Semantic similarity between words and sentences using lexical database and word embeddings

by

Atish Shivaji Pawar
Lakehead University

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

Lakehead University

Semantic similarity between words and sentences using lexical database and word embeddings

by

Atish Shivaji Pawar
Lakehead University

Supervisory Committee

Dr. Vijay Mago, Supervisor
(Department of Computer Science)

Dr. Salimur Choudhury, Departmental Member
(Department of Computer Science)

Dr. Rachid Benlamri, Outside Member
(Department of Software Engineering)

ABSTRACT

Calculating the semantic similarity between sentences is a long-standing problem in the area of natural language processing. The semantic analysis field has a crucial role to play in the research related to the text analytics. The meaning of the word in general English language differs as the context changes. Hence, the semantic similarity varies significantly as the domain of operation differs. For this reason, it is crucial to consider the appropriate definition of the words when they are compared semantically.

We present an unsupervised method that can be applied across multiple domains by incorporating corpora based statistics into a standardized semantic similarity algorithm. To calculate the semantic similarity between words and sentences, the proposed method follows an edge-based approach using a lexical database. When tested on both benchmark standards and mean human similarity dataset, the methodology achieves a high correlation value for both word (Pearsons Correlation Coefficient = 0.8753) and sentence similarity (PCC = 0.8793) while comparing Rubenstein and Goodenough standard; and the SICK dataset (PCC = 0.8324) outperforming other unsupervised models.

We use the semantic similarity algorithm and extend it to compare the *Learning Objectives* from course outlines. The course description provided by instructors is an essential piece of information as it defines what is expected from the instructor and what he/she is going to deliver during a particular course. One of the key components of a course description is the Learning Objectives section. The contents of this section are used by program managers who are tasked to compare and match two different courses during the development of *Transfer Agreements* between various institutions. This research introduces the development of semantic similarity algorithms to calculate the similarity between two learning objectives of the same domain. We present a methodology which deals with the semantic similarity by using a previously established algorithm and integrating it with the domain corpus to utilize domain statistics. The *disambiguated* domain serves as a supervised learning data for the algorithm. We also introduce Bloom Index to calculate the similarity between action verbs in the Learning Objectives referring to the Bloom's taxonomy.

We also study and present the approach to calculate the semantic similarity between words under the word2vec model for a specific domain. We present a methodology to compile a corpus for a specific domain using Wikipedia. We then present a case to show the variance in the semantic similarity between words using different corpora. The core contributions of this thesis are a semantic similarity algorithm for words and sentences, and the corpus compilation of a specific domain to train the word2vec model. We also provide the practical uses of algorithms and the implementation.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

Foremost, I offer my sincere gratitude to my advisor Dr. Vijay Mago who gave me this excellent opportunity and supported me throughout my thesis with his patience and knowledge whilst allowing me the room to work in my own way. My sincere thanks to Dr. Salimur Choudhary for the all the encouragement and suggestions during the study. I also thank Dr. Rachid Benlamri for reviewing the thesis and for the valuable suggestions. I would also like to thank Andrew Heppner for the crucial edits and my fellow labmates for all the stimulating conversations, suggestions and all the fun. I would like to acknowledge the financial support from the Ontario Council on Articulation and Transfer. Last but not the least, I would like to thank my family and every other person who has helped me directly or indirectly throughout the journey.

Because we don't know when we will die, we get to think of life as an inexhaustible well. Yet everything happens only a certain number of times, and a very small number really. How many more times will you remember a certain afternoon of your childhood, an afternoon that is so deeply a part of your being that you can't even conceive of your life without it? Perhaps four, five times more, perhaps not even that. How many more times will you watch the full moon rise? Perhaps 20. And yet it all seems limitless.

Paul Bowles

DEDICATION

To my Father, Mother, and Brother, for their unconditional support

Morning Glory

Walking across a moor, looking at the bright sky
With a glimpse of birds flying by
I remembered the old times with a sigh
It was the morning glory of goodbye

Everything was at its place
With a glazing grace
Sun was rising with snails pace
Morning rays made my memories erase

The tap of my cat brought me back
Well, it gave me a little heart attack
I told her to go sit on a tack
But she is one tough nut to crack

The cold wind whispered in my ear
With such breeze, there's no fear
The gaiety was just sheer
There was no place for a tear

Some days feel like nothing is meant to be
But morning glory is the real glee
And there was me
Dear me

- Atish

Structure of thesis

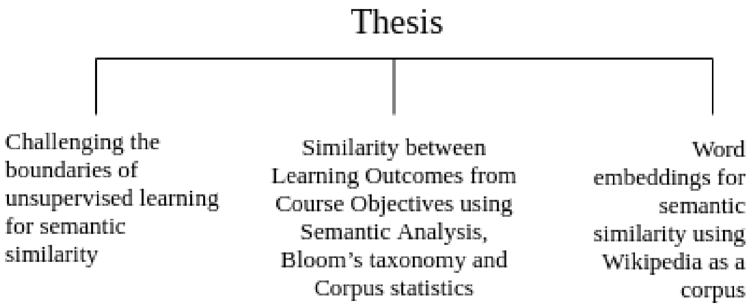

Thesis
Challenging the boundaries of unsupervised learning for semantic similarity
Similarity between Learning Outcomes from Course Objectives using Semantic Analysis, Bloom's taxonomy and Corpus statistics
Word embeddings for semantic similarity using Wikipedia as a corpus

Figure 1: Structure of thesis

The thesis is essentially divided into three segments and each segment constitutes an article. Each article is further divided into various sections.

Chapter 1

Challenging the boundaries of unsupervised learning for semantic similarity

1.1 Introduction

In general, semantic similarity is a measure of conceptual distance between two objects, based on the correspondence of their meanings [37]. Semantic similarity between sentences in natural language processing(NLP) is considered a complex task as the meaning of words changes significantly when the context is changed. As Jiang quotes, *"In many cases, humans have little difficulty in determining the intended meaning of an ambiguous word, while it is extremely difficult to replicate this process computationally"* [28]. Determination of semantic similarity in NLP has a wide range of applications. In internet-related applications, the uses of semantic similarity include estimating relatedness between search engine queries [17] and generating keywords for advertising on the web [3]. In biomedical applications, semantic similarity has become a valuable tool for analyzing the results in gene clustering, gene expression and disease gene prioritization [56] [38] [55]. In addition to this, semantic similarity is also beneficial in information retrieval on web [63], text summarization [12] and text categorization [31]. Hence, such applications need to have a robust algorithm to estimate the semantic similarity which can be used across variety of domains.

Methodologies used to calculate semantic similarity are highly varied across multiple domains and the databases and algorithms used in one specific domain do not translate well onto other domains. Since the concept of calculating semantic similarities has a common underlying conceptual foundation regardless of domain, a methodology with a robust algorithm that can accurately estimate semantic similarity while incorporating a variety of domain specific predefined standard language measures is desirable. To improve the existing algorithms that determine the closeness of implications of the objects under comparison, it is clear that a domain specific predefined standard measure which readily describes the relatedness of the meanings in context is necessary. If we use natural language to compare the natural language sentences, then it would be a recursive problem with no stopping condition. Hence, it is essential to have some predefined measures.

This research aims to improve on existing algorithms and increase robustness through the integration of interchangeable domain specific corpora and through the use of

lexical databases. Lexical databases have fixed vocabulary structures and edge-based word structure that supports the determination of semantic similarity [13]. Many approaches utilizing lexical databases have been developed and proven to be very useful in the area of semantic analysis [6][58][50][36][38][28].
The main contribution of this research is a robust unsupervised semantic similarity algorithm which requires low computational resources and outperforms existing algorithms relative to the Rubenstein and Goodenough(R&G) benchmark standard [59] and achieves a good correlation with respect to the SICK dataset [45].

The following section of this chapter contains a review of related works. Section 1.3 provides a systematic review of our methodology. Section 1.4 explains the idea of traversal in a lexical database along with detailed visual diagrams and the computation with an illustrative example. Section 1.5 contains the result of our algorithm for the 65 noun word pairs from R&G [59] and sentence similarity for the sentence pairs in pilot data set [53] and and sentence similarity for the sentence pairs in SICK dataset [45]. Section 1.6 discusses the results and performance of the algorithm in relation to previous methodologies. Finally, Section 1.7 briefly outlines the outcomes of this research with conclusions.

1.2 Related Work

Recent work in the area of natural language processing has contributed valuable solutions to calculate the semantic similarity between words and sentences. This section reviews some related work to investigate the strengths and limitations of previous methods and to identify the particular difficulties in computing semantic similarity. Related works can roughly be classified into following major categories:

- Word co-occurrence methods
- Similarity based on a lexical database
- Methods based on web search engine results
- Methods based on word vectors using recursive neural networks and deep neural networks

Word co-occurrence methods are commonly used in Information Retrieval (IR) systems [47]. This method has a word list of meaningful words and every query is considered as a document. A vector is formed for the query and for documents. The relevant documents are retrieved based on the similarity between query vector and document vector [9]. This method has obvious drawbacks such as:

- It ignores the word order of the sentence.
- It does not take into account the meaning of the word in the context of the sentence.

But it has following advantages:

- It matches documents regardless the size of documents
- It successfully extracts keywords from documents [46]

Using the lexical database methodology, similarity is computed using a predefined word hierarchy which has words, meanings, and relationships with other words and are stored in a tree-like structure [36]. While comparing two words, it takes into account the path distance between the words as well as the depth of the *subsumer* in the hierarchy. The subsumer refers to the relative root node concerning the two words being compared. It also uses a word corpus to calculate the 'information content'of the word which influences the final similarity. This methodology has the following limitations:

- The appropriate meaning of the word is not considered while calculating the similarity, rather it takes the best matching pair even if the meaning of the word is totally different in two distinct sentences.
- The information content of a word from a corpus, differs from corpus to corpus. Hence, final result differs for every corpus.

The third methodology computes *relatedness* based on web search engine results utilizing the number of search results [10]. This technique does not necessarily give the similarity between words as words with opposite meanings frequently occur together on the web pages which influences the final similarity index. After implementing the method to calculate the Google Similarity Distance [1], we found that the results are not encouraging. The search engines used to calculate the Google Similarity Distance

[1] Interested readers can contact me (apawar1@lakeheadu.ca) for code and results

are Google and Bing.

Recently, the models based on neural networks have produced significant improvements in the results related to semantic similarity [23][51][61][34][9]. One revolutionary model proposed by Tai, Socher, and Manning (2015) [61] uses Glove vectors and subsequently Tree-LSTM. Tree-LSTMs generalize the order-sensitive chain-structure of standard LSTMs to tree-structured network topologies. A *siamese adaptation* of LSTM proposed by Mueller(2016) [51] outperforms the state of the art models. The authors explain the dependency of their model on a simple Manhattan matric. Their method forms a highly structured space whose geometry reflects complex semantic relationships. Performance evaluations for all aforementioned neural network models are trained on SICK dataset and tested on the same dataset. Despite improvements, these models perform poorly when tested on sentences which do not follow the grammar and structure of SICK sentences.

Overall, above-mentioned methods compute the semantic similarity without considering the context of the word according to the sentence. The algorithm proposed in this thesis addresses aforementioned issues by disambiguating the words in sentences and forming semantic vectors dynamically for comparing sentences and words.

1.3 Methodology

The method to calculate the semantic similarity between two sentences is divided into two modules:
Pass 1: Maximize the similarity
Pass 2: Bound the similarity

1.3.1 Pass 1: Maximize the similarity

The proposed methodology considers the text as a sequence of words and deals with all the words in sentences separately according to their semantic and syntactic structure. The information content of the word is related to the frequency of the meaning of the word in a lexical database or a corpus. Figure 1.1 depicts the procedure to calculate the similarity between two sentences. Unlike other existing methods that

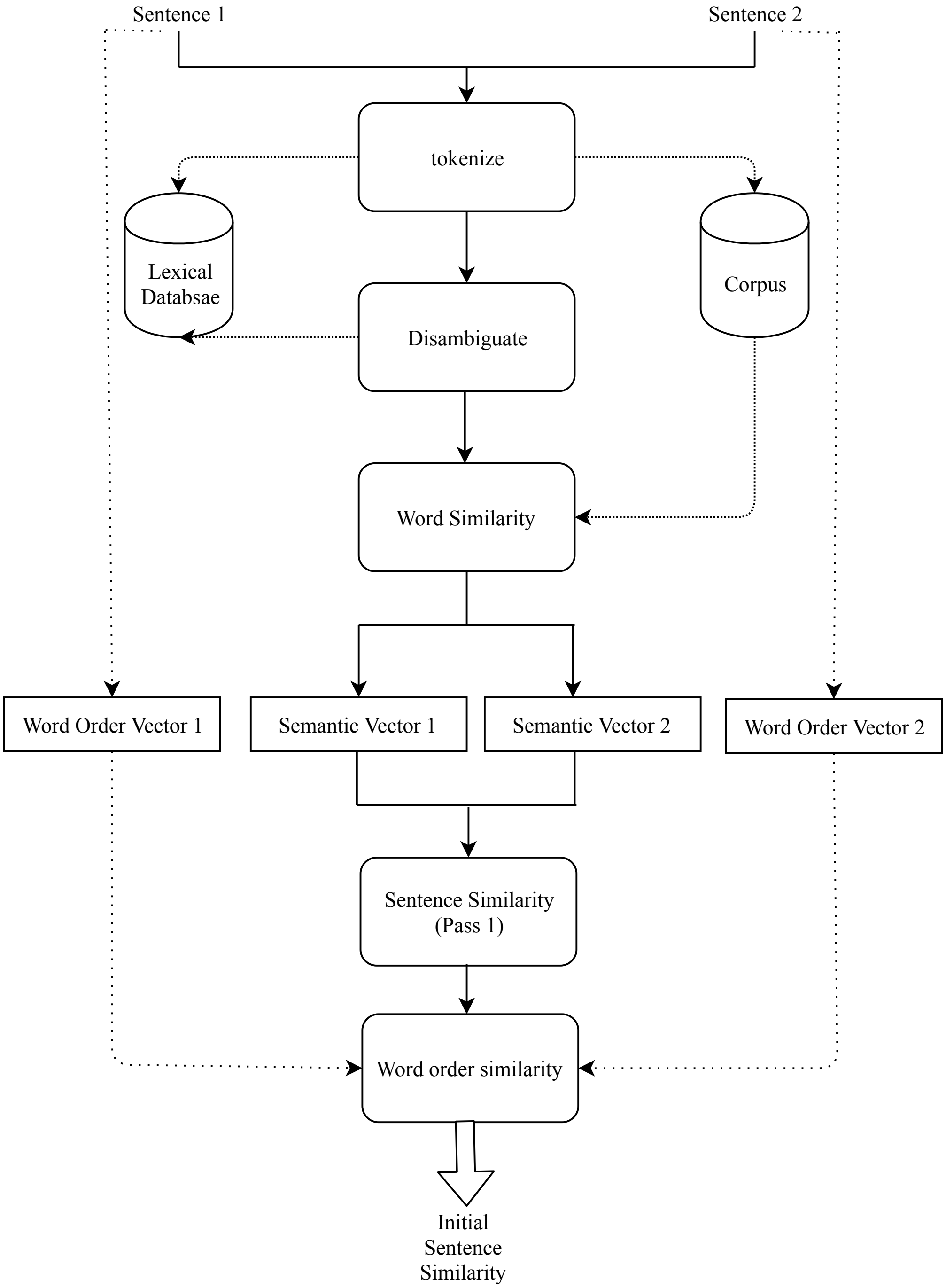


Figure 1.1: Pass 1 of the proposed sentence similarity methodology

use the fixed structure of vocabulary, the proposed method uses a lexical database to compare the appropriate meaning of the word. A semantic vector is formed for each sentence which contains the weight assigned to each word for every other word from the second sentence in comparison. This step also takes into account the information content of the word, for instance, word frequency from a standard corpus. Semantic similarity is calculated based on two semantic vectors. An order vector is formed for each sentence which considers the syntactic similarity between the sentences. Finally, semantic similarity is calculated based on semantic vectors and order vectors. *Pass 1* is divided into three parts:

- Word similarity
- Sentence similarity
- Word order similarity

The following section further describes each of the steps in more details.

1.3.1.1 Word Similarity

To compute the word similarity, the proposed method uses the sizeable lexical database for the English language, WordNet [49], from the Princeton University.

Identifying words for comparison Before calculating the semantic similarity between words, it is essential to determine the words for comparison. We use word tokenizer and ‘parts of speech tagging technique’ as implemented in natural language processing toolkit, NLTK [8]. This step filters the input sentence and tags the words into their ‘part of speech’(POS) and labels them accordingly. *WordNet* has path relationships between noun-noun and verb-verb only. Such relationships are absent in WordNet for the other parts of speeches. Hence, it is not possible to get a numerical value that represents the link between other parts of speech except nouns and verbs. We deal with other parts of speeches in pass 2 of the algorithm.
Example: ‘A voyage is a long journey on a ship or in a spacecraft’
Table 1.1 represents the words and the corresponding parts of speeches. The parts of speeches are as per the Penn Treebank [44].

Table 1.1: Parts of speeches

Word	**Part of Speech**
A	DT - Determiner
voyage	NN - Noun
is	VBZ - Verb
a	DT - Determiner
long	JJ - Adjective
journey	NN - Noun
on	IN - Preposition
a	DT - Determiner
ship	NN - Noun
or	CC - Coordinating conjunction
in	IN - Prepostion
a	DT - Determiner
spacecraft	NN - Noun

Associating word with a sense The primary structure of the WordNet is based on *synonymy*. Every word has *synsets* according to the meaning of the word in the context of a statement. The distance between *synsets* in comparison varies as we change the meaning of the word.

Consider an example where we calculate the shortest path distance between words 'river' and 'bank.' WordNet has only one synset for the word 'river'. We will calculate the path distance between synset of 'river' and three synsets of word 'bank'. Table 1.2 represents the synsets and corresponding definitions for the words 'bank' and 'river'.

Shortest distances for the Synset pairs are represented in Table 1.3. When comparing two sentences, we have many such word pairs which have multiple synsets. Therefore, not considering the proper synset in context of the sentence, could introduce errors at the early stage of similarity calculation. Hence, sense of the word has a significant effect on the overall similarity measure. Identifying the sense of the word is an area of research called 'word sense disambiguation'. We use 'max similarity' algorithm, Eq. (1.1), to perform word sense disambiguation [54] as implemented in Pywsd, an NLTK based Python library [62]. In Eq.(1.1), a is a query word and i

Table 1.2: Synsets and corresponding definitions from WordNet for words *bank* and *river*

Synset	**Definition**
Synset('river.n.01')	a large natural stream of water (larger than a creek)
Synset('bank.n.01')	sloping land (especially the slope beside a body of water)
Synset('bank.n.09')	a building in which the business of banking transacted
Synset('bank.n.06')	the funds held by a gambling house or the dealer in some gambling games

Table 1.3: Synsets and corresponding shortest path distances from WordNet

Synset Pair	**Shortest Path Distance**
Synset('river.n.01') - Synset('bank.n.01')	8
Synset('river.n.01') - Synset('bank.n.09')	10
Synset('river.n.01') - Synset('bank.n.06')	11

represents all the words in context.

$$argmax_{synset(a)} = (\sum_{i}^{n} max_{synset(i)}(sim(i,a))) \tag{1.1}$$

Shortest path distance between synsets Shortest path distance between synsets is the number of connecting edges between them in the lexical database, WordNet. The following example explains, in detail, the method used to calculate the shortest path distance. Referring to Figure 1.2, consider two words, viz.:
w1 = motorcycle and *w2 = car*
We are referring to *Synset('motorcycle.n.01')* for 'motorcycle' and *('car.n.01')* for 'car'.
The traversal path is : motorcycle → motor vehicle → car. Hence, the shortest path distance between motorcycle and car is 2. Listing A.1 represents the code to calculate the shortest path distance.

In WordNet, the gap between words increases as similarity decreases. Utilizing

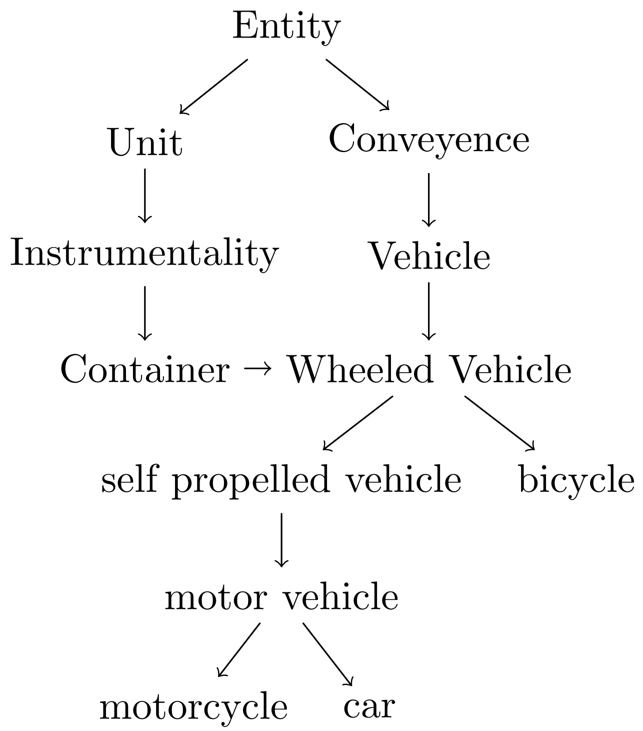


Figure 1.2: Hierarchical structure from WordNet

this property, we use the previously established monotonically decreasing function [36]:

$$f(l) = e^{-\alpha l} \tag{1.2}$$

where l is the shortest path distance and α is a constant. The selection of exponential function is to ensure that the value of *f(l)* lies between 0 to 1.

Hierarchical distribution of words In WordNet, the primary relationship between the synsets is the super-subordinate relation, also called *hyperonymy*, *hyponymy* or *ISA* relation [49]. This relationship connects the general concept synsets to the synsets that have specific characteristics. For example, Table 1.4 represents the word ‘vehicle’ and its *hyponyms.*

The hyponyms of ‘vehicle’ have more specific properties and represent the particular set, whereas ‘vehicle’ has more general properties. Hence, words at the upper layer of

Table 1.4: Synset and corresponding hyponyms from WordNet

Synset	**Hyponyms**

Synset(‘vehicle.n.01’)	Synset(‘bumper_car.n.01’)
	Synset(‘craft.n.02’)
	Synset(‘military_vehicle.n.01’)
	Synset(‘rocket.n.01’)
	Synset(‘skibob.n.01’)
	Synset(‘sled.n.01’)
	Synset(‘steamroller.n.02’)
	Synset(‘wheeled_vehicle.n.01’)

the hierarchy have more general features and less semantic information, as compared to words at the lower layer of the hierarchy [36].

Hierarchical distance plays an important role when the path distances between word pairs are same. For instance, referring to Figure 1.2, consider following word pairs: *car* - *motorcycle* and *bicycle* - *self_propelled_vehicle*.

The shortest path distance between both the pairs is 2, but the pair *car* - *motorcycle* has more semantic information and specific properties than *bicycle* - *self_propelled_vehicle*. Hence, we need to scale up the similarity measure if the word pair *subsume* words at the lower level of the hierarchy and scale down if they subsume words at the upper level of the hierarchy. To include this behavior, we use a previously established function [36]:

$$g(h) = \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}} \tag{1.3}$$

Listing A.2 represents the code to calculate the hierarchical distance. For WordNet, the optimal values of α and β are 0.2 and 0.45 respectively as reported previously [8].

Information content of the word The meaning of the word differs as we change the domain of operation. We can use this behavior of natural language to make the similarity measure domain-specific. It is used to influence the similarity measure if the domain operation is predetermined. Listing A.3 represents the code snippet to fetch the synset having maximum frequency for a word in WordNet.

To illustrate the *Information Content* of the word in action, consider the word: *bank*. The most frequent meaning of the word *bank* in the context of Potamology (the study of rivers) is *sloping land (especially the slope beside a body of water)*. The most

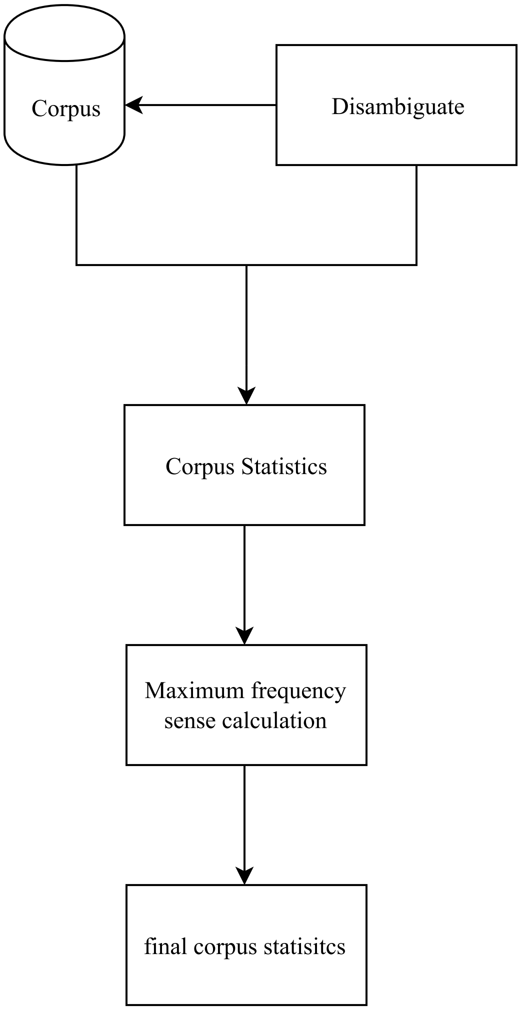


Figure 1.3: Method to calculate the frequency of a synset in a corpus
Corpus: An external corpus
Disambiguate: Function to identify appropriate synset for every word in the corpus
Corpus Statistics: An external file with corpus features
Maximum frequnecy sense calculation: A function to determine the sysnet with maximum frequency for every word
Final corpus statistics: An external file containing records from previous function

frequent meaning of the word *bank* in the context of Economics would be *a financial institution that accepts deposits and channels the money into lending activities.*

When applying the *Word Disambiguation Approach* described in subsection 1.3.1, the final similarity of the word would be different for every corpus. The corpus, belonging to particular domain, works as supervised learning data for the algorithm. We first disambiguate the whole corpus to get the sense of the word and further calculate the frequency of the particular sense. These statistics for the corpus work as the *knowledge base* for the algorithm. Figure 1.3 represents the steps involved in the analysis of corpus statistics.

1.3.1.2 Sentences' similarity

As Li [36] states, the meaning of the sentence is reflected by the words in the sentence. Hence, we can use the semantic information from subsection 1.3.1 to calculate the final similarity measure. Previously established methods to estimate the semantic similarity between sentences use the static approaches like using a precompiled list of words and phrases. The problem with this technique is the precompiled list of words and phrases which may not necessarily reflect the correct semantic information in the current context while comparing sentences.

The dynamic approach includes the formation of a joint word vector which compiles words from sentences and uses it as a baseline to form individual vectors. This method introduces inaccuracies in similarity calculations, particularly for the long sentences and the paragraphs containing multiple sentences.

Unlike these methods, our method forms the semantic value vectors for the sentences and aims to keep the size of the semantic value vector to the minimum. Formation of semantic vector begins after the subsection 1.3.1. This approach avoids the overhead involved to form semantic vectors separately unlike in previously discussed methods. Also, in this stage, we eliminate prepositions, conjunctions and interjections. Hence, these connectives are automatically eliminated from the semantic vector. We determine the size of the vector, based on the number of tokens from subsection 1.3.1. Every unit of the semantic vector is initialized to null to void the foundational effect. Initializing the semantic vector to a unit positive value discards the negative/null effects, and overall semantic similarity will be a reflection of the most similar words in the sentences. Listing A.4 represents the code to form the semantic vectors. Let's see an example.

S1= “A jewel is a precious stone used to decorate valuable things that you wear, such as rings or necklaces.”
S2= “A gem is a jewel or stone that is used in jewellery.”

List of tagged words for *S1*:
[(‘jewel’, Synset(‘jewel.n.01’)), Synset(‘jewel.n.02’)],
[(‘stone’, Synset(‘stone.n.02’)), Synset(‘stone.n.13’)],
[(‘used’, Synset(‘use.v.03’)), Synset(‘use.v.06’)],
[(‘decorate’, Synset(‘decorate.v.01’)), Synset(‘dress.v.09’)],
[(‘valuable’, Synset(‘valuable.a.01’)), Synset(‘valuable.s.02’)],
[(‘things’, Synset(‘thing.n.04’)), Synset(‘thing.n.12’)],
[(‘wear’, Synset(‘wear.v.01’)), Synset(‘wear.v.09’)],
[(‘rings’, Synset(‘ring.n.08’)), Synset(‘band.n.12’)],
[(‘necklaces’, Synset(‘necklace.n.01’)), Synset(‘necklace.n.01’)]

Length of the list of tagged words for *S1* is 9

List of tagged words for *S2*:
[(‘gem’, Synset(‘jewel.n.01’)), Synset(‘jewel.n.01’)],
[(‘jewel’, Synset(‘jewel.n.01’)), Synset(‘jewel.n.02’)],
[(‘stone’, Synset(‘gem.n.02’)), Synset(‘stone.n.13’)],
[(‘used’, Synset(‘use.v.03’)), Synset(‘use.v.06’)]
[(‘jewellery’, Synset(‘jewelry.n.01’)), Synset(‘jewelry.n.01’)]

Length of the list of tagged words for *S2* is 5

We eliminate words like *a, is, to, that, you, such, as, or*; hence further reducing the computing overhead. The resultant semantic vectors contain semantic information concerning all the words from both the sentences. For example, the semantic vector for *S1* is:

V1 = [0.99742103, 0.90118787, 0.42189901, 0.0, 0.0, 0.40630945, 0.0, 0.59202, 0.81750916]

Vector *V1* has semantic information from *S1* as well as from *S2*; because every word from *S1* is compared with every word from *S2* which implies that the resultant similarity is relative. Similarly, vector *V2* also has semantic information from *S1* and *S2*. To establish a similarity value using two vectors, we use the magnitude of the normalized vectors.

$$S = ||V1||.||V2|| \tag{1.4}$$

We make this method adaptable to longer sentences by introducing a variable(ζ) which is calculated dynamically at runtime. With the utilization of ζ, this method can also be used to compare paragraphs with multiple sentences.

Algorithm 1 Semantic similarity between sentences

```
    Input: Two tokenized sentences S1 and S2
    Output: Maximized semantic similarity between sentences
procedure SENTENCE_SIMILARITY
    S1_list_of_tagged_tokens(L1) ← disambiguate
    S2_list_of_tagged_tokens(L2) ← disambiguate
    vector_length ← max(length(S1),length(S2))
    V1, V2 ← vector_length(null)
    V1, V2 ← vector_length(word_similarity(S1,S2))
    $\zeta$=0 , $C1$=0, $C2$=0
    while $L1$ do
        if word_similarity_value > benchmark_similarity_value then
            $C1$ ← C1+1
    while $L2$ do
        if word_similarity_value > benchmark_similarity_value then
            $C2$ ← C2+1
    $\zeta \leftarrow sum(C1, C2)/\gamma$
    $S \leftarrow ||V1||.||V2||$
    if $sum(C1, C2) = 0$ then
        $\zeta \leftarrow vector\_length/2$
    $\delta \leftarrow S/\zeta$
```

Determination of ζ The words with maximum similarity have more impact on the magnitude of the vector. Using this property, we establish ζ for the sentences in comparison. According to R&G, the benchmark synonymy value of two words is 0.8025 [59]. Using this value as a determination standard, we calculate all the cells from *V1* and *V2* with the value greater than 0.8025. ζ is given by:

$$\zeta = sum(C1, C2)/\gamma \tag{1.5}$$

where $C1$ is count of valid elements in *V1* and $C2$ is count of valid elements in *V2*. γ is set to 1.8, determined by grid search over the correlation with R&G. Now, using Eq. 1.4 and Eq. 1.5, we establish similarity as:

$$\delta = S/\zeta \tag{1.6}$$

Algorithm 1 explains procedure.

1.3.1.3 Word Order Similarity

Along with semantic nature of the sentences, we need to consider the word order in the sentences. The word order similarity, simply put, is the aggregation of comparisons of word indices in two sentences. The semantic similarity approach based on words and the lexical database doesn't take into account the grammar of the sentence. Li [36] assigns a number to each word in the sentence and forms a word order vector according to their occurrence and similarity. They also consider the semantic similarity value of words to decide the word order vector. If a word from sentence 1 is not present in sentence 2, the number assigned to the index of this word in word order vector corresponds to the word with maximum similarity. This case is not always valid and introduces errors in the final semantic similarity index. For the methods which calculate the similarity by chunking the sentence into words, it is not always necessary to decide the word order similarity. For such techniques, the word order similarity actually matters when two sentences contain same words in different order. Otherwise, if the sentences contain different words, the word order similarity should be an optional construct. For such sentences, the impact of word order similarity is negligible as compared to the semantic similarity. Hence, in our approach, we implement word order similarity as an optional feature. Consider following classical example:

- *S1*: A quick brown dog jumps over the lazy fox.
- *S2*: A quick brown fox jumps over the lazy dog.

The edge-based approach using lexical database will produce a result showing that both *S1* and *S2* are same, but since the words appear in a different order we should scale down the overall similarity as they represent different meaning. We start with the formation of vectors *V1* and *V2* dynamically for sentences *S1* and *S2* respectively. Initialization of vectors is performed as explained in subsection 1.3.1.2. Instead of forming joint word set, we treat sentences relatively to keep the size of vector minimum.

The process starts with the sentence having maximum length. Vector *V1* is formed with respect to sentence 1 and cells in *V1* are initialized to index values of words in *S1* beginning with 1. Hence *V1* for *S1* is:

$V1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]$

Now, we form *V2* concerning *S1* and *S2*. To form *V2*, every word from *S2* is compared with *S1*. If the word from *S2* is absent in *S1*, then the cell in *V2* is filled with the index value of the word in sentence *S2*. If the word from *S2* matches with a word from *S1*, then the index of the word from *S1* is filled in *V2*.

In the above example, consider words 'fox' and 'dog' from sentence 2. The word 'fox' from *S2* is present in *S1* at the index 9. Hence, entry for 'fox' in *V2* would be 9. Similarly, the word 'dog' form *S2* is present in the *S1* at the index 4. Hence, entry for 'dog' in *V2* would be 9. Following the same procedure for all the words, we get *V2* as:

$V2 = [1, 2, 3, 9, 5, 6, 7, 8, 4]$

Finally, word order similarity is given by:

$$W_s = ||V1 - V2|| / ||V1 * V2|| \tag{1.7}$$

In this case, W_s is 0.067091.

1.3.2 Pass 2: Bound the similarity

The first pass of the algorithm returns the maximized similarity(δ) between two sentences. The second pass of the algorithm aims at computing a more robust similarity by reducing the ancillary similarity which causes skeweness in results by considering syntactical structure, adjectives and adverbs, and negations in the sentences. Skewness in this context implies the deviation of the similarity(δ) from the similarity in the SICK dataset.

We propose three approaches for the *Pass 2* of the algorithm.

- Model 1: Recurrence of words
- Model 2: Negation and stanford POS tagger model
- Model 3: Spacy's dependency parser model

Model 1: Recurrence of words

We consider the number of occurrences of a word with same meaning in the sentence. If a word occurs multiple times in the sentence, then we should reduce the impact of the word on the overall similarity. To illustrate this property of occurrences, consider following example:
S1: Explain the term Database and Database Management System DBMS, as well as the use of Primary and Foreign Key.
S2: Understand the fundamental concepts of relational database and implement a relational database.
The word *Database* occurs twice in both sentences. The impact it has on the final similarity is more than the actual information it adds to the sentence. Hence, while assigning the similarity value for such word pairs, we divide the subsequent occurrences by the number of occurrences.

$$V[word] = similarity/number\_of\_occurrences \tag{1.8}$$

where V represents the semantic vector. In this example, the value of similarity for *database* would be reduced to half as it occurs twice.

Model 2: Negation and stanford POS tagger model

The intuitive idea behind this model is to build a concise list containing syntactical information for both sentences and subsequently processing the lists to arrive at a decision value [42]. We focus on verbs, adverbs, and adjectives primarily. In this model, we use Stanford POS tagger, thesaurus.com Python API [1] and a list of English language contractions from Wikipedia [2]. We start by resolving the contractions to get the necessary form of the sentences. Both the sentences are tagged in their respective parts of speeches using Stanford's *bidirectional distsim tagger* [43]. A list is formed for both the sentences in following order:

1. The length of lists is determined by the length of the list containing POS of the sentences.
   $l = max(s1\_tagged, s2\_tagged)$

2. All the elements in the list are initialized to zero.

3. If the word is verb, adverb, adjective or negation, then the corresponding bit is set to represent the POS of the word.

Both the lists are compared as depicted in Figure 1.4. A decision is made explicitly for each verb, adverb, and adjective. If opposite sense is encountered in the sentences, then similarity δ is amended using following formula:

$$\omega = \delta/\theta \tag{1.9}$$

θ is set to 1.5. Through grid search we found that θ at 1.5 gives the highest correlation with the SICK values. Figure 1.5 represents the normal distribution using Gaussian curve of correlation with respect to θ. The correlation is determined concerning 4927 sentences from the SICK dataset [45].

Model 3: Spacy's dependency parser model

The model based on dependency parsing outperforms Model 1(subsection 1.3.2) and Model 2(subsection 1.3.2). We use Spacy's [24] dependency parser to get the dependecy grammar of the sentence. We follow a similar approach as in Model 2(subsection 1.3.2) by forming a list representing the dependency information of the sentence. We assemble the following information from dependency parsing:
$cell = \{token, token.pos, token.dep\}$
The above cell format represents a cell in the list. A *token* is a word from a sentence, *token.pos* is part of speech of the token in a sentence, *token.dep* depicts the dependency in the sentence. We maintain information about root, nouns, and verbs from both the sentences separately.

The goal of this approach is to keep track of the syntactical differences by incrementing a global dependency variable. We start the comparison with the roots of both the sentences. If roots are not similar or if the synsets of roots do not intersect each

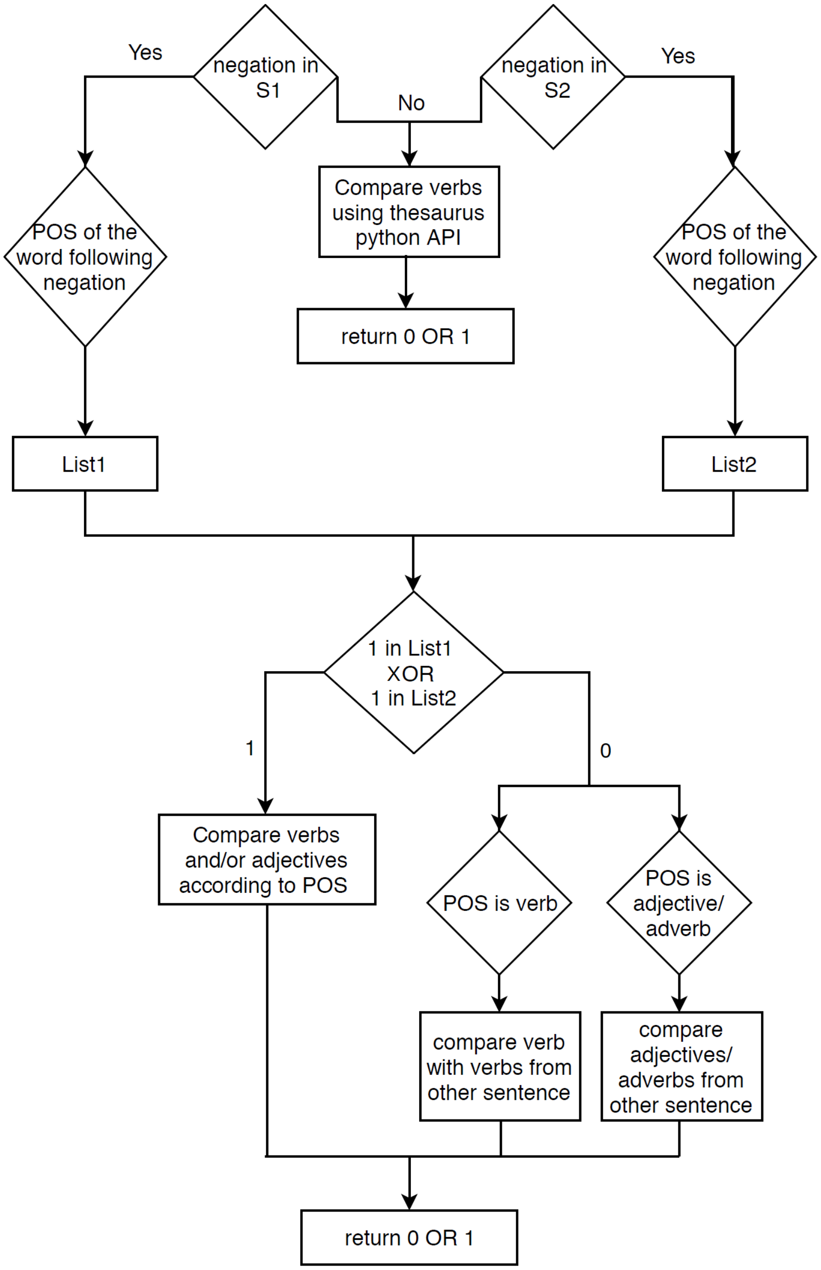

negation in S1
negation in S2
Yes
No
Yes
POS of the word following negation
Compare verbs using thesaurus python API
POS of the word following negation
return 0 OR 1
List1
List2
1 in List1 XOR 1 in List2
1
0
Compare verbs and/or adjectives according to POS
POS is verb
POS is adjective/ adverb
compare verb with verbs from other sentence
compare adjectives/ adverbs from other sentence
return 0 OR 1

Figure 1.4: Decision making for negation: 1 signifies the negation and 0 signifies no negation

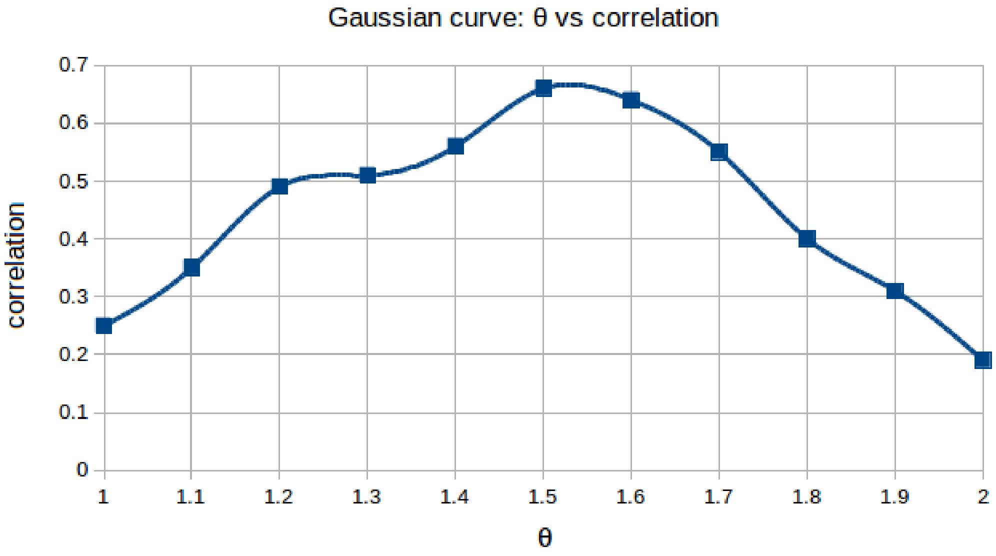


Figure 1.5: Normal distribution of θ over correlation

other, then we increment the dependency variable by 1. Next, we compare the lists containing nouns and accordingly increment the dependency variable. We consider the length of the lists containing nouns and the dependency of the nouns in the sentence. Similarly, we compare the lists containing verbs.

We check the negation explicitly. We maintain a list of words conveying negation. We use the SICK dataset to compile this list. If we encounter a word from the list of negation words, then we increase the dependency variable(dep_var) by 1. Length of sentences is also an important factor affecting the semantics of the sentences. We use following formula to calculate the *shift* between two sentences.

$$shift = \epsilon * log(abs(s1\_length - s2\_length) + 1) \tag{1.10}$$

where *S1_length* and *S2_length* are lengths of sentences 1 and 2 respectively. We establish a dependency index(dep_index) using following formula:

$$dep\_index = (\epsilon * tan^{-1}(dep\_var)) + shift \tag{1.11}$$

where ϵ is a constant, dep_var is dependency variable and *shift* represents the length difference from Eq. (1.10). ϵ is set to 0.10 through grid search over correlation on SICK dataset. Finally, we use this dep_index as a measure indicating the syntactical difference between two sentences. We establish final similarity as:

$$\omega = \delta - dep\_index \tag{1.12}$$

where ω is the final semantic similarity.

1.4 Implementation using Semantic nets

The database used to implement the proposed methodology is WordNet and statistical information from WordNet is used calculate the information content of the word. This section describes the prerequisites to implement the methods described in sections 1.3 and 1.4.

1.4.1 The Database - WordNet

WordNet is a lexical semantic dictionary available for online and offline use, developed and hosted at Princeton. The version used in this study is WordNet 3.0 which has 117,000 synonymous sets, *Synsets*. Synsets for a word represent the possible meanings of the word when used in a sentence. WordNet currently has synset structure for nouns, verbs, adjectives and adverbs. These lexicons are grouped separately and do not have interconnections; for instance, nouns and verbs are not interlinked.

The main relationship connecting the synsets is the super-subordinate(ISA-HASA) relationship. The relation becomes more general as we move up the hierarchy. The root node of all the noun hierarchies is ‘Entity’ like nouns, verbs are arranged into hierarchies as well.

Shortest path distance and hierarchical distances from WordNet

The WordNet relations connect the same parts of speech. Thus, it consists of four subnets of nouns, verbs, adjectives and adverbs respectively. Hence, determining the similarity between cross-domains is not possible.

The shortest path distance is calculated by using the tree-like hierarchical structure. To find the shortest path, we climb up the hierarchy from both the synsets and determine the meeting point which is also a synset. This synset is called *subsumer* of the respective synsets. The shortest path distance equals the hops from one synset to another.

We consider the position of subsumer of two synsets to determine the hierarchical distance. Subsumer is found by using the *hyperonymy* (ISA) relation for both the synsets. The algorithm moves up the hierarchy until a common synset is found. This common synset is the subsumer for the synsets in comparison. A set of hypernyms is formed individually for each synset and the intersubsection of sets contains the subsumer. If the intersubsection of these sets contain more than one synset, then the synset with the shortest path distance is considered as a subsumer.

The Information content of the word

For general purposes, we use the statistical information from WordNet for the information content of the word. WordNet provides the frequency of each synset in the WordNet corpus. This frequency distribution is used in the implementation of subsection 1.3.1.

1.4.2 Illustrative example

This subsection explains in detail the steps involved in the calculation of semantic similarity between two sentences. Consider following two sentences:

- *S1*: A gem is a jewel or stone that is used in jewellery.
- *S2*: A jewel is a precious stone used to decorate valuable things that you wear, such as rings or necklaces.

Following segment contains the parts of speeches and corresponding synsets used to determine the similarity.
For *S1* the tagged words are:

Synset('jewel.n.01') : a precious or semiprecious stone incorporated into a piece of jewelry
Synset('jewel.n.01') : a precious or semiprecious stone incorporated into a piece of jewelry
Synset('gem.n.02') : a crystalline rock that can be cut and polished for jewelry
Synset('use.v.03') : use up, consume fully
Synset('jewelry.n.01') : an adornment (as a bracelet or ring or necklace) made of

precious metals and set with gems (or imitation gems)

For *S2* the tagged words are:

Synset(‘jewel.n.01’) : a precious or semiprecious stone incorporated into a piece of jewelry
Synset(‘stone.n.02’) : building material consisting of a piece of rock hewn in a definite shape for a special purpose
Synset(‘use.v.03’) : use up, consume fully
Synset(‘decorate.v.01’) : make more attractive by adding ornament, colour, etc.
Synset(‘valuable.a.01’) : having great material or monetary value especially for use or exchange
Synset(‘thing.n.04’) : an artifact
Synset(‘wear.v.01’) : be dressed in
Synset(‘ring.n.08’) : jewelry consisting of a circlet of precious metal (often set with jewels) worn on the finger
Synset(‘necklace.n.01’) : jewelry consisting of a cord or chain (often bearing gems) worn about the neck as an ornament (especially by women)

After identifying the synsets for comparison, we find the shortest path distances between all the synsets and take the best matching result to form the semantic vector. The intermediate list is formed which contains the words and the identified synsets. L1 and L2 below represent the intermediate lists.

L1: [(‘gem’, Synset(‘jewel.n.01’))],
[(‘jewel’, Synset(‘jewel.n.01’))], [(‘stone’, Synset(‘gem.n.02’))], [(‘used’, Synset(‘use.v.03’))],
[(‘jewellery’, Synset(‘jewelry.n.01’))]

L2: [(‘jewel’, Synset(‘jewel.n.01’))],
[(‘stone’, Synset(‘stone.n.02’))], [(‘used’, Synset(‘use.v.03’))], [(‘decorate’, Synset(‘decorate.v.01’))]
[(‘valuable’, Synset(‘valuable.a.01’))],
[(‘things’, Synset(‘thing.n.04’))], [(‘wear’, Synset(‘wear.v.01’))], [(‘rings’, Synset(‘ring.n.08’))],
[(‘necklaces’, Synset(‘necklace.n.01’))]

Now we begin to form the semantic vectors for *S1* and *S2* by comparing every synset from *L1* with every synset from *L2*. The intermediate step here is to determine the size of semantic vector and initialize it to null. In this example, the size of the semantic vector is 9 by referring to the method explained in subsection 1.3.1. The following part contains the cross comparison of *L1* and *L2*.

Cross-comparison with all the words from *S1* and *S2* is essential because if a word from statement *S1* best matches with a word from *S2*, it does not necessarily mean that it would be true if the case is reversed. This scenario can be observed with the words *jewel* from Table 1.5 and *things* from Table 1.5. *things* best matches with *jewel* with index of 0.4063 whereas *jewel* from Table 1.5 best matches with *jewel* from Table 1.6.

After getting the similarity values for all the word pairs, we need to determine an index entry for the semantic vector. The entry in the semantic vector for a word is the highest similarity value from the comparison with the words from other sentence.

Table 1.5: L1 compared with L2

Words	**Similarity**
gem - jewel	**0.908008550956**
gem - stone	0.180732071642
gem - used	0.0
gem - decorate	0.0
gem - valuable	0.0
gem - things	0.284462910289
gem - wear	0.0
gem - rings	0.485032351325
gem - necklaces	0.669319889871
jewel - jewel	**0.997421032224**
jewel - stone	0.217431543606
jewel - used	0.0
jewel - decorate	0.0
jewel - valuable	0.0
jewel - things	0.406309448212
jewel - wear	0.0
jewel - rings	0.456849659596
jewel - necklaces	0.41718607131
stone - jewel	0.475813717007
stone - stone	**0.901187866267**

stone - used	0.0
stone - decorate	0.0
stone - valuable	0.0
stone - things	0.198770510639
stone - wear	0.0
stone - rings	0.100270000776
stone - necklaces	0.0856785820827
used - jewel	0.0
used - stone	0.0
used - used	**0.42189900525**
used - decorate	0.0
used - valuable	0.0
used - things	0.0
used - wear	0.0
used - rings	0.0
used - necklaces	0.0
jewellery - jewel	0.509332774797
jewellery - stone	0.220266070205
jewellery - used	0.0
jewellery - decorate	0.0
jewellery - valuable	0.0
jewellery - things	0.346687374295
jewellery - wear	0.0
jewellery - rings	0.592019999822
jewellery - necklaces	**0.81750915958**

Table 1.6: L2 compared with L1

Words	**Similarity**
jewel - gem	0.908008550956
jewel - jewel	**0.997421032224**
jewel - stone	0.475813717007
jewel - used	0.0
jewel - jewellery	0.509332774797

stone - gem	0.180732071642
stone - jewel	0.217431543606
stone - stone	**0.901187866267**
stone - used	0.0
stone - jewellery	0.220266070205
used - gem	0.0
used - jewel	0.0
used - stone	0.0
used - used	**0.42189900525**
used - jewellery	0.0
decorate - gem	0.0
decorate - jewel	0.0
decorate - stone	0.0
decorate - used	0.0
decorate - jewellery	0.0
valuable - gem	0.0
valuable - jewel	0.0
valuable - stone	0.0
valuable - used	0.0
valuable - jewellery	0.0
things - gem	0.284462910289
things - jewel	**0.406309448212**
things - stone	0.198770510639
things - used	0.0
things - jewellery	0.346687374295
wear - gem	0.0
wear - jewel	0.0
wear - stone	0.0
wear - used	0.0
wear - jewellery	0.0
rings - gem	**0.485032351325**
rings - jewel	0.456849659596
rings - stone	0.100270000776
rings - used	0.0
rings - jewellery	0.592019999822

necklaces - gem	0.669319889871
necklaces - jewel	0.41718607131
necklaces - stone	0.0856785820827
necklaces - used	0.0
necklaces - jewellery	**0.81750915958**

For instance, for the word *gem*, from Table 1.6, the corresponding semantic vector entry is 0.90800855 as it is the maximum of all the compared similarity values. Hence, we get *V1* and *V2* as following:

- *V1*= [0.90800855, 0.99742103, 0.90118787, 0.42189901, 0.81750916, 0.0, 0.0, 0.0, 0.0]
- *V2*= [0.99742103, 0.90118787, 0.42189901, 0.0, 0.0, 0.40630945, 0.0, 0.59202, 0.81750916]

The intermediate step here is to calculate the dot product of the magnitude of normalized vectors: *V1* and *V2* as explained in subsection 1.3.1.
$S = 3.472426$
The following segment explains the determination of ζ with reference to subsection 1.3.1.
From Algorithm 1, *C1* for *V1* is 4. *C2* for *V2* is 3. Hence, ζ is $(4{+}3)/1.8 = 3.89$.
Now, the final similarity is
$\delta = S/\zeta = 3.472426/3.89 = 0.8929$.
We execute Pass 2 of the algorithm using dependency parser model. Here *length_difference* for $S1$ and $S2$ is 7. Hence we obtain a *shift* of 0.2079. Next the *dependency_variable* computed is 5 considering roots, negation if any, count and index of nouns and verbs in both the sentences. We obtain *dependency_index* of 0.1373.

Finally,δ = 0.8929-(0.2079+0.1373) = 0.5477

1.5 Experimental Results

To evaluate the algorithm, we used three standard datasets:

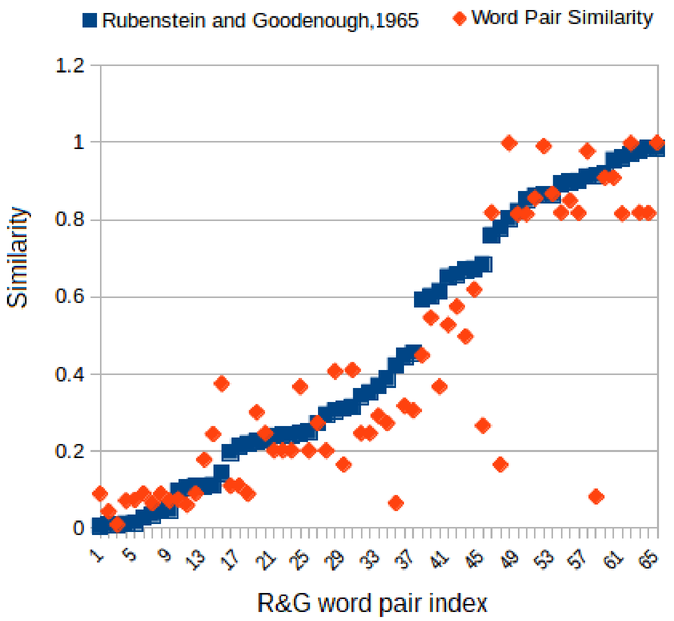


Figure 1.6: Perfomance of word similarity method vs Standard by Rubenstein and Goodenough

- Rubenstein and Goodenough word pairs [59]
- Sentence similarity for Rubenstein and Goodenough word pairs [53]
- SICK test dataset [45]

Table 1.7: Linear regression parameter values for proposed methodology

Slope	0.84312603549362108
Intercept	0.017742354112473213
r-value	0.87536955005374539
p-value	1.4816200698817255e-21
stderr	0.058665976202757132

These three datasets have been used in many investigations over the years and have been established as a stable source of semantic similarity measure. The word similarity obtained in this experiment is assisted by the standard sentences in Pilot Short Text Semantic Similarity Benchmark Data Set by James O'Shea [53]. The aim of this methodology is to achieve results as close as possible to the benchmark

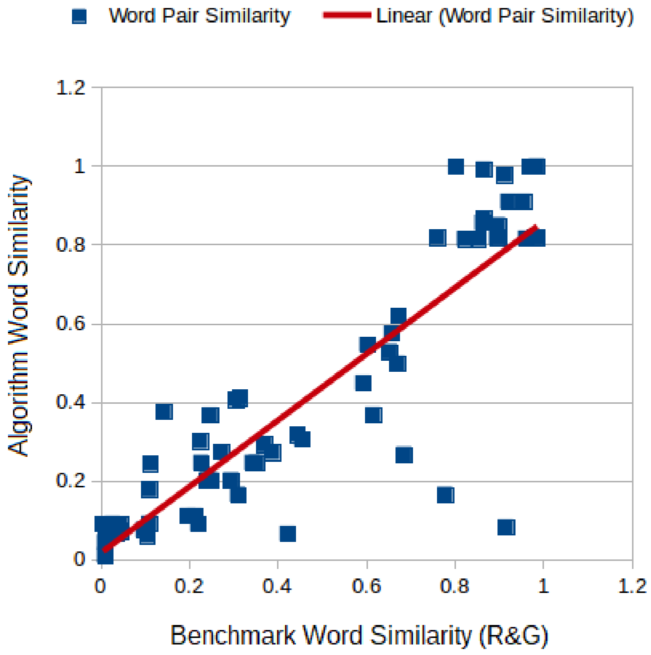


Figure 1.7: Linear Regression model word similarity method against Standard by Rubenstein and Goodenough

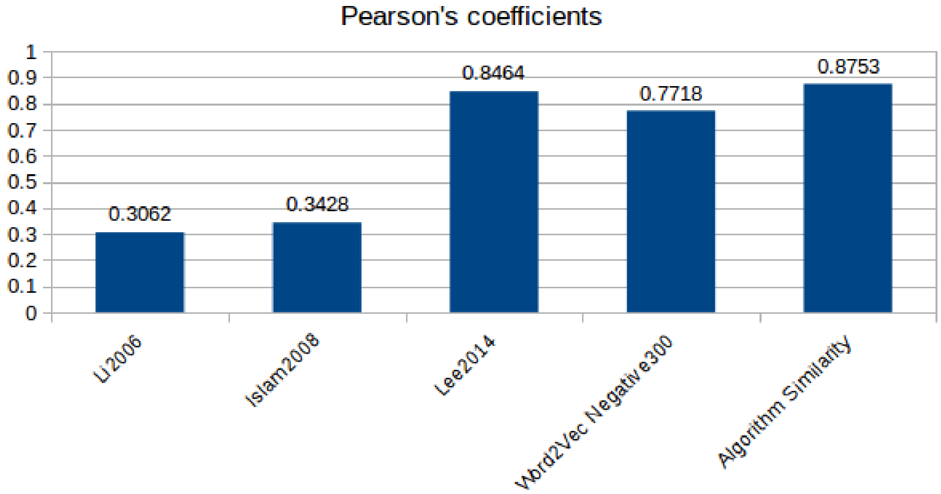


Figure 1.8: Pearson's coefficients various algorithms against Standard by Rubenstein and Goodenough

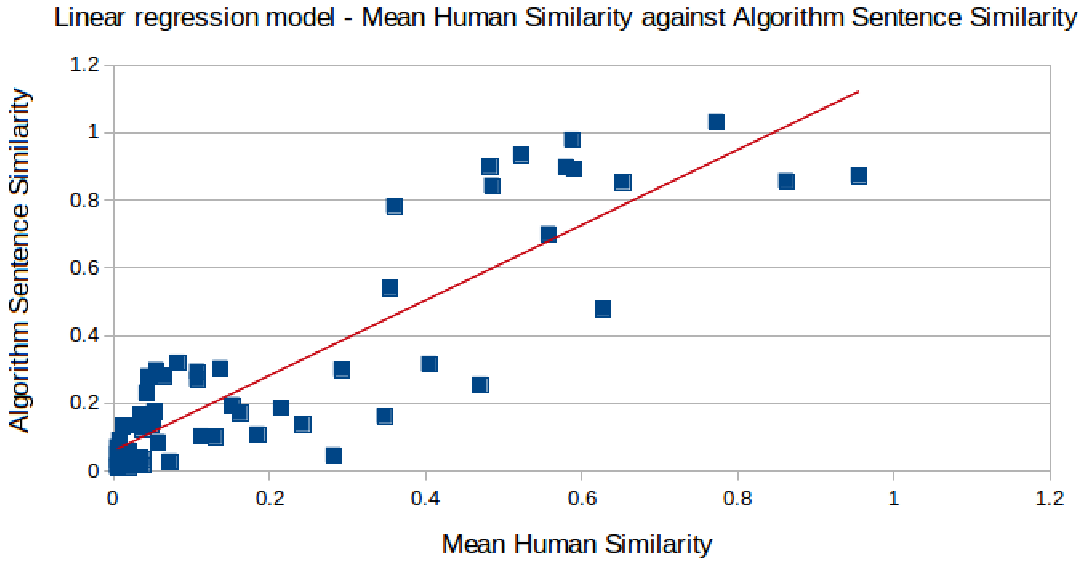


Figure 1.9: Linear regression model- Mean Human Similarity Vs Algorithm Sentence Similarity

standards [45] [59]. The definitions of the words are obtained from the Collins Cobuild dictionary.

1.5.1 Word similarity

Our algorithm achieved a good Pearson correlation coefficient of 0.8753695501 for word similarity which is higher than the existing algorithms. From Table 1.9 and Table 1.10 (see Appendix), we conclude that proposed method outperforms other methods for word similarity. The Pearson's coefficients for proposed method and other similar methods are showed in Figure 1.8 where proposed method achieved highest correlation.
Figure 1.6 represents the correlation results for 65 pairs from various algorithms against the R&G benchmark standard. Figure 1.7 represents the linear regression against the R&G standard. Table 1.7 shows the values of parameters for linear regression for word similarity.

1.5.2 Sentence similarity: R&G

Table 1.11 contains the mean human sentence similarity values from Pilot Short Text Semantic Similarity Benchmark Data Set by James O'Shea [53]. As Li [36] explains, when a survey was conducted by 32 participants to establish a measure for semantic similarity, they were asked to mark the sentences, not the words. Hence, word similarity is compared with the R&G [59] whereas sentence similarity is compared with mean human similarity. Our algorithm's sentence similarity achieved good Pearson correlation coefficient of 0.8794 with mean human similarity outperforming previous methods. Li [36] obtained correlation coefficient of 0.816 and Islam [27] obtained correlation coefficient of 0.853. The results for R&G sentences are displayed in Appendix Table 1.11. Out of 65 sentence pairs, 5 pairs were eliminated because of their definitions from Collins Cobuild dictionary [60]. The reasons and results are discussed in the discussion subsection.

1.5.3 Sentence similarity: SICK

To evaluate the sentence similarity algorithm, we used the SICK dataset which is considered as a stable measure of semantic correlation and has been used as a task in *SemEval 2014: semantic relatedness*. Our aim is to achieve semantic similarity as

Table 1.8: Results on the SICK semantic relatedness subtask. For our experiments, we report correlations and MSEs for 3 different models. Results are grouped as (1)Previously reported supervised models (2)Proposed unsupervised models

Method	**Pearson's r**	**Spearman's ρ**	**MSE**
Supervised methods			
Illinois-LH (Lai and Hockenmaier, 2014) [34]	0.7993	0.7538	0.3692
UNAL-NLP (Jimenez et al., 2014) [29]	0.8070	0.7489	0.3550
Meaning Factory (Bjerva et al., 2014) [9]	0.8268	0.7721	0.3224
ECNU (Zhao et al., 2014) [66]	0.8414	-	-
Constituency Tree-LSTM(Kai et al., 2015) [61]	0.8582	0.7966	0.2734
Dependency Tree-LSTM(Kai et al., 2015) [61]	0.8676	0.8083	0.2532
ConvNet(he2015multi) [22]	0.8686	0.8047	0.2606
MaLSTM(Mueller et al., 2016) [51]	0.8822	0.8345	0.2286
Proposed unsupervised models			
Recurrence of words	0.5878	0.5147	0.5585
Negatives and stanford POS tagger model	0.6645	0.5964	4389
Spacy's dependency parser model	**0.7958(0.8324)**	**0.6854**	**0.3784**

close as to the semantic similarity established in the SICK dataset. Sample results for SICK dataset are displayed in Table 1.12. We present the results obtained from the three proposed models. Table 1.8 represents the correlations obtained for each model.

Model 1: Recurrence of words

Model 1 utilizes the property that the reoccurring words in the sentences contain less semantic information than the words occurring once. This property is useful when dealing with longer sentences. There are very few incidences in the SICK dataset which possess this property. We obtained a correlation of 0.58 concerning SICK dataset.

Model 2: Negation and stanford POS tagger model

We obtained a fairly good correlation of 0.66 for model 2 which uses the Stanford POS tagger. This model performs well when all the words in both sentences are tagged correctly. It incurred few inaccuracies when negation is encountered in either or both sentences. The reason behind this behavior is the word following negation is tagged with a different POS than the corresponding word from the other sentence. Hence the negation calculation fails.

Model 3: Spacy's dependency parser model

The dependency parser model performed best and obtained a correlation of 0.79 which is the best performing unsupervised model until now, as the knowledge of authors. We also encountered a few *outliers*. *Outliers* are the cases where either the disambiguation function fails to identify the correct synset for the word or the dependency parser fails to form the fitting dependency model. Our algorithm's measure obtained the correlation of 0.83.

1.6 Computational Comparison

1.6.1 Comparison of hardware requirements(space complexity) for the proposed method vs. recent methods

The computational requirements for the proposed method are fundamental and do not require high-end hardware unlike the recent methods based on deep neural networks. The proposed method can be deployed on a very low capability CPU and does not require significant main memory except to load the dependent python modules; whereas methods based on deep neural networks first require high-end hardware such as GPUs to train the word embeddings model and then require a significant amount of main memory to load the trained model.

1.6.2 Comparison of the complexity of the proposed method vs. recent methods

The time complexity of the proposed algorithm is directly proportional to the number of words in the compared sentences. Since every word from both the sentences is compared with every word from other the sentence, the overall complexity of the algorithm can be estimated as $O(n^2)$. Consider an example, comparing the sentences containing ten words each, required 0.8 seconds to compute the similarity. This period also includes the time required to load the python modules. The subsequent comparisons when modules are already loaded took 0.5 seconds to compute the similarity.

1.7 Discussion & Future Work

Our algorithm's similarity measure achieved a good Pearson correlation coefficient of 0.8753 with R&G word pairs [59]. This performance outperforms all the previous methods. Table 1.9 represents the comparison of similarity from proposed method, Lee [35] and GoogleNews English Negatives300-word2vec with the R&G. Table 1.10 depicts the comparison of algorithm similarity against Islam [27] and Li [36] for the 30 noun pairs and performs better.

For sentence similarity, the pairs *17: coast-forest, 24: lad-wizard, 30: coast-hill, 33: hill-woodland and 39: brother-lad* are not considered. The reason for this is, the definition of these word pairs have more than one common or synonymous words. Hence, the overall sentence similarity does not reflect the true sense of these word pairs as they are rated with low similarity in mean human ratings. For example, the definition of 'lad' is given as: 'A lad is a young man or boy.' and the definition of 'wizard' is: 'In legends and fairy stories, a wizard is a man who has magic powers.' Both sentences have similar or closely related words such as: 'man-man', 'boy-man' and 'lad-man'. Hence, these pairs affect overall similarity measure more than the actual words compared 'lad-wizard'.

There's a scope of improvement in the *disambiguation* function. We observed that the disambiguation function fails to identify the correct synset for the word occasionally. One of the examples is the word 'can' which is always identified as the helping verb despite of the context.

1.8 Conclusions

This chapter presented an unsupervised approach to calculate the semantic similarity between two words, sentences or paragraphs which is applicable across multiple domains. The ability to accurately predict semantic similarity using a robust algorithm, a standardized lexical database and interchangable corpora with low computing overhead is beneficial to professionals in all domains requiring semantic similarity calculations. The algorithm initially disambiguates both the sentences and tags them in their parts of speeches. The disambiguation approach ensures the right meaning of the word for comparison. The similarity between words is calculated based on a previously established edge-based approach. The information content from a corpus can be used to influence the similarity in particular domain. Semantic vectors containing similarities between words are formed for sentences and further used for sentence similarity calculation. Word order vectors are also formed to calculate the impact of the syntactic structure of the sentences. Since word order affects less on the overall similarity than that of semantic similarity, word order similarity is weighted to a smaller extent. The methodology has been tested on previously established data sets which contain standard results as well as mean human results. Our algorithm achieved a good Pearson correlation coefficient of 0.8753 for word similarity concerning the benchmark standard and 0.8794 for sentence similarity with respect to mean human similarity and 0.7958 concerning the SICK dataset.

Chapter 2

Similarity between Learning Objectives from Course Outlines using Semantic Analysis, Blooms taxonomy and Corpus statistics

2.1 Introduction

The Learning Outcomes or Learning Objectives(LO) of a course define what the student is expected to learn by taking the course. LOs form a crucial part of any course description; hence these objectives of a course are considered as a base criterion to compare the two courses [19]. If a student is transferring from institution A to institution B and is also attempting to transfer credits from institution A, then accurate comparison of courses is essential in deciding if the student is eligible to receive credit at institution B. This task of examining the LOs from two courses is currently completed by a variety of personnel including Enrollment and Admissions Staff, Program Chairs, Faculty and Transfer Officers, Program Managers. For the sake of simplification, we are assuming that **Program Managers** are the personnel responsible for creating Transfer Program Agreements between institutions.

In addition to assessing transfer credit on a course by course basis, post-secondary institutes also develop transfer credit agreements called ‘articulation agreements’ by which they agree to give graduates of a specific program a set amount of transfer credit. The development of articulation agreements requires human intelligence and expertise from numerous stakeholders and content experts which is resource intensive and time consuming [33]. This process requires human intelligence and expertise to evaluate the course objectives. Similarly, Program Managers depend on domain experts to finalize the decision. Domain experts are persons who have knowledge of a particular field. This process depends on the human interference throughout; hence is resource and time consuming.

Our intelligent system aims to automate the process of assessing similarities between learning outcomes to assist institutional stakeholders in deciding whether a given student is eligible to recieve credits or not, by extracting the learning outcomes from the course objectives and comparing them semantically. Bloom’s taxonomy is often used to help guide the development of learning outcomes [5] when structuring the learning outcomes. Bloom’s taxonomy provides general keyword guidelines, and a hierarchical structure to be used when defining the learning outcomes [32], see Figure 2.1 [15]. But within our testing and course outline data set, we found that many instructors have not applied the language associated with Bloom’s Taxonomy in their learning outcomes. So, in our methodology, we limit the influence of Bloom’s

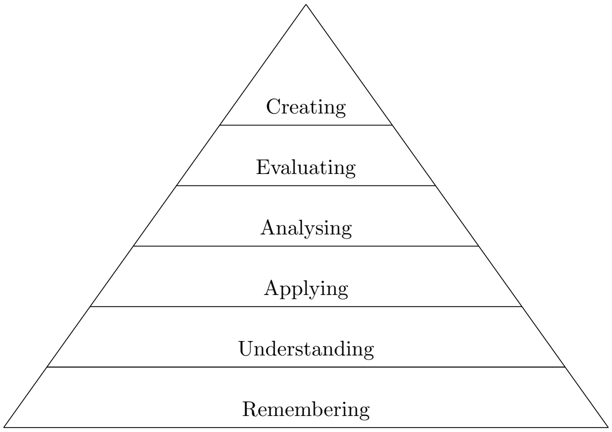


Figure 2.1: Hierarchical Structure of Bloom's Taxonomy

taxonomy by analyzing only the verbs. We use the hierarchical distribution of verbs in Bloom's taxonomy to compare learning objectives. Each layer in Bloom's taxonomy, as depicted in Figure 2.1, has a list of verbs associated with it [25]. The main contributions of this research are:

- Development of LO similarity measures using semantic analysis
- Utilizing Bloom's Taxonomy to determine the level of learning associated with the LOs
- Demonstrating the effect and the usage of corpus statistics

Section 2.2 of this chapter elaborates the methodology step by step. Section 2.3 describes the implementation in detail. Section 2.4 analyses the experimental results. Finally, section 2.5 explains the results in brief and draws the conclusion.

2.2 Methodology

The proposed methodology uses a semantic similarity algorithm discussed in chapter 1 and extends it to work with Bloom's taxonomy and a corpus related to the specific domain. Figure 2.2 shows the modules for computing the similarity between two learning objectives. The components of Figure 2.2 are explained below:
LO1 and ***LO2*** are the two learning objectives for comparison.
Similarity Algorithm is the algorithm explained in chapter 1.
Corpus is a compiled corpus for a particular domain.

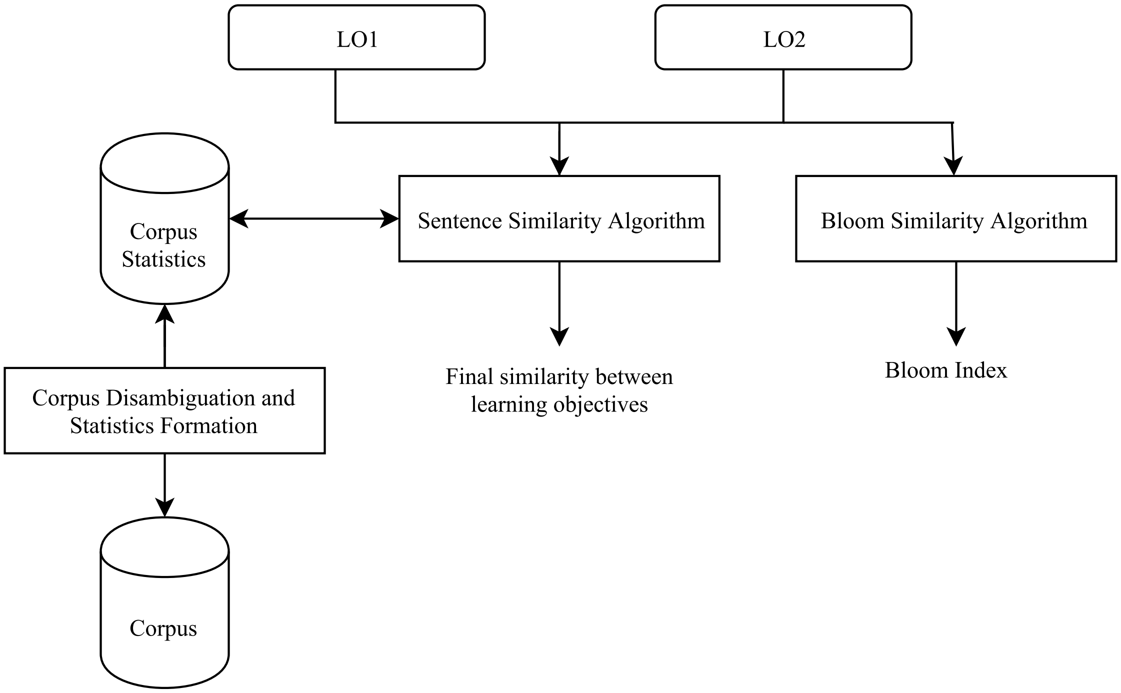

LO1
LO2
Corpus Statistics
Sentence Similarity Algorithm
Bloom Similarity Algorithm
Final similarity between learning objectives
Bloom Index
Corpus Disambiguation and Statistics Formation
Corpus

Figure 2.2: The proposed methodology

Corpus Disambiguation and Statistics Formation is the algorithm to extract the domain-specific properties from the corpus.
Corpus Statistics is a file containing records of words and corresponding synsets.
Bloom Similarity Algorithm is the algorithm explained in section 2.2.2.

The semantic similarity algorithm uses *Synsets* from Wordnet to calculate the semantic similarity between the sentences. This methodology aims to identify the correct Synsets according to the meaning of the word in sentence using corpus statistics. The methodology is divided into the following subsections:

- Semantic similarity algorithm
- Bloom's taxonomy
- Corpus statistics

2.2.1 Semantic similarity algorithm

The semantic similarity algorithm used in this method is an edge-based approach which uses WordNet, a lexical database. The method to calculate the semantic similarity between two sentences is divided into three parts:

- Word similarity
- Sentence similarity
- Word order similarity

This method first calculates the semantic similarity between words considering the meaning of the word in the context of the statement. The best result is then used to form a semantic vector for both the sentences separately. The semantic vectors formed are used to calculate the semantic similarity. The word order vector is constructed by considering the syntactic structure of the sentences, i.e., the occurrences of words concerning each other. This suite of algorithms is explained in chapter 1.

2.2.2 Bloom's taxonomy

As discussed in section 2.1, a well-structured course objective describes what students will be able to learn and to do as a result of the course [4]. Bloom's taxonomy is well-known, established, the hierarchically structured model which contains action verbs in a level of the hierarchy. As we move up the hierarchy, the difficulty level of action verb increases. The upper three layers, *Analysis*, *Synthesis* and *Evaluation* demonstrate the verbs associated with higher levels of critical thinking. We implement Bloom's taxonomy as separate part of methodology and restrain its influence on the main sentence similarity methodology. We explain the reason in the following subsection.

Problem with integrating Bloom's taxonomy with the principal method

Though Bloom's taxonomy is the suggested standard for designing the course outline, we have found that a considerable number of course drafts differ significantly from the norm. Hence, treating such LOs as well structured and integrating it with the primary methodology violates the purpose. Therefore to use the Bloom's taxonomy, we establish the *Bloom's Index*. The Bloom's Index represents the learning gap between two learning outcomes according to the verbs in LOs.

Bloom's Index

We start with identifying the action verbs in learning outcomes. Two lists are formed containing the action verbs from each LO respectively. We use Stanford POS Tagger [43] to tag the words and identify action verbs. Each layer in the hierarchy is given a numerical value starting from 1 and going up to 6 as we move up the hierarchy. The absolute distance between the numerical values of layers of verbs yields the distance between two verbs. We use this relative distance to calculate the index for each pair of the verb. The absolute Bloom's index for each pair is given by:

$$absolue\_bloom\_index = \alpha \times distance + \beta \tag{2.1}$$

where α = -0.20 and β = 1. The *absolue_bloom_index* represents the absolute similarity between two verbs according to the Bloom hierarchy. If both verbs fall into the same category, then they represent the same learning level; hence for such verb pairs it is logical to assign a similarity index which represents maximum similarity. Since, most of the similarity algorithms follow the range from 0 to 1 for the similarity index, we follow the same standard and establish the maximum Bloom similarity as 1. Since the hierarchy is divided into 6 levels uniformaly, and the range for bloom index is 0 to 1, we set incremental or decremental distance as 0.2.

We add the absolute bloom indices of all the verb pairs and get the Total Bloom Index.

$$Total\_Bloom\_Similarity = \sum Absolute\_Bloom\_indices \tag{2.2}$$

Now, to limit the value of Bloom_Index to 0 to 1, we use the total number of comparisons for verb pairs. Finally, Bloom index is given by:

$$Bloom\_Index = Total\_Bloom\_Similarity/number\_of\_comparisons \tag{2.3}$$

Listing A.5 represents the code to the Bloom's Index.

2.2.3 Corpus statistics

The selection of corpus affects the similarity index by a considerable amount. Learning objectives have some peculiar words which are associated with a specific domain of education (i.e. Economics, Physics, English Literature, Software Engineering), therefore using a general-purpose corpus is insufficient to assess semantic similarity within the context. In general, the meaning of the words differs when the context is changed. Hence, use of the general-purpose corpus does not assure proper meaning and context for words. No single corpus serves the purpose as the terminologies used in LOs are different for every domain. For example, the terminologies used in Computer Science are different from that of Economics and Chemistry. Our similarity algorithm uses Synsets from the WordNet to calculate the semantic similarity between the words. A word can have multiple synsets with different meanings. Hence, it is essential to identify the appropriate Synset.

This methodology simulates the use of corpus as a supervised learning model. The corpus is then *disambiguated*, i.e., we find the appropriate sense for each word in the corpus. Identifying sense of the word is part of "word sense disambiguation" research area. We use 'max similarity' algorithm, Eq. 1.1, to identify the sense of the words [54], as implemented in Pywsd, an NLTK based python library [8]. In this stage, we identify the meaning of the word and the synset corresponding to this definition from the WordNet. This information is stored in conjunction with each other to use efficiently for further calculations. The format used is: Word → Synset → Meaning of the word

This information also serves as a replica of 'Educational Ontologies' synchronous with WordNet. A replica of Educational Ontologies means we get the synsets corresponding to a particular domain and we can traverse and get the distance between these synsets using WordNet. Then we run a separate thread to establish the frequencies of the synsets and group them according to the meaning. The process is repeated everytime the corpus is changed or updated, and new storage is created for every run. In case of rare events, if the disambiguation function fails to tag a word, then we use the statistics from the WordNet. WordNet has the predefined frequency distribution of definition of the words. We use this frequency for the failed words.

2.2.4 Information content of the word

We also consider the number of occurrences of a word in the sentence. If the word occurs multiple times in the sentence, then we should reduce the impact of the word on the overall similarity. To illustrate this property of occurrences, consider following example:

LO1: Explain the term Database and Database Management System DBMS, as well

Algorithm 2 Corpus statistics compilation

```
Input: Corpus of a specific domain
Output: Word-Synset-Frequency record file
procedure SYNSET_FREQUENCY_DETERMINATION
    $Remove\_ASCII\_characters$
    $Remove\_stop\_words$
    $Remove\_words\_with\_frequenct < n$
    $sent\_list \leftarrow Sentence\_tokenize\_corpus$
    while $sent\_list$ do
        if $len(disambiguate\_sentence) > 0$ then
            $update\_synset\_frequency\_file \leftarrow disambiguated\_synsets$
    $determine\_words\_with\_max\_frequency\_synset$
    $rearrange\_synset\_frequency\_file\_according\_to\_frequency$
```

as the use of Primary and Foreign Key.

LO2: Understand the fundamental concepts of relational database and implement a relational database.

The word *Database* occurs twice in both LOs. The impact it has on the final similarity is more than the actual information it adds to the sentence. Hence, while assigning the similarity value for such word pairs, we divide the subsequent occurrences by the number of occurrences.

$$sim\_value = similarity\_index/number\_of\_occurrences \tag{2.4}$$

2.3 Implementation

We use previously established Sentence Similarity algorithm and modify it as explained in section 2.2. The database used to implement the proposed methodology is WordNet and the statistical information from WordNet. A corpus of the 'Chemistry' domain is compiled using the corpus compilation method explained in Chapter 3, section 3.2.1.

2.3.1 The Databse - WordNet

WordNet is a lexical semantic dictionary available for online and offline use, developed and hosted at Princeton. The WordNet version used for this study is WordNet 3.0

which has 117,000 synonymous sets, Synsets. Synsets of a word represent possible meanings of the word in the context of a sentence. The central relationship connecting the synsets is the super-subordinate(ISA-HASA) relationship. We use this connection to find the shortest path distance and use this distance to establish similarity between word pairs.

2.3.2 Corpus statistics

We present a simulation of formation of corpus statistics using a small corpus. Consider following LOs from the corpus.
LO1: Describe the subatomic composition of atoms, ions and isotopes.
LO2: An electrical force linking atoms and molecular bonds in chemicals.
LO3: Write electronic configurations of atoms and ions and relate to the structure of the Periodic Table.

Table 2.1 and Table 2.2 represent the Synsets for *LO1* and textitLO2 according to information retrieved from the corpus. Similarly, all the LOs from the corpus are disambiguated to get the data. We then calculate the frequency distribution of synsets corresponding to words. For instance, the frequency of synset *Synset('atom.n.01')* is 6 in the corpus statistics file. Hence, whenever the word *atom* occurs in the LO, the synset considered for semantic similarity calculation is *Synset('atom.n.01')*. Identically, the statistics are formed for all the synsets and words. Having a well-performing word disambiguation function is crucial to get the precise information from the corpus.

2.3.3 Bloom's Taxonomy

To implement Bloom's Taxonomy, we consider the traditional hierarchical structure. We use the verbs listed in Blooms Taxonomy of Measurable Verbs [25] arranged in the hierarchy. Each level in the hierarchy is assigned a number starting at 1 with the base 'Remembering' and going up to 6 with 'Creating'. To tag the verbs in the LOs, we use the Stanford POS tagger [43].

Table 2.1: Disambiguated data for LO1

Term	Synset	Meaning
subatomic	Synset('subatomic.a.01')	of or relating to constituents of the atom or forces within the atom
composition	Synset('composition.n.03')	a mixture of ingredients
atoms	Synset('atom.n.01')	(physics and chemistry) the smallest component of an element having the chemical properties of the element
ions	Synset('ion.n.01')	a particle that is electrically charged (positive or negative); an atom or molecule or group that has lost or gained one or more electrons
isotopes	Synset('isotope.n.01')	one of two or more atoms with the same atomic number but with different numbers of neutrons

2.3.4 Illustrative Example

This subsection explains the working of methodology to calculate the Bloom's Index and the usage of corpus statistics.

Calculating Bloom's Index

Consider following sentences:
S1: Discuss the application of the scientific method to the study of human thinking, development, disorders, therapy, and social processes
S2: Identify major health informatics applications and develop basic familiarity with healthcare IT products
From S1, we tag one verb, *discuss*.
From S2, we tag two verbs,viz., *identify* and *develop*.
Here we have 2 comparisons: disucss↔identify and develop↔discuss. Hierarchical distance between disucss↔identify is 1, similarly Hierarchical distance between develop↔discuss is 3. Using Eq.(2.1), we get absolute_blooms_index as 0.8 and 0.4 respectively. Now using Eq.(2.3), we get the Bloom's Index as (0.8+0.4)/2=0.6.

Corpus Statistics

Consider two LOs listed below:
LO1: Describe the subatomic composition of atoms, ions and isotopes.

Table 2.2: Disambiguated data for LO2

Term	**Synset**	**Meaning**
electrical	Synset('electrical.a.01')	relating to or concerned with electricity
force	Synset('power.n.05')	one possessing or exercising power or influence or authority
linking	Synset('connect.v.01')	connect, fasten, or put together two or more pieces
atoms	Synset('atom.n.01')	(physics and chemistry) the smallest component of an element having the chemical properties of the element
bond	Synset('bond.n.01')	an electrical force linking atoms
chemistry	Synset('chemistry.n.02')	the chemical composition and properties of a substance or object
chemical	Synset('chemical.n.01')	material produced by or used in a reaction involving changes in atoms or molecules

LO3: Write electronic configurations of atoms and ions and relate to the structure of the Periodic Table.

Table 2.1 and Table 2.3 depicts the words, synsets, and meanings for LO1 and LO3 respectively.

From Table 2.1, considering the meanings of the words, we can conclude that the disambiguation worked fine and we have appropriate synsets for the further calculations; whereas, from Table 2.3, we conclude that there are some inaccuracies with the words such as *structure* and *table*. The meaning of these words we get after disambiguation is different from their contextual sense in the sentence. The expected meaning of table here is *a tabular array (a set of data arranged in rows and columns)*, and structure is *a structure (the manner of construction of something and the arrangement of its parts)*.

Using right set of LOs corresponding to the appropriate domain, we get the synset with the correct meaning. Even while disambiguating the corpus, the disambiguation function can identify inaccurate meaning for a word. Using frequency of the occurrence of meaning in corpus deprecates this inaccuracy since we always get the synset which has appropriate meaning for the word in the context of domain.

Table 2.3: Disambiguated data for LO3

electronic	Synset('electronic.a.02')	of or concerned with electrons
configurations	Synset('shape.n.01')	any spatial attributes (especially as defined by outline)
atoms	Synset('atom.n.01')	(physics and chemistry) the smallest component of an element having the chemical properties of the element
ions	Synset('ion.n.01')	a particle that is electrically charged (positive or negative); an atom or molecule or group that has lost or gained one or more electrons
structure	Synset('structure.n.03')	the complex composition of knowledge as elements and their combinations
Table	Synset('table.n.05')	a company of people assembled at a table for a meal or game

2.4 Experimental Results

To evaluate the algorithm, we used real Learning Objectives from various course outlines. A survey was conducted and users were asked if they can make a decision based on the resultant semantic similarity and the Bloom Similarity. All the users at least possessed a Bachelors degree and had taken at least introductory courses in Chemistry. Users were asked to verify the results of the compared learning objectives. Out of 15 users, 10 users agreed that 75% or more of the results were useful; 1 user agreed that 65% or more of the results were useful and 4 users agreed that 50% or more of the results were useful.

Table 2.4 shows the semantic similarity between real-time LOs.

Table 2.4: Similarity between LOs

LO1	LO2	Proposed Algorithm Similarity
Acquire knowledge: memorize factual information and laws; assimilate scientific concepts; learn chemical calculations	To predict the physical and chemical properties of organic molecules from structures.	0.243231930716

Students will develop both problem solving and critical thinking skills, and they will use these skills to solve problems utilizing chemical principles.	use knowledge of intermolecular forces to predict the physical properties of molecular and extended-network elements and compounds;	0.295240282004
apply chemical knowledge to integrate knowledge gained in other courses and to better understand the connections between the various branches of science;	To become familiar with the structures of organic molecules, especially those found in nature or those with important biological effects.	0.260902736198
understand and utilize the terminology and concepts of chemistry to acquire and communicate scientific information and to solve basic chemical problems;	To predict the physical and chemical properties of organic molecules from structures	0.240184283928
solve problems involving the physical properties of matter in the solid, liquid and gaseous states;	Students will gain an appreciation of the scientific discipline of chemistry and the principles used by chemists to solve complex problems.	0.223101105502
understand the basis of the unique properties of mixtures and perform related calculations;	memorize factual information and laws; assimilate scientific concepts; learn chemical calculations	0.289648142927
apply knowledge of thermochemistry to calculate enthalpy changes associated with chemical and physical processes;	solve problems involving the physical properties of matter in the solid, liquid and gaseous states;	0.113466429084

Write electronic configurations of atoms and ions and relate to the structure of the Periodic Table.	Describe the subatomic composition of atoms, ions and isotopes.	0.852869346717
Students will learn and apply the method of inquiry used by chemists to solve chemical problems.	Describe the role of chemists and chemistry in drug design.	0.214912072273
Examine, integrate, and assess any provided or collected chemical data.	Draw scientific conclusions from experimental results or data.	0.900301710749

2.5 Conclusion & Future Work

This chapter presented an approach to calculate the semantic similarity between learning objectives using Corpus Statistics and Blooms taxonomy. The crucial part of the algorithm is the disambiguation of words in the context of their use. Having fewer datapoints may lead to detecting the wrong meaning of the word. Hence, using a corpus, we make sure that the algorithm always selects the appropriate sense of the word. We use corpus statistics from the disambiguated corpus. The meaning with the highest frequency is considered by the algorithm to find the proper synset from the WordNet. The methodology has been tested on real-time learning objectives, and we have achieved positive results.

While our initial experiment shows promise in terms of the accuracy of assessing semantic similarity between Learning Outcomes, we are currently extending this experimentation phase to include assessing semantic similarity between multiple courses from high affinity post-secondary credentials to test this research on a broader scale. Future work includes expanding the domains, increasing efficiency of algorithms by using different file structures and forming WordNet-like ontologies for specifically the education domain. Use of fuzzy cognitive maps to form ontologies is also a possibility [18][40][39][41]. For the purpose of this research we selected Bloom's Taxonomy based on it's visibility in documents published by the Higher Education Quality Council of Ontario, however we recognize that future applications of this work could include Bigg's SOLO taxonomy [7] as well.

Chapter 3

Word embeddings for semantic similarity using Wikipedia as a corpus

3.1 Introduction

The recent advancements in using neural networks for natural language processing has lead to significant improvements [23][51][61][34][9] in the area of semantic analysis. The models based on neural networks are able to achieve state-of-the-art performace; for instance, the *Siamese Recurrent Architectures for Learning Sentence Similarity* achieved the correlation of 0.8822 with the SICK dataset [51]. We use the similar architecture to determine the semantic similarity between *learning objectives* which is the application domain for this research. *Learning objectives* contain peculiar words which are rarely used in general English and might have a different meaning. For instance, the computer science domain has hundreds of programming languages such as python, java, lua, etc. Such terms used in learning objectives make it impossible to use lexical databases such as WordNet because of the limitation of words. Also, it has become increasingly difficult to maintain and update the WordNet as it is resource consuming and requires human intervention to redefine or add new relations between words.

Word embeddings are shallow, two-layered neural network models which capture the linguistic context of words. Word embeddings represent the words converted into numerical format. The basic idea that shapes the word embedding representation is “a word is characterized by the company it keeps” [14]. The intuitive idea is to use a text corpus and form vectors for each word in the corpus, such as *one-hot vectors*. The distributional representation of the vectors is based on the usage of the word in the corpus. One of the algorithms used to create word embeddings is Word2vec developed by Thomas Mikolov, et al. at Google in 2013. The results of word2vec are sensitive to parameterization. The parameters of word2vec represent the way the text from the corpus is utilized to form the word vectors. The parameters of the word2vec are set according to the purpose. Some of the important training parameters are:

- *Sub-sampling*: Sub-sampling is related to the concept of the word frequency and its effect on the linguistic importance of that word. Words with high frequency often provide little information. A threshold can be set to remove such words and improve the computational efficiency.

- *Dimensionality*: Dimensionality represents the size of one-dimensional vectors used to represent the words. It is independent of the size of vocabulary. Generally, the dimensionality is set between 100 to 1000. Higher dimensionality results in higher accuracy, but at a point of saturation, there is no marginal gain in the accuracy.

- *Context window*: The concept of context window is used to capture the semantic information of the word. Context window represents the number of words before and after the given word to be included as the context of the word. According to Mikolov, the recommended value of the context window for word2vec is 10 for the skip-gram model [48].

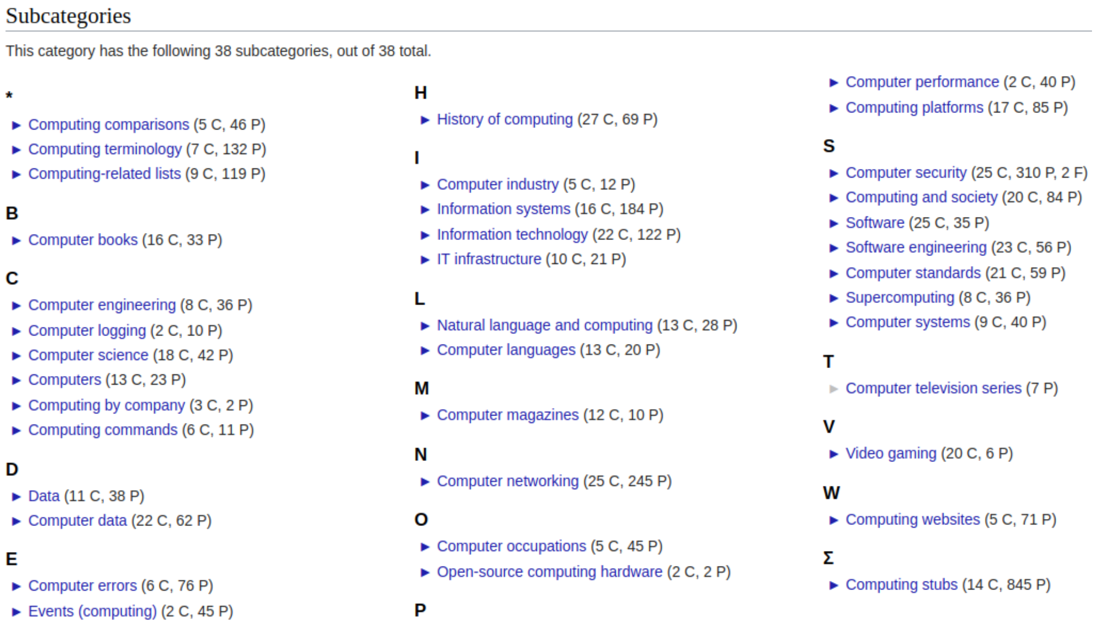
The application domain of this research is to calcuate the semantic similarities between learning objectives from course outlines. In this research, we present the usage of Wikipedia as corpus and the effect of corpus on the word similarities.

3.2 Methodology

The proposed methodology uses word vectors formed using word2vec model's *skip-gram* approach using *hierarchical softmax*. In linguistic modeling, the skip-gram approach is a generalization of n-grams where words need not be consecutive. The methodology can be divided into following subparts:

- Building a domain specific corpus
- Word Similarity

For implementation, we use previously reported training algorithms and architectures [30].



Subcategories

This category has the following 38 subcategories, out of 38 total.

*

- Computing comparisons (5 C, 46 P)
- Computing terminology (7 C, 132 P)
- Computing-related lists (9 C, 119 P)

B

- Computer books (16 C, 33 P)

C

- Computer engineering (8 C, 36 P)
- Computer logging (2 C, 10 P)
- Computer science (18 C, 42 P)
- Computers (13 C, 23 P)
- Computing by company (3 C, 2 P)
- Computing commands (6 C, 11 P)

D

- Data (11 C, 38 P)
- Computer data (22 C, 62 P)

E

- Computer errors (6 C, 76 P)
- Events (computing) (2 C, 45 P)

H

- History of computing (27 C, 69 P)

I

- Computer industry (5 C, 12 P)
- Information systems (16 C, 184 P)
- Information technology (22 C, 122 P)
- IT infrastructure (10 C, 21 P)

L

- Natural language and computing (13 C, 28 P)
- Computer languages (13 C, 20 P)

M

- Computer magazines (12 C, 10 P)

N

- Computer networking (25 C, 245 P)

O

- Computer occupations (5 C, 45 P)
- Open-source computing hardware (2 C, 2 P)

P

- Computer performance (2 C, 40 P)
- Computing platforms (17 C, 85 P)

S

- Computer security (25 C, 310 P, 2 F)
- Computing and society (20 C, 84 P)
- Software (25 C, 35 P)
- Software engineering (23 C, 56 P)
- Computer standards (21 C, 59 P)
- Supercomputing (8 C, 36 P)
- Computer systems (9 C, 40 P)

T

- Computer television series (7 P)

V

- Video gaming (20 C, 6 P)

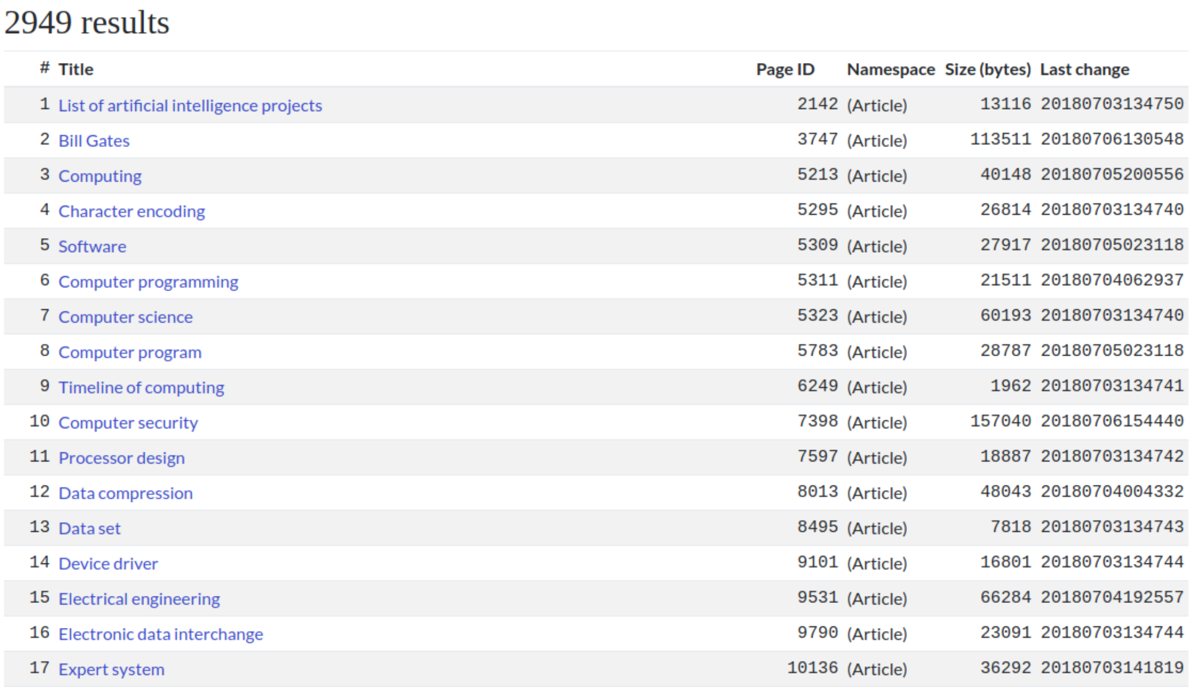
W

- Computing websites (5 C, 71 P)

Σ

- Computing stubs (14 C, 845 P)

Figure 3.1: Subcategories for *computing* domain



2949 results

#	Title	Page ID	Namespace	Size (bytes)	Last change
1	List of artificial intelligence projects	2142	(Article)	13116	20180703134750
2	Bill Gates	3747	(Article)	113511	20180706130548
3	Computing	5213	(Article)	40148	20180705200556
4	Character encoding	5295	(Article)	26814	20180703134740
5	Software	5309	(Article)	27917	20180705023118
6	Computer programming	5311	(Article)	21511	20180704062937
7	Computer science	5323	(Article)	60193	20180703134740
8	Computer program	5783	(Article)	28787	20180705023118
9	Timeline of computing	6249	(Article)	1962	20180703134741
10	Computer security	7398	(Article)	157040	20180706154440
11	Processor design	7597	(Article)	18887	20180703134742
12	Data compression	8013	(Article)	48043	20180704004332
13	Data set	8495	(Article)	7818	20180703134743
14	Device driver	9101	(Article)	16801	20180703134744
15	Electrical engineering	9531	(Article)	66284	20180704192557
16	Electronic data interchange	9790	(Article)	23091	20180703134744
17	Expert system	10136	(Article)	36292	20180703141819

Figure 3.2: Level 1 Subcategories for *computing* domain

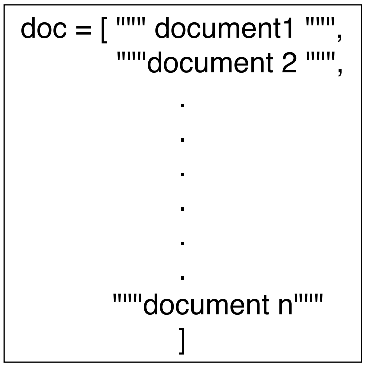
3.2.1 Building a domain specific corpus

Learning objectives from course outline contain peculiar words corresponding to the field. For instance, word ‘Python’, in the domain of computer science, means ‘A programming language’ whereas it could mean ‘A species of reptiles’ in a more general sense. Hence, using a general-purpose corpus does not serve the purpose. So, building a domain-specific corpus and training the model with the corpus is a crucial part of the methodology. We chose Wikipedia as a source for compiling a corpus [65]. For this research, we decided to focus on a particular domain and work on the corpus compiled from Wikipedia. We chose ‘Computing’ as the main sub-category. Wikipedia is divided into multiple sub-categories, and each sub-category can have multiple categories and pages. Figure 3.1 represents the sub-categories for computing domain.

We use the *petscan API* to get the Wikipedia structure of a particular category [64]. The sample output for petscan API is displayed in Figure 3.2. We treat this hierarchy as a tree-traversal problem and use DFS to traverse the tree. The total number of Wikipedia articles collected is 160,624. We use *Wikipedia python API* [20] to retrieve and parse the articles to get the textual content from the article webpage.

We store the corpus as a python file which enables us to compile the corpus to find if there are any non-ascii characters. Filtering such characters is a necessary step before training the model. Every article is stored as a list element in the file for simpler iterations. A code to clean-up the Wikipedia textual data and compiling a corpus is given in Listing A.6.

The structure of the corpus file is displayed in Figure 3.3.



```
doc = [ """ document1 """,
          """document 2 """,
                  .
                  .
                  .
                  .
                  .
                  .
          """document n"""
                  ]
```

Figure 3.3: Structure of corpus

3.2.2 Word Similarity

After creating a corpus of *computing* domain, we train the Word2Vec model with the compiled corpus. We use the *gensim*'s implementation of Word2Vec [57]. Listing A.7 depicts the code snippet to train the word2vec.

We follow the same sentence similarity methodology explained in chapter 1.

3.3 Experimental Results

We tested the performance of the model regarding semantic similarity between words on various corpora. We present the results of semantic similarity between keywords in the computer science domain. During our research, we found that there is no gold standard for the words in academics to compare the results. Hence we rely on the general human knowledge to make sense of the results. We use a corpus with ‘computing’ as a root category from Wikipedia and collect textual data from the pages till level 4. Table 3.1 represents the results from the compiled corpus and *GoogleNews Negatives 300 corpus*. The results for the GoogleNews Negatives 300 corpus are

Table 3.1: Comparison of semantic similairy between words

Word 1	**Word 2**	**Computing corpus similarity**	**GoogleNews Negatives 300 similarity**
java	python	0.9218	0.5626
language	python	0.8074	0.5372
software	people	0.0406	0.5325
computing	semantics	0.7222	0.5986
computing	processing	0.7118	0.6369
software	application	0.8095	0.6955
software	platform	0.7674	0.6777
software	tea	0.4869	0.5476
java	coffee	0.4698	0.8252
database	network	0.7180	0.6710
ethernet	network	0.82355	0.7097
ethernet	protocol	0.8464	0.6478
network	protocol	0.8353	0.6323
drivers	firmware	0.8586	0.5794
compiler	debugger	0.8659	0.8052
windows	linux	0.6868	0.8089
windows	door	0.5977	0.7273
windows	.exe	0.6475	Word *.exe* not in vocabulary
augmented	simulated	0.7742	0.5840
google	baidu	0.8655	Word *baidu* not in vocabulary
parallelism	model	0.6577	0.5384
data	mining	0.6335	0.5623
minerals	mining	0.6943	0.7877
ibm	mainframe	0.8681	0.6312
servers	cloud	0.7686	0.6652
servers	river	0.4744	0.5161
cook	servers	0.3717	0.5619
learner	classifier	0.8595	0.6031
abstract	namespace	0.7671	0.6200

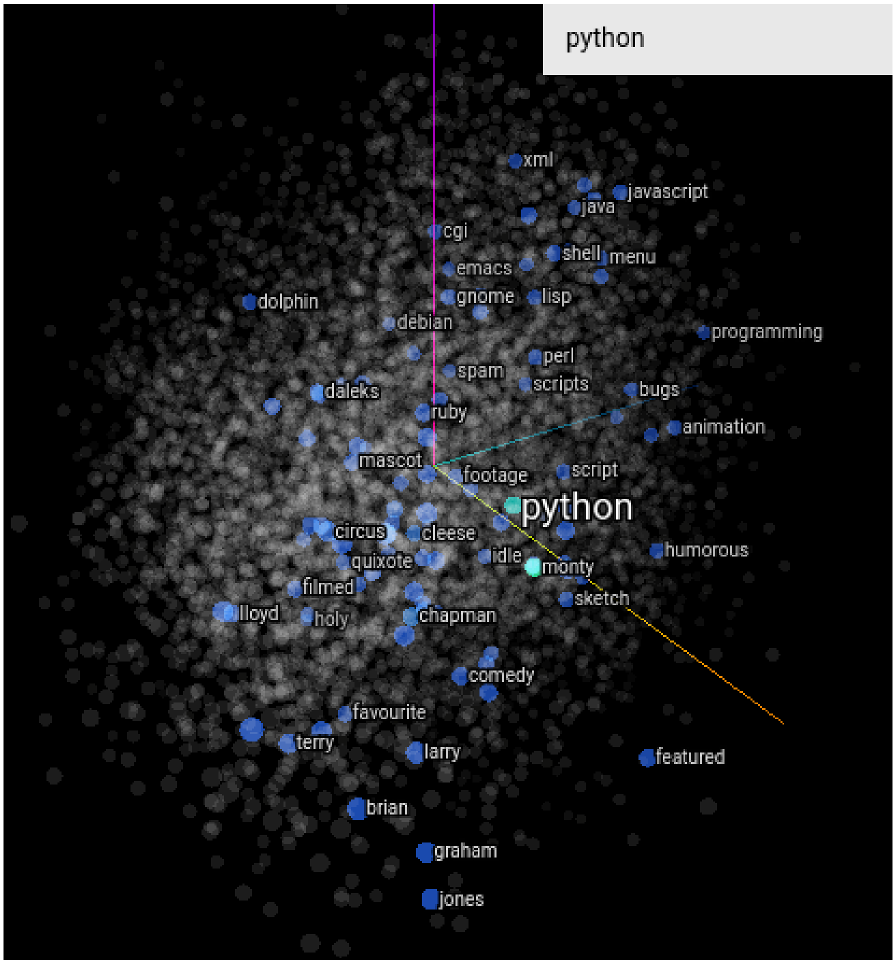
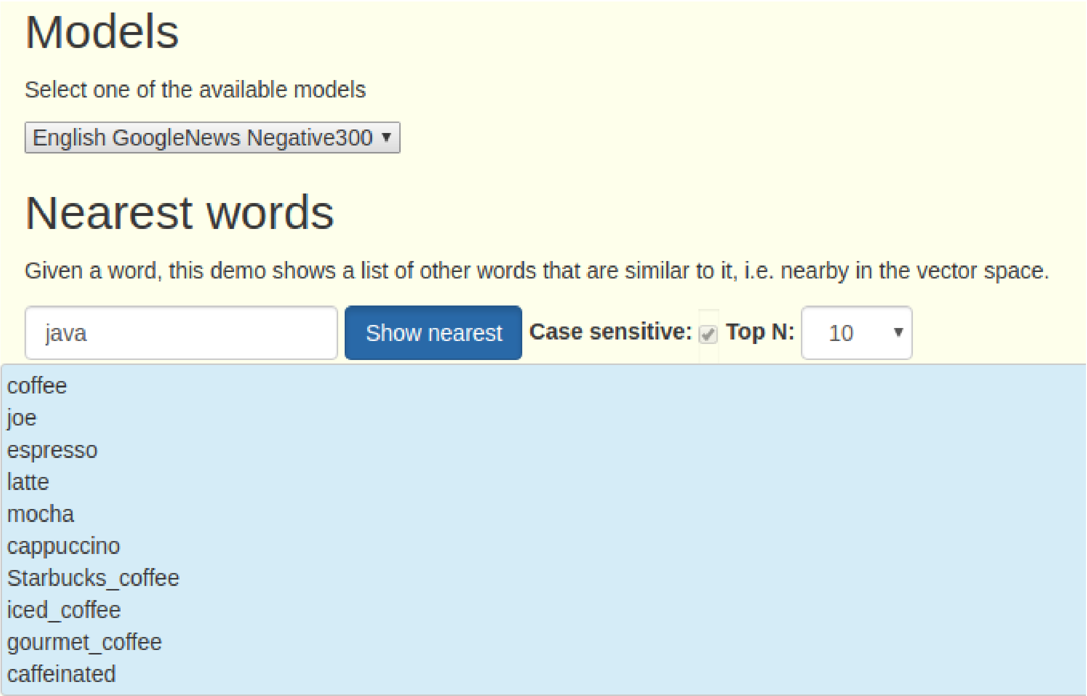


Figure 3.4: Word distribution in a trained word2vec model [26]

taken from the online demo by Turku BioNLP Group [21]. Figure 3.4 shows the word distribution in a word2vec model trained using GoogleNews Negatives 300.

From the Table 3.1, we can see that the corpus affects the similarity significantly. Figure 3.5 shows the top 10 words similar to ‘java’ regarding the English GoogleNews Negative300 corpus. These results would be inaccurate for the computing domain, as ‘java’ is not related to coffee, joe, espresso, etc. in the context of computing. Hence, to get the precise similarity between words under a particular field, we need a domain specific corpus.



Models

Select one of the available models

English GoogleNews Negative300 ▾

Nearest words

Given a word, this demo shows a list of other words that are similar to it, i.e. nearby in the vector space.

java

Case sensitive: ☑ Top N: 10 ▾

```
coffee
joe
espresso
latte
mocha
cappuccino
Starbucks_coffee
iced_coffee
gourmet_coffee
caffeinated
```

Figure 3.5: Top 10 words similar to *java*: English GoogleNews Negative300

3.3.1 Some of the notable differences in similarities for word pairs using general purpose corpus vs. the computing corpus

Concerning the computing domain, java and python both are programming languages and should have a higher similarity value. From Table 3.1, we can see that, for the first two word-pair comparisons, the semantic similarity from the general purpose corpus is less than the corpus for the computing domain. Similarly, for the word pair "computing - processing," the general purpose corpus similarity is 0.6369 whereas similarity is 0.7118 for the computing domain which makes more sense as *computing* and *processing* are closely related to each other in the context of computing. For the words "java - coffee," the general purpose corpus similarity is 0.8252, and the computing corpus similarity is 0.4698. Hence using general purpose corpus for the computing domain would be inaccurate as java and coffee are not closely related to each other in the context of computing. We can observe the similar pattern for all the other listed word pairs. Also, some of the essential words in the computing domain are not found in the general purpose corpus, e.g. '.exe' and 'Baidu'. Hence, from these observations, we can conclude that using a general purpose for the domain-specific

NLP tasks, does serve the intended purpose.

3.4 Computational Requirements

The memory requirement for word2vec depends on the size of the corpus used to train the model. Generally, for a large corpora, it is essential to use a GPU and libraries like Keras [11] or CUDA [52] which utilize GPU capabilities. The other option is to have a large amount of main memory(RAM) so that the trained model can be loaded into main memory. The memory required to train the model depends directly on dimensions of word vectors and vocabulary of the corpus. The *gensim* implementation of word2vec has requirement of O**(#dimensions * #vocabulary)** of memory. The GoogleNews word2vec model is **300 dimensions * 3M vocab** and can be used on a 16GB machine.

3.5 Conclusion & Future Work

In this research, we present a method to compile a domain-specific corpus using Wikipedia. We also present a case that the semantic similarity between words varies significantly regarding the corpus used to train the word2vec model. We also show the difference between similarities of domain-specific words using a general purpose corpus and a domain-specific corpus.

One of the crucial areas for the future work is to determine the ideal properties of the corpus. With the current philosophy, a corpus is considered as well-compiled if it follows the Zipf's law [67][68]. This case is not true as similarities for words are poor regarding the Brown corpus [16] which follows the Zipf's law[1]. Another area of improvement is determining the optimal parameters for the word2vec model.

[1] Interested readers can contact the author for results of the word similarity under the model trained using Brown corpus.

Appendix A

Table A.1: Rubenstein and Goodenough Vs Lee2014 Vs Proposed Algorithm Similarity

R&G No	R&Gpair	R&G Similarity	Lee2014	GoogleNews English Negative300-Word2Vec	Proposed Algorithm Similarity
1	cord smile	0.005	0.01	0.509055	0.0899021679
2	noon string	0.01	0.005	0.510825	0.0440401486
3	rooster voyage	0.01	0.0125	0.53135	0.010051669
4	fruit furnace	0.0125	0.0475	0.536605	0.0720444643
5	autograph shore	0.015	0.005	0.517325	0.0742552483
6	automobile wizard	0.0275	0.02	0.48596	0.0906955651
7	mound stove	0.035	0.005	0.6226	0.0656419906
8	grin implement	0.045	0.005	0.4999	0.0899021679
9	asylum fruit	0.0475	0.005	0.5289	0.0720444643
10	asylum monk	0.0975	0.0375	0.5693	0.0757289762
11	graveyard madhouse	0.105	0.0225	0.64695	0.0607950554
12	boy rooster	0.11	0.0075	0.6424	0.0907164485
13	glass magician	0.11	0.1075	0.51861	0.1782144411
14	cushion jewel	0.1125	0.0525	0.56235	0.2443794293
15	monk slave	0.1425	0.045	0.5957	0.3750880747
16	asylum cemetery	0.1975	0.0375	0.5462	0.1106378337

17	coast forest	0.2125	0.0475	0.6180485	0.1106378337
18	grin lad	0.22	0.0125	0.624005	0.0899021679
19	shore woodland	0.225	0.0825	0.55845	0.3011198804
20	monk oracle	0.2275	0.1125	0.65177	0.2464473057
21	boy sage	0.24	0.0425	0.582975	0.2017739882
22	automobile cushion	0.2425	0.02	0.56675	0.2018466921
23	mound shore	0.2425	0.035	0.56582	0.2018466921
24	lad wizard	0.2475	0.0325	0.665	0.3673305438
25	forest graveyard	0.25	0.065	0.614505	0.2015952767
26	food rooster	0.2725	0.055	0.55915	0.2732326922
27	cemetery woodland	0.295	0.0375	0.69096	0.2015952767
28	shore voyage	0.305	0.02	0.60215	0.4075214431
29	bird woodland	0.31	0.0125	0.670121	0.1651985693
30	coast hill	0.315	0.1	0.58055	0.4103617321
31	furnace implement	0.3425	0.05	0.5117	0.2464473057
32	crane rooster	0.3525	0.02	0.618035	0.2465928735
33	hill woodland	0.37	0.145	0.63675	0.2918421392
34	car journey	0.3875	0.0725	0.549245	0.2730713984
35	cemetery mound	0.4225	0.0575	0.60302	0.0656419906
36	glass jewel	0.445	0.1075	0.5872465	0.3176716099
37	magician oracle	0.455	0.13	0.6261	0.3057403627
38	crane implement	0.5925	0.185	0.51159	0.4486585394
39	brother lad	0.6025	0.1275	0.67975	0.5462290271
40	sage wizard	0.615	0.1525	0.669055	0.3675115617
41	oracle sage	0.6525	0.2825	0.72125	0.5279307332
42	bird cock	0.6575	0.035	0.681	0.5750838807
43	bird crane	0.67	0.1625	0.6514	0.4978503715
44	food fruit	0.6725	0.2425	0.687046	0.6196075053
45	brother monk	0.685	0.045	0.6116	0.2664571358

46	asylum madhouse	0.76	0.215	0.6262695	0.8185286992
47	furnace stove	0.7775	0.3475	0.80415	0.1651985693
48	magician wizard	0.8025	0.355	0.74315	0.9985079423
49	hill mound	0.8225	0.2925	0.7311	0.8148010746
50	cord string	0.8525	0.47	0.59475	0.8148010746
51	glass tumbler	0.8625	0.1375	0.733755	0.8561402541
52	grin smile	0.865	0.485	0.9302	0.9910074537
53	serf slave	0.865	0.4825	0.7249	0.8673305438
54	journey voyage	0.895	0.36	0.8415	0.8185286992
55	autograph signature	0.8975	0.405	0.6566	0.8499457067
56	coast shore	0.9	0.5875	0.75415	0.8179120223
57	forest woodland	0.9125	0.6275	0.82085	0.9780261147
58	implement tool	0.915	0.59	0.60617	0.0822919486
59	cock rooster	0.92	0.8625	0.7393	0.9093502924
60	boy lad	0.955	0.58	0.7943	0.9093502924
61	cushion pillow	0.96	0.5225	0.6258	0.8157293861
62	cemetery graveyard	0.97	0.7725	0.8212403	0.9985079423
63	automobile car	0.98	0.5575	0.791915	0.8185286992
64	gem jewel	0.985	0.955	0.8105	0.8175091596
65	midday noon	0.985	0.6525	0.77637	0.9993931059

Table A.2: Proposed Algorithm Similarity Vs Islam2008 Vs Li2006

R&G No	R&G pair	Proposed Algorithm Similarity	A.Islam2008	Lietal.2006
1	cord smile	0.0899021679	0.06	0.33
5	autograph shore	0.0742552483	0.11	0.29
9	asylum fruit	0.0720444643	0.07	0.21
12	boy rooster	0.0907164485	0.16	0.53
17	coast forest	0.1106378337	0.26	0.36
21	boy sage	0.2017739882	0.16	0.51

25	forest graveyard	0.2015952767	0.33	0.55
29	bird woodland	0.1651985693	0.12	0.33
33	hill woodland	0.2918421392	0.29	0.59
37	magician oracle	0.3057403627	0.2	0.44
41	oracle sage	0.5279307332	0.09	0.43
47	furnace stove	0.1651985693	0.3	0.72
48	magician wizard	0.9985079423	0.34	0.65
49	hill mound	0.8148010746	0.15	0.74
50	cord string	0.8148010746	0.49	0.68
51	glass tumbler	0.8561402541	0.28	0.65
52	grin smile	0.9910074537	0.32	0.49
53	serf slave	0.8673305438	0.44	0.39
54	journey voyage	0.8185286992	0.41	0.52
55	autograph signature	0.8499457067	0.19	0.55
56	coast shore	0.8179120223	0.47	0.76
57	forest woodland	0.9780261147	0.26	0.7
58	implement tool	0.0822919486	0.51	0.75
59	cock rooster	0.9093502924	0.94	1
60	boy lad	0.9093502924	0.6	0.66
61	cushion pillow	0.8157293861	0.29	0.66
62	cemetery graveyard	0.9985079423	0.51	0.73
63	automobile car	0.8185286992	0.52	0.64
64	gem jewel	0.8175091596	0.65	0.83
65	midday noon	0.9993931059	0.93	1

Table A.3: Sentence Similarity from proposed methodology compared with human mean similarity from Li2006

R&G number	Sentence 1	Sentence 2	Mean Human Similarity	Algorithm Sentence Similarity

1	Cord is strong, thick string.	A smile is the expression that you have on your face when you are pleased or amused, or when you are being friendly.	0.01	0.0125
2	A rooster is an adult male chicken.	A voyage is a long journey on a ship or in a spacecraft.	0.005	0.1593
3	Noon is 12 o'clock in the middle of the day.	String is thin rope made of twisted threads, used for tying things together or tying up parcels.	0.0125	0.03455
4	Fruit or a fruit is something which grows on a tree or bush and which contains seeds or a stone covered by a substance that you can eat.	A furnace is a container or enclosed space in which a very hot fire is made, for example to melt metal, burn rubbish or produce steam.	0.0475	0.1388
5	An autograph is the signature of someone famous which is specially written for a fan to keep.	The shores or shore of a sea, lake, or wide river is the land along the edge of it.	0.0050	0.0701
6	An automobile is a car.	In legends and fairy stories, a wizard is a man who has magic powers.	0.0200	0.0088
7	A mound of something is a large rounded pile of it.	A stove is a piece of equipment which provides heat, either for cooking or for heating a room.	0.0050	0.3968
8	A grin is a broad smile.	An implement is a tool or other pieces of equipment.	0.0050	0.0099
9	An asylum is a psychiatric hospital.	Fruit or a fruit is something which grows on a tree or bush and which contains seeds or a stone covered by a substance that you can eat.	0.0050	0.01456
10	An asylum is a psychiatric hospital.	A monk is a member of a male religious community that is usually separated from the outside world.	0.0375	0.0175

11	A graveyard is an area of land, sometimes near a church, where dead people are buried.	If you describe a place or situation as a madhouse,you mean that it is full of confusion and noise.	0.0225	0.1339
12	Glass is a hard transparent substance that is used to make things such as windows and bottles.	A magician is a person who entertains people by doing magic tricks.	0.0075	0.0911
13	A boy is a child who will grow up to be a man.	A rooster is an adult male chicken.	0.1075	0.2921
14	A cushion is a fabric case filled with soft material, which you put on a seat to make it more comfortable.	A jewel is a precious stone used to decorate valuable things that you wear, such as rings or necklaces.	0.0525	0.1745
15	A monk is a member of a male religious community that is usually separated from the outside world.	A slave is someone who is the property of another person and has to work for that person.	0.0450	0.1394
16	An asylum is a psychiatric hospital.	A cemetery is a place where dead peoples bodies or their ashes are buried.	0.375	0.03398
17	The coast is an area of land that is next to the sea.	A forest is a large area where trees grow close together.	0.0475	0.3658
18	A grin is a broad smile.	A lad is a young man or boy.	0.0125	0.0281
19	The shores or shore of a sea, lake, or wide river is the land along the edge of it.	Woodland is land with a lot of trees.	0.0825	0.2192
20	A monk is a member of a male religious community that is usually separated from the outside world.	In ancient times, an oracle was a priest or priestess who made statements about future events or about the truth.	0.1125	0.1011
21	A boy is a child who will grow up to be a man.	A sage is a person who is regarded as being very wise.	0.0425	0.2305

22	An automobile is a car.	A cushion is a fabric case filled with soft material, which you put on a seat to make it more comfortable.	0.0200	0.0330
23	A mound of something is a large rounded pile of it.	The shores or shore of a sea, lake, or wide river is the land along the edge of it.	0.0350	0.0386
24	A lad is a young man or boy.	In legends and fairy stories, a wizard is a man who has magic powers.	0.0325	0.2939
25	A forest is a large area where trees grow close together.	A graveyard is an area of land, sometimes near a church, where dead people are buried.	0.0650	0.2787
26	Food is what people and animals eat.	A rooster is an adult male chicken.	0.0550	0.2972
27	A cemetery is a place where dead peoples bodies or their ashes are buried.	Woodland is land with a lot of trees.	0.0375	0.1240
28	The shores or shore of a sea, lake, or wide river is the land along the edge of it.	A voyage is a long journey on a ship or in a spacecraft.	0.0200	0.0304
29	A bird is a creature with feathers and wings, females lay eggs, and most birds can fly.	Woodland is land with a lot of trees.	0.0125	0.1334
30	The coast is an area of land that is next to the sea.	A hill is an area of land that is higher than the land that surrounds it.	0.1000	0.8032
31	A furnace is a container or enclosed space in which a very hot fire is made, for example to melt metal, burn rubbish or produce steam.	An implement is a tool or other piece of equipment.	0.0500	0.1408

32	A crane is a large machine that moves heavy things by lifting them in the air.	A rooster is an adult male chicken.	0.0200	0.0564
33	A hill is an area of land that is higher than the land that surrounds it.	Woodland is land with a lot of trees.	0.1450	0.6619
34	A car is a motor vehicle with room for a small number of passengers.	When you make a journey, you travel from one place to another.	0.0725	0.02610
35	A cemetery is a place where dead peoples bodies or their ashes are buried.	A mound of something is a large rounded pile of it.	0.0575	0.0842
36	Glass is a hard transparent substance that is used to make things such as windows and bottles.	A jewel is a precious stone used to decorate valuable things that you wear, such as rings or necklaces.	0.1075	0.2692
37	A magician is a person who entertains people by doing magic tricks.	In ancient times, an oracle was a priest or priestess who made statements about future events or about the truth.	0.1300	0.1000
38	A crane is a large machine that moves heavy things by lifting them in the air.	An implement is a tool or other piece of equipment.	0.1850	0.1060
39	Your brother is a boy or a man who has the same parents as you.	A lad is a young man or boy.	0.1275	0.8615
40	A sage is a person who is regarded as being very wise.	In legends and fairy stories, a wizard is a man who has magic powers.	0.1525	0.1920
41	In ancient times, an oracle was a priest or priestess who made statements about future events or about the truth.	A sage is a person who is regarded as being very wise.	0.2825	0.0452
42	A bird is a creature with feathers and wings, females lay eggs, and most birds can fly.	A crane is a large machine that moves heavy things by lifting them in the air.	0.0350	0.1660

43	A bird is a creature with feathers and wings, females lay eggs, and most birds can fly.	A cock is an adult male chicken.	0.1625	0.1704
44	Food is what people and animals eat.	Fruit or a fruit is something which grows on a tree or bush and which contains seeds or a stone covered by a substance that you can eat.	0.2425	0.1379
45	Your brother is a boy or a man who has the same parents as you.	A monk is a member of a male religious community that is usually separated from the outside world.	0.0450	0.2780
46	An asylum is a psychiatric hospital.	If you describe a place or situation as a madhouse, you mean that it is full of confusion and noise.	0.2150	0.1860
47	A furnace is a container or enclosed space in which a very hot fire is made, for example, to melt metal, burn rubbish, or produce steam.	A stove is a piece of equipment which provides heat, either for cooking or for heating a room.	0.3475	0.1613
48	A magician is a person who entertains people by doing magic tricks.	In legends and fairy stories, a wizard is a man who has magic powers.	0.3550	0.5399
49	A hill is an area of land that is higher than the land that surrounds it.	A mound of something is a large rounded pile of it.	0.2925	0.2986
50	Cord is strong, thick string.	String is thin rope made of twisted threads, used for tying things together or tying up parcels.	0.4700	0.2530
51	Glass is a hard transparent substance that is used to make things such as windows and bottles.	A tumbler is a drinking glass with straight sides.	0.1375	0.2643

52	A grin is a broad smile.	A smile is the expression that you have on your face when you are pleased or amused, or when you are being friendly.	0.4850	0.7204
53	In former times, serfs were a class of people who had to work on a particular persons land and could not leave without that persons permission.	A slave is someone who is the property of another person and has to work for that person.	0.4825	0.7695
54	When you make a journey, you travel from one place to another.	A voyage is a long journey on a ship or in a spacecraft.	0.3600	0.7201
55	An autograph is the signature of someone famous which is specially written for a fan to keep.	Your signature is your name, written in your own characteristic way, often at the end of a document to indicate that you wrote the document or that you agree with what it says.	0.4050	0.3146
56	The coast is an area of land that is next to the sea.	The shores or shore of a sea, lake, or wide river is the land along the edge of it.	0.5875	0.7945
57	A forest is a large area where trees grow close together.	Woodland is land with a lot of trees.	0.6275	0.4770
58	An implement is a tool or other pieces of equipment.	A tool is any instrument or simple piece of equipment that you hold in your hands and use to do a particular kind of work.	0.5900	0.7590
59	A cock is an adult male chicken.	A rooster is an adult male chicken.	0.8625	0.8560
60	A boy is a child who will grow up to be a man.	A lad is a young man or boy.	0.5800	0.8296
61	A cushion is a fabric case filled with soft material, which you put on a seat to make it more comfortable.	A pillow is a rectangular cushion which you rest your head on when you are in bed.	0.5225	0.7626

62	A cemetery is a place where dead peoples bodies or their ashes are buried.	A graveyard is an area of land, sometimes near a church, where dead people are buried.	0.7725	0.8750
63	An automobile is a car.	A car is a motor vehicle with room for a small number of passengers.	0.5575	0.7001
64	Midday is 12 oclock in the middle of the day.	Noon is 12 oclock in the middle of the day.	0.9550	0.8726
65	A gem is a jewel or stone that is used in jewellery.	A jewel is a precious stone used to decorate valuable things that you wear, such as rings or necklaces.	0.6525	0.5477

Table A.4: Sentence Similarity from proposed methodology compared with SICK similarity

SICK index	Sentence 1	Sentence 2	SICK similarity	Algorithm similarity
6	There is no boy playing outdoors and there is no man smiling	A group of kids is playing in a yard and an old man is standing in the background	0.575	0.5054
7	A group of boys in a yard is playing and a man is standing in the background	The young boys are playing outdoors and the man is smiling nearby	0.675	0.6689
8	A group of children is playing in the house and there is no man standing in the background	The young boys are playing outdoors and the man is smiling nearby	0.5	0.5054
10	A brown dog is attacking another animal in front of the tall man in pants	A brown dog is attacking another animal in front of the man in pants	0.975	0.8892
11	A brown dog is attacking another animal in front of the man in pants	A brown dog is helping another animal in front of the man in pants	0.66625	0.5966

13	Two dogs are wrestling and hugging	There is no dog wrestling and hugging	0.575	0.6306
15	A brown dog is attacking another animal in front of the man in pants	There is no dog wrestling and hugging	0.425	0.4920
16	Two dogs are wrestling and hugging	A brown dog is attacking another animal in front of the tall man in pants	0.475	0.4950
17	Two dogs are wrestling and hugging	A brown dog is helping another animal in front of the man in pants	0.325	0.5034
19	A person in a black jacket is doing tricks on a motorbike	A man in a black jacket is doing tricks on a motorbike	0.975	0.95

Listing A.1: Code snippet for shortest path distance

```
def length_between_synsets(synset_1, synset_2):
    set_1 = set()
    set_2 = set()
    length = sys.maxint
    if synset_1 is None or synset_2 is None:
        return length
    if synset_1 == synset_2:
        length = 0.0
    else:
        # "check for common elements in set"
        for x in synset_1.lemmas():
            set_1.add(str(x.name()))
        for y in synset_2.lemmas():
            set_2.add(str(y.name()))
        if len(set_1.intersubsection(set_2)) > 0:
            length = 1
        else:
            length = synset_1.shortest_path_distance(synset_2)
            if length is None:
                length = 0.0
    return math.exp(-ALPHA * length)
```

Listing A.2: Code snippet for hierarchical distance

```
def hierarchical_distance(synset_1, synset_2):
    distance = sys.maxint
    hypernyms_dict_1 = {}
    hypernyms_dict_2 = {}
    if synset_1 is None or synset_2 is None:
        return distance
    if synset_1 == synset_2:
        distance = max(x[1] for x in synset_1.hypernym_distances())

    else:
        for x in synset_1.hypernym_distances():
            hypernyms_dict_1[x[0]] = x[1]
        for y in synset_2.hypernym_distances():
            hypernyms_dict_2[y[0]] = y[1]
        least_common_subsumers =
            set(hypernyms_dict_1.keys()).intersubsection(set(hypernyms_dict_2.keys()))

        if len(least_common_subsumers) > 0:
            lcs_distances = []
            for least_common_subsumer in least_common_subsumers:
                d1 = 0
                if hypernyms_dict_1.has_key(least_common_subsumer):
                    d1 = hypernyms_dict_1[least_common_subsumer]

                d2 = 0
                if hypernyms_dict_2.has_key(least_common_subsumer):
                    d2 = hypernyms_dict_2[least_common_subsumer]

                lcs_distances.append(max([d1, d2]))

            distance = max(lcs_distances)

        else:
```

```
        distance = 0
    return (math.exp(BETA * distance) - math.exp(-BETA * distance)) / (
       math.exp(BETA * distance) + math.exp(-BETA * distance))
```

Listing A.3: Synset with maximum frequency of occurence

```
def return_max_frequency_synset(word,ps):
    i = 0
    old_frequency = 0
    dict = {}
    return_syn=None
    syns = wordnet.synsets(word,pos=ps)
    for s in syns:
        for l in s.lemmas():
            new_frequency=str(l.count())
            if int(new_frequency)>=int(old_frequency):
                return_syn=syns[i]

        i+=1
    return return_syn
```

Listing A.4: Formation of sematic vectors for sentences

```

def form_value_vector(d1, d2):
    # form a semantic vector for sentences
    #d1 and d2 are the list of tagged words for s1 and s2 respectively

    global similarity_values
    global data
    global div_index
    a = len(d1)
    b = len(d2)
    length = max(a, b)
    div_index = min(a, b)
    avg = 0
    i = 0
    j = 0
```

```
semantic_vector = np.zeros(length)
semantic_vector.fill(0)
disambiguate_similarity=0.0
max_frequency_similarity=0.0
for w1 in d1:
    previous_avg=0
    for w2 in d2:
  #tuple[0][1] is disambiguated tuple and tuple[1] is maximum frequency
      tuple

        if ".n." in str(w1[0][1]) and ".n." in str(w2[0][1]):
            disambiguate_similarity = word_similarity(w1[0][1], w2[0][1])

            max_frequency_similarity = word_similarity(w1[1], w2[1])
        if ".v." in str(w1[0][1]) and ".v." in str(w2[0][1]):
            disambiguate_similarity = word_similarity(w1[0][1], w2[0][1])
            max_frequency_similarity = word_similarity(w1[1], w2[1])

        key = w1[0][0], w2[0][0]
        previous_key = ' '.join(key)

        if disambiguate_similarity == 0.0:
            previous_avg = max_frequency_similarity
        elif max_frequency_similarity == 0.0:
            previous_avg = disambiguate_similarity

        else:
            previous_avg = (disambiguate_similarity +
                max_frequency_similarity) / 2.0


        data[previous_key] = previous_avg
        similarity_values = json.dumps(data)

        if avg < previous_avg:
            avg = previous_avg
    semantic_vector[i] = avg
```

```
        i = i + 1
        avg = 0
        key = ""
    return semantic_vector
```

Listing A.5: Bloom's Index

```
def stanford_pos_tagger(statement):
    home = '/home/atish'
    from nltk.tag.stanford import StanfordPOSTagger as POS_Tag
    _path_to_model = home +
        '/stanford-postagger/models/english-bidirectional-distsim.tagger'
    _path_to_jar = home + '/stanford-postagger/stanford-postagger.jar'
    st = POS_Tag(model_filename=_path_to_model, path_to_jar=_path_to_jar)
    return st.tag(statement.split())

def bloom_lookup(v): #returns the numerical value of the domain to which
    the verb belongs
    #Action words for bloom taxonomy
    knowledge = ["select", "list", "name", "define", "describe",
        "memorize", "label", "identify", "locate", "recite","state",
        "recognize"]
    comprehension = ["discuss","match", "restate", "paraphrase",
        "rewrite", "give examples", "express", "illustrate",
        "explain","defend", "distinguish", "summarize", "interrelate",
        "interpret", "extend"]
    application = ["use","write","organize", "generalize", "dramatize",
        "prepare", "produce", "choose", "sketch", "apply", "solve", "draw",
        "show", "paint"]
    analysis = ["compare", "analyze","analyse", "classify", "point out",
        "distinguish", "categorize", "differentiate","subdivide", "infer",
        "survey", "select", "prioritize"]
    synthesis = ["compose", "originate", "hypothesize", "develop",
        "design", "combine", "construct", "plan", "create","invent",
        "organize"]
    evaluation = ["judge", "relate", "weight", "criticize", "support",
        "evaluate", "consider", "critique", "recommend","summarize",
        "appraise", "compare","predict"]
    if v in knowledge:
        return 1
    if v in comprehension:
        return 2
    if v in application:
        return 3
    if v in analysis:
        return 4
    if v in synthesis:
        return 5
    if v in evaluation:
        return 6


def bloom_verbs(list_of_tagged_tokens): #returns verbs
    list_of_verbs={}
    for token in list_of_tagged_tokens:
        #print token
        pos= str(token).split(’,’)[1]
        pos= pos.split("’")[1]
        #print pos
        if pos=="VB": #if it is a verb, then look it up in bloom taxonomy
            #print token[0]
            action_verb=str(token[0]).lower()
            list_of_verbs[action_verb]=bloom_lookup(action_verb)
            #bt=bloom_lookup(token[0].lower())
    #print list_of_verbs
    return list_of_verbs

def bloom_index(action_verbs_list_1,action_verbs_list_2):
    comparisons=0
    total_bloom_similarity=0
    for verb1,value1 in action_verbs_list_1.iteritems():
        for verb2,value2 in action_verbs_list_2.iteritems():
            #print verb1, value1
            #print verb2,value2
            distance=abs(value1-value2)
            print "Distance:",distance
            absolue_bloom_index= -0.20*distance+1.0
            print "AB",absolue_bloom_index
            total_bloom_similarity=total_bloom_similarity+absolue_bloom_index
            print total_bloom_similarity
            comparisons=comparisons+1
        print "\n"
    print "comparisons=",comparisons
    print "Bloom index=", (total_bloom_similarity / comparisons)

d1=stanford_pos_tagger("apply chemical knowledge to integrate knowledge
    gained in other courses and to better understand the connections
    between the various branche")
action_verbs_list_1=bloom_verbs(d1)
#print "\n"
d2=stanford_pos_tagger("To become familiar with the structures of organic
    molecules, especially those found in nature or those with important
    biological effects.")
action_verbs_list_2=bloom_verbs(d2)
bloom_index(action_verbs_list_1,action_verbs_list_2)
```

Listing A.6: Corpus compilation

```
import wikipedia
count=0
count2=0
#read names of Wikipedia pages from collected from petscan API
petscan = open("wiki_data_petscan_level4.py", "a")
petscan.write("doc=[")
with open(’level4tsv’) as f:
    for line in f:
        count2=count2+1
        #page = line.split(",")[1].strip("’").strip(’"’)
        title = line.split(" ")[0].split("\t")[1]

        if petscan.closed:
            petscan = open("wiki_data_petscan_level4.py", "a")

      #remove ascii characters from the text so that corpus can be compied
          using python
        try:
            cs = wikipedia.page(title)
            petscan.write('"""')
            all_ascii = ''.join(char for char in cs.content if ord(char) <
                128)
            if all_ascii.endswith('"'):
                all_ascii = ''.join(char for char in cs.content if ord(char)
                    < 128)[:-1]
            if all_ascii.startswith('"'):
                all_ascii = ''.join(char for char in cs.content if ord(char)
                    < 128)[1:]
            # doc.append(all_ascii)
            petscan.write(all_ascii)
            petscan.write('""",')
            count = count + 1
        except:
            pass
        #flush the IO streams after dumping textual data from 5 pages
        if count % 5 == 0:
            petscan.flush()
            petscan.close()
            print "Dumped:",count2


if petscan.closed:
    petscan = open("wiki_data_petscan_level4.py", "a")
    petscan.write("]")
```

Listing A.7: Training word2vec using compiled corpus

```
from gensim.models import Word2Vec
from wikiCorpus import doc
```

```
stoplist = set('for a of the and to in'.split())
texts = [[word for word in document.lower().split() if word not in
    stoplist]for document in doc]
# remove words that appear only once
from collections import defaultdict
frequency = defaultdict(int)
for text in texts:
    for token in text:
        frequency[token] += 1
texts = [[token for token in text if frequency[token] > 1]
        for text in texts]
model = Word2Vec(texts,size=300,window=10,min_count=2, workers=4)
model.save("CompModel")
#print model.similarity('language','python')
#model.wv.save_word2vec_format('CompModel_format_level4')
```

Bibliography

[1] thesaurus.com api. `https://github.com/Manwholikespie/thesaurus-api`.

[2] Wikipedia english language contractions. `https://en.wikipedia.org/wiki/Wikipedia:ListofEnglishcontractions`.

[3] Vibhanshu Abhishek and Kartik Hosanagar. Keyword generation for search engine advertising using semantic similarity between terms. In *Proceedings of the ninth international conference on Electronic commerce*, pages 89–94. ACM, 2007.

[4] Gina M Almerico and Russell K Baker. Blooms taxonomy illustrative verbs: Developing a comprehensive list for educator use. *Florida Association of Teacher Educators Journal*, 1(4):1–10, 2004.

[5] Lorin W Anderson, David R Krathwohl, P Airasian, K Cruikshank, R Mayer, P Pintrich, James Raths, and M Wittrock. A taxonomy for learning, teaching and assessing: A revision of blooms taxonomy. *New York. Longman Publishing. Artz, AF, & Armour-Thomas, E.(1992). Development of a cognitive-metacognitive framework for protocol analysis of mathematical problem solving in small groups. Cognition and Instruction*, 9(2):137–175, 2001.

[6] Alan D Baddeley. Short-term memory for word sequences as a function of acoustic, semantic and formal similarity. *The Quarterly Journal of Experimental Psychology*, 18(4):362–365, 1966.

[7] John B Biggs and Kevin F Collis. *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. Academic Press, 2014.

[8] Steven Bird. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72. Association for Computational Linguistics, 2006.

[9] Johannes Bjerva, Johan Bos, Rob Van der Goot, and Malvina Nissim. The meaning factory: Formal semantics for recognizing textual entailment and determining semantic similarity. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 642–646, 2014.

[10] Danushka Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. Measuring semantic similarity between words using web search engines. *16th international conference on World Wide Web*, 7:757–766, 2007.

[11] François Chollet. Keras, 2015.

[12] Günes Erkan and Dragomir R Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, 22:457–479, 2004.

[13] Christiane Fellbaum. *WordNet*. Wiley Online Library, 1998.

[14] John R Firth. A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*, 1957.

[15] Mary Forehand. Blooms taxonomy. *Emerging perspectives on learning, teaching, and technology*, 41(4):47–56, 2010.

[16] W Nelson Francis and Henry Kucera. Brown corpus. *Department of Linguistics, Brown University, Providence, Rhode Island*, 1, 1964.

[17] André Freitas, João Oliveira, Seán ORiain, Edward Curry, and João Pereira da Silva. Querying linked data using semantic relatedness: a vocabulary independent approach. *Natural Language Processing and Information Systems*, 8:40–51, 2011.

[18] Philippe J Giabbanelli, Thomas Torsney-Weir, and Vijay Kumar Mago. A fuzzy cognitive map of the psychosocial determinants of obesity. *Applied soft computing*, 12(12):3711–3724, 2012.

[19] Lori Goff, Michael K Potter, Eleanor Pierre, Thomas Carey, Amy Gullage, Erika Kustra, Rebecca Lee, Valerie Lopes, Leslie Marshall, Lynn Martin, et al. Learning outcomes assessment a practitioner's handbook. 2015.

[20] Jonathan Goldsmith. Wikipedia api. https://pypi.org/project/wikipedia/.

[21] Turku NLP Group. Word embeddings induced using the word2vec method. http://bionlp-www.utu.fi/wv_demo/.

[22] Hua He, Kevin Gimpel, and Jimmy Lin. Multi-perspective sentence similarity modeling with convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1576–1586, 2015.

[23] Zhen He, Shaobing Gao, Liang Xiao, Daxue Liu, Hangen He, and David Barber. Wider and deeper, cheaper and faster: Tensorized lstms for sequence learning. In *Advances in Neural Information Processing Systems*, pages 1–11, 2017.

[24] M Honnibal. Spacy (version 1.3. 0), 2016.

[25] Verbs Cognitive Level Illustrative. Blooms taxonomy of measurable verbs. *Center for Teaching & Learning— The University of Georgia— ctl. uga. edu*, 706:1355.

[26] Google Inc. Word distribution. https://projector.tensorflow.org/.

[27] Aminul Islam and Diana Inkpen. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2(2):10–36, 2008.

[28] Jay J Jiang and David W Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. *arXiv preprint cmp-lg/9709008*, 1997.

[29] Sergio Jimenez, George Duenas, Julia Baquero, and Alexander Gelbukh. Unalnlp: Combining soft cardinality features for semantic textual similarity, relatedness and entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 732–742, 2014.

[30] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[31] Youngjoong Ko, Jinwoo Park, and Jungyun Seo. Improving text categorization using the importance of sentences. *Information processing & management*, 40(1):65–79, 2004.

[32] David R Krathwohl. A revision of bloom's taxonomy: An overview. *Theory into practice*, 41(4):212–218, 2002.

[33] Frankie Santos Laanan. Transfer student adjustment. *New directions for community colleges*, 2001(114):5–13, 2001.

[34] Alice Lai and Julia Hockenmaier. Illinois-lh: A denotational and distributional approach to semantics. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 329–334, 2014.

[35] Ming Che Lee, Jia Wei Chang, and Tung Cheng Hsieh. A grammar-based semantic similarity algorithm for natural language sentences. *The Scientific World Journal*, 2014, 2014.

[36] Yuhua Li, David McLean, Zuhair A Bandar, James D O'shea, and Keeley Crockett. Sentence similarity based on semantic nets and corpus statistics. *IEEE transactions on knowledge and data engineering*, 18(8):1138–1150, 2006.

[37] Dekang Lin et al. An information-theoretic definition of similarity. In *Icml*, volume 98, pages 296–304, 1998.

[38] Phillip W. Lord, Robert D. Stevens, Andy Brass, and Carole A. Goble. Investigating semantic similarity measures across the gene ontology: the relationship between sequence and annotation. *Bioinformatics*, 19(10):1275–1283, 2003.

[39] Vijay K Mago, Laurens Bakker, Elpiniki I Papageorgiou, Azadeh Alimadad, Peter Borwein, and Vahid Dabbaghian. Fuzzy cognitive maps and cellular automata: An evolutionary approach for social systems modelling. *Applied Soft Computing*, 12(12):3771–3784, 2012.

[40] Vijay K Mago, Hilary K Morden, Charles Fritz, Tiankuang Wu, Sara Namazi, Parastoo Geranmayeh, Rakhi Chattopadhyay, and Vahid Dabbaghian. Analyzing the impact of social factors on homelessness: a fuzzy cognitive map approach. *BMC medical informatics and decision making*, 13(1):94, 2013.

[41] Vijay Kumar Mago, Anjali Mago, Poonam Sharma, and Jagmohan Mago. Fuzzy logic based expert system for the treatment of mobile tooth. In *Software Tools and Algorithms for Biological Systems*, pages 607–614. Springer, 2011.

[42] Vijay Kumar Mago, Bhanu Prasad, Ajay Bhatia, and Anjali Mago. A decision making system for the treatment of dental caries. In *Soft computing applications in business*, pages 231–242. Springer, 2008.

[43] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[44] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[45] Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, Roberto Zamparelli, et al. A sick cure for the evaluation of compositional distributional semantic models. In *LREC*, pages 216–223, 2014.

[46] Yutaka Matsuo and Mitsuru Ishizuka. Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13(01):157–169, 2004.

[47] Charles T Meadow. *Text information retrieval systems.* Academic Press, Inc., 1992.

[48] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[49] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[50] George A Miller and Walter G Charles. Contextual correlates of semantic similarity. *Language and cognitive processes*, 6(1):1–28, 1991.

[51] Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. In *AAAI*, pages 2786–2792, 2016.

[52] CUDA Nvidia. Programming guide, 2010.

[53] James O'Shea, Zuhair Bandar, Keeley Crockett, and David McLean. Pilot short text semantic similarity benchmark data set: Full listing and description. *Computing*, 2008.

[54] Ted Pedersen, Satanjeev Banerjee, and Siddharth Patwardhan. Maximizing semantic relatedness to perform word sense disambiguation. *University of Minnesota supercomputing institute research report UMSI*, 25:2005, 2005.

[55] Ted Pedersen, Serguei VS Pakhomov, Siddharth Patwardhan, and Christopher G Chute. Measures of semantic similarity and relatedness in the biomedical domain. *Journal of biomedical informatics*, 40(3):288–299, 2007.

[56] Catia Pesquita, Daniel Faria, Andre O Falcao, Phillip Lord, and Francisco M Couto. Semantic similarity in biomedical ontologies. *PLoS computational biology*, 5(7):e1000443, 2009.

[57] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. pages 45–50, May 2010.

[58] Philip Resnik et al. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *J. Artif. Intell. Res.(JAIR)*, 11:95–130, 1999.

[59] Herbert Rubenstein and John B Goodenough. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633, 1965.

[60] John M Sinclair. *Looking up: An account of the COBUILD project in lexical computing and the development of the Collins COBUILD English language dictionary.* Collins Elt, 1987.

[61] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[62] Liling Tan. Pywsd: Python implementations of word sense disambiguation (wsd) technologies [software]. https://github.com/alvations/pywsd.

[63] Giannis Varelas, Epimenidis Voutsakis, Paraskevi Raftopoulou, Euripides GM Petrakis, and Evangelos E Milios. Semantic similarity methods in wordnet and their application to information retrieval on the web. In *Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 10–16. ACM, 2005.

[64] wikimedia. Petscan api. https://petscan.wmflabs.org/.

[65] Torsten Zesch, Iryna Gurevych, and Max Mühlhäuser. Analyzing and accessing wikipedia as a lexical semantic resource. *Data Structures for Linguistic Resources and Applications*, pages 197–205, 2007.

[66] Jiang Zhao, Tiantian Zhu, and Man Lan. Ecnu: One stone two birds: Ensemble of heterogenous measures for semantic relatedness and textual entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 271–277, 2014.

[67] George Kingsley Zipf. Selected studies of the principle of relative frequency in language. 1932.

[68] George Kingsley Zipf. The meaning-frequency relationship of words. *The Journal of General Psychology*, 33(2):251–256, 1945.