_1 and w_2 following the orders (w_1, w_i, w_2) and (w_2, w_i, w_1) respectively where w_i is a co-occurrence context representing an association between them.

3.1.2 Summing the frequencies of Google tri-grams

For each set of extracted Google tri-grams, we sum their frequencies. After that we add the two sums obtained from these two sets, defined in Eq. 3.1 where $M12$ and $M21$ are

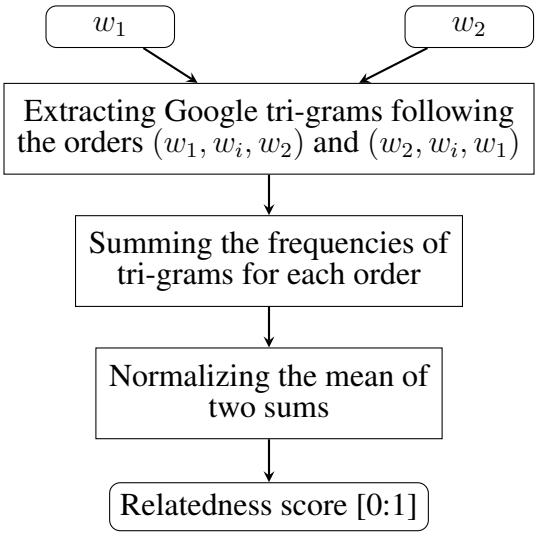


Figure 3.1: Steps of computing word relatedness [36].

the number of Google tri-grams in each set respectively. $C(w_1 w_i w_2)$ is the frequency of a Google tri-gram.

$$u(w_1, w_2) = \sum^{M12} C(w_1 w_i w_2) + \sum^{M21} C(w_2 w_i w_1) \tag{3.1}$$

3.1.3 Normalizing the mean of two sums

The mean of the two sums is computed by dividing $u(w_1, w_2)$ by 2. Then it is normalized with respect to the frequency of the words denoted by $C(w_1)$ and $C(w_2)$ along with the frequency of the most frequent Google uni-gram (C_{max}), designated in Eq. 3.2 [36]. The outcome of this normalization is the relatedness score [0:1] between the words, w_1 and w_2.

$$RT(w_1, w_2) = \begin{cases} \frac{log \frac{(u(w_1,w_2)/2){C_{max}}^2}{C(w_1)C(w_2)min(C(w_1),C(w_2))}}{-2 \times log \frac{min(C(w_1),C(P_2))}{C_{max}}} & \text{if } \frac{(u(w_1,w_2)/2){C_{max}}^2}{C(w_1)C(w_2)min(C(w_1),C(w_2))} > 1 \\ \frac{log 1.01}{-2 \times log \frac{min(C(w_1),C(w_2))}{C_{max}}} & \text{if } \frac{(u(w_1,w_2)/2){C_{max}}^2}{C(w_1)C(w_2)min(C(w_1),C(w_2))} \leq 1 \\ 0 & \text{if } u(w_1, w_2) = 0 \end{cases} \tag{3.2}$$

3.2 Terminology used in phrase relatedness

The terminologies used in measuring phrase relatedness are described below:

Definition 1 (Phrase) *Phrase P is a word n-gram, $(w_1, w_2, ...w_n)$ with a particular meaning.*

Definition 2 (Bi-gram context) *Bi-gram context of a phrase P is a bi-gram (z_1, z_2) such that either of the following augmented n-grams*

$$(w_1, w_2...w_n, z_1, z_2) = (P, z_1, z_2)$$
$$(z_1, w_1, w_2...w_n, z_2) = (z_1, P, z_2)$$
$$(z_1, z_2, w_1, w_2...w_n) = (z_1, z_2, P)$$

exist in the Google-n-gram corpus. Since the Google-n-gram corpus is limited to n=5, that is why we limit the phrases we consider to uni-grams and bi-grams and therefore our augmented n-grams are limited to three to four grams.

The intuition of considering bi-gram context is that if we consider tri-gram context, then the number of contexts in the Google-n-gram corpus becomes small and the uni-gram context does not bear semantics as bi-gram context. Iosif and Potamianos [33] extracted left and right contexts of two words each from the web documents to measure similarity between them. Sample bi-gram contexts for the uni-gram phrase "bachelor" and bi-gram phrase "large number" are extracted by placing them in the left most, middle and right position within the Google tri-grams and Google-4-grams respectively as shown in Table 3.1 Table 3.2.

Phrase position	Uni-gram phrase in Google tri-gram	Bi-gram context
Left most	bachelor **lives alone**	lives alone
Middle	**nice** bachelor **person**	nice..person
Right most	**good looking** bachelor	good looking

Table 3.1: Positions of the uni-gram phrase (bachelor) in Google tri-grams and corresponding bi-gram contexts marked bold.

Phrase position	Bi-gram phrase in Google-4-gram	Bi-gram context
Left most	large number **of files**	of files
Middle	**very** large number **generator**	very..generator
Right most	**multiply a** large number	multiply a

Table 3.2: Positions of the bi-gram phrase (large number) in Google-4-grams and corresponding bi-gram contexts marked bold.

Definition 3 (Overlapping bi-gram context) *Given two phrases P_1 and P_2, we extract two sets of Google-(n=3,4)-grams for them denoted by S_1 and S_2 respectively such that any Google-(n=3,4)-gram in S_1 can be either of the form of (P_1, z_1, z_2), (z_3, P_1, z_4) and (z_5, z_6, P_1) and any Google-(n=3,4)-gram in S_2 can be either of the form of (P_2, z_7, z_8), (z_9, P_2, z_{10}) and (z_{11}, z_{12}, P_2) where each of $\{(z_1, z_2), (z_3, z_4), (z_5, z_6)\}$ and $\{(z_7, z_8), (z_9, z_{10}), (z_{11}, z_{12})\}$ represents a single bi-gram context of P_1 and P_2 respectively which are obtained by placing the phrases P_1 and P_2 in the left most, middle and right most positions correspondingly within a single Google-(n=3,4)-gram.*

Any two bi-gram contexts of P_1 and P_2 are overlapped if either of the following patterns is satisfied with respect to the position of P_1 and P_2 within their corresponding Google-(n=3,4)-grams as defined in the following.

Phrase position	*Google-(n=3,4)-gram containing P_1 and bi-gram context*	*Google-(n=3,4)-gram containing P_2 and bi-gram context*	*Overlapping pattern*
Left most	(P_1, z_1, z_2)	(P_2, z_7, z_8)	$(z_1 = z_7, z_2 = z_8)$
Middle	(z_3, P_1, z_4)	(z_9, P_2, z_{10})	$(z_3 = z_9, z_4 = z_{10})$
Right most	(z_5, z_6, P_1)	(z_{11}, z_{12}, P_2)	$(z_5 = z_{11}, z_6 = z_{12})$

Consider the Google-4-grams “large number of data” and “vast amount of data” where “large number” and “vast amount” are the target phrases and “of data” is the overlapping bi-gram context.

Definition 4 (Non-overlapping bi-gram context) *Any two bi-gram contexts of P_1 and P_2 are non-overlapped if either of the following patterns is satisfied with respect to the position of P_1 and P_2 within their corresponding Google-(n=3,4)-grams as defined in the following.*

Phrase position	*Google-(n=3,4)-gram containing P_1 and bi-gram context*	*Google-(n=3,4)-gram containing P_2 and bi-gram context*	*Non-overlapping pattern*
Left most	(P_1, z_1, z_2)	(P_2, z_7, z_8)	$z_1 \neq z_7$ *or* $z_2 \neq z_8$ *or both*
Middle	(z_3, P_1, z_4)	(z_9, P_2, z_{10})	$z_3 \neq z_9$ *or* $z_4 \neq z_{10}$ *or both*
Right most	(z_5, z_6, P_1)	(z_{11}, z_{12}, P_2)	$z_5 \neq z_{11}$ *or* $z_6 \neq z_{12}$ *or both*

Definition 5 (All bi-gram context) *All Bi-gram context of a phrase includes both the overlapping and non-overlapping bi-gram contexts that are extracted from the Google-(n=3,4)-grams where the target phrase appears.*

Definition 6 (Sum-Ratio (SR)) *Sum-Ratio refers to the product of the sum and ratio between the minimum (min) and maximum (max) of two numbers. Given two numbers a and b, the Sum-Ratio of a and b is defined as follows.*

$$\begin{aligned} Sum(a,b) &= a+b && \textit{Total strength} \\ Ratio(a,b) &= min(a,b)/max(a,b) && \textit{Relative strength} \\ Sum\text{-}Ratio(a,b) &= Sum \times Ratio && \textit{Quantified Total strength} \end{aligned}$$

The Sum-Ratio of two numbers indicates the strength between them. The objective of Sum-Ratio is to maximize the sum of two numbers with respect to their ratio. For instance, the strength between 1 and 1000 is $(1+1000) \times (min(1,1000)/max(1,1000)) = 1.001$ and between 500 and 501 is $(500+501) \times (500/501) = 999.002$. The sum of the first and second two numbers is same (e.g., 1001), but the strengths are different due to having different ratios $(1/1000) = 0.001$ and $(500/501) = 0.998$, respectively.

Definition 7 (Relatedness strength) *Relatedness strength is the strength between two phrases P_1 and P_2 which is computed using the strength (Sum-Ratio value) between the counts of two Google-(n=3,4)-grams that contain P_1 and P_2 respectively and a overlapping bi-gram context.*

Given a pair of phrases P_1 and P_2, any n-gram pair of the form (tri-gram, 4-gram) or (4-gram, 4-gram) that contains P_1 and P_2 and a overlapping bi-gram context, represents an association between P_1 and P_2. In order to measure the degree of association between two phrases, we use the counts of Google-(n=3,4)-grams in which they appear. To measure the strength between two counts, the proposed phrase relatedness method uses simple Sum-Ratio technique. Consider the counts 200 and 300 for two Google-(n=3,4)-grams, "bachelor marry woman" and "unmarried man marry woman" respectively where the target phrases are "bachelor" and 'unmarried man". The SR value is $(200+300) \times (200/300) = 333.33$, representing the relatedness strength between these two Google-(n=3,4)-grams which is used to compute relatedness between the phrases "bachelor" and 'unmarried man".

Definition 8 (Phrase relatedness function (f)) *Phrase relatedness function (f) is a function that takes two phrases P_1 and P_2 as input and computes the relatedness strength between them by applying Sum-Ratio technique on the overlapping bi-gram contexts in conjunction with the cosine similarity computed from all bi-gram contexts.*

3.3 Phrase detection

We elicit the bi-grams from texts as phrases using the frequencies of the Google bi-grams. The elicited bi-grams are not the actual phrases but more likely to be considered as phrases if they are highly frequent, asserted in the Google-Book-Ngram-Viewer (`https://books.google.com/ngrams/info`). A bi-gram contains two words, hence it can be considered as a multi-word term. Frantzi et al. [22] proposed a domain-independent method for the automatic extraction of multi-word terms from machine-readable special language corpora. Bonin et al. [14] presented a novel approach to multi-word terminology extraction combining automatic term recognition approach with contrastive ranking technique.

We adopt a very naive approach to detect the bi-gram phrases from texts using the mean (u_{bg}) and standard deviation (sd_{bg}) of all Google bi-gram frequencies (counts) which are computed once. To detect the bi-gram phrases, the whole text is split by stop-words that produces a list of c-grams [1]. For each c-gram having length [2] greater than or equal to two, the unsupervised phrase detection algorithm (UPD) 1 is called. There are mainly two cases in terms of the length of c-gram, that UPD 1 deals with.

Case 1: If the c-gram is a bi-gram and its frequency is greater than $u_{bg} + sd_{bg}$, then we add it to the list of bi-gram phrases.

Case 2: If the length of c-gram is greater than two, we generate an array of bi-grams from the c-gram and find the most frequent bi-gram (mbg) among them; then we check whether the frequency of the mbg is greater than $u_{bg} + sd_{bg}$. If the condition is true, then we add the mbg to the list of bi-gram phrases and split the c-gram into two parts (e.g., left, right) by mbg. After splitting, the part of the c-gram situated to the left and right of mbg are denoted as lng and rng respectively. Then for each of the lng and rng, we examine the two cases: **Case 1** and **Case 2** recursively.

[1] c-gram: A chunk of uni-grams where no uni-gram is a stop-word.
[2] $length(c\text{-}gram)$ = number of uni-grams within the c-gram.

Algorithm 1 Unsupervised Phrase Detection

Input:
- $c\text{-}gram$ = A chunk of uni-grams where no uni-gram is a stop-word
- u_{bg} = Mean of all Google bi-gram frequencies
- sd_{bg} = Standard deviation of all Google bi-gram frequencies

Output: List of bi-gram phrases

function $UPD(c\text{-}gram)$
  if $length(c\text{-}gram) < 2$ **then** ▷ $length(c\text{-}gram)$ = number of uni-grams
    $Return$
  else if $length(c\text{-}gram) = 2$ **then**
    if $freq(c\text{-}gram) > (u_{bg} + sd_{bg})$ **then** ▷ Frequency of Google-(c=2)-gram
      Include the $c\text{-}gram$ in the list of bi-gram phrases
    end if
  else if $length(c\text{-}gram) > 2$ **then**
    Find mbg = the most frequent bi-gram from $c\text{-}gram$
    if $freq(mbg) > (u_{bg} + sd_{bg})$ **then**
      Include the mbg in the list of bi-gram phrases
      $lng = LeftNG(c\text{-}gram, mbg)$ ▷ lng= Part of $c\text{-}gram$, left to mbg
      $rng = RightNG(c\text{-}gram, mbg)$ ▷ rng= Part of $c\text{-}gram$, right to mbg
      $UPD(lng)$
      $UPD(rng)$
    end if
  end if
end function

Time complexity of the phrase detection algorithm is analyzed in section C.1 of Complexity analysis C.

3.4 Computing Phrase Relatedness

The proposed text relatedness approach uses phrase relatedness to compute the relatedness between two texts. In order to compute phrase relatedness we propose a function f that computes relatedness strength between two phrases P_1 and P_2 using the Google-n-gram corpus [15]. Then the strength is normalized in between 0 and 1 using NGD [19] in conjunction with NGD´ [24]. The steps of computing phrase relatedness are depicted in Fig. 3.2.

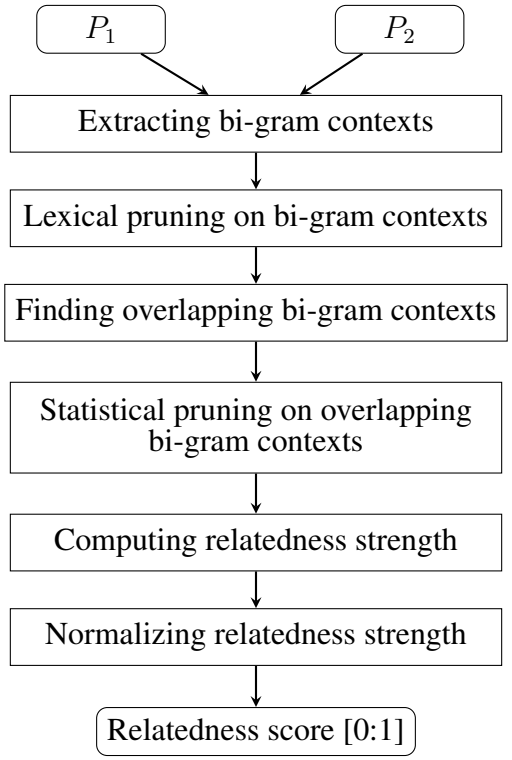


Figure 3.2: Steps of computing phrase relatedness.

3.4.1 Extracting bi-gram contexts

The relatedness between two phrases is computed based on the bi-gram contexts extracted from Google-(n=3,4)-grams. Therefore f needs to extract all possible bi-gram contexts

from Google-(n=3,4)-grams in which the target phrases appear. Since it uses Google tri-grams and 4-grams for uni-gram and bi-gram phrases respectively, it can extract bi-gram contexts in three possible ways as shown in Table 3.1 and Table 3.2. Sample extracted (non) overlapping bi-gram contexts for the phrases P_1="large number" and P_2="vast amount" are given in Table 3.3. Also the frequency (count) of each Google-4-gram and the phrases are shown here.

large number, count=1000		vast amount, count=2000		Bi-gram context	
Google-4-gram	Count	Google-4-gram	Count	Overlap	Non-overlap
large number **of death**	100	vast amount **of death**	200	of death	
consider the large number	300	**consider the** vast amount	400	consider the	
very large number **generator**	150	**very** vast amount **generator**	300	very..generator	
large number **of data**	200	vast amount **of data**	300	of data	
large number **of items**	100	vast amount **of land**	200		of items, of land

Table 3.3: Sample (non) Overlapping Google-4-grams for phrases "large number" and "vast amount".

3.4.2 Lexical pruning on the bi-gram contexts

Some phrases along with their bi-gram contexts do not convey meaningful insight due to the improper positioning of the stop-words within the bi-gram contexts. Therefore we perform lexical pruning [3] based on the position of the stop-words inside the bi-gram contexts. When the target phrase is placed at the left most position within a Google-(n=3,4)-gram followed by a bi-gram context, then the Google-(n=3,4)-gram is pruned if the right most word of that bi-gram context is a stop-word. When the phrase is in the middle surrounding two context words within a Google-(n=3,4)-gram, then the Google-(n=3,4)-gram is pruned if both the surrounding context words are the stop-words. When the phrase is situated at the right most position within a Google-(n=3,4)-gram preceded by a bi-gram context, then the Google-(n=3,4)-gram is pruned if the left most word of the bi-gram context is a stop-word. After performing the lexical pruning, we have a set of non-pruned Google-(n=3,4)-grams containing the bi-gram contexts for a particular phrase. The pruning conditions with examples are given in Table 3.4.

Consider the phrase "large number" and the container Google-4-gram "large number

[3]Performing pruning on the bi-gram contexts implies to the pruning of the Google-(n=3,4)-grams from which those contexts are extracted.

Phrase position	Phrase	Sample bi-gram contexts		Pruning conditions on bi-gram contexts
		Meaningful (Kept)	Meaningless (Pruned)	
Left most	large number	large number **of death**	large number **of the**	Right most uni-gram is a stop-word
Middle	different kind	**facing** different kind **problem**	**in** different kind **to**	Both uni-grams are stop-words
Right most	certain circumstance	**state under** certain circumstance	**of under** certain circumstance	Left most uni-gram is a stop-word

Table 3.4: Conditions for the lexical pruning on bi-gram contexts.

of death" given in Table 3.4. The phrase "large number" and the bi-gram context "of death" all together refer to the amount of death. Hence "of death" is kept. In contrast, the phrase "large number" and the bi-gram context "of the" all together do not express any significant meaning as "large number of death", therefore "of the" has been pruned. The Google-4-gram "facing different kind problem" states with regard to the types of problems that are being faced where "different kind" is a phrase and "facing..problem" is a bi-gram context. The bi-gram context "state under" precedes the phrase "certain circumstance" in the Google-4-gram "state under certain circumstance" represents a situation of the state.

3.4.3 Finding overlapping bi-gram contexts

In this step, we find the overlapping bi-gram contexts between the two sets of non-pruned Google-(n=3,4)-grams extracted for two phrases. The Google-(n=3,4)-grams having overlapping contexts are being separated from the non-overlapping Google-(n=3,4)-grams in order to determine the relatedness strength between them to ultimately compute the relatedness strength between the phrases. The set of overlapping bi-gram contexts for the phrases "large number" and "vast amount" is {"of death", "consider the", "very..generator", "of data"} as shown in Table 3.3.

3.4.4 Statistical pruning on the overlapping bi-gram contexts

Each Google-(n=3,4)-gram pair with overlapping bi-gram context possesses a relatedness strength. We presume that if most of the Google-(n=3,4)-gram pairs have higher relatedness strengths, the relatedness score between two phrases tends to be higher and vice versa.

However some strengths do not lay within the group of maximum strengths [4]. The strengths that are not within the group of maximum strengths are designated as outliers and because of the outliers, the relatedness score between two phrases becomes inconsistent. That is why we apply statistical pruning on the relatedness strengths to prune the Google-(n=3,4)-gram pairs having outliers.

Since our aim is to find the group of maximum strengths and prune the outliers, therefore we adopt the Normal Distribution [12] for statistical pruning. In Normal Distribution, most of the samples tend to be around their center (mean) and from the center they spread out to the left and right directions. In order to characterize the distribution of the samples, Normal Distribution uses two statistical properties which are the mean and standard deviation of those samples. It has been shown that in Normal Distribution most of the samples exist within the mean $\pm$ standard deviation.

In order to perform statistical pruning, we divide each Google-(n=3,4)-gram count (frequency) within a pair by the count of its corresponding (n=1,2)-gram phrase, resulting in a normalized count. A single Google-(n=3,4)-gram pair has two Google-(n=3,4)-grams, so there are two normalized counts for a pair. The minimum and maximum among the two normalized counts are determined. After that we calculate the ratio (e.g., minimum/maximum) between them. Following that for each Google-(n=3,4)-gram pair, we multiply the ratio with the sum of the two Google-(n=3,4)-gram counts that produces a resultant product (e.g., strength). Later on we compute the mean (u_{sr}) and standard deviation (sd_{sr}) from the strengths of the Google-(n=3,4)-gram pairs. If the strength is within the $u_{sr} \pm sd_{sr}$, it has been kept otherwise pruned.

How statistical pruning is performed, is illustrated using the counts of the bi-gram phrases and Google-4-grams from Table 3.3. Consider the Google-4-gram pair ("large number of data", "vast amount of data") where the target phrases are "large number" and "vast amount" respectively. The counts of the phrases "large number" and "vast amount" are 1000 and 2000 correspondingly; and the counts of the 4-grams "large number of data" and "vast amount of data" are 200 and 300 respectively. The normalized counts of the 4-grams "large number of data" and "vast amount of data" are $200/1000 = 0.2$ and $300/2000 = 0.15$ respectively and the ratio between them is $min(0.2, 0.15) \div$

[4]Group of maximum strength: The group in which most of the strengths are close to each other with respect to their standard deviation. Example: Among the strengths {1, 20, 25, 30, 70}, the group of maximum strength is {20, 25, 30} and 1 and 70 are the outliers.

$max(0.2, 0.15) = 0.15/0.2 = 0.75$ which is multiplied with the sum of these 4-gram counts (e.g., $200 + 300 = 500$) produces the resultant product (e.g., strength) $500 \times 0.75 = 375$. The strengths for other Google-(n=3,4)-gram pairs (e.g., 300, 466.67, 450) are calculated similarly. The u_{sr} and sd_{sr} from these strengths are 397.92 and 76.49 respectively. The $u_{sr} + sd_{sr}$ = 474.41 and $u_{sr} - sd_{sr}$ = 321.43 are used to identify the group of maximum strengths; in order to find that the strengths which are in between the $u_{sr} \pm sd_{sr}$ are kept and others are pruned. Therefore the strengths 375, 450 and 466.67 along with their Google-4-gram pairs are kept and the strength 300 along with the 4-gram pair ("large number of death","vast amount of death") is pruned.

3.4.5 Computing relatedness strength

We compute the relatedness strength between two phrases P_1 and P_2 using the overlapping and all bi-gram contexts.

Using overlapping bi-gram contexts

For each non-pruned Google-(n=3,4)-gram pair having overlapping bi-gram context, the relatedness strength is calculated following the Sum-Ratio technique. We sum the two Google-(n=3,4)-gram counts and find the minimum and maximum among them. After that we calculate the ratio (e.g., minimum/maximum) between them. Then the Sum-Ratio value is calculated by multiplying the sum with ratio which signifies the relatedness strength for a Google-(n=3,4)-gram pair. By summing up the relatedness strength of each Google-(n=3,4)-gram pair, we get the relatedness strength between the phrases P_1 and P_2 denoted by $RSOB(P_1, P_2)$ and defined in Eq. 3.3. GP_1 and GP_2 are the Google-(n=3,4)-grams that contain P_1 and P_2, respectively and a overlapping bi-gram context. $C(GP_1)$ and $C(GP_2)$ are the counts of GP_1 and GP_2 respectively. $C(P)$ is the count of phrase P where P is a Google-(n=1,2)-gram. k is the number of non-pruned Google-(n=3,4)-gram pairs.

$$RSOB(P_1, P_2) = \sum^{k} \frac{min(C(GP_1), C(GP_2))}{max(C(GP_1), C(GP_2))} \times sum(C(GP_1), C(GP_2)) \qquad (3.3)$$

In this step, the Google-(n=3,4)-gram count is not normalized with respect to its corresponding (n=1,2)-gram phrase count because the sum of the relatedness strengths from all non-pruned Google-(n=3,4)-gram pairs is normalized with respect to the counts of the phrases in the formula of Normalized Google Distance [19].

Using all bi-gram contexts

All bi-gram contexts of a phrase P_1 include both the non-pruned overlapping and non-overlapping bi-gram contexts that are extracted from the Google-(n=3,4)-grams where P_1 appears. Two vectors V_1 and V_2 in Vector Space Model are constructed for the phrases P_1="large number" and P_2="vast amount", respectively using their corresponding all bi-gram Contexts from Table 3.3, as shown in Table 3.5. The values of V_1 and V_2 are the presence or absence of a bi-gram context belonging to the phrases P_1 and P_2, correspondingly. The relatedness strength between P_1 and P_2 using all bi-gram contexts is designated as $cosSim(V_1, V_2)$, and computed by the cosine similarity between the vectors V_1 and V_2, defined in Eq. 3.4. The similarity score refers to the number of commonalities between the vectors in terms of the bi-gram contexts of P_1 and P_2, respectively.

	consider the	very..generator	of data	of items	of land
V_1	1	1	1	1	0
V_2	1	1	1	0	1

Table 3.5: Vectors V_1 and V_2 from non-pruned (non) overlapping bi-gram contexts of the phrases "large number" and "vast amount", respectively.

$$cosSim(V_1, V_2) = \frac{V_1.V_2}{||V_1||\ ||V_2||} \tag{3.4}$$

Computing overall relatedness strength

The overall relatedness strength $f(P_1, P_2)$ between the phrases P_1 and P_2 is computed by quantifying the relatedness strength obtained from non-pruned overlapping bi-gram contexts, $RSOB(P_1, P_2)$ with respect to the relatedness strength obtained from all bi-gram contexts, $cosSim(V_1, V_2)$, as defined in Eq. 3.5. In order to quantify, $RSOB(P_1, P_2)$ is multiplied with $cosSim(V_1, V_2)$. To illustrate the reason, why we multiply $RSOB(P_1, P_2)$ with $cosSim(V_1, V_2)$, let us consider the Google-4-grams shown in Table 3.3. The $RSOB$ (P_1, P_2) and $cosSim(V_1, V_2)$ between the phrases "large number" and "vast amount" are 1083.33 and 0.75, respectively. After multiplying them by Eq. 3.5, we get the $f(P_1, P_2) = 812.49$ which is less than 1083.33 because $cosSim(V_1, V_2)$ = 0.75. If all the bi-gram contexts of two phrases are overlapped, then we get the maximum value from $f(P_1, P_2)$ since $cosSim(V_1, V_2)$ is 1, and vice versa.

$$f(P_1, P_2) = RSOB(P_1, P_2) \times cosSim(V_1, V_2) \tag{3.5}$$

3.4.6 Normalizing overall relatedness strength

The overall relatedness strength $f(P_1, P_2)$ between the phrases P_1 and P_2 is normalized by the normalization approach used in NGD [19], as defined in Eq. 3.6. N is the total number of web documents used in the Google-n-gram corpus. The value of NGD is unbounded, ranging from 0 to ∞. Therefore $NGDf$ adopts the variation of NGD (e.g., NGD´ [24]) in order to bound the normalized value in between 0 and 1 which is the ultimate relatedness score between the phrases P_1 and P_2.

$$NGDf(P_1, P_2) = e^{-2\times \frac{max(log\ \ C(P_1), log\ \ C(P_2)) - log\ \ f(P_1, P_2)}{log\ \ N - min(log\ \ C(P_1), log\ \ C(P_2))}} \tag{3.6}$$

Time complexity of computing relatedness between two phrases is investigated in section C.3 of Complexity analysis C.

3.5 Computing Text Relatedness

We represent the text as BoWP model that includes both words and phrases. At first, the punctuations are removed from text. The phrases are extracted from text using the phrase detection algorithm described in section 3.3. Other than phrases the rest of the text is split into non stop-words.

The relatedness between two texts is calculated by the phrase-pair relatedness together with word-pair relatedness following the notion of text relatedness in GTM [36]. The difference between GTM and our approach is that we incorporate phrase relatedness to show that the combination of word and phrase relatedness improves the text relatedness result. The word-pair relatedness is computed by the GTM word relatedness module, defined in Eq. 3.2. We consider a (word, bi-gram) or (bi-gram, bi-gram) pair generated from two texts as a phrase-pair and to compute phrase-pair relatedness, we use the Eq. 3.6. The steps of computing text relatedness are given below.

Step 1: We assume that the two texts $A = \{a_1, a_2, a_3, ..., a_p\}$ and $B = \{b_1, b_2, b_3, ..., b_q\}$ have p and q tokens respectively, and $p \leq q$. Otherwise we switch A and B. Each token can be word or bi-gram phrase.

Step 2: We count the number of common tokens (δ) in both A and B where $\delta \leq p$. The common tokens are removed [31, 34, 36] from A and B. So, $A = \{a_1, a_2, a_3, ..., a_{p-\delta}\}$ and $B = \{b_1, b_2, b_3, ..., b_{q-\delta}\}$. If all tokens match e.g., $p - \delta = 0$, go to step **Step 5**. The reason

for removing common tokens is that if they are not removed, they dominate the $SAvg$ score in **Step 4** since the relatedness scores between common tokens are always 1 and as a result other non-common content-bearing tokens become obscured, leading to a inconsistent text relatedness score.

Step 3: We construct a $(p-\delta) \times (q-\delta)$ 'semantic relatedness matrix' (Say, $M = (\alpha_{ij})_{(p-\delta)\times(q-\delta)}$) using the following process. We set $\alpha_{ij} \leftarrow relatedness(a_i, b_j) \times w^2$ where $i = 1...p-\delta$, $j = 1...q-\delta$, $w =$ weighting factor to boost the relatedness score. The value of w is the average number of words within a word or phrase-pair. The reason for boosting is that same relatedness score of a phrase-pair is more weighted than that of a word-pair. If (a_i, b_j) is a word-pair, $relatedness(a_i, b_j) =$ GTM word-pair relatedness; otherwise $relatedness(a_i, b_j) =$ phrase-pair relatedness obtained from Eq. 3.6.

$$M = \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,j} & \cdots & \alpha_{1,q-\delta} \\ \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,j} & \cdots & \alpha_{2,q-\delta} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \alpha_{i,1} & \alpha_{i,2} & \cdots & \alpha_{i,j} & \cdots & \alpha_{i,q-\delta} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \alpha_{p-\delta,1} & \alpha_{p-\delta,2} & \cdots & \alpha_{p-\delta,j} & \cdots & \alpha_{p-\delta,q-\delta} \end{pmatrix}$$

Step 4: For each row we compute the mean (u) and standard deviation (sd) of the relatedness scores and select the scores which are larger than $u + sd$. The idea is to find more related tokens among $(q-\delta)$, for each $(p-\delta)$ tokens. The average of the selected scores is computed for a row and for $(p-\delta)$ rows we get $(p-\delta)$ averages. We sum the $(p-\delta)$ average values denoted by $SAvg$.

Step 5: In order to compute relatedness between the texts A and B, we use the GTM [36] normalization with minor modification, given in Eq. 3.7.

$$rel.(A, B) = \frac{(2|\delta| + SAvg) \times (2|A| + 2|B|)}{2 \cdot 2|A| \cdot 2|B|} \tag{3.7}$$

Number of words in A, B and δ are denoted by $|A|$, $|B|$, $|\delta|$ respectively. Since we multiply the weighting factor w with relatedness score while constructing the semantic relatedness matrix M, hence $|A|$, $|B|$, $|\delta|$ are multiplied by 2. Let us illustrate an example, why we multiply by 2. Consider two texts $A =$"certain circumstance" and $B =$"particular case"

where “certain circumstance” and “particular case” are the phrases. The maximum relatedness score between them can be 1 which is multiplied by the weighting factor w. The average number of words of the phrase-pair, (“certain circumstance”,“particular case”) is 2; so the weighting factor $w = 2$. The maximum value of $SAvg = 1 \cdot 2 \cdot 2 = 4$. If we use the Eq. 3.7, then the maximum relatedness score between the texts, A and B is $\frac{4\times(2\cdot2+2\cdot2)}{2\times2\cdot2\times2\cdot2} = 1$.

Time complexity of computing relatedness between two texts is discussed in section C.4 of Complexity analysis C.

Chapter 4

Evaluation

The proposed phrase and text relatedness approach have been evaluated to determine **i)** how phrase relatedness compares with existing methods and **ii)** whether text relatedness has been improved by incorporating phrase relatedness with word relatedness.

4.1 Evaluation of phrase relatedness

To evaluate the relatedness scores computed by the proposed phrase relatedness method $NGDf$, the adjective-noun (AdjN) and noun-noun (NN) section from Mitchell and Lapata's [42] dataset have been used. Each section contains 108 phrase-pairs. Hartung and Frank [30] and Reddy et al. [47] evaluated their phrase relatedness methods using AdjN and NN phrase-pairs from Mitchell and Lapata's [42] dataset, respectively.

We evaluate the relatedness scores generated by $NGDf$ against the scores obtained from different phrase relatedness methods[1] using Pearson correlation[2] r. $NGDf$ outperforms all the other measures by achieving the highest Pearson's r for combined 216 phrase-pairs, shown in Table 4.1.

To find whether the difference between two correlations is statistically significant, we use the procedure mentioned in [61]. For 108 AdjN phrase-pairs, the correlation differences between $NGDf$ and the rest except $cosSim$ are statistically significant at 0.05 level. For 108 NN phrase-pairs, the correlation differences between $NGDf$ and the rest except Annesi et al. [4] are statistically significant at 0.05 level. For combined 216 phrase-pairs, the correlation differences between $NGDf$ and the rest except Annesi et al. [4] are statistically significant at 0.05 level.

OMIOTIS [53] and UMBC [29] have phrase relatedness services where they split the

[1] Pearson's r is not computed using Mitchell and Lapata's [42] system due to the unavailability of their individual phrase-pair score. Moreover, in an attempt to reproduce Mitchell and Lapata's [42] method, Hartung and Frank [30] get Spearman's $\rho = 0.34$ instead of $\rho = 0.46$ on 108 adjective-noun pairs.

[2] We prefer Pearson's r to Spearman's ρ because Agirre et al. [2] stated that Pearson's r is more informative than Spearman's ρ. Spearman's ρ considers the rank differences while Pearson's r takes into account the value differences. Moreover, SemEval-2012 [1] and SemEval-2013 [2] used Pearson's r for evaluation task.

Method type	Phrase rel. method	AdjN	NN	Combined
Knowledge based	OMIOTIS [53]	0.491^*	0.352^*	0.399^*
	UMBC [29]	0.550^*	0.357^*	0.436^*
Computational Distributional Semantics (CDS) [7]	Hartung and Frank [30]	0.502^*	$\times$	$\times$
Syntactic structure in CDS	Annesi et al. [4]	0.602^*	**0.703**	0.639
Web-based	Jaccard [49]	0.324^*	0.007^*	0.163^*
	Simpson [13]	0.360^*	-0.003^*	0.184^*
	Dice [39, 51]	0.332^*	0.031^*	0.169^*
	PMI [54]	0.223^*	0.252^*	0.186^*
	NGD [19]	0.247^*	0.176^*	0.160^*
Google-n-gram based	cosSim	0.722	0.536^*	0.589^*
	NGDf	**0.743**	0.636	**0.656**

Table 4.1: Pearson's r on 108 AdjN, 108 NN and 216 combined phrase-pairs. For a specific dataset, the highest correlation (r) is marked bold. The correlation (r) of proposed phrase relatedness approach $NGDf$ is statistically significant at 0.05 level with respect to the correlation (r) marked by ($*$).

phrases into words and use word-pair relatedness obtained from WordNet to calculate the phrase-pair relatedness. Due to the lack of entry in WordNet, we get more zero (or near to zero) relatedness scores by OMIOTIS and UMBC than other methods. For example, UMBC gives zero relatedness score for 29 phrase-pairs among 216. This reduces the overall performance of OMIOTIS and UMBC.

Annesi et al. [4] follows the syntactic structure of the phrases; because of this, the correlation r on NN phrase-pairs is higher than that of AdjN pairs. Consider the syntactic structures of two AdjN phrases, *phrase1=adj1-noun1, phrase2=adj2-noun2*. *rel(phrase1, phrase2) = rel(adj1, adj2) ($\times$ or +) rel(noun1, noun2)* where rel refers to the relatedness function. Since they multiply or add two relatedness scores generated from two different types of word-pairs e.g., $(adj1, adj2)$ and $(noun1, noun2)$, therefore the correlation r for AdjN phrase-pairs is less than that of NN pairs. For NN pairs any operation $(\times\ or\ +)$ is performed on the relatedness scores, obtained from the same type of word-pairs.

The web-based methods [35]: Jaccard [49], Simpson [13], Dice [39, 51], PMI [54] and NGD [19] use the co-occurrence counts of two phrases from the web documents. Their results vary due to normalizing the co-occurrence count by different relatedness measures.

We consider $cosSim(V_1, V_2)$ as a baseline method that uses simple Cosine similarity to compute the similarity between two phrases where the vectors V_1 and V_2 are constructed for the phrases P_1 and P_2 respectively, using the bi-gram contexts extracted from Google-(n=3,4)-grams in which the phrases appear. For combined 216 phrase-pairs the correlation r from $cosSim(V_1, V_2)$ is lower than that of proposed method $NGDf$ since $cosSim(V_1, V_2)$ does not take into account the relatedness strengths between the counts of the Google-(n=3,4)-grams having overlapping bi-gram contexts.

4.2 Evaluation of text relatedness

To evaluate the proposed unsupervised text relatedness method $TrWP$, eleven datasets have been used. Five datasets from Semeval-2012 [1]: SMTnews12, SMTeur12, MSRpar12, MSRvid12, OnWN12; Four from Semeval-2013 [2]: FNWN13, OnWN13, HDL13, SMT13; other two from STS131 [43] and ABC1225 [38]. Different properties of these datasets for instance Min/Max (Minimum/Maximum), Avg (Average) and STD (Standard Deviation) of the average number of words of each text pair before and after the stop-word removal are shown in Table 4.2.

Text rel. dataset		#Text pairs	Before stop-word removal			After stop-word removal		
			Min/Max	Avg	STD	Min/Max	Avg	STD
Short text	SMTNews12	399	3/23	11.59	4.28	1/11	5.14	2.44
	SMTeur12	459	1/18	10.39	4.55	1/8	4.08	1.93
	MSRpar12	750	7/31	17.17	4.86	2/19	8.71	2.93
	MSRvid12	750	3/13	6.41	1.53	1/6	3.16	0.77
	OnWN12	750	2/27	7.36	3.19	1/14	3.30	1.60
	FNWN13	189	6/39	20.14	8.03	2/17	8.34	3.31
	OnWN13	561	5/17	6.97	1.82	1/8	2.54	1.25
	HDL13	750	3/20	6.91	1.58	2/9	5.06	1.05
	SMT13	750	1/87	24.51	11.09	1/40	10.9	5.40
	STS131	131	7/27	13.52	3.24	1/11	4.92	1.57
Long text	ABC1225	1225	48/124	80.66	12.34	24/62	41.94	6.54

Table 4.2: Properties of the datasets used.

We categorize these eleven datasets into two major groups which are short text and long text. All the datasets except ABC-1225 are treated as short text since the average of the average number of words of each text pair (before and after the stop-word removal) in those datasets are relatively smaller than that of ABC-1225. The maximum Avg, before and after

the stop-word removal among these ten short text datasets are 24.516 and 10.9 respectively whereas in ABC-1225 the Avg, before and after the stop-word removal are 80.66 and 41.94 respectively.

4.2.1 Comparing $TrWP$ with Unsupervised text relatedness method

We compare the results of the proposed $TrWP$ with the results obtained from classical BoW text relatedness methods (e.g., GTM [36] and LSA [37]). Experimental results show that $TrWP$ performs better than GTM and LSA by achieving higher weighted mean [1] of Pearson r than of GTM and LSA respectively, given in Table 4.3.

Text rel. dataset		Unsupervised Text rel. method		
		TrWP (r)	GTM [36] (r)	LSA [37]) (r)
Short text	SMTNews12	**0.431**	0.406*	0.379
	SMTeur12	**0.518**	0.479*	0.326*
	MSRpar12	**0.457**	0.442	0.178*
	MSRvid12	**0.798**	0.774*	0.557*
	OnWN12	**0.673**	0.628*	0.632*
	FNWN13	0.469	**0.470**	0.348*
	OnWN13	**0.811**	0.798*	0.224*
	HDL13	**0.704**	0.689*	0.482*
	SMT13	0.327	**0.349**	0.303
	STS131	**0.781**	0.761	0.759
Long text	ABC1225	**0.545**	0.469*	0.537
Weighted mean		**0.587**	0.559	0.426

Table 4.3: Pearson's r on the eleven datasets obtained from the unsupervised text relatedness methods. Weighted mean [1] of Pearson's r for these three text relatedness methods is calculated. For a specific dataset, the highest correlation (r) is marked bold. The correlation (r) of proposed text relatedness approach $TrWP$ is statistically significant at 0.05 level with respect to the correlation (r) marked by ($*$).

For significance testing, we use the procedure described in [61]. For the datasets: SMTNews12, SMTeur12, MSRvid12, OnWN12, OnWN13, HDL13 and ABC1225, the correlation differences between $TrWP$ and GTM [36] are statistically significant at 0.05 level. For the datasets: SMTeur12, MSRpar12, MSRvid12, OnWN12, FNWN13, OnWN13 and HDL13, the correlation differences between $TrWP$ and LSA [37] are statistically significant at 0.05 level.

4.2.2 Comparing $TrWP$ with Supervised text relatedness method

We compare the result of $TrWP$ with the result of the top ranked SemEval-2012 [1] and SemEval-2013 [2] text relatedness systems which are UKP [6] and UMBC [29] respectively. The correlation (r) from $TrWP$ and UKP [6] on SemEval-2012 [1] datasets, their weighted means and rankings are shown in Table 4.4. The correlation (r) from $TrWP$ and UMBC [29] on SemEval-2013 [2] datasets, their weighted means and rankings are shown in Table 4.5.

Text rel. dataset		Unsupervised	Supervised
		TrWP (r)	UKP [6] (r)
Short text	SMTNews12	0.431	**0.493**
	SMTeur12	0.518	**0.528**
	MSRpar12	0.457	**0.683**
	MSRvid12	0.798	**0.873**
	OnWN12	**0.673**	0.664
Weighted mean		0.597	**0.677**
Ranking out of 89 systems		24	1

Table 4.4: Pearson's r from $TrWP$ and UKP on SemEval-2012 [1] datasets. Weighted mean [1] of Pearson's r and the rankings of $TrWP$ and UKP are given. For a specific dataset, the highest correlation (r) is marked bold.

Text rel. dataset		Unsupervised	Supervised
		TrWP (r)	UMBC [29] (r)
Short text	FNWN13	0.469	**0.581**
	OnWN13	**0.811**	0.752
	HDL13	0.704	**0.764**
	SMT13	0.327	**0.380**
Weighted mean		0.585	**0.618**
Ranking out of 90 systems		3	1

Table 4.5: Pearson's r from $TrWP$ and UMBC on SemEval-2013 [2] datasets. Weighted mean [1] of Pearson's r and rankings of $TrWP$ and UMBC are given. For a specific dataset, the highest correlation (r) is marked bold.

On the dataset OnWN12 and OnWN13, the correlation (r) of $TrWP$ is higher than that of UKP and UMBC correspondingly. However the weighted mean (r) of $TrWP$ are lower than that of UKP and UMBC respectively. $TrWP$ does not perform better on the SemEval-2012 datasets and its ranking is 24 among 89 text relatedness systems. However on the SemEval-2013 datasets, it stands third among 90 text relatedness systems. $TrWP$ is totally unsupervised and independent of the datasets used. On the other hand, both UKP

and UMBC are supervised and trained on a particular training dataset. UKP combines different types of text relatedness measures (e.g., string-based, WordNet-based) and UMBC uses different tools and resources such as monolingual corpora, WordNet, KB Similarity, lemmatizer, POS tagger, and time and date resolution [2].

4.2.3 Discussion on short text dataset

The datasets from Semeval-2012 [1], Semeval-2013 [2] and STS131 [43] are taken into account as short text datasets. Several properties of these datasets are shown in Table 4.2. Among them STD of a dataset stands for the standard deviation of the average number of words of each text pair. Higher STD of a dataset implies that a significant number of text pairs contain more words and a significant number of text pairs have few words. Text pairs having more words, more likely to contain both words and phrases, therefore the relatedness scores for these text pairs are generated by aggregating both the word and phrase-pair relatedness. In contrast, there is a less possibility of having phrases for the text pairs that consists of few words, hence the relatedness scores for these text pairs are computed using only word-pair relatedness.

Computing text relatedness using both word and phrase-pair relatedness is different from computing text relatedness using only word-pair relatedness because when we combine both word and phrase-pair relatedness, we use the weighting factor (e.g., w) to boost the phrase relatedness score. For a particular dataset, when a significant number of text relatedness scores are computed in two different ways, and combined to find the correlation with ground truth, then the correlation degrades.

After removing the stop-words, the STD of FNWN-13 and SMT-13 are 3.31 and 5.40 respectively which are higher than the STD of other short text datasets. Hence in these two datasets, a significant number of text pairs have both words and phrases, and a significant number of text pairs contain only words. As a result a significant number of text relatedness scores are generated from the proposed text relatedness method $TrWP$, by combining both word and phrase-pair relatedness as well as by word-pair relatedness. Since a significant number of text relatedness scores are computed in two different ways and combined, that is why $TrWP$ does not perform better than GTM on these two datasets.

4.2.4 Discussion on long text dataset

We consider ABC-1225 as long text dataset and discuss the reasons why the proposed text relatedness method $TrWP$ performs better than GTM on this dataset. **i)** Although the STD of ABC-1225 is higher than that of other ten short text datasets, each text pair has a significant number of words which increases the possibility of having more phrases, implying that most of the relatedness scores are generated from one system (e.g., $TrWP$). **ii)** Since long text contains more phrases, so by treating a phrase as a single unit preserves the semantics of the whole text more accurately.

Chapter 5

Conclusion and Future Work

The proposed text relatedness method $TrWP$ using word and phrase relatedness performs better than the classical BoW text relatedness methods GTM and LSA. Moreover the results of $TrWP$ are comparable to the results of the top ranked supervised text relatedness systems of SemEval-2012 and SemEval-2013. To compute phrase relatedness, we propose a function f based on the Sum-Ratio technique along with the statistical pruning. f is unsupervised, does not require any knowledge base and independent of the syntactic structure of the phrases. Unlike other phrase relatedness methods based on word relatedness, f considers the whole phrase as a single unit without losing inner semantic meaning within a phrase.

In the future, we plan to examine f on a new dataset of phrase-pairs of two to three words. We also plan to apply the statistical pruning on the GTM word relatedness module to eliminate the useless Google tri-grams to enhance the word relatedness scores which will be used in the proposed text relatedness method so as to improve the text relatedness result. In addition, we will investigate the text classification and clustering using the proposed text relatedness method.

Bibliography

[1] Eneko Agirre, Johan Bos, Mona Diab, Suresh Manandhar, Yuval Marton, and Deniz Yuret, editors. **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*. Association for Computational Linguistics, Montréal, Canada, 7-8 June 2012.

[2] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. *SEM 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43, Atlanta, Georgia, USA, June 2013. Association for Computational Linguistics.

[3] L Allison and T I Dix. A bit-string longest-common-subsequence algorithm. *Inf. Process. Lett.*, 23(6):305–310, December 1986.

[4] Paolo Annesi, Valerio Storch, and Roberto Basili. Space projections as distributional models for semantic composition. In *Proceedings of the 13th International Conference on Computational Linguistics and Intelligent Text Processing - Volume Part I*, CICLing'12, pages 323–335, Berlin, Heidelberg, 2012. Springer-Verlag.

[5] A.M. Bakr, N.A. Yousri, and M.A. Ismail. Efficient incremental phrase-based document clustering. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 517–520, Nov 2012.

[6] Daniel Bär, Chris Biemann, Iryna Gurevych, and Torsten Zesch. UKP: Computing semantic textual similarity by combining multiple content similarity measures. In *SemEval 2012*, pages 435–440, Montréal, Canada, 7-8 June 2012. Association for Computational Linguistics.

[7] Marco Baroni. Composition in distributional semantics. *Language and Linguistics Compass*, 7(10):511–522, 2013.

[8] Pierpaolo Basile, Annalina Caputo, and Giovanni Semeraro. Encoding syntactic dependencies by vector permutation. In *Proceedings of the GEMS 2011 Workshop on GEometrical Models of Natural Language Semantics*, GEMS '11, pages 43–51, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[9] Pierpaolo Basile, Annalina Caputo, and Giovanni Semeraro. A study on compositional semantics of words in distributional spaces. In *ICSC*, pages 154–161. IEEE Computer Society, 2012.

[10] William Blacoe and Mirella Lapata. A comparison of vector-based representations for semantic composition. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 546–556, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[11] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[12] G. Bohm and G. Zech. *Introduction to statistics and data analysis for physicists*. DESY, 2010.

[13] Danushka Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. A web search engine-based approach to measure semantic similarity between words. *IEEE Trans. on Knowl. and Data Eng.*, 23(7):977–990, July 2011.

[14] Francesca Bonin, Felice Dell'Orletta, Simonetta Montemagni, and Giulia Venturi. A contrastive approach to multi-word extraction from domain-specific corpora. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, may 2010. European Language Resources Association (ELRA).

[15] T. Brants and A. Franz. Web 1T 5-gram corpus version 1.1. *Linguistic Data Consortium*, 2006.

[16] JohnA. Bullinaria and JosephP. Levy. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior Research Methods*, 39(3):510–526, 2007.

[17] Davide Buscaldi, Ronan Tournier, Nathalie Aussenac-Gilles, and Josiane Mothe. Irit: textual similarity combining conceptual similarity with an n-gram comparison method. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics - Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, SemEval '12, pages 552–556, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[18] Hung Chim and Xiaotie Deng. Efficient phrase-based document similarity for clustering. *IEEE Trans. on Knowl. and Data Eng.*, 20(9):1217–1229, September 2008.

[19] Rudi L. Cilibrasi and Paul M. B. Vitanyi. The google similarity distance. *IEEE Trans. on Knowl. and Data Eng.*, 19(3):370–383, March 2007.

[20] Katrin Erk. Vector space models of word meaning and phrase meaning: A survey. *Language and Linguistics Compass*, 6(10):635–653, 2012.

[21] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: the concept revisited. *ACM Trans. Inf. Syst.*, 20(1):116–131, January 2002.

[22] Katerina T. Frantzi, Sophia Ananiadou, and Jun-ichi Tsujii. The c-value/nc-value method of automatic recognition for multi-word terms. In *Proceedings of the Second European Conference on Research and Advanced Technology for Digital Libraries*, ECDL '98, pages 585–604, London, UK, UK, 1998. Springer-Verlag.

[23] Wael H. Gomaa and Aly A. Fahmy. A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):13–18, April 2013.

[24] Jorge Gracia, Raquel Trillo, Mauricio Espinoza, and Eduardo Mena. Querying the web: A multiontology disambiguation method. In *Proceedings of the 6th International Conference on Web Engineering*, ICWE '06, pages 241–248, New York, NY, USA, 2006. ACM.

[25] Weiwei Guo and Mona T. Diab. Modeling sentences in the latent space. In *ACL (1)*, pages 864–872, 2012.

[26] Weiwei Guo and Mona T. Diab. Improving lexical semantics for sentential semantics: Modeling selectional preference and similar words in a latent variable model. In *HLT-NAACL*, pages 739–745. The Association for Computational Linguistics, 2013.

[27] Khaled M. Hammouda and Mohamed S. Kamel. Phrase-based document similarity based on an index graph model. In *ICDM*, pages 203–210. IEEE Computer Society, 2002.

[28] K.M. Hammouda and M.S. Kamel. Efficient phrase-based document indexing for web document clustering. *Knowledge and Data Engineering, IEEE Transactions on*, 16(10):1279–1296, Oct 2004.

[29] Lushan Han, Abhay L. Kashyap, Tim Finin, James Mayfield, and Johnathan Weese. UMBC_EBIQUITY-CORE: Semantic Textual Similarity Systems. In *Proceedings of the Second Joint Conference on Lexical and Computational Semantics*. Association for Computational Linguistics, June 2013.

[30] Matthias Hartung and Anette Frank. Assessing interpretable, attribute-related meaning representations for adjective-noun phrases in a similarity prediction task. In *Proceedings of the GEMS 2011 Workshop on GEometrical Models of Natural Language Semantics*, GEMS '11, pages 52–61, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[31] Chukfong Ho, Masrah Azrifah Azmi Murad, Rabiah Abdul Kadir, and Shyamala C. Doraisamy. Word sense disambiguation-based sentence similarity. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, pages 418–426, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[32] Ellis Horowitz, Sartaj Sahni, and Sanguthevar Rajasckaran. *Computer Algorithms: C++*. W. H. Freeman & Co., New York, NY, USA, 1996.

[33] Elias Iosif and Alexandros Potamianos. Unsupervised semantic similarity computation between terms using web documents. *IEEE Trans. on Knowl. and Data Eng.*, 22(11):1637–1647, November 2010.

[34] Aminul Islam and Diana Inkpen. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Trans. Knowl. Discov. Data*, 2:10:1–10:25, July 2008.

[35] Aminul Islam, Evangelos Milios, and Vlado Kešelj. Comparing word relatedness measures based on google-n-grams. In *COLING (Posters)*, pages 495–506, 2012.

[36] Aminul Islam, Evangelos Milios, and Vlado Kešelj. Text similarity using google tri-grams. In *Proceedings of the 25th Canadian conference on Advances in Artificial Intelligence*, Canadian AI'12, pages 312–317, Berlin, Heidelberg, 2012. Springer-Verlag.

[37] T.K. Landauer, P.W. Foltz, and D. Laham. Introduction to latent semantic analysis. *Discourse Processes*, 25(2-3):259–284, 1998.

[38] Michael D. Lee, Brandon Pincombe, and Matthew Welsh. An empirical evaluation of models of text document similarity. In *Proceedings of the 27th Annual Conference of the Cognitive Science Society (CogSci2005)*, pages 1254–1259. Erlbaum, 2005.

[39] Dekang Lin. Automatic retrieval and clustering of similar words. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 2*, ACL '98, pages 768–774, Stroudsburg, PA, USA, 1998. Association for Computational Linguistics.

[40] Dekang Lin. An information-theoretic definition of similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 296–304, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[41] Kevin Lund and Curt Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208, 1996.

[42] Jeff Mitchell and Mirella Lapata. Composition in distributional models of semantics. *Cognitive Science*, 34(8):1388–1429, 2010.

[43] James O'Shea, Zuhair Bandar, and Keeley Crockett. A new benchmark dataset with production methodology for short text semantic similarity algorithms. *ACM Trans. Speech Lang. Process.*, 10(4):19:1–19:63, January 2014.

[44] Ted Pedersen, Siddharth Patwardhan, and Jason Michelizzi. Wordnet::similarity: measuring the relatedness of concepts. In *Demonstration Papers at HLT-NAACL 2004*, HLT-NAACL–Demonstrations '04, pages 38–41, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[45] Maria Soledad Pera and Yiu-Kai Ng. Utilizing phrase-similarity measures for detecting and clustering informative rss news articles. *Integr. Comput.-Aided Eng.*, 15(4):331–350, December 2008.

[46] Maria Soledad Pera and Yiu-Kai Ng. Spamed: A spam e-mail detection approach based on phrase similarity. *J. Am. Soc. Inf. Sci. Technol.*, 60(2):393–409, February 2009.

[47] Siva Reddy, Ioannis Klapaftis, Diana McCarthy, and Suresh Manandhar. Dynamic and static prototype vectors for semantic composition. In *Proceedings of 5th International Joint Conference on Natural Language Processing (IJCNLP-2011)*, pages 705–713, Chiang Mai, Thailand,[Best Paper Award], November 2011. Asian Federation of Natural Language Processing.

[48] G. Salton, C. Buckley, and C. T. Yu. An evaluation of term dependence models in information retrieval. In *Proceedings of the 5th Annual ACM Conference on Research and Development in Information Retrieval*, SIGIR '82, pages 151–173, New York, NY, USA, 1982. Springer-Verlag New York, Inc.

[49] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[50] Shailendra Kumar Shrivastava, J. L. Rana, and R. C. Jain. Article: Text document clustering based on phrase similarity using affinity propagation. *International Journal of Computer Applications*, 61(18):38–44, January 2013. Published by Foundation of Computer Science, New York, USA.

[51] Frank Smadja, Kathleen R. McKeown, and Vasileios Hatzivassiloglou. Translating collocations for bilingual lexicons: A statistical approach. *Comput. Linguist.*, 22(1):1–38, March 1996.

[52] George Tsatsaronis, Iraklis Varlamis, and Michalis Vazirgiannis. Text relatedness based on a word thesaurus. *J. Artif. Int. Res.*, 37(1):1–40, January 2010.

[53] George Tsatsaronis, Iraklis Varlamis, Michalis Vazirgiannis, and Kjetil Nrvg. Omiotis: A thesaurus-based measure of text relatedness. In Wray L. Buntine, Marko Grobelnik, Dunja Mladenic, and John Shawe-Taylor, editors, *ECML/PKDD (2)*, volume 5782 of *Lecture Notes in Computer Science*, pages 742–745. Springer, 2009.

[54] Peter D. Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In *Proceedings of the 12th European Conference on Machine Learning*, EMCL '01, pages 491–502, London, UK, 2001. Springer-Verlag.

[55] Peter D. Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *J. Artif. Int. Res.*, 37(1):141–188, January 2010.

[56] Manuel Vilares, FranciscoJ. Ribadas, and Jess Vilares. Phrase similarity through the edit distance. In Fernando Galindo, Makoto Takizawa, and Roland Traunmller, editors, *Database and Expert Systems Applications*, volume 3180 of *Lecture Notes in Computer Science*, pages 306–317. Springer Berlin Heidelberg, 2004.

[57] Yao-Feng Wang, Yue-Jie Zhang, Zhi-Ting Xu, and Tao Zhang. Research on dual pattern of unsupervised and supervised word sense disambiguation. In *Machine Learning and Cybernetics, 2006 International Conference on*, pages 2665–2669, Aug 2006.

[58] Justin Washtell. Compositional expectation: A purely distributional model of compositional semantics. In *Proceedings of the Ninth International Conference on Computational Semantics*, IWCS '11, pages 285–294, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[59] MichaelB.W. Wolfe and SusanR. Goldman. Use of latent semantic analysis for predicting psychological phenomena: Two issues and proposed solutions. *Behavior Research Methods, Instruments, & Computers*, 35(1):22–31, 2003.

[60] Oren Zamir and Oren Etzioni. Grouper: A dynamic clustering interface to web search results. In *Proceedings of the Eighth International Conference on World Wide Web*, WWW '99, pages 1361–1374, New York, NY, USA, 1999. Elsevier North-Holland, Inc.

[61] Guang Y. Zou. Toward using confidence intervals to compare correlations. *Psychological methods*, 12(4):399–413, December 2007.

Appendix A

Augmenting Google tri-gram and 4-gram files

To compute phrase relatedness, we extract bi-gram context by placing the phrase at the middle and right most position in a Google-(n=3,4)-gram, in addition to the usual left most position. Therefore the default orders (g_1, g_2, g_3) and (g_1, g_2, g_3, g_4) of tri-gram and 4-gram files respectively are augmented. (g_1, g_2, g_3) is augmented with two extra different orders which are (g_2, g_1, g_3) and (g_3, g_1, g_2) where g_1, g_2 and g_3 are the first, second and third uni-grams of the original tri-gram files respectively. The augmented order of the 4-gram files are (g_2, g_3, g_1, g_4) and (g_3, g_4, g_1, g_2), where g_1, g_2, g_3 and g_4 are the first, second, third and fourth uni-grams of the original 4-gram files respectively. These are done once and it reduces the look up time for contexts in Google-(n=3,4)-gram files.

Appendix B

Splitting the Google-n-gram files

We use the Google tri-gram and 4-gram files to extract the bi-gram contexts for uni-gram and bi-gram phrases respectively. Both the augmented and original files are in the same size, having 209.66 MB in an average. We split the augmented and original Google-(n=1,2,3,4)-gram files into smaller files (e.g., average file size is 38.65 KB) so that we can compute the relatedness between two phrases within reasonable amount of time. However this amount of time is not optimized since the average computation time for each phrase-pair is about one second.

Both the Google-1-gram and 3-gram files are split using the first two to four characters of the first uni-gram in each Google-(n=1,3)-gram. Hence the Google-1-gram as well as 3-gram files can be split into maximum $26^4 = 456976$ and $3 \times 26^4 = 1370928$ files respectively.

The Google-2-gram and 4-gram files are split using the first two characters of each first and second uni-gram of each Google-(n=2,4)-gram. Therefore the Google-2-gram and 4-gram files can be split into maximum $26^{(2+2)} = 456976$ and $3 \times 26^{(2+2)} = 1370928$ files respectively. $3 =$ one original order $+$ two augmented orders of the Google-(n=3,4)-gram files. $4 =$ number of splitting characters. $26 =$ number of English alphabets. How the splitting of Google-(n=1,3)-gram and Google-(n=2,4)-gram are performed, are illustrated in Algorithm 2 and 3 respectively. The basic idea of Algorithm 2 is to generate a file name based on the first two to four characters of the first uni-gram within a Google-(n=1,3)-gram and store the Google-(n=1,3)-gram in that file. Similarly the Algorithm 3 is used to generate a file name based on the first two characters of the first and second uni-gram of a Google-(n=2,4)-gram and store the Google-(n=2,4)-gram in that file.

The time complexities for splitting Google-(n=1,3)-gram and Google-(n=2,4)-gram files are analyzed in section C.5 and C.6 respectively.

Algorithm 2 Splitting Google-(n=1,3)-gram files

Input: List of Google-1-gram files; or List of Google-3-gram files in a particular order (e.g., (g_1, g_2, g_3), (g_2, g_1, g_3) and (g_3, g_1, g_2) where g_1, g_2 and g_3 are the first, second and third uni-grams of the original Google-3-gram files respectively), $min2 = 2$ and $max4 = 4$, minimum and maximum number of splitting characters respectively, $roDir =$ root output directory (e.g., .../1g/ or .../3g/ to store the split Google-1-gram or 3-gram files correspondingly), $SOrder =$ split file name order (e.g., 1 for the Google-1-gram files and 123, 213, 312 for the Google-3-gram files in three different orders)

Output: Maximum 456976 split files

for each file fl in the list of files **do**
  for each line li in fl **do**
    $ug1 =$ First uni-gram of Google-(n=1,3)-gram in li
    $lug1 =$ Number of characters in $ug1$
    if $lug1 \geq min2$ **then**
      $minL =$ Find the minimum between $lug1$ and $max4$
      $osDir$, output sub directory for a (n=1,3)-gram
      $tFile$, temporary file name for a (n=1,3)-gram
      for $i = 1$ to $minL$ **do**
        $osDir = concat(osDir, ug1[i], '/')$ ▷ $concat$ multiple strings
        $tFile = concat(tFile, ug1[i])$
      end for
      $oFile$, Final output file name for a Google-(n=1,3)-gram
      $SOrder = 1$, if fl is a Google-1-gram file
      $SOrder = 123$, if fl is a Google-3-gram file in the order of (g_1, g_2, g_3)
      $SOrder = 213$, if fl is a Google-3-gram file in the order of (g_2, g_1, g_3)
      $SOrder = 312$, if fl is a Google-3-gram file in the order of (g_3, g_1, g_2)
      $oFile = concat(roDir, osDir, tFile, SOrder)$
      Append li in the file $oFile$
    end if
  end for
end for

Algorithm 3 Splitting Google-(n=2,4)-gram files

Input: List of Google-2-gram files; or List of Google-4-gram files in a particular order (e.g., (g_1, g_2, g_3, g_4), (g_2, g_3, g_1, g_4) and (g_3, g_4, g_1, g_2), where g_1, g_2, g_3 and g_4 are the first, second, third and fourth uni-grams of the original Google-4-gram files respectively), $min2 = 2$, minimum number of splitting characters, $roDir =$ root output directory (e.g., .../2g/ or .../4g/ to store the split Google-2-gram or 4-gram files correspondingly), $SOrder =$ split file name order (e.g., 12 for the Google-2-gram files and 1234, 2314, 3412 for the Google-4-gram files in three different orders)

Output: Maximum 456976 split files

for each file fl in the list of files **do**
  for each line li in fl **do**
    $ug1 =$ First uni-gram of Google-(n=2,4)-gram in li
    $ug2 =$ Second uni-gram of Google-(n=2,4)-gram in li
    $lug1 =$ Number of characters in $ug1$
    $lug2 =$ Number of characters in $ug2$
    if $lug1 \geq min2$ and $lug2 \geq min2$ **then**
      $osDir$, output sub directory for a (n=2,4)-gram
      $tFile$, temporary file name for a (n=2,4)-gram
      for $i = 1$ to $min2$ **do**
        $osDir = concat(osDir, ug1[i], '/')$ ▷ $concat$ multiple strings
        $tFile = concat(tFile, ug1[i])$
      end for
      for $i = 1$ to $min2$ **do**
        $osDir = concat(osDir, ug2[i], '/')$
        $tFile = concat(tFile, ug2[i])$
      end for
      $oFile$, Final output file name for a Google-(n=2,4)-gram
      $SOrder = 12$, if fl is a Google-2-gram file
      $SOrder = 1234$, if fl is a Google-4-gram file in the order of (g_1, g_2, g_3, g_4)
      $SOrder = 2314$, if fl is a Google-4-gram file in the order of (g_2, g_3, g_1, g_4)
      $SOrder = 3412$, if fl is a Google-4-gram file in the order of (g_3, g_4, g_1, g_2)
      $oFile = concat(roDir, osDir, tFile, SOrder)$
      Append li in the file $oFile$
    end if
  end for
end for

Appendix C

Complexity analysis

In this section, we analyze the time complexity of different modules used in our system which are phrase detection, computing word and phrase relatedness, and computing text relatedness by combining the word and phrase relatedness. In addition we show the time complexity to split the Google-(n=1,3)-gram and Google-(n=2,4)-gram files respectively.

C.1 Time complexity of phrase detection algorithm

The input of the proposed phrase detection algorithm (UPD) 1 is a n-gram and the output is the list of bi-gram phrases detected from the n-gram.

In order to detect the phrases, UPD finds the most frequent bi-gram (mbg) from the list of bi-grams generated from a n-gram. mbg is used to split the n-gram into left and right parts denoted as lng and rng respectively. Then for each lng and rng, UPD is called recursively.

Hence we can treat UPD as a binary splitting algorithm resembling to the Quick-Sort [32]. For simplicity we consider each mbg as a bi-gram phrase. Lets us consider that the n-gram has n uni-grams and mbg is the most frequent bi-gram phrase among n uni-grams that splits the n uni-grams into two parts v and $n - v$, representing the number of uni-grams to the left and right of mbg respectively. Notice that UPD needs to find mbg within the v and $n - v$ uni-grams. So the time complexity to detect all mbg within a n-gram is given in Eq. C.1. Before splitting n uni-grams we linearly traverse all the bi-grams within the n uni-grams that requires approximately $n\alpha$ time where α is a constant.

$$T(n) = T(v) + T(n - v) + n\alpha \tag{C.1}$$

We assume that in every step of splitting, both left and right part of n-gram contain equal number of uni-grams, so $v = n/2$. After substituting $v = n/2$ in Eq. C.1, we get the

following recurrence.

$$
\begin{aligned}
T(n) &= T(n/2) + T(n - n/2) + n\alpha \\
&= T(n/2) + T(n/2) + n\alpha \\
&= 2T(n/2) + n\alpha \\
&= 2^1 T(n/2^1) + n\alpha && \text{After 1st level of split} \\
&= 2[T(n/4) + T(n/2 - n/4) + (n/2)\alpha] + n\alpha \\
&= 2[2T(n/4) + (n/2)\alpha] + n\alpha \\
&= 4T(n/4) + 2n\alpha \\
&= 2^2 T(n/2^2) + 2n\alpha && \text{After 2nd level of split} \\
&= 4[T(n/8) + T(n/8) + (n/4)\alpha] + 2n\alpha \\
&= 4[2T(n/8) + (n/4)\alpha] + 2n\alpha \\
&= 8T(n/8) + 3n\alpha \\
&= 2^3 T(n/2^3) + 3n\alpha && \text{After 3rd level of split} \\
&= ..... \\
&= 2^v T(n/2^v) + vn\alpha && \text{Continuing likewise till } v^{th} \text{ level of split}
\end{aligned}
\tag{C.2}
$$

Notice that this recurrence continues until $v = log(n)$ (e.g., $n = 2^v$). Thus, by putting $v = log(n)$ and $n = 2^v$ in Eq. C.2, the time complexity to detect all bi-gram phrases from n-gram is shown in Eq. C.3.

$$
\begin{aligned}
T(n) &= nT(1) + nlog(n) + log(n)n\alpha \\
&\approx O(nlog(n))
\end{aligned}
\tag{C.3}
$$

C.2 Time complexity of computing word relatedness

In order to compute the word relatedness, we extract the count of each word (e.g., w_1, w_2) from the Google uni-gram corpus that contains $N = 10^7$ uni-grams. To find the co-occurrences of two words in two different orders (e.g., $(w_1..w_2)$, $(w_2..w_1)$), we scan through two Google tri-gram files where each of them consists of $N = 10^7$ tri-grams. So the total time complexity to computing word relatedness is $2 \times O(N) + 2 \times O(N) \approx O(N)$.

C.3 Time complexity of computing phrase relatedness

The time complexity to compute relatedness between two phrases P_1 and P_2 is calculated by aggregating the time complexity of each step as shown in Table C.1.

No.	Steps of computing phrase relatedness	Time complexity
1	Extracting bi-gram contexts	$C1 = O(S) + O(S)$
2	Lexical pruning on bi-gram contexts	$C2 = O(S) + O(S)$
3	Finding overlapping bi-gram contexts	$C3 = O(S \times S)$
4	Statistical pruning on overlapping bi-gram contexts	$C4 = O(S) + O(S)$
5	Computing relatedness strength	$C5 = O(S) + O(S \times S)$
6	Extracting the phrase count	$C6 = O(S) + O(S)$
7	Normalizing the relatedness strength	β
Total time complexity		$= C1 + C2 + C3 + C4 + C5 + C6 + \beta$ $\approx O(S \times S) = O(S^2)$

Table C.1: Time complexity of each step in phrase relatedness computation. Steps are numbered at the left column. S is the average number of Google-n-grams in a file after splitting the original Google-n-gram files. $O(S)$ = Time complexity to scan through a file. β is a constant. Time complexity of each step is denoted by $C1, C2, C3, C4, C5$ and $C6$ respectively.

S is the average number of Google-n-grams in a file after splitting, which is equal to $184.34 \approx 185$ and $O(S)$ is the time complexity to scan through that file. For simplicity, we assume that we extract two sets of bi-gram contexts for the phrases P_1 and P_2 from two files. So the time complexity in step 1, is the sum of time complexity for extracting bi-gram contexts from each file, denoted as $C1 = O(S) + O(S)$.

In step 2, lexical pruning is performed on the same set of bi-gram contexts; therefore time complexity of step 2 is same as step 1.

In step 3, we find the overlapping bi-gram contexts by comparing each bi-gram context of a particular set with the bi-gram contexts of another set. Hence, the time complexity in step 3 is quadratic designated as $C3 = O(S \times S)$.

Once we have the set of overlapping bi-gram contexts, we perform statistical pruning in step 4 by following two sub-steps. At first, we calculate the mean and standard deviation by scanning through the counts of the overlapping contexts. After that we prune the overlapping bi-gram contexts through comparing their counts with the mean and standard deviation. The time complexity of each sub-step is linear, so the time complexity in step 4 is $C4 = O(S) + O(S)$.

In step 5 the relatedness strength between P_1 and P_2 is computed from the relatedness strengths of the overlapping bi-gram contexts, that requires to scan through the bi-gram contexts only once referring to the linear time complexity. Moreover we calculate the cosine similarity by matching each bi-gram context extracted for P_1 with the bi-gram contexts extracted for P_2. So, the complexity for computing cosine similarity is quadratic (e.g., $O(S \times S)$). The total time complexity in step 5 is $C5 = O(S) + O(S \times S)$.

In step 6, individual phrase count is obtained by looking up the bi-gram counts from a file, that entails the time complexity $O(S)$. For two phrases, the total time complexity in this step is $C6 = O(S) + O(S)$.

The final step 7 requires a constant time β to normalize the relatedness strength using the formula of NGD [19].

By summing up the time complexities of these seven steps, we get a polynomial function of degree 2 as shown in Table C.1. Therefore the time complexity for computing phrase relatedness is $O(S^2)$.

C.4 Time complexity of computing text relatedness

Time complexity to compute relatedness between two texts A and B is the aggregation of the time complexity for detecting phrases and the time complexity for combining word and phrase relatedness. For the sake of simplicity, we ignore the time to pre-process the texts (e.g., removing stop-words and punctuation). At first the n-gram tokens are elicited from the texts. A n-gram token is a sequence of uni-grams (words). The minimum length of a token is one and maximum length can be the length of the text. The length of the n-gram token and text is measured based on the number of uni-grams within them. The maximum length of a text that we find among the text-relatedness datasets is 127.

After pre-processing we get two sets of n-gram tokens for the texts A and B. The number of n-gram tokens in text A and B are denoted as t_a and t_b correspondingly. For each n-gram token of text A, the phrase detection algorithm is called. So the time complexity to extract all bi-gram phrases from text A, is $t_a \times nlog(n)$ where n is the number of uni-grams within a n-gram token and $nlog(n)$ is the time complexity to extract all bi-gram phrases from it. Similarly, the time complexity to find all bi-gram phrases from text B is $t_b \times nlog(n)$.

Now we construct a 'semantic relatedness matrix', M having t_a rows and t_b columns.

Each cell of the matrix contains the relatedness score between a word or bi-gram phrase of text A and a word or bi-gram phrase of text B. So each matrix cell contains the relatedness score of a (word, word) or (word, bi-gram) or (bi-gram, bi-gram) pair. We take into account the (word, bi-gram) and (bi-gram, bi-gram) as a phrase-pair. Therefore a matrix cell contains the relatedness score either for a word-pair or for a phrase-pair. This is because the time complexity to compute relatedness score inside a matrix cell is the sum of time complexity for computing both word and phrase relatedness (e.g., $O(N) + O(S^2)$). There are $t_a \times t_b$ cells in M. So the total time complexity to construct the matrix M is $t_a \times t_b \times (O(N) + O(S^2))$.

The texts A and B may have some common words or bi-gram phrases and the relatedness scores between them are always one. The time complexity to compute relatedness between the common words or bi-gram phrases of A and B is $O(min(t_a, t_b))$ where $min(t_a, t_b)$ is the maximum common words or bi-gram phrases among two texts. Finally we aggregate the relatedness score obtained from the matrix M and from common words or phrases. Then normalize the aggregated score that entails a constant time γ.

By summing up the above time complexities, we get the time complexity to compute relatedness between the texts A and B, which is $(t_a \times nlog(n) + t_b \times nlog(n))$ + $t_a \times t_b \times (O(N) + O(S^2))$ + $O(min(t_a, t_b))$ + γ. The number of n-gram tokens t_a and t_b are small, and in fact the maximum value of t_a and t_b can be 127. Ignoring t_a, t_b and the constant γ the total complexity to compute relatedness between the texts A and B is $2 \times nlog(n) + O(N) + O(S^2) \approx nlog(n) + O(N) + O(S^2)$.

C.5 Time complexity of splitting Google-(n=1,3)-gram files

We split each original Google-(n=1,3)-gram files into smaller files. There is one original Google-1-gram file and 97 original Google-3-gram files. We augment the Google-3-gram files into two additional orders that produces extra $97 \times 2 = 194$ Google-3-gram files. So, we have total $1 + 97 + 194 = 292$ files.

In each file there are $N = 10^7$ Google-(n=1,3)-grams where each of them is placed in a single line. The time complexity to process $N = 10^7$ Google-(n=1,3)-grams is $O(N)$.

For each Google-(n=1,3)-gram we extract maximum first 4 characters from the first uni-gram to create a name for the smaller file. Therefore the time complexity to process a single Google-(n=1,3)-gram is $(4 + \lambda)$ where λ is a constant time for other computations

(e.g., string concatenation).

Therefore the total time complexity of splitting 292 Google-(n=1,3)-gram files is $292 \times O(N) \times (4 + \lambda) \approx O(N)$.

C.6 Time complexity of splitting Google-(n=2,4)-gram files

Computing time complexity for splitting Google-(n=2,4)-gram files is same as computing time complexity for splitting Google-(n=1,3)-gram files. There are total 424 files including both original and augmented ones. In each file there are $N = 10^7$ Google-(n=2,4)-grams. For each Google-(n=2,4)-gram we extract first two characters from the first two uni-grams to create a name for the smaller file.

Therefore the total time complexity of splitting 424 Google-(n=2,4)-gram files is $424 \times O(N) \times (4 + \lambda) \approx O(N)$.