Department of Computer Systems and Computation
Polytechnic University of Valencia

Text similarity by using GloVe word vector representations

MASTER'S THESIS

Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

Author: Iván Sánchez Rodríguez

Tutor: Encarna Segarra Soriano
Lluís Felip Hurtado Oliver

Course 2016-2017

Resumen

Los *word embeddings* son representaciones de palabras en forma de vector que permiten mantener cierta información semántica de estas. Existen diferentes maneras de aprovechar la información semántica que las palabras tienen, así como también existen diferentes maneras de generar los vectores de palabras que representan dichas palabras (por ejemplo, los modelos Word2Vec frente al modelo GloVe). Si se usa la información que los *word embeddings* capturan, se pueden construir aproximaciones para comparar información semántica entre frases o incluso documentos, en lugar de palabras. En este proyecto, proponemos el uso de la herramienta GloVe, presentada por la Universidad de Stanford, para entrenar representaciones vectoriales de palabras en español, así como su uso para comparar diferencias semánticas entre frases en español y comparar el rendimiento frente a resultados previos en los que otros modelos fueron utilizados, por ejemplo, Word2Vec.

Palabras clave: Similitud entre frases, Global Vectors, GloVe, representación vectorial de frases, diferencia semántica, similitud semántica, vectores de palabras en español, similitud entre textos.

Abstract

Word embeddings are word representations in the form of vectors that allow to maintain certain semantic information of the words. There exist different ways of taking profit of the semantic information the words have, as there exist different ways of generating the word vectors that represent those words (e.g. Word2Vec model vs. GloVe model). By using the semantic information the word embeddings capture, we can build approximations to compare semantic information between phrases or even documents instead of words. In this project, we propose the use of the GloVe tool, presented by Stanford University, to train Spanish word embeddings, use them to compare semantic differences between Spanish phrases and compare the accuracy of the system with prior results in which other models were used, for example, Word2Vec.

Key words: Phrase similarity, Global Vectors, GloVe, phrase embeddings, semantic difference, spanish word embeddings, text similarity, word embeddings, word vector representations.

Contents

Contents **v**
List of Figures **vii**
List of Tables **vii**

1 Introduction **1**
1.1 Motivation 3
1.2 Objectives 3
1.3 Structure of the report 4

2 Related work **5**
2.1 Classification of the task and methods 5
2.1.1 String-Based Similarity 5
2.1.2 Corpus-Based Similarity 7
2.1.3 Knowledge-Based Similarity 8
2.1.4 Description of the SemEval 2015 Task on Semantic Text Similarity 9
2.2 State-of-the-art systems in Spanish text similarity 11
2.2.1 Best systems of SemEval 2015 11
2.2.2 Other state-of-the-art systems based on word embeddings . 12

3 Tools and methods for learning word embeddings **15**
3.1 Word2Vec 15
3.1.1 Feedforward Neural Net Language Model (NNLM) 15
3.1.2 Hierarchical Softmax 16
3.1.3 Continuous Bag-of-Words Model 16
3.1.4 Continuous Skip-gram Model 16
3.2 GloVe: Global Vectors 17
3.2.1 Co-occurrence probability ratios 18
3.2.2 Weighted least squares regression model and cost function . 18
3.2.3 GloVe method summarized 19
3.3 Comparison between Word2Vec and GloVe models 19

4 Experimentation.
Method and evaluation **21**
4.1 Experiment description and setup 21
4.2 Experiment results 25
4.3 Analysis and comparison of results 25

5 Further experimentation **27**
5.1 Increasing training iterations 27
5.2 COMB method using all six similarity measures 28
5.3 Vector Aggregation without stop words 29
5.3.1 COMB method with Vector Aggregation without stop words 29

5.4 Exploring the Combinatorial Space of the COMB method 30
6 Final conclusions and future work **33**
6.1 Final conclusions 33
6.2 Future work 34
Bibliography **35**

Appendix
A Auxiliar scripts **41**
A.1 `remove_tokens.py` 41
A.2 `experiment.py` 43

List of Figures

2.1 String-Based Similarity Measures 6
2.2 Corpus-Based Similarity Measures 8
2.3 Knowledge-Based Similarity Measures 9

3.1 Representation of the architecture of CBOW model 17
3.2 Representation of the architecture of Skip-gram model 18

5.1 GloVe: Decreasing cost function value as iterations go through . . . 28

List of Tables

2.1 Examples of pairs of phrases from the evaluation set with their respective target value. 10
2.2 Results of the runs of the best three teams of the Spanish STS of the Task 2 of SemEval 2015 12
2.3 Best run per team 12
2.4 Results of López Solaz, Tomás, et al. [47], compared with ExBThemis 13

4.1 Short summary of the used corpora 22
4.2 Results of every similarity measure for the Training set (Model 1) . 25
4.3 Results of every similarity measure for the Training set (Model 2) . 25
4.4 Comparison of our results (Model 1) with the teams at SemEval 2015 and [47] 26
4.5 Comparison of our results (Model 2) with the teams at SemEval 2015 and [47] 26

5.1 Comparison between Model 1 trained with 20 and 50 iterations . . 27
5.2 Comparison between the results of the original combination method and the improved combination method 28
5.3 List of the Spanish stop words provided in the NLTK Corpus 29
5.4 Comparison between VA-NoSW methods with their respective VA method 29
5.5 Comparison between the original combination method and the combination method of all the similarities 30
5.6 Comparison between the best combination and COMB-8 30

5.7 Examples of the results of our best system for the pairs of phrases shown before. 31

CHAPTER 1
Introduction

Computational Linguistics (CL) is the field of Computer Science that aims to model the human natural language in order to automatically treat digital text to accomplish an innumerable amount of tasks that are related to text processing, analysis, synthesis, and calculations of text characteristics that are involved in other problems, such as semantic parsing, text similarity, answer selection, etc.

From the view of CL, generally, every problem is related to some sort of linguistic representation, ideally, digital text. Texts are made of atomic pieces, known as words. Practically, a word is the linguistic unit, that generally possesses an intrinsic meaning (i.e. it expresses something) that grouped in conjunction with other words by following a set of grammatical rules, makes a sentence, building a more complex idea. Technically, in text, a word is a set of characters delimited by a blank space or a punctuation mark.

As in every other Computer Science field, all data to be processed must first become encoded in order to be understandable by the computer. In the case of CL, there are several ways of encoding words [72, 53], depending on several factors, that have relative advantages and disadvantages. For example, we can represent a text with a Bag of Words (BoW) model: in this case, the words take the shape of a bit, that can have the value of 0 or 1 depending on the presence of that word in the text. Nevertheless, one of the potential downsides of the BoW model is that it does not capture the meaning of the words since it only takes into account if that word appears in a given text. Another example, more related to this project is the representation of words known as word embeddings. Word embeddings are n-dimensional vectors of real values that are built with a large corpus of plain text. One of the main characteristics of word embeddings that make it special is the capability of keeping a relative meaning of the words, and that has opened up a whole world in text representation and processing.

However, the ideal way of representing text generally depends on the problem and its nature. In the field of CL, there is a broad set of different problems and challenges that require different representations of text to be tackled. Some well-known problems related to the field of CL are the following:

- Word Sense Disambiguation (WSD): Given a polysemic word in an arbitrary text, determine the meaning of the word [59].
- Part-of-speech Tagging (PoS Tagging): Given a text, determine the parts of speech pertaining to each word (e.g. verb, noun, adjective, etc.) [27].
- Cross-language Information Retrieval (CLIR): Given an Information Retrieval system and a query, process the query to be able to obtain potential results to the query in different languages [60].
- Metonymy Resolution: Given a piece of text, find all the metonymies and classify them into the correspondent type of metonymy [50].
- Named-entity recognition (NER): Given a text, find entities (i.e. persons, companies, places) and classify them into their respective class [57].
- Semantic Dependency Parsing (SDP): Given a sentence, recover sentence-internal predicate–argument relationships for all content words [61].
- Semantic Role Labeling: Given a sentence, find semantic arguments associated with the predicate or the verb and classify them into a set of specific roles [18].
- Sentiment Analysis: Given a text, identify sentiment related characteristics, such as polarity, attitude and emotions to classify it as a positive, negative or neutral opinion [41].
- Commonsense Reasoning: Given a text, generate assumptions and practical inferences of non-existent explicit information [42].
- Lexical Simplification: Given a text, replace words and/or short phrases (generally by synonyms) in order to make it simpler so it can be understood by a wider range of readers [73].
- Plagiarism Detection: Given two texts, determine if the compared text is a plagiarism of the source text or calculate a plagiarism score between the two texts [64].
- Text Classification: Given a text and a set of possible topics, classify the text into the correspondent topic [49].
- Author Profiling: Given a text, identify who is the author (from a given set of possible authors), or infer certain characteristics of the author, such as gender, age, personality, language variety, ideological or organizational affiliation, etc. [69]
- Semantic Text Similarity: Given two texts, calculate a semantic similarity score between the two texts [26].

Most, if not all of these problems have been presented in different ways in tasks of SemEval[1], which is an ongoing series of evaluations of computational semantic analysis systems in which researchers and research groups can participate

[1] https://www.aclweb.org/aclwiki/index.php?title=SemEval_Portal

in order to solve a task related to CL, and, in the end, a ranking of all submitted systems following some performance metrics (depending on each task) is presented in order to compare all the systems.

In this project, we are going to introduce and tackle the task of Semantic Text Similarity for the Spanish language by using word embeddings generated with the Stanford tool GloVe[2]. Also, we will compare the results with similar work in which other methods of calculating the word embeddings have been used (Word2Vec) in order to find differences between distinct word embedding tools and models, and finally, given the word embeddings obtained with GloVe, determine if improvements can be made by slightly modifying the methods that best work using Word2Vec embeddings to calculate phrase-level semantic similarity.

1.1 Motivation

As [24] introduces, supposing that a practical and valid method of calculating the semantic difference between two short texts exists, there are many applications in Natural Language Processing (NLP) that can take advantage of it. For example, in the field of information retrieval and image retrieval from the Web, one of the best techniques for improving retrieval effectiveness is by using semantic similarity [63].
The use of text similarity is also useful for boosting accuracy results in relevance feedback and text categorization [30, 43], as for methods for automatic evaluation of machine translation [44, 62], evaluation of text coherence [75, 35], word sense disambiguation [37, 71], formatted documents classification [75] and text summarization [15, 39]. Also, it has been proved that for data sharing systems such as federated databases, message passing or data integration systems, web services, data management systems, etc., lexical and syntactical differences between shared variables can be solved by using semantic text similarity [48].
Semantic text similarity can also be used to build a text similarity join operator, that can be used to join two relations if their join attributes are textually similar to each other, which can be useful in several domains, such as integration of data from heterogeneous resources, mining of data, cleansing of data, etc. [14]

1.2 Objectives

The objective of this thesis is to determine and prove whether a system using word embeddings generated with GloVe can perform better than state-of-the-art systems that use the collection of models Word2Vec to build the word vector representations for their final use in the field of text similarity. We compare both methods (GloVe and Word2Vec) in several ways in order to determine which aspects of the word embeddings are different for the task of semantic text similarity. After analyzing the results, we also aim to use the currently generated word embeddings with GloVe in several different ways to improve the performance of our model.

[2] https://nlp.stanford.edu/projects/glove/

1.3 Structure of the report

This report is structured as follows. The next chapter (chapter 2) is focused on related work, especially, previous work made by the professors of the University of Sevilla, that is closely related to the work of this thesis, as the main objective of this thesis is to reproduce their approximation of [47] but using word embeddings generated with GloVe in order to compare the used tools.
Chapter 3 introduces the two most famous tools to build word embeddings: Word2Vec and GloVe; in this chapter, a simple description of each tool/method is explained and a comparison between the two is made in order to understand the main differences between the two.
In Chapter 4, that is the most extensive, we explain the setup of our experiment and the steps made to reproduce [47] with GloVe. Also, in Chapter 4, we compare the results obtained.
In Chapter 5, we discuss some variations of the experiment explained in Chapter 4 and compare the result of all the different experiments.
Finally, in Chapter 6, we conclude the thesis making some final conclusions and establishing some possibilities of future work.

CHAPTER 2
Related work

This chapter discusses prior related work and research made in the field of semantic text similarity. Firstly, a classification of the tackled task is made, along with the already existing methods to solve the challenge. We also introduce the different classes these methods can belong to. Secondly, we are going to mention three systems that are the state of the art in this task and then, we are going to reference and explain the adaptation made in [47] in order to obtain state-of-the-art results with their system for this task, but for Spanish texts, as our approach is closely related to their work.

2.1 Classification of the task and methods

As already stated, finding the similarity between words is elemental to calculate the similarity between sentences, paragraphs, and documents. There are two kinds of similarity between words: lexical similarity and semantic similarity. Two words are lexically similar if they are built using a similar sequence of characters. Two words are semantically similar if they mean similar things, are opposite of each other, one is a type of the other, used in a similar way, or are in the same context [19].

While lexical similarity can be calculated through String-Based algorithms, semantic similarity is calculated using Corpus-Based algorithms or Knowledge-Based algorithms. String-Based algorithms ensure that a string comparison metric is used in order to compare the similarity of the different sequences of characters. Corpus-Based algorithms measure the semantic similarity of words using information gathered in a large corpus, and Knowledge-Based algorithms calculate the semantic similarity between words using information obtained from semantic networks.

2.1.1. String-Based Similarity

String-Based similarity measures focus on the comparison between the string that delimits the two segments of text being compared. There are two sub-types of string comparison, one is Character-Based and the other is Term-Based. Character-Based similarity compares the sequence of characters of the segments of text (i.e.

the letters of the words), while Term-Based similarity compares the blocks of the segments of texts (i.e. the words of the texts).

In the first group [19] lay the Longest Common Substring (LCS) algorithm, the Jaro Method [25] used in record linkage, Damerau-Levenshtein algorithm [20], Needleman-Wunsch and Smith-Waterman algorithms, used in bioinformatics; and character N-grams, where the distance can be calculated by dividing the number of similar n-grams by the total number of n-grams [5].
In the second group [19] we can find the Manhattan Distance, Cosine Similarity, Dice's Coefficient, Euclidean Distance, Jaccard Similarity, Matching Coefficient and Overlap Coefficient. A schematic representation of the classification of String-Based algorithms is represented in the Figure 2.1 [19].

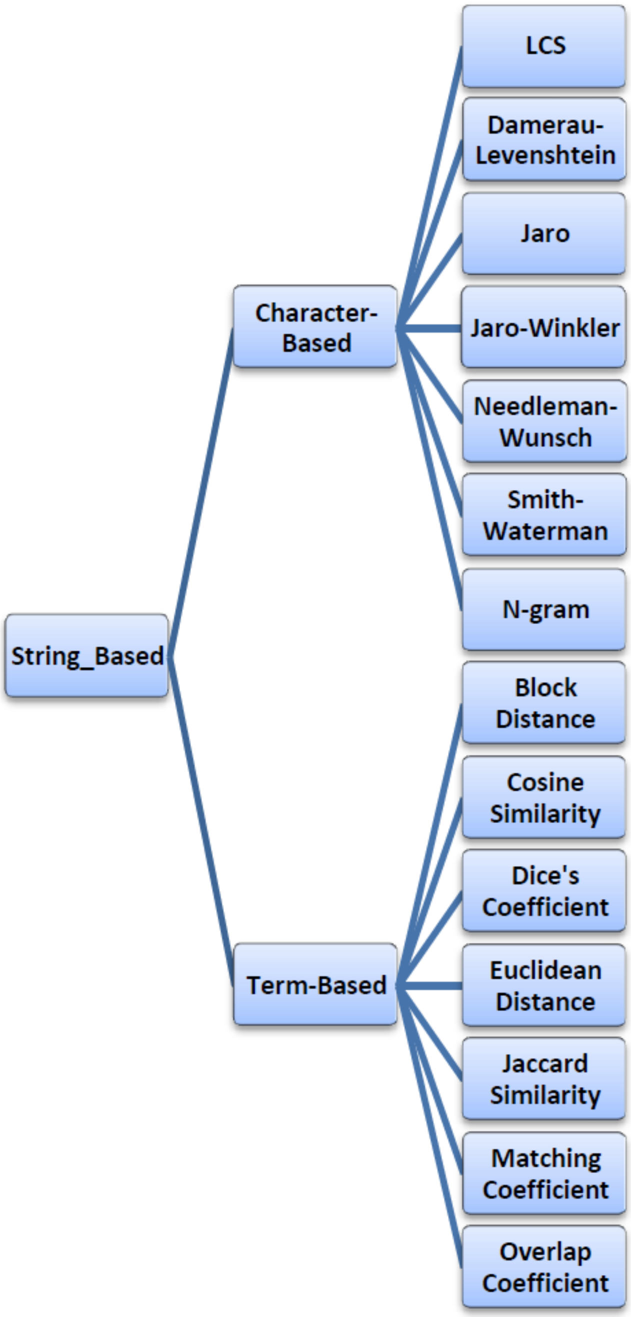

Figure 2.1: String-Based Similarity Measures

2.1.2. Corpus-Based Similarity

Corpus-Based algorithms measure the semantic similarity of words using information gathered in a large corpus. The next methods fall into this category of similarity computation methods:

- **Hyperspace Analogue to Language (HAL)**[46, 45] is a method in which a square matrix of size N is created, where N is the number of distinct terms that appear in the training corpus. Then, a sliding window is passed through the text, counting how many times each word (columns) appears in the context of the currently evaluated word (row), and adding a value (that is weighted depending on the distance to the currently evaluated word inside the window) to the count in the matrix every time the words are in the same context.

- **Latent Semantic Analysis (LSA)**[34] is the most popular form of Corpus-Based Similarity. In this technique, the assumption of the distributional hypothesis is made, meaning that words that are close in meaning will occur in similar pieces of text. Similarly to HAL, a matrix is constructed from a large piece of text (corpus) taking into account the cooccurrences inside a defined context (usually, a sliding window). Then, a mathematical technique called singular value decomposition (SVD) is used to reduce the number of columns while preserving the similarity structure among rows, leaving the word vectors as rows in the matrix.

- **Generalized Latent Semantic Analysis (GLSA)**[51] is a framework for calculating semantic term and document vectors [19]. It amplifies the LSA method by focusing on term vectors instead of the dual document-term representation. GLSA demands a specification of the semantic association between the terms (e.g. a context; given, for example, by a sliding window) and a method of dimensionality reduction (e.g. SVD, like in LSA; PCA, LDA, etc.).

- **Explicit Semantic Analysis (ESA)**[17] is a measure to compare two arbitrary texts. One famous approach is the Wikipedia-Based technique, which represents texts or words as high-dimensional vectors and each vector entry represents the TF-IDF weight between the term and one Wikipedia article. In this way, there is a manner of representing the meaning of the word by calculating how important is this word in each Wikipedia article.
  The multilingual generalization of this technique is known as **cross-language explicit semantic analysis (CL-ESA)** [68].

- **Pointwise Mutual Information - Information Retrieval (PMI-IR)**[74] uses AltaVista's Advanced Search system to calculate probabilities that two words appear close to each other in a web, which translates into a semantic similarity. The **Second-order co-occurrence pointwise mutual information (SOC-PMI)** [23] first calculates the PMI-IR to sort the list of important semantically close words of both target words. Then, it calculates the similarity between the two sets of sorted words to better determine the similarity of two words that do not co-occur frequently.

- **Normalized Google Distance (NGD)** [12] is derived from the number of hits returned by the Google search engine after doing a query with the keywords of which the similarity has to be calculated.

- **Extracting DIStributionally similar words using CO-occurrences (DISCO)** [32, 31] is also based in the distributional hypothesis and a sliding window. The similarities are based on the statistical analysis of very large text collections. This method has two similarity measures: DISCO1 and DISCO2. Similarly to PMI-IR and SOC-PMI, DISCO1 is the first order similarity between two words, while DISCO2 is the second-order similarity between two words, that is calculated using the results of DISCO1 to perform the similarity measures between the set of closest words to each of the two input words.

A schematic representation of the classification of Corpus-Based algorithms is represented in the Figure 2.2 [19].

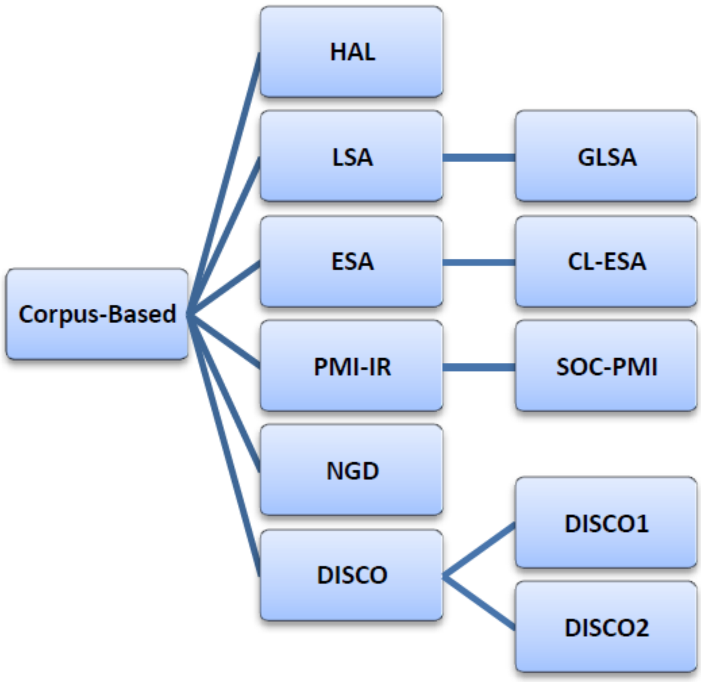

Figure 2.2: Corpus-Based Similarity Measures

2.1.3. Knowledge-Based Similarity

Knowledge-based algorithms calculate the semantic similarity between words using information obtained from semantic networks. One well-known semantic network is WordNet [54]. WordNet is a large lexical database of English nouns, verbs, adjectives and adverbs, that are grouped into sets of synonyms (known as synsets). Synsets can also be considered sets of semantically related words.

Knowledge-based algorithms can be divided into two groups [19]: measures of semantic similarity and measures of semantic relatedness. The former covers much more senses of relatedness than the latter, as semantic similarity takes into account relations such as is-a-kind-of, is-a-specific-example-of, is-a-part-of, is-the-opposite-of [66].

Inside the semantic similarity class just described, there are two subgroups: methods based on information content, such as Resnik (res) [70], Lin (lin) [40] and Jiang & Conrath (jcn) [26], pertain to one group, while methods based on path length, such as Leacock & Chodorow (lch) [36], Wu & Palmer (wup) [77] and Path Length (path) [38], pertain to the other group.
In the semantic relatedness class, three similarity measures can be found: St.Onge (hso) [22], Lesk (lesk) [2] and vector pairs (vector) [65].

A schematic representation of the classification of Knowledge-Based algorithms is represented in the Figure 2.3 [19].

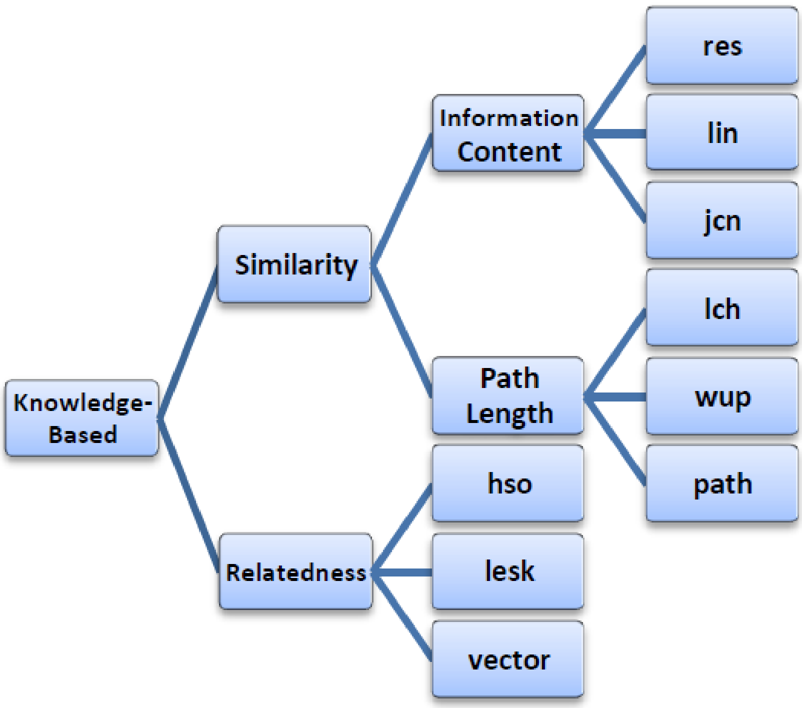

Figure 2.3: Knowledge-Based Similarity Measures

2.1.4. Description of the SemEval 2015 Task on Semantic Text Similarity

In this task, the submitted systems had to compute how similar were two sentences of text, by returning a similarity score between 0 and 4. The similarity score 0 means that the similarity between the sentences is null, in other words, it means that there is no semantic relation between the two sentences at all. The similarity score 4 means that the sentences are semantically equivalent. To evaluate the systems, an evaluation set of pairs of sentences was used with human annotations (also known as ground truth or Gold Standard annotations in SemEval) of the similarity score. The Gold Standard annotations were created calculating the mean of the annotations of different expert judges in semantic text similarity.

Some examples of input pairs of phrases and target values can be found in table 2.1.

Compared pair of phrases	Target value
El espécimen es excepcional por las partes conservadas: un cráneo y mandíbula y un molde interno de la caja craneal. El espécimen comprende la mayor parte de la cara y mandíbula con los dientes y un molde interno de la caja craneal.	4
Time “100´´ es una lista de las 100 personas más influyentes según la revista Time. La primera lista fue publicada en 1999 con las 100 personas más infuyentes del siglo 20.	3
La “marinera´´ es un baile de pareja suelto, el más conocido de la costa del Perú. La marinera es el baile nacional del Perú, y su ejecución busca hacerse con derroche de gracia, picardía y destreza.	2
La “cripta de Santa Leocadia´´ está situada en el interior de la catedral de Oviedo, Asturias. Esteban Báthory fue sepultado en la cripta de la catedral de Wawel en Cracovia.	1
El río atraviesa la importante ciudad de Puebla de Zaragoza, la cuarta más poblada del país. El “Grêmio Esportivo Bagé´´ es un club de fútbol brasileño, de la ciudad de Bagé en el estado de Rio Grande do Sul.	0

Table 2.1: Examples of pairs of phrases from the evaluation set with their respective target value.

Evaluation of the systems

The final evaluation was made by using the commonly used metric in semantic text similarity tasks: the Pearson Correlation Coefficient (PCC), also known as *Pearson r*, *linear* or *product-moment* correlation.
The PCC between two vectors of real values, $\mathbf{x}$ and $\mathbf{y}$ is defined [6] as:

$$r(\mathbf{x},\mathbf{y}) = \frac{\sum_i (x_i - \bar{\mathbf{x}})(y_i - \bar{\mathbf{y}})}{\sqrt{\sum_i (x_i - \bar{\mathbf{x}})^2}\sqrt{\sum_i (y_i - \bar{\mathbf{y}})^2}} \tag{2.1}$$

One important property of PCC is that:

$$-1 \leq r(\mathbf{x},\mathbf{y}) \leq 1 \tag{2.2}$$

$\mathbf{x}$ and $\mathbf{y}$ can be considered two ordered lists of real numbers, where x_0 is related to y_0, x_1 is related to y_1, and so on. In that case, the value $r(\mathbf{x},\mathbf{y})$ calculates the linearity between the values of $\mathbf{x}$ and $\mathbf{y}$. If the linearity is total (i.e. $r(\mathbf{x},\mathbf{y}) = 1$), it means that $\mathbf{x}$ is a linear combination of $\mathbf{y}$ or vice versa. If the linearity is nonexistent (i.e. $r(\mathbf{x},\mathbf{y}) = 0$), it means that there is absolutely no relation between the behavior of $\mathbf{x}$ and $\mathbf{y}$. If the linearity is inverse (i.e. $-1 \leq r(\mathbf{x},\mathbf{y}) \leq 0$), it means that $\mathbf{x}$ is inversely proportional to $\mathbf{y}$, i.e. one increases as the one decreases, proportionally. If the linearity is relatively strong (i.e. $0.6 < r(\mathbf{x},\mathbf{y})$), it means that there is some relationship (higher relationship the higher the value of $r(\mathbf{x},\mathbf{y})$) between the variables in $\mathbf{x}$ and the variables in $\mathbf{y}$.
In the case of semantic text similarity, PCC is useful to calculate the grade of linearity between the ground truth and the predictions the system makes. The better PCC, the more the predictions fit the ground truth, hence, the better the system is predicting the real similarity value between the texts. Therefore, the objective is to build a system which predictions, paired with the gold standard values of similarity, returns the PCC as close to 1 as possible.

Classification of the task

Given the introduction to the different methods of calculating the similarity between texts, we can classify (and will confirm it in chapter 4) that our approach to this task is based on a Corpus-Based Similarity approach.

2.2 State-of-the-art systems in Spanish text similarity

The majority of recent contributions in the field of semantic text similarity comes from the participation of research groups in the related tasks of SemEval. Thanks to the work of these groups, a lot of techniques, ideas, and even frameworks have emerged to help develop text similarity systems. One example of an open source framework for text similarity is **DKPro Similarity** [3].

In this section, three state-of-the-art systems that showed the best performance in the results of the Spanish Text Similarity task of SemEval 2015 [1] will be described. Then, one last system presented by the professors of the University of Sevilla [47] that has achieved a better performance than the best system of SemEval 2015 will be introduced. As we will describe in the evaluation part of Chapter 4, the metric used in this task for the evaluation is the Pearson Correlation Coefficient.

2.2.1. Best systems of SemEval 2015

The three best groups of SemEval 2015 were **ExB Themis** [21], **UMDuluth-BlueTeam** [28] and **RTM-DCU** [10], that ended being first, second and third in the rank, respectively.
ExB Themis joins three techniques: vector representation of texts through BOW, sequential alignment with the help of similarity techniques, and the use of machine learning to combine the different calculated metrics.
UMDuluth-BlueTeam takes a system previously used for the English language task and utilizes the Google Translator in order to translate the inputs for their system. The main idea of this system is to build an alignment system based on different metrics such as proportionality, number of adjectives, verbs, nouns, etc.; and the size of the texts.
RTM-DCU submitted a system based on Referential Translation Machines, which is founded on how similar are the two texts being compared when they are translated into another language.

While ExBThemis and RTM-DCU submitted three runs, UMDuluth-BlueTeam only submitted one. The ExBThemis system was clearly the best, obtaining more than 10 points more than the second best system, UMDuluth-BlueTeam. The results of all the runs submitted for each system and the ranking of the teams depending on the best run are shown in Table 2.2 and Table 2.3 respectively.

System	*Pearson*	*Rank*
ExBThemis-trainMini	0.70550	1
ExBThemis-trainEs	0.70545	2
ExBThemis-trainEn	0.67630	3
UMDuluth-BlueTeam-run1	0.59364	4
RTM-DCU-1stST.tree	0.58233	5
RTM-DCU-2ndST.rr	0.58233	6
RTM-DCU-3rdST.SVR	0.58233	7

Table 2.2: Results of the runs of the best three teams of the Spanish STS of the Task 2 of SemEval 2015

Team	*Pearson*	*Rank*
ExBThemis	0.70550	1
UMDuluth-BlueTeam	0.59364	2
RTM-DCU	0.58233	3

Table 2.3: Best run per team

2.2.2. Other state-of-the-art systems based on word embeddings

Word embeddings are used in a lot of systems for different purposes, for example, they have been used in different manners for Phrase-Based Machine Translation [78], calculating document distances [33], learn semantic hierarchies [16], improve document ranking [58], speech recognition [7], information retrieval [13], clinical abbreviation disambiguation [76], etc. One of the main uses of word embeddings is to calculate the similarity between texts, as they are enough powerful they can yield good results by being used without any additional external information, as can be seen in [29, 47].
In [29], the authors aim to make as few assumptions as possible and to build a generic model that requires no prior knowledge of the natural language (such as parse trees) and no external resources of structured semantic information.
In [47], the authors aim to improve the best results of SemEval 2015 by only using word embeddings. In both approaches, it is clear that the only resource of data is the huge amount of unlabeled text data extracted from the web (mainly Wikipedia), which is an appealing characteristic of the model, since this kind of data not expensive to obtain.
Our objective is to take [47] as our main reference and reproduce their approach using GloVe word embeddings and see if their results can be improved, therefore, we proceed to make a detailed explanation of the approach taken by them.

The authors of [47] describe a method that is built by combining several similarity indicators based on word embeddings to calculate similarities between the words of the phrases. The first two indicators are obtained doing word vector aggregation to build phrase embeddings from word embeddings, and then, calculating two simple distances between phrase embeddings by using the Euclidean distance and the cosine similarity. The third indicator is obtained doing an alignment of the phrases that are being compared and using word embeddings to make the alignment of the words that cannot be matched directly. To combine the indicators, a method based on supervised machine learning for regression is used.

There were two models trained in order to get the word embeddings, the Model 1, which was trained by only using the Spanish Wikipedia[1], and the Model 2, that was trained by adding Europarl [2] and Ancora-ES [3] corpora as training input for the estimation of the word embeddings. This estimation was made using the tool Word2Vec. Word vectors of 300 dimensions were generated, the number of iterations was 20, and the option negative sample was used to remove noise words. The training set consisted of 324 pairs of sentences for training and 251 pairs of sentences for evaluation. The training split corresponds to the test split of Task 10 of SemEval 2014 "Multilingual Semantic Textual Similarity". The test split corresponds to the test split of Task 2 of SemEval 2015 "Semantic Textual Similarity". The results of the experiments are shown in the Table 2.4

System	*Pearson*	Δ
ExB Themis	0.706	-
Euclidean distance [E] (Model 1)	0.509	-19.7
Euclidean distance [E] (Model 2)	0.642	-6.4
Cosine similarity [C] (Model 1)	0.467	-23.9
Cosine similarity [C] (Model 2)	0.646	-5.9
Alignment [A] (Model 1)	0.692	-1.4
Alignment [A] (Model 2)	0.687	-1.8
Combined [E+C+A] (Model 1)	0.713	0.7
Combined [E+C+A] (Model 2)	**0.723**	**1.8**

Table 2.4: Results of López Solaz, Tomás, et al. [47], compared with ExBThemis

In table 2.4, the column Δ represents the absolute improvement of the correspondent method (given by the row) with respect to the best SemEval 2015 system, ExB Themis.

[1]https://dumps.wikimedia.org/eswiki/latest/
[2]http://www.statmt.org/europarl/
[3]http://clic.ub.edu/corpus/

CHAPTER 3
Tools and methods for learning word embeddings

This chapter focuses on two of the most famous existing methods to unsupervisedly learn word embeddings from large corpora, that are Word2Vec and GloVe. There are way more than two methods for estimating continuous representations of words; a couple of well-known classic examples are Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA). However, the main downside of this methods is the high computational cost, since they must be estimated from a large corpus, and the operations made in order to estimate the vectors are not scalable enough. That is the reason for the popularity of Word2Vec models from [52]. In this paper of 2013, Mikolov et al. presented a novel efficient way of calculating continuous word vector representations and developed software that implemented the models that are explained in their paper, giving to the scientific community a valuable tool to keep investigating on word embeddings. From that time, other efficient methods of unsupervisedly estimating word vectors have grown in the scientific community, and one of the most famous is GloVe, that stands for Global Vectors, and is a model created by the professors of the University of Stanford, and described in their paper [67].

3.1 Word2Vec

The intuitive idea behind Word2Vec models is to train deep neural networks in order to predict the context given a word, and vice versa. The model to predict the context $[..., w(t-2), w(t-1), w(t+1), w(t+2), ...]$ given a word $w(t)$ is known as the Skip-gram model, while the model to predict the word that goes in the middle given the context is known as the Continuous Bag of Words (CBOW) model. The figures 3.1 and 3.2 show the architecture of both models.

3.1.1. Feedforward Neural Net Language Model (NNLM)

While the Skip-gram model is based on the CBOW model, the CBOW model is based on the Feedforward Neural Net Language Model (NNLM), which was originally introduced by Bengio et al. in [8]. The probabilistic feedforward neural network language model is a deep neural network model of four layers: input (I), projection (P), hidden (H) and output (O). The input layer (I) takes the N previous words in *1-of-V* encoding, V being the size of the vocabulary. Then, a projection using the shared projection matrix encoded by the projection layer (P) is made. After that, the hidden layer (H) computes the probability distribution over all the words in the vocabulary and with the help of a softmax activation function, the output layer (O) shows the probability of each word being the next one.

The literature specifies that the dominating term of the cost function resides in the size of the vocabulary, V, because the output layer size increases as V increases and the softmax function gets harder to compute; and H, that is the hidden layer size, that depends on the chosen value of N (the context, that usually is $N = 10$), but normally, H ranges from 500 to 2000.

3.1.2. Hierarchical Softmax

There are solutions to significantly reduce the computational cost of the output probability. The most used method is to use the hierarchical variation of the softmax function [56, 55]. The hierarchical softmax is founded on the idea of building a Huffman tree based on word frequencies, meaning that the most frequent word is the one with the shortest path from the root to the leaf that represents the word itself. The normalization of the probabilities of each target word is calculated by continuously multiplying prior calculated probabilities of the tree (i.e. the probability of the branches), and this reduces the complexity from $H \times V$ to $log_2(V) \times H$.

3.1.3. Continuous Bag-of-Words Model

The Continuous Bag-of-Words model, known as CBOW model is a model derived from the NNLM model, but in this case, the non-linear hidden layer (H) is removed so the projection layer is shared for all words of the context. In the CBOW model, the position of the words does not influence the projection, and that is why it is called a bag-of-words model. This model is used to predict the word given the context.

3.1.4. Continuous Skip-gram Model

The Continuous Skip-gram Model (or simply Skip-gram model, *SG*) is based on the CBOW model, but it aims to adjust its parameters to try to maximize the classification of the current word given a context. If we consider the conditional probability $p(c|w)$, the goal of the SG model is to calculate the parameters θ of the distribution, $p(c|w;\theta)$, that maximize the probability of the corpus:

$$\arg\max_{\theta} \prod_{w \in Text} \left[\prod_{c \in C(w)} p(c|w;\theta) \right] \tag{3.1}$$

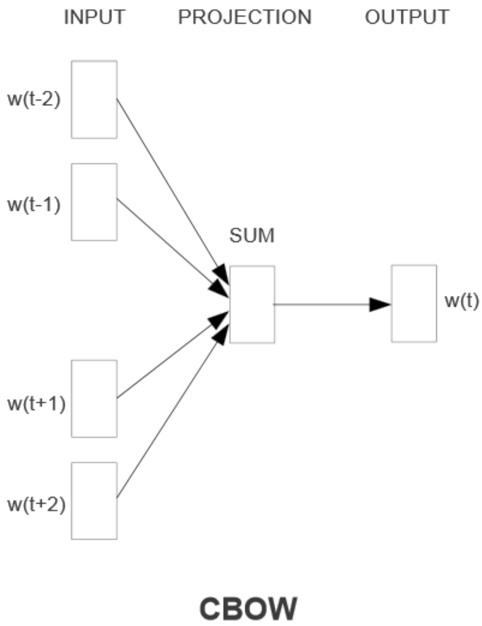

Figure 3.1: Representation of the architecture of CBOW model

that can also be represented as:

$$\arg\max_{\theta} \prod_{(w,c)\in D} p(c|w;\theta) \tag{3.2}$$

where D is the set of all word-context pairs in the corpus.

To do this, each current word is used as input to a log-linear classifier with a continuous projection layer in order to predict words that are near the center (current) word within a range.

3.2 GloVe: Global Vectors

The intuitive idea behind GloVe model is to build a very big matrix, X, of co-occurrence words from a corpus. Each cell of the matrix, X_{ij} represents how many times has the row word, w_i, appeared in some context, c_j. By doing a simple normalization of the values for each row of the matrix, we can obtain the probability distribution of every context given a word. Once the probabilities have been calculated, the relationship between words can be calculated by doing the ratio between the probabilities of the context given those two words. For example, given two random contexts formed by only one word, *dog* and *teapot*, and a target word *tail*, we expect $P(\text{“}dog\text{”}|\text{“}tail\text{”})$ to be higher than $P(\text{“}teapot\text{”}|\text{“}tail\text{”})$, therefore, we expect the ratio $\frac{P(\text{“}dog\text{”}|\text{“}tail\text{”})}{P(\text{“}teapot\text{”}|\text{“}tail\text{”})}$ to be greater than 1. In the same way, the ratio between the probabilities of two similar contexts (with respect to probability) tends to be close to 1.

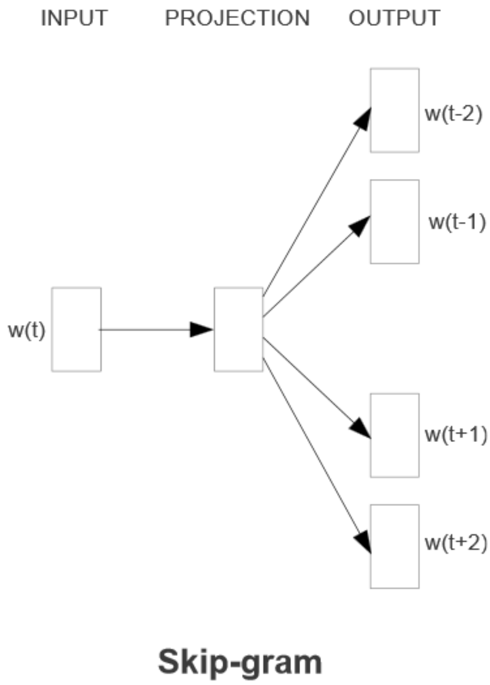

Figure 3.2: Representation of the architecture of Skip-gram model

3.2.1. Co-occurrence probability ratios

Given this reasoning, the first step of the creation of the word vectors is to use the ratios instead of the raw probabilities, so given the words i, j and k, the ratio $\frac{P_{ij}}{P_{ik}}$ can be expressed in the terms of those words:

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ij}}{P_{ik}} \tag{3.3}$$

where the probabilities and therefore the ratios are extracted from the corpus and the left-hand side of the equation may depend on some parameters. As the authors state, the number of possibilities for F is big, but taking several assumptions and derivations, they end up with a soft constraint to the matrix values that define the word vectors:

$$\overrightarrow{w}_i^T \overrightarrow{w}_j + b_i + b_j = \log X_{ij} \tag{3.4}$$

3.2.2. Weighted least squares regression model and cost function

But there is a major problem with this formulation, that is that it it weighs all co-occurrences equally, so the result word vectors y very sensitive to noise, so the authors propose a weighted least squares regression model that takes into account the noise by adding weight to each term of the function. The cost function of the model is expressed as:

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - log X_{ij}\right)^2 \tag{3.5}$$

and by taking a function f that does not skew the objective with the presence of values that are distinctively high:

$$f(x) = \begin{cases} (x/x_{max})^{\alpha} & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases} \tag{3.6}$$

so, when a pair of words that are extremely common is found (i.e. $X_{ij} > x_{max}$), the function cuts it off and returns 1. In the other case, the function returns a value in the range $(0, 1)$ that is weighted by the value of α, which is demonstrated to give the best performance outputs when $\alpha = 3/4$.

Finally, by means of gradient descent and calculating the derivative of the cost function with respect to the important parameters (w_i, w_j, b_i, b_j), the values of the vectors get rectified through iterations until the cost function reaches a local minimum and the vectors reach a state of convergence.

3.2.3. GloVe method summarized

The algorithm of the GloVe tool acts as follows:

1. Given a vocabulary, a corpus, a window size (N) and a minimum co-occurrence count, the algorithm takes a sliding window of size N and passes it through the entire corpus, counting the co-occurrences of every word with every other word. The result is the sparse matrix X_{ij}.
2. Initialization of the model parameters. Those are the word vector matrix W of size ($V \times D$, where V denotes the size of the vocabulary and D denotes the specified vector dimensions; and a vector of bias for each term. Each term is initiated randomly in the range of $(-0.5, 0.5)$
3. The co-occurrences are shuffled and the algorithm calculates the cost function associated with the initial phase of the algorithm.
4. For each iteration, gradients of the cost function J are derived with respect to the parameters and the parameters are updated accordingly to a learning rate.

3.3 Comparison between Word2Vec and GloVe models

Despite both Word2Vec and GloVe models outputs are the estimated word vectors, it is clearly visible that the two methods are far from being similar. Although there are some parts of the methods that are similar, such as the pass of the sliding window over the text or the use of some gradient descent method to adjust parameters, the basis on which the two methods are founded are completely different.

While Word2Vec is a predictive model, GloVe is a count-based model [11, 4]. On the one hand, Word2Vec seeks to maximize the log-likelihood of the probabilities of the corpus (i.e. the words and their contexts) given the model parameters (i.e. the word vectors, θ). This is done by learning the vectors with the help of a learning algorithm such as the backpropagation algorithm for feedforward neural networks, in order to improve the predictive ability of the model, given by the loss function:

$$\mathrm{J}_\theta = \prod_{(w,c)\in D} p(c|w;\theta) \tag{3.7}$$

On the other hand, count-based models such as GloVe, generally learn the word vectors by building a matrix of co-occurrences and doing some sort of dimensionality reduction, preserving the meaningful columns. Although it depends on the parameters of each model, GloVe is generally faster than Word2Vec [67] because it does not need to go through the entire corpus in each iteration. However, most papers report Word2Vec being slightly more accurate in terms of results of their respective tasks [9].

CHAPTER 4
Experimentation. Method and evaluation

In this chapter, the method used and the steps followed in our experiments will be discussed. First, we will make a detailed description of the steps made in order to reproduce the experiments in [47] but using GloVe. Then, we will explain the different setups of the experiments, and finally, we will gather the results of all the experiments and make a reasoned comparison.

4.1 Experiment description and setup

For the first GloVe model (Model 1) We have downloaded the latest Spanish dump of the Wikipedia to process it. This is an XML file that contains all the information of the articles of the Spanish Wikipedia. Note that since latest dumps vary upon the date, the same experiments done with the latest dumps may slightly vary over time. The corpus we have been worked with corresponds to the one of June of 2017.

To extract the articles into a plain text file, the Python script WikiExtractor[1] was used.
After the conversion of the Wikipedia dump into a text file, a tokenization using the Stanford Tokenizer was used. Since the Stanford Tokenizer is a tool used in several Stanford Tools, it is not distributed alone, so we chose to utilize the one integrated into the Stanford Parser Tool[2]. For the execution of the tokenizer, the options `-preserveLines` and `-lowerCase` were used.

However, these steps lead to a plain text file that still has commas, dots, and several escape sequences that the tokenizer uses in order to specify brackets and parenthesis as tokens, which must be removed. In order to remove this kind of tokens that should not appear in the corpus that GloVe is fed with, the implementation of a simple python script was done, in order to remove unwanted tokens and to unify the entire text into a single line, which is recommended for the GloVe tool. This script, named as `remove_tokens.py` is annexed at the end of this document.

[1]https://github.com/attardi/wikiextractor
[2]https://nlp.stanford.edu/software/lex-parser.html

After this processing, the resulting text file (2.8 GB) is composed of a single line of approximately 470,000,000 words. This text is the text in which the GloVe tool bases its first iteration, and it is when the sliding window is used to determine the context of each word in the vocabulary.
After that, GloVe was run with the text file as input to calculate the word vectors. For this first experiment, we set `VOCAB_MIN_COUNT=30`, `VECTOR_SIZE=300`, `MAX_ITER=20`, `WINDOW_SIZE=15` and `X_MAX=10`, as we wanted to approximate this experiment as much as possible to the same experiment but done with Word2Vec in [47].

For the second model (Model 2), the Spanish version of the corpus Europarl (346.5 MB, composed by 56.060.785 words) and the corpus Ancora-ES were added to the Spanish Wikipedia corpus (2.9 MB, composed by 451.918 words). The parameters were the same as for Model 1. A short summary of the corpora characteristics can be found in table 4.1

Corpus Name	*Word count*	*Size*
Wikipedia (Spanish)	474.529.339	2.8 GB
Europarl (Spanish)	56.060.785	346.5 MB
Ancora-ES	451.918	2.9 MB
Wikipedia + Europarl + Ancora-ES	531.042.042	3.2 GB

Table 4.1: Short summary of the used corpora

The word vectors were trained in a machine running an *Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz* processor, in *Ubuntu 16.04.1 LTS*. The training time for Model 1 was 4 hours (approximately 13 minutes per iteration), while the training time for Model 2 was 5 hours (approximately 16 minutes per iteration).

Once the word vectors were generated, a python script was programmed in order to do the entire experiment. The code of this script can be found in the annex, in the section `experiment.py`. This script loads the word vectors into dictionaries and both training and test sentence pairs along with the respective target values (i.e. ground truth similarity measures). After loading the sentence pair, a simple cleaning of the text is made by changing all characters to lower case and removing punctuation marks. After that, a vector aggregation of the sentences is calculated by summing the word vectors and dividing by the number of word vectors that were summed, to normalize the resulting sentence vector:

$$\overrightarrow{V_d} = \frac{\sum_{i=0}^{n} \overrightarrow{v_i}}{n} \tag{4.1}$$

where $\overrightarrow{v_i}$ is the word vector of the word i of the sentence and $\overrightarrow{V_d}$ is the phrase vector.

Once the phrase embeddings of all training and test sentence pairs are calculated, the script performs the calculation of the Euclidean distance, cosine similarity, and alignment distance method described in [47] for every pair of phrases. For the Euclidean distance, a transformation into a “Euclidean similarity” is done, creating a Euclidean distance: given a Euclidean distance, d_e, the calculated Euclidean similarity, s_e takes the form of:

$$s_e = \frac{1}{1 + d_e} \tag{4.2}$$

With respect to the alignment method, which was proposed in [47], the algorithm in pseudo-code is described in 4.1

Algorithm 4.1 Similarity Alignment Method

Require: t_1 and t_2 token sets. m word embedding model
Ensure: s similarity score between $t1$ and $t2$

```
voc = t1 ∪ t2
for all w in voc do
    bow[w] = 0   // Initialization
end for
for all w in t1 do
    if w in t2 then
        bow[w] = 1
    else if w in m then
        bow[w] = max(m.similarity(w, t2))
    else
        continue
    end if
end for
for all w in t2 do
    if w in m then
        bow[w] = max(m.similarity(w, t1))
    else
        continue
    end if
end for
values = (w for w in selected_words)
sim = sum(values)/values.size()
return sim
=0
```

This alignment algorithm performs a bidirectional alignment by looking to align every possible word in the first sentence 1 with every word in sentence 2 and vice versa. When a direct alignment cannot be made (i.e. the word in sentence 1 is not found in sentence 2), the alignment is made by calculating the similarities between the word of the sentence 1 and every possible word of sentence 2 and choosing the closest word.

This alignment method has two prior versions depending on how *selected_words* is described. On the one hand, the selected words are all the words in the dictionary, that is, all the words in the pair of sentences being evaluated, regardless their presence in the model m and therefore their alignment, which could not have been accomplished. We call this "*Counting All*" (CA). On the other hand, the selected words are only those words in which the alignment between the pair of phrases could be made. This means that if a word in the dictionary had a value of 0 because it could not be found in the model m, it does not count as a part of the normalization. We call this "*Counting Only Appearances*" (COA). As the authors of [47] do not state whether they count all words or not, for this experiment we try both versions, CA and COA.

Also, note that in the alignment method, a similarity measure is used in order to find the closest word when it is not found directly, so either the cosine (CS) or the Euclidean similarity (ES) can be used. As the authors in [47] do not specify this either, we calculate the alignment similarities by using both cosine and Euclidean similarities for this step, so we can compare later. However, this leads to another two posterior variations of the alignment algorithm, one that is done with the cosine similarity (CS) and other that is calculated with the Euclidean similarity (ES), meaning that, combined with the two prior versions, there are four different alignment method variations: CA-CS, CA-ES, COA-CS and COA-ES. For this first experiment, we use all of them and see which one gives best results on training and evaluate it with the test set.

For the combination of methods, a linear function takes the similarity results of the methods to build a new similarity. The learning of the factor of each similarity result (so as the bias term of the linear function) is made by calculating a linear least squares solution for the linear function that best explains the target results, given by the 324 pairs of labeled phrases (training set). For the testing phase, an inference using the linear model and the weighs calculated in training is made with the 251 pair of phrases and then the results are compared with the labels and using the PCC to calculate the correlation between the results and the target.

In summary, six vectors of sentence similarities are calculated, which store the calculated similarities between each pair of sentences:

1. VA-CS: Vector Aggregation and Cosine Similarity

2. VA-ES: Vector Aggregation and Euclidean Similarity

3. CA-CS: Alignment method, Counting All and Cosine Similarity

4. CA-ES: Alignment method, Counting All and Euclidean Similarity

5. COA-CS: Alignment method, Counting Only Appearances and Cosine Similarity

6. COA-ES: Alignment method, Counting Only Appearances and Euclidean Similarity

And by taking a subset of those vectors, a combined method (COMB) is built by using the information each method outputs with a linear function. In the first experiment, we combine VA-CS, VA-ES and the best result of the alignment in training to

4.2 Experiment results

Once the word vectors are trained, the similarities VA, CA and COA can be calculated without any kind of training method, so they are ready to be used. In a real-case scenario, the procedure would be to evaluate the quality of the different similarities by calculating the PCC in a development set, take the best results, and apply them to perform the inference by the system. Note that this inference could not be tested, and it might not be optimal for the new inference domain. In this experiment, we are going to simulate that environment by looking which similarity reports best results for the training set and establishing our outcome score by applying that same system to the training test.

The PCC values of the systems fed with Model 1 and Model 2 for the training and test sets are shown in table 4.2 and 4.3 respectively.

Similarity measure	*Pearson for training set*	*Pearson for test set*	∇
VA-ES	0.647	0.592	5.5
VA-CS	0.507	0.561	-5.4
CA-ES	0.683	0.643	4.0
CA-CS	0.690	0.679	1.1
COA-ES	0.712	0.658	5.4
COA-CS	0.713	0.682	3.1
COMB (VA-ES + VA-CS + COA-CS)	0.749	0.683	6.6

Table 4.2: Results of every similarity measure for the Training set (Model 1)

Similarity measure	*Pearson for training set*	*Pearson for test set*	∇
VA-ES	0.646	0.574	7.2
VA-CS	0.517	0.547	-3.0
CA-ES	0.680	0.650	3.0
CA-CS	0.697	0.682	1.5
COA-ES	0.715	0.653	6.2
COA-CS	0.721	0.681	4.0
COMB (VA-ES + VA-CS + COA-CS)	0.752	0.675	7.7

Table 4.3: Results of every similarity measure for the Training set (Model 2)

In both tables 4.2 and 4.3, the "∇" column represents the absolute decrement of the PCC of the test results when compared to the training results. However, it is worth noting that the only measure in which the training set is used to adjust parameters in order to make predictions for the evaluation set is in the combination method, since for all the other methods, only the word vectors are used, which are trained using an external source of information.

4.3 Analysis and comparison of results

From table 4.2, some prior conclusions can be taken before even comparing with the results of [47]. First and most important, it is the fact that despite the best result of our model for training is relatively good, the results of the same system in the evaluation are not that good. However, even the two systems that are independent of any external implementations details (VA-ES and VA-CS) fail to give the same results as in training, which makes us think that the training and test labeling made in SemEval 2014 and SemEval 2015 respectively, are not consistent enough, which is understandable, as calculating a bounded similarity measure between two texts can be a highly subjective task. However, if that is not taken into account, along with the fact that VA-CS gives better results on test that on training, we can see a correlation between the improvements of each system between training and test, which shows us that if the combined method in test is failing to get the same results as in training, is just because the similarity measures in which it is based are failing as well.

If we take a look at the results of [47], we will see that despite our results do not show an overall improvement when compared with the best result, they are still good if we compare them to the other SemEval 2015 teams, and that shows the strength of the method itself. Tables 4.4 and 4.5 show our results compared with the best teams of SemEval 2015 and [47].

Something that is also remarkable is the fact that the Model 2 performs worse than the Model 1 even given the fact that the word vectors have been trained with roughly 12 % more data. This shows that GloVe does not necessarily benefit from more data to train the word embeddings, unlike the experiments in [47], that show an improvement with Word2Vec when the Model 2 is used.

System	*Pearson*	Δ
Baseline	0.529	0.0
RTM-DCU	0.582	5.3
UMDuluth	0.594	6.5
COMB (VA-ES + VA-CS + COA-CS)	0.683	15.4
ExBThemis	0.706	17.7
T. López et al. [47]	**0.713**	**18.4**

Table 4.4: Comparison of our results (Model 1) with the teams at SemEval 2015 and [47]

System	*Pearson*	Δ
Baseline	0.529	0.0
RTM-DCU	0.582	5.3
UMDuluth	0.594	6.5
COMB (VA-ES + VA-CS + COA-CS)	0.675	14.6
ExBThemis	0.706	17.7
T. López et al. [47]	**0.723**	**19.4**

Table 4.5: Comparison of our results (Model 2) with the teams at SemEval 2015 and [47]

In both 4.4 and 4.5 tables, the Δ column represents the absolute improvement of the system represented of that row with respect to the baseline.

CHAPTER 5
Further experimentation

This chapter focuses on possible improvements to our method. We try different variations of the original method discussed in the last chapter and see how these changes improve or worsen the performance results of the system.

5.1 Increasing training iterations

The most straightforward method to try to obtain better results using GloVe is by increasing the training iterations when estimating the word vectors from the corpus. This has been done for the Model 1, meaning that instead of the default parameters specified in section 4.1, we trained the model with the same parameters but with `MAX_ITER=50`. Table 5.1 shows the differences between the original model and this one.

Similarity method	*Training iter.*	*Pearson for training*	*Pearson for test*
VA-ES	20	0.647	0.592
	50	0.647	0.574
VA-CS	20	0.507	0.561
	50	0.554	0.543
CA-ES	20	0.683	0.643
	50	0.676	0.650
CA-CS	20	0.690	0.679
	50	0.718	0.688
COA-ES	20	0.712	0.658
	50	0.716	0.649
COA-CS	20	0.713	0.682
	50	0.741	0.685
COMB (VA-ES + VA-CS + COA-CS)	20	0.749	0.683
	50	**0.758**	**0.686**

Table 5.1: Comparison between Model 1 trained with 20 and 50 iterations

Note that almost all methods benefit from increasing the training iterations. However, the proportion of the increased PCC is minimal, showing that the model already reached a local optimum and that those extra 30 iterations were beyond

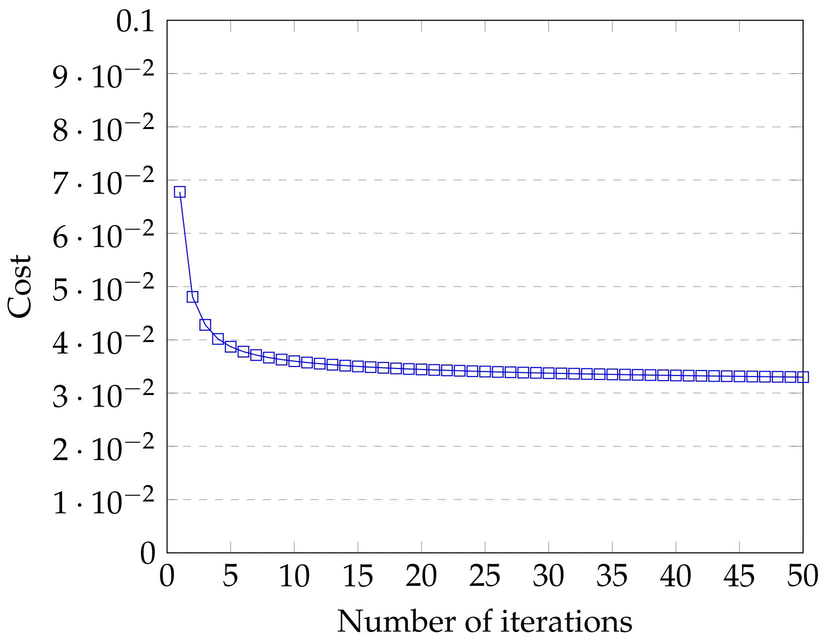

Figure 5.1: GloVe: Decreasing cost function value as iterations go through

the point where the phenomenon of diminishing returns started affecting the model. In other words, at that point, we are already out of the curve that shows the increasing performance as more iterations occur in the training phase, as can be seen in plot 5.1.

5.2 COMB method using all six similarity measures

Given the combination method used, which relies on a linear combination of the already existing similarities to build a new similarity that fits as much as possible the target similarity values, a new idea is to bring more similarities to the pool so the resulting linear expression is more versatile because of the increased number of terms. To do this, instead of doing the combination (COMB-3) of three methods (VA-ES + VA-CS + COA-CS), we combine (COMB-6) all the six methods (VA-ES + VA-CS + CA-ES + CA-CS + COA-ES + COA-CS). Table 5.2 shows the differences between the two combination methods.

Combination method	*Pearson for training set*		*Pearson for test set*	
Model	1	2	1	2
COMB-3	0.749	0.752	**0.683**	**0.675**
COMB-6	0.752	0.755	0.682	0.667

Table 5.2: Comparison between the results of the original combination method and the improved combination method

We can see a slight improvement in training, but also a slight worsening in the evaluation, probably because the linear function is overfitting. However, the changes are minimal, so we conclude that the extra information contributed to the system by the three added methods is insignificant.

5.3 Vector Aggregation without stop words

Since vector aggregation relies on building a vector that summarizes the phrase meaning, we consider that stop words do not add any useful information building phrase embeddings for two reasons: the first one is that just because stop words are very common, they appear in almost every phrase, making the resulting vectors skew in the vector space, resulting in a less informed phrase vector. The second one is that the intrinsic meaning a stop word can have makes it useless to determine a phrase meaning.

So for the two VA methods (VA-CS and VA-ES), a new method for each one is derived, VA-CS-NoSW and VA-ES-NoSW, that are the two variations but with the stop words being removed out of the phrases before joining the word vectors to build the resulting phrase embedding. The set of Spanish stop words we use is the one that can be found in the *NLTK Corpus* package of *NLTK* for *Python*. A subsample of 30 of all the 313 Spanish stop words of *NLTK* can be found in table 5.3.

Spanish stop words
de, la, que, el, en, y, a, los, del, se, las, por, un, para, con, no, una, su, al, lo, como, más, pero, sus, le, ya, o, este, sí, porque

Table 5.3: List of the Spanish stop words provided in the NLTK Corpus

The table 5.4 shows the results of both the NoSW variations with their respective original methods for both Model 1 and Model 2.

Similarity method	*Pearson for training set*		*Pearson for test set*	
Model	1	2	1	2
VA-ES	0.647	0.646	0.592	0.574
VA-ES-NoSW	**0.727**	**0.734**	**0.640**	**0.620**
VA-CS	0.507	0.517	0.561	0.547
VA-CS-NoSW	**0.708**	**0.72**	**0.643**	**0.629**

Table 5.4: Comparison between VA-NoSW methods with their respective VA method

The results above show a clear improvement in all cases of the VA methods when the stop words are removed.

5.3.1. COMB method with Vector Aggregation without stop words

Because removing the stop words has proven to be a good choice when building phrase embeddings by the means of vector aggregation, we could think that the new info provided by this two new similarity measures (VA-ES-NoSW and VA-CS-NoSW) can be good for the combination model. In this case, the six original similarity measures plus the two new ones will be used. The table 5.5 shows the

Combination method	*Pearson for training set*		*Pearson for test set*	
Model	1	2	1	2
COMB-3	0.749	0.752	0.683	0.675
COMB-8	**0.776**	**0.785**	**0.710**	**0.692**

Table 5.5: Comparison between the original combination method and the combination method of all the similarities

results of the new combination model (COMB-8) when compared to the original (COMB-3).

As the results confirm, the extra information provided by the VA-NoSW methods do increase our results by a significant amount, reaching state-of-the-art results, and almost improving the results of [47].

5.4 Exploring the Combinatorial Space of the COMB method

In section 5.3.1 we have not only shown that two better VA methods can be obtained if we remove the stop words of the phrase before building the phrase embedding, but we have also demonstrated that they add new info to the combined method, as the contribution of the two new NoSW methods introduce clear improvements to our system.

However, in section 5.2 we have seen that it is not always a good choice to add more similarities to the combined method, as overfitting can occur, making our model give good results for training but worse results for the evaluation set. In an attempt to improve our model even more, we have tried all possible combinations of the eight similarity measures to see which one is giving the best results on training and evaluate if the best result in the evaluation phase is also improved. Given N similarity models, the number of all possible combinations is determined by $\sum_{i=1}^{N} \binom{N}{i}$, which in our case is $\sum_{i=1}^{8} \binom{8}{i} = 255$.

Nevertheless, it would be naive to check every possibility of, for example, choosing only one model to create the combined model, and hope it will perform better than a more complex model combining more methods, so we have established a lower bound on the number of models that should be used to build the combined model in 5, so the number of models to evaluate is $\sum_{i=5}^{8} \binom{8}{i} = 93$.

The table 5.6 shows the results of the best combination found in training compared to our best combination until now (COMB-8).

Combination method	*Pearson for training set*		*Pearson for test set*	
Model	1	2	1	2
COMB-8	0.776	0.785	0.710	0.692
Best COMB	0.776	0.784	0.710	0.693

Table 5.6: Comparison between the best combination and COMB-8

In both models, the best combination was the result of combining all models except VA-ES, but as the results show, the difference is negligible, so we can conclude that every similarity is bringing useful information to the system, as there is no subset of combinations that performs better than just combining all the similarities.

Finally, the estimated similarity values of our best system for the pair of phrases we have shown as an example in 2.1 can be found in 5.7.

Compared pair of phrases	Target value	Estimated value
El espécimen es excepcional por las partes conservadas: un cráneo y mandíbula y un molde interno de la caja craneal. El espécimen comprende la mayor parte de la cara y mandíbula con los dientes y un molde interno de la caja craneal.	4	3.57
Time "100´´ es una lista de las 100 personas más influyentes según la revista Time. La primera lista fue publicada en 1999 con las 100 personas más infuyentes del siglo 20.	3	2.81
La "marinera´´ es un baile de pareja suelto, el más conocido de la costa del Perú. La marinera es el baile nacional del Perú, y su ejecución busca hacerse con derroche de gracia, picardía y destreza.	2	1.89
La "cripta de Santa Leocadia´´ está situada en el interior de la catedral de Oviedo, Asturias. Esteban Báthory fue sepultado en la cripta de la catedral de Wawel en Cracovia.	1	1.27
El río atraviesa la importante ciudad de Puebla de Zaragoza, la cuarta más poblada del país. El "Grêmio Esportivo Bagé´´ es un club de fútbol brasileño, de la ciudad de Bagé en el estado de Rio Grande do Sul.	0	0.46

Table 5.7: Examples of the results of our best system for the pairs of phrases shown before.

CHAPTER 6
Final conclusions and future work

6.1 Final conclusions

From this experiments we can conclude that word embeddings are indeed a powerful tool not only to capture semantic information of words, but also to build more complex systems to calculate, analyze and compare the meaning of more complex data, like phrases. We have seen that if the representation space of the vectors is big enough, we can build phrase embeddings that approximate considerably good the meaning of the phrase, showing that a correlation of 70% can be achieved when calculating the similarity between two phrases, and this is accomplished only by using a single trained word embedding model, meaning that there is no need to train any other model as extension in order to calculate these similarities.

Despite our inability to achieve better results than the best system to date, (and although we are only a tenth of a percent point worse), we have discovered ways of increasing the performance of our simple system that was built originally to compare our results with the state-of-the-art systems. This means that those improvements that we made to our model to increase its accuracy are potential improvements that can be done to other systems, like the one built with Word2Vec in [47], in order to achieve even better results. We have also seen that even if both approaches are built only upon word embeddings, the behavior of the used metrics vary, showing that the word embeddings themselves are considerably disparate.

Also, a light evaluation of the training time using GloVe has shown that training word embeddings for a large corpus like Wikipedia, made of more than 450 million words, is more than feasible, because the entire training only takes a few hours in a normal computer. Therefore, GloVe can be more attractive in those cases where several models have to be trained and/or when the corpus the model is trained with is very big.

Finally, we have seen that in the case of GloVe, extending the corpus that is used to train the word embeddings does not necessarily improve the performance, even if the text comes from standard corpora like Europarl or Ancora-ES. This is most likely because the word vectors of GloVe are more domain-specific, meaning that if texts from Wikipedia are going to be evaluated by using word embeddings, those word embeddings should also be trained from Wikipedia texts.

6.2 Future work

As future work, we would like to test our two improvements made to the first GloVe model, which are (1) the removal of the stop words before building phrase embeddings by vector aggregation and (2) considering more than three similarities to combine to calculate the new similarity, but with Word2Vec, to see if the best results of [47] can be improved.

We also establish a new line of work in which several or all the computer similarities are combined together by a more powerful machine learning method than calculating the linear least squares solutions. We are especially interested in training an Artificial Neural Network that takes as input the similarities reported by N methods and outputs a new similarity, which is desired to be as close to the target as possible.

Other alignment methods to calculate the similarity between two phrases are also a good option, since they are very likely to provide complementary information that the similarities described in this thesis do not capture, and therefore potentially improve any further combination methods. However, care must be taken when learning methods based on combinations to prevent overfitting as much as possible.

Finally, we would like to evaluate the steps taken in this work in English and see if the results are similar or not, and if they are, compare the results of our system in English with state-of-the-art systems in English semantic text similarity.

Bibliography

[1] Eneko Agirre, Carmen Banea, Claire Cardie, Daniel M Cer, Mona T Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Inigo Lopez-Gazpio, Montse Maritxalar, Rada Mihalcea, et al., *Semeval-2015 task 2: Semantic textual similarity, english, spanish and pilot on interpretability.*, SemEval@ NAACL-HLT, 2015, pp. 252–263.

[2] Satanjeev Banerjee and Ted Pedersen, *An adapted lesk algorithm for word sense disambiguation using wordnet*, International Conference on Intelligent Text Processing and Computational Linguistics, Springer, 2002, pp. 136–145.

[3] Daniel Bär, Torsten Zesch, and Iryna Gurevych, *Dkpro similarity: An open source framework for text similarity.*, ACL (Conference System Demonstrations), 2013, pp. 121–126.

[4] Marco Baroni, Georgiana Dinu, and Germán Kruszewski, *Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors.*, ACL (1), 2014, pp. 238–247.

[5] Alberto Barrón-Cedeno, Paolo Rosso, Eneko Agirre, and Gorka Labaka, *Plagiarism detection across distant language pairs*, Proceedings of the 23rd International Conference on Computational Linguistics, Association for Computational Linguistics, 2010, pp. 37–45.

[6] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen, *Pearson correlation coefficient*, pp. 1–4, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[7] Samy Bengio and Georg Heigold, *Word embeddings for speech recognition*, Fifteenth Annual Conference of the International Speech Communication Association, 2014.

[8] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin, *A neural probabilistic language model*, Journal of machine learning research **3** (2003), no. Feb, 1137–1155.

[9] Giacomo Berardi, Andrea Esuli, and Diego Marcheggiani, *Word embeddings go to italy: A comparison of models and training datasets.*, IIR, 2015.

[10] Ergun Biçici, *Rtm-dcu: Predicting semantic similarity with referential translation machines*, (2015).

[11] Kuan-Yu Chen, Shih-Hung Liu, Berlin Chen, Hsin-Min Wang, and Hsin-Hsi Chen, *Novel word embedding and translation-based language modeling for extractive speech summarization*, Proceedings of the 2016 ACM on Multimedia Conference, ACM, 2016, pp. 377–381.

[12] Rudi L Cilibrasi and Paul MB Vitanyi, *The google similarity distance*, IEEE Transactions on knowledge and data engineering **19** (2007), no. 3.

[13] Stéphane Clinchant and Florent Perronnin, *Aggregating continuous word embeddings for information retrieval*, Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality, 2013, pp. 100–109.

[14] William W Cohen, *Data integration using similarity joins and a word-based information representation language*, ACM Transactions on Information Systems (TOIS) **18** (2000), no. 3, 288–321.

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

[16] Ruiji Fu, Jiang Guo, Bing Qin, Wanxiang Che, Haifeng Wang, and Ting Liu, *Learning semantic hierarchies via word embeddings.*, ACL (1), 2014, pp. 1199–1209.

[17] Evgeniy Gabrilovich and Shaul Markovitch, *Computing semantic relatedness using wikipedia-based explicit semantic analysis.*, IJcAI, vol. 7, 2007, pp. 1606–1611.

[18] Daniel Gildea and Daniel Jurafsky, *Automatic labeling of semantic roles*, Computational linguistics **28** (2002), no. 3, 245–288.

[19] Wael H Gomaa and Aly A Fahmy, *A survey of text similarity approaches*, International Journal of Computer Applications **68** (2013), no. 13.

[20] Patrick AV Hall and Geoff R Dowling, *Approximate string matching*, ACM computing surveys (CSUR) **12** (1980), no. 4, 381–402.

[21] Christian Hänig, Robert Remus, and Xose De La Puente, *Exb themis: Extensive feature extraction from word alignments for semantic textual similarity.*, SemEval@ NAACL-HLT, 2015, pp. 264–268.

[22] Graeme Hirst, David St-Onge, et al., *Lexical chains as representations of context for the detection and correction of malapropisms*, WordNet: An electronic lexical database **305** (1998), 305–332.

[23] Aminul Islam and Diana Inkpen, *Second order co-occurrence pmi for determining the semantic similarity of words*, Proceedings of the International Conference on Language Resources and Evaluation, 2006, pp. 1033–1038.

[24] ———, *Semantic text similarity using corpus-based word similarity and string similarity*, ACM Transactions on Knowledge Discovery from Data (TKDD) **2** (2008), no. 2, 10.

[25] Matthew A Jaro, *Probabilistic linkage of large public health data files*, Statistics in medicine **14** (1995), no. 5-7, 491–498.

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

[27] Daniel Jurafsky and James H. Martin, *Speech and language processing*, Always learning, Prentice Hall, Pearson Education International, 2014.

[28] Sakethram Karumuri, Viswanadh Kumar Reddy Vuggumudi, and Sai Charan Raj Chitirala, *Umduluth-blueteam: Svcsts-a multilingual and chunk level semantic similarity system.*, SemEval@ NAACL-HLT, 2015, pp. 107–110.

[29] Tom Kenter and Maarten De Rijke, *Short text similarity with word embeddings*, Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, ACM, 2015, pp. 1411–1420.

[30] Youngjoong Ko, Jinwoo Park, and Jungyun Seo, *Improving text categorization using the importance of sentences*, Information processing & management **40** (2004), no. 1, 65–79.

[31] Peter Kolb, *Disco: A multilingual database of distributionally similar words*, Proceedings of KONVENS-2008, Berlin (2008).

[32] ______, *Experiments on the difference between semantic similarity and relatedness*, (2009).

[33] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger, *From word embeddings to document distances*, International Conference on Machine Learning, 2015, pp. 957–966.

[34] Thomas K Landauer and Susan T Dumais, *A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge.*, Psychological review **104** (1997), no. 2, 211.

[35] Mirella Lapata and Regina Barzilay, *Automatic evaluation of text coherence: Models and representations*, IJCAI, vol. 5, 2005, pp. 1085–1090.

[36] C Leacock and M Chodorow, *Combining local context and wordnet sense similarity for word sense identification. wordnet, an electronic lexical database*, 1998.

[37] Michael Lesk, *Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone*, Proceedings of the 5th annual international conference on Systems documentation, ACM, 1986, pp. 24–26.

[38] Yuhua Li, Zuhair A Bandar, and David McLean, *An approach for measuring semantic similarity between words using multiple information sources*, IEEE Transactions on knowledge and data engineering **15** (2003), no. 4, 871–882.

[39] Chin-Yew Lin and Eduard Hovy, *Automatic evaluation of summaries using n-gram co-occurrence statistics*, Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1, Association for Computational Linguistics, 2003, pp. 71–78.

[40] Dekang Lin, *Extracting collocations from text corpora*, First workshop on computational terminology, Montreal, Canada, 1998, pp. 57–63.

[41] Bing Liu, *Sentiment analysis and opinion mining (introduction and survey)*, Morgan & Claypool Publishers.

[42] Hugo Liu and Push Singh, *Commonsense reasoning in and over natural language*, Knowledge-based intelligent information and engineering systems, Springer, 2004, pp. 293–306.

[43] Tao Liu and Jun Guo, *Text similarity computing based on standard deviation*, Advances in Intelligent Computing (2005), 456–464.

[44] Ying Liu and Chengqing Zong, *Example-based chinese-english mt*, Systems, Man and Cybernetics, 2004 IEEE International Conference on, vol. 7, IEEE, 2004, pp. 6093–6096.

[45] Kevin Lund and Curt Burgess, *Producing high-dimensional semantic spaces from lexical co-occurrence*, Behavior Research Methods, Instruments, & Computers **28** (1996), no. 2, 203–208.

[46] Kevin Lund, Curt Burgess, and Ruth Ann Atchley, *Semantic and associative priming in high-dimensional semantic space*, Proceedings of the 17th annual conference of the Cognitive Science Society, vol. 17, 1995, pp. 660–665.

[47] Tomás López Solaz, José Antonio Troyano Jiménez, Francisco Javier Ortega Rodríguez, and Fernando Enríquez de Salamanca Ros, *An approach to the use of word embeddings in a textual similarity task for spanish texts*, Procesamiento del Lenguaje Natural (2016), no. 57, 67–74.

[48] Jayant Madhavan, Philip A Bernstein, AnHai Doan, and Alon Halevy, *Corpus-based schema matching*, Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on, IEEE, 2005, pp. 57–68.

[49] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze, *Introduction to information retrieval*, Cambridge University Press, New York, NY, USA, 2008.

[50] Katja Markert and Malvina Nissim, *Metonymy resolution as a classification task*, 2002.

[51] Irina Matveeva, Gina-Anne Levow, Ayman Farahat, and Christian Royer, *Generalized latent semantic analysis for term representation*, Proc. of RANLP, 2005.

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

[53] Tomas Mikolov and One Hacker Way, *Using neural networks for modeling and representing natural languages.*, COLING (Tutorials), 2014, pp. 3–4.

[54] George A Miller, *Wordnet: a lexical database for english*, Communications of the ACM **38** (1995), no. 11, 39–41.

[55] Andriy Mnih and Geoffrey E Hinton, *A scalable hierarchical distributed language model*, Advances in neural information processing systems, 2009, pp. 1081–1088.

[56] Frederic Morin and Yoshua Bengio, *Hierarchical probabilistic neural network language model.*, Aistats, vol. 5, 2005, pp. 246–252.

[57] David Nadeau and Satoshi Sekine, *A survey of named entity recognition and classification*, Lingvisticae Investigationes **30** (2007), no. 1, 3–26.

[58] Eric Nalisnick, Bhaskar Mitra, Nick Craswell, and Rich Caruana, *Improving document ranking with dual word embeddings*, Proceedings of the 25th International Conference Companion on World Wide Web, International World Wide Web Conferences Steering Committee, 2016, pp. 83–84.

[59] Roberto Navigli, *Word sense disambiguation: A survey*, ACM Computing Surveys **41** (2009), no. 2, 1–69 (en).

[60] Jian-Yun Nie, *Cross-Language Information Retrieval*, Synthesis Lectures on Human Language Technologies **3** (2010), no. 1, 1–125 (en).

[61] Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Yi Zhang, *SemEval 2014 Task 8: Broad-coverage semantic dependency parsing*, Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014), 2014, pp. 63–72.

[62] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu, *Bleu: a method for automatic evaluation of machine translation*, Proceedings of the 40th annual meeting on association for computational linguistics, Association for Computational Linguistics, 2002, pp. 311–318.

[63] Eui-Kyu Park, Dong-Yul Ra, and Myung-Gil Jang, *Techniques for improving web retrieval effectiveness*, Information processing & management **41** (2005), no. 5, 1207–1223.

[64] Alan Parker and James O. Hamblen, *Computer algorithms for plagiarism detection*, IEEE Transactions on Education (IEEE T EDUC) **14**, no. 2, 94–99.

[65] Siddharth Patwardhan, *Incorporating dictionary and corpus information into a context vector measure of semantic relatedness*, Ph.D. thesis, University of Minnesota, Duluth, 2003.

[66] Siddharth Patwardhan, Satanjeev Banerjee, and Ted Pedersen, *Using measures of semantic relatedness for word sense disambiguation*, CICLing, vol. 2588, Springer, 2003, pp. 241–257.

[67] Jeffrey Pennington, Richard Socher, and Christopher D Manning, *Glove: Global vectors for word representation.*, EMNLP, vol. 14, 2014, pp. 1532–1543.

[68] Martin Potthast, Benno Stein, and Maik Anderka, *A wikipedia-based multilingual retrieval model*, Advances in Information Retrieval (2008), 522–530.

[69] Francisco Rangel, Paolo Rosso, Moshe Moshe Koppel, Efstathios Stamatatos, and Giacomo Inches, *Overview of the author profiling task at pan 2013*, CLEF Conference on Multilingual and Multimodal Information Access Evaluation, CELCT, 2013, pp. 352–365.

[70] Philip Resnik, *Using information content to evaluate semantic similarity in a taxonomy*, arXiv preprint cmp-lg/9511007 (1995).

[71] Hinrich Schütze, *Automatic word sense discrimination*, Computational linguistics **24** (1998), no. 1, 97–123.

[72] Richard Socher, Francois Chaubard, and Rohit Mundra, *CS 224d: Deep Learning for NLP1*, (2016).

[73] Lucia Specia, Sujay Kumar Jauhar, and Rada Mihalcea, *Semeval-2012 task 1: English lexical simplification.*, SemEval@NAACL-HLT (Eneko Agirre, Johan Bos, and Mona T. Diab, eds.), The Association for Computer Linguistics, 2012, pp. 347–355.

[74] Peter Turney, *Mining the web for synonyms: Pmi-ir versus lsa on toefl*, Machine Learning: ECML 2001 (2001), 491–502.

[75] Katarzyna Wegrzyn-Wolska and Piotr S Szczepaniak, *Classification of rss-formatted documents using full text similarity measures*, International Conference on Web Engineering, Springer, 2005, pp. 400–405.

[76] Yonghui Wu, Jun Xu, Yaoyun Zhang, and Hua Xu, *Clinical abbreviation disambiguation using neural word embeddings*, Proceedings of the 2015 Workshop on Biomedical Natural Language Processing (BioNLP), 2015, pp. 171–176.

[77] Zhibiao Wu and Martha Palmer, *Verbs semantics and lexical selection*, Proceedings of the 32nd annual meeting on Association for Computational Linguistics, Association for Computational Linguistics, 1994, pp. 133–138.

[78] Will Y Zou, Richard Socher, Daniel M Cer, and Christopher D Manning, *Bilingual word embeddings for phrase-based machine translation.*, EMNLP, 2013, pp. 1393–1398.

APPENDIX A
Auxiliar scripts

A.1 remove_tokens.py

```python
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

import re
import argparse
import os
from time import time
from nltk.tokenize import RegexpTokenizer

def parse_args():
    parser = argparse.ArgumentParser(description='Remove tokens and normalize text')
    parser.add_argument('input_path', metavar='input',
                        help='relative path to the input file')
    parser.add_argument('output_path', metavar='output',
                        help='relative path to the output file')
    return parser.parse_args()

def deleteContent(pfile):
    pfile.seek(0)
    pfile.truncate()

if __name__ == "__main__":
     args = parse_args()

     f = open(args.input_path, 'r')
     o = open(args.output_path, 'a')
     deleteContent(o)

     print ('Removing tokens...')
     t0 = time()

     for line in f.readlines():
```

```
        line = (re.sub(u'''<[^>]*>|-rrb-|-lrb-|-rsb-|-lsb-|\.|,|°''', '', line))
        tokenizer = RegexpTokenizer('[a-záéíóúàèìòù0-9ñçŭĭńŕýĺǵśźćńüöëïäčďěňřšťůž]+')
        line = tokenizer.tokenize(line)
        line = ' '.join(line)

        line = (re.sub('\s\s+|\n|\r', ' ', line))
        if(line not in ["", " ", "\n", "\r"]):
            o.write(" "+line)

    f.close()
    o.close()
    print ('Done in %.3f seconds.' % (time()-t0))
    print ('Results saved in file "%s"' % args.output_path)
```

A.2 experiment.py

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

from __future__ import division
import argparse
import re
import numpy as np

from scipy.spatial.distance import cosine
from scipy.spatial.distance import euclidean
from scipy.stats import pearsonr

from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords


parser = argparse.ArgumentParser()
parser.add_argument('--vocab_file', default='vocab.txt', type=str)
parser.add_argument('--vectors_file', default='vectors.txt', type=str)
parser.add_argument('--training_X',
default='../data/train/input_wikipedia.txt', type=str)
parser.add_argument('--training_y',
default='../data/train/tags_wikipedia.txt', type=str)
parser.add_argument('--test_X',
default='../data/test/input_wikipedia.txt', type=str)
parser.add_argument('--test_y',
default='../data/test/tags_wikipedia.txt', type=str)
parser.add_argument('--verbose', default='0', type=int)


args = parser.parse_args()
verbose = args.verbose

print('\nReading vocabulary file...')
with open(args.vocab_file, 'r') as f:
    words = [x.rstrip().split(' ')[0] for x in f.readlines()]
print('Reading vectors file...\n')
with open(args.vectors_file, 'r') as f:
    vectors = {}
    for line in f:
        vals = line.rstrip().split(' ')
        vectors[vals[0]] = map(float, vals[1:])

vocab_size = len(words)
# vocab takes the word as string and returns the ID
```

```
vocab = {w: idx for idx, w in enumerate(words)}
# ivocab takes the ID and returns the word
ivocab = {idx: w for idx, w in enumerate(words)}

vector_dim = len(vectors[ivocab[0]])
W = np.zeros((vocab_size, vector_dim))
for word, v in vectors.iteritems():
    if word == '<unk>':
        continue
    W[vocab[word], :] = v

# Normalize each word vector to unit variance
print('Normalizing vectors...')
W_norm = np.zeros(W.shape)
d = (np.sum(W ** 2, 1) ** (0.5))
W_norm = (W.T / d).T # W_norm takes the ID and returns the word embedding

# Summary:
# vocab: takes the word and returns ID
# W_norm: takes the ID and returns the embedding

def get_embedding(word):
    return W_norm[vocab[word]]

with open(args.training_X,'r') as f:
    content = f.readlines()
    X_train = [x.strip() for x in content]
    X_train = [x.split('\t') for x in X_train]

with open(args.training_y,'r') as f:
    content = f.readlines()
    y_train = [x.strip() for x in content]
    y_train = [float(x) for x in y_train]

with open(args.test_X,'r') as f:
    content = f.readlines()
    X_test = [x.strip() for x in content]
    X_test = [x.split('\t') for x in X_test]

with open(args.test_y,'r') as f:
    content = f.readlines()
    y_test = [x.strip() for x in content]
    y_test = [float(x) for x in y_test]

def clean_text(text):

    tokenizer = RegexpTokenizer(
    '[a-záéíóúàèìòù0-9ñçŭĭńŕýĺǵśźćńüöëïäčďěňřšťůž]+')

```

```
    cleaned_text = re.sub(''':|;|'|"|\?|¿|¡|!|$|%|/|\(|\)|\-|
    \.|,|º|...|#|«|»|¡|²|³''', '', text)
    cleaned_text = re.sub('Á', 'á', cleaned_text)
    cleaned_text = re.sub('É', 'é', cleaned_text)
    cleaned_text = re.sub('Í', 'í', cleaned_text)
    cleaned_text = re.sub('Ó', 'ó', cleaned_text)
    cleaned_text = re.sub('Ú', 'ú', cleaned_text)

    cleaned_text = cleaned_text.lower()

    cleaned_text = tokenizer.tokenize(cleaned_text)
    cleaned_text = ' '.join(cleaned_text)

    return cleaned_text

for i in range(len(X_train)):
    X_train[i][0] = clean_text(X_train[i][0])
    X_train[i][1] = clean_text(X_train[i][1])

for i in range(len(X_test)):
    X_test[i][0] = clean_text(X_test[i][0])
    X_test[i][1] = clean_text(X_test[i][1])

for split in ['train', 'test']:

    if(split == 'train'):
        X = X_train
        y = y_train
    elif(split == 'test'):
        X = X_test
        y = y_test
        # Word embeddings of the train/test phrases
    X_WE = []
    # Word embeddings of the train/test phrases without stop words
    # (for vector aggregation purposes)
    X_WE_no_sw = []

    string_sw_vector = []

    for i in range(len(X)): # for each couple of phrases
        firstPhraseEmbeddings = []
        secondPhraseEmbeddings = []

        firstPhraseEmbeddings_no_sw = []
        secondPhraseEmbeddings_no_sw = []

        for j in range(len(X[i][0].split(' '))): # 1st phrase
            word = X[i][0].split(' ')[j]
            try:
```

```
            embedding = get_embedding(word)
            firstPhraseEmbeddings.append(embedding)
            if word not in stopwords.words('spanish'):
                firstPhraseEmbeddings_no_sw.append(embedding)
        except KeyError:
            if verbose > 0:
                print '''KeyError: Word Embedding of word "%s"
                cannot be found. Skipping word...''' % word

            # phrase to compare (2nd)
    for j in range(len(X[i][1].split(' '))):
        word = X[i][1].split(' ')[j]
        try:
            embedding = get_embedding(word)
            secondPhraseEmbeddings.append(embedding)
            if word not in stopwords.words('spanish'):
                secondPhraseEmbeddings_no_sw.append(embedding)
        except KeyError:
            if verbose > 0:
                print '''KeyError: Word Embedding of word "%s"
                cannot be found. Skipping word...''' % word

    X_WE.append([firstPhraseEmbeddings, secondPhraseEmbeddings])
    X_WE_no_sw.append([firstPhraseEmbeddings_no_sw,
    secondPhraseEmbeddings_no_sw])

    # Phrase Embeddings of the train/test phrases
X_PE = []
# Phrase Embeddings without taking into account stopwords
X_PE_no_sw = []

# Build phrase embeddings by vector aggregation (summing all
# vectors in a phrase and dividing by the number of summed vectors)
for dualphrase in X_WE:
    phrase1 = dualphrase[0]
    phrase2 = dualphrase[1]
    summed_embeddings1 = map(sum, zip(*phrase1))
    summed_embeddings2 = map(sum, zip(*phrase2))

    summed_embeddings1 = [x/len(dualphrase[0]) for x in
    summed_embeddings1]
    summed_embeddings2 = [x/len(dualphrase[1]) for x in
    summed_embeddings2]

    X_PE.append([summed_embeddings1, summed_embeddings2])

# Build phrase embeddings without stopwords
for dualphrase in X_WE_no_sw:
    phrase1 = dualphrase[0]
```

```
        phrase2 = dualphrase[1]
        summed_embeddings1 = map(sum, zip(*phrase1))
        summed_embeddings2 = map(sum, zip(*phrase2))

        summed_embeddings1 = [x/len(dualphrase[0]) for x in
        summed_embeddings1]
        summed_embeddings2 = [x/len(dualphrase[1]) for x in
        summed_embeddings2]

        X_PE_no_sw.append([summed_embeddings1, summed_embeddings2])


    # remember: y is the vector of true scores for X
    cosine_similarities = []
    euclidean_similarities = []

    cosine_similarities_no_sw = []
    euclidean_similarities_no_sw = []

    for i in range(len(X_PE)):
        cos_similarity = (1.0-cosine(X_PE[i][0], X_PE[i][1]))
        cosine_similarities.append(cos_similarity)

        # Euclidean similarity based on Euclidean distance.
        # This is only one of many ways
        euc_similarity = (1/(1 + euclidean(X_PE[i][0], X_PE[i][1])))
        euclidean_similarities.append(euc_similarity)

    # In another loop for readability purposes
    for i in range(len(X_PE_no_sw)):
        cos_similarity = (1.0-cosine(X_PE_no_sw[i][0], X_PE_no_sw[i][1]))
        cosine_similarities_no_sw.append(cos_similarity)

        euc_similarity = (1/(1 + euclidean(X_PE_no_sw[i][0], X_PE_no_sw[i][1])))
        euclidean_similarities_no_sw.append(euc_similarity)





    print ('\n')
    print ('########################################')
    if(split=='train'):
        print ('############# Training set ############')
    elif(split=='test'):
        print ('############### Test set ###############')
    print ('########################################')

```

```
    print ('\n')
    print ('Pearson Correlation Coefficient for Euclidean: %.3f' %
    pearsonr(euclidean_similarities, y)[0])
    print ('Pearson Correlation Coefficient for Cosine: %.3f' %
    pearsonr(cosine_similarities, y)[0])
    print ('Pearson Correlation Coefficient for Euclidean (no stopwords): %.3f' %
    pearsonr(euclidean_similarities_no_sw, y)[0])
    print ('Pearson Correlation Coefficient for Cosine (no stopwords): %.3f' %
    pearsonr(cosine_similarities_no_sw, y)[0])


    aligned_cosine_similarities_COA = []
    aligned_euclidean_similarities_COA = []

    aligned_cosine_similarities_CA = []
    aligned_euclidean_similarities_CA = []

    for dualphrase in X:
        phrase1 = dualphrase[0].split()
        phrase2 = dualphrase[1].split()

        t1 = set(phrase1)
        t2 = set(phrase2)

        #vocabset = t1.union(t2)

        bow_euclidean = {}
        bow_cosine = {}

        # Initialize dictionaries
        for w in t1:
            bow_euclidean[w] = 0
            bow_cosine[w] = 0


        for w in t2:
            bow_euclidean[w] = 0
            bow_cosine[w] = 0

        # Alignment algorithm described in Lopez-Solaz, T. et al.
        for w in t1:
            if w in t2:
                bow_euclidean[w] = 1
                bow_cosine[w] = 1
            elif w in vocab: # Check if there is a word embedding in our model
                embedding = get_embedding(w)
                # Calculate the similarity between w and each word of phrase2
                embeddings_phrase2 = [get_embedding(x) for x in t2 if x
                in vocab]
```

```
                cos_sims = [1.0-cosine(embedding, x) for x in
                embeddings_phrase2]
                euc_sims = [(1.0/(1 + euclidean(embedding, x)))
                for x in embeddings_phrase2]
                max_cosine_sim = max(cos_sims)
                max_euclidean_sim = max(euc_sims)

                bow_euclidean[w] = max_euclidean_sim
                bow_cosine[w] = max_cosine_sim

            else:
                continue

        for w in t2:
            if w in vocab:
                embedding = get_embedding(w)
                # Calculate the similarity between w and each word of phrase1
                embeddings_phrase1 = [get_embedding(x) for x in t1 if x
                in vocab]
                cos_sims = [1.0-cosine(embedding, x) for x
                in embeddings_phrase1]
                euc_sims = [(1.0/(1 + euclidean(embedding, x))) for x
                in embeddings_phrase1]
                max_cosine_sim = max(cos_sims)
                max_euclidean_sim = max(euc_sims)

                bow_euclidean[w] = max_euclidean_sim
                bow_cosine[w] = max_cosine_sim

            else:
                continue

        # COA = Counting Only Appearances
        values_cos_COA = [bow_cosine[w] for w in bow_cosine.keys()
        if bow_cosine[w] > 0] # select those similarities that are not 0
        values_euc_COA = [bow_euclidean[w] for w in bow_euclidean.keys()
        if bow_euclidean[w] > 0]

        aligned_cosine_similarity_COA = sum(
        values_cos_COA)/len(values_cos_COA)
        aligned_euclidean_similarity_COA = sum(
        values_euc_COA)/len(values_euc_COA)

        aligned_cosine_similarities_COA.append(
        aligned_cosine_similarity_COA)
        aligned_euclidean_similarities_COA.append(
        aligned_euclidean_similarity_COA)

        # CA = Counting All
```

```
        values_cos_CA = [bow_cosine[w] for w in bow_cosine.keys()]
        values_euc_CA = [bow_euclidean[w] for w in bow_euclidean.keys()]

        aligned_cosine_similarity_CA = sum(
        values_cos_CA)/len(values_cos_CA)
        aligned_euclidean_similarity_CA = sum(
        values_euc_CA)/len(values_euc_CA)

        aligned_cosine_similarities_CA.append(
        aligned_cosine_similarity_CA)
        aligned_euclidean_similarities_CA.append(
        aligned_euclidean_similarity_CA)


    print ('''Pearson Correlation Coefficient for Aligned,
    Counting Only Appearances (Euclidean): %.3f''' %
    pearsonr(aligned_euclidean_similarities_COA, y)[0])
    print ('''Pearson Correlation Coefficient for Aligned,
    Counting Only Appearances (Cosine): %.3f''' %
    pearsonr(aligned_cosine_similarities_COA, y)[0])

    print ('''Pearson Correlation Coefficient for Aligned,
    Counting All (Euclidean): %.3f''' %
    pearsonr(aligned_euclidean_similarities_CA, y)[0])
    print ('''Pearson Correlation Coefficient for Aligned,
    Counting All (Cosine): %.3f''' %
    pearsonr(aligned_cosine_similarities_CA, y)[0])


    x1 = euclidean_similarities
    x2 = cosine_similarities
    x3 = euclidean_similarities_no_sw
    x4 = cosine_similarities_no_sw
    x5 = aligned_euclidean_similarities_COA
    x6 = aligned_cosine_similarities_COA
    x7 = aligned_euclidean_similarities_CA
    x8 = aligned_cosine_similarities_CA

    # Original experiment
    # x = [x1, x2, x6]

    # COMB method using all six similarity measures
    # x = [x1, x2, x5, x6, x7, x8]

    # COMB method with Vector Aggregation without stop words
    x = [x1, x2, x3, x4, x5, x6, x7, x8]


    # Stack 1-D arrays of similarities as columns into a 2-D array
```

```
    X = np.column_stack(x+[[1]*len(x[0])])

    if(split=='train'):
        beta_hat = np.linalg.lstsq(X,y)[0]

    print ('LLS solution weights = ' + str(beta_hat))

    estimated_output = np.dot(X,beta_hat)

    print ('''Pearson Correlation Coefficient for Linear Least Squares
    (Using training set solution (weights) to LLS): %.3f''' %
     pearsonr(estimated_output, y)[0])
```