University Institute of Lisbon

Department of Information Science and Technology

Sparse Distributed Representations as Word Embeddings for Language Understanding

André de Vasconcelos Santos Silva

A dissertation submitted in partial fulfillment of the requirements for the Degree of

Master in Computer Engineering

Supervisor

Doctor Ricardo Daniel Santos Faro Marques Ribeiro,

Assistant Professor

ISCTE-IUL

October, 2018

Acknowledgements

I would like to thank my wife and son for being so comprehensive and supportful throughout this year of intense work. They have been my source of motivation and inspiration that helped me accomplish all my objectives.

I'm also grateful to my supervisor Prof. Ricardo Ribeiro who always guided me to the right direction. Without his guidance I wouldn´t have achieved my research goals.

Last but not least a very special gratitude for my parents, from whom I owe everything.

Resumo

A designação *word embeddings* refere-se a representações vetoriais das palavras que capturam as similaridades semânticas e sintáticas entre estas. Palavras similares tendem a ser representadas por vetores próximos num espaço *N* dimensional considerando, por exemplo, a distância Euclidiana entre os pontos associados a estas representações vetoriais num espaço vetorial contínuo. Esta propriedade, torna as *word embeddings* importantes em várias tarefas de Processamento Natural da Língua, desde avaliações de analogia e similaridade entre palavras, às mais complexas tarefas de categorização, sumarização e tradução automática de texto.

Tipicamente, as *word embeddings* são constituídas por vetores densos, de dimensionalidade reduzida. São obtidas a partir de aprendizagem não supervisionada, recorrendo a consideráveis quantidades de dados, através da otimização de uma função objetivo de uma rede neuronal.

Este trabalho propõe uma metodologia para obter *word embeddings* constituídas por vetores binários esparsos, ou seja, representações vetoriais das palavras simultaneamente binárias (e.g. compostas apenas por zeros e uns), esparsas e com elevada dimensionalidade. A metodologia proposta tenta superar algumas desvantagens associadas às metodologias do estado da arte, nomeadamente o elevado volume de dados necessário para treinar os modelos, e simultaneamente apresentar resultados comparáveis em várias tarefas de Processamento Natural da Língua.

Os resultados deste trabalho mostram que estas representações, obtidas a partir de uma quantidade limitada de dados de treino, obtêm performances consideráveis em tarefas de similaridade e categorização de palavras. Por outro lado, em tarefas de analogia de palavras apenas se obtém resultados consideráveis para a categoria gramatical dos substantivos. As *word embeddings* obtidas com a metodologia proposta, e comparando com o estado da arte, superaram a performance de oito *word embeddings* em tarefas de similaridade, e de duas *word embeddings* em tarefas de categorização de palavras.

Palavras-chave: Representação vetorial de palavras, Modelo Semântico Distribuído, Clustering de Texto, Vetores binários esparsos, Redes Neuronais

Abstract

Word embeddings are vector representations of words that capture semantic and syntactic similarities between them. Similar words tend to have closer vector representations in a N dimensional space considering, for instance, Euclidean distance between the points associated with the word vector representations in a continuous vector space. This property, makes word embeddings valuable in several Natural Language Processing tasks, from word analogy and similarity evaluation to the more complex text categorization, summarization or translation tasks.

Typically state of the art word embeddings are dense vector representations, with low dimensionality varying from tens to hundreds of floating number dimensions, usually obtained from unsupervised learning on considerable amounts of text data by training and optimizing an objective function of a neural network.

This work presents a methodology to derive word embeddings as binary sparse vectors, or word vector representations with high dimensionality, sparse representation and binary features (e.g. composed only by ones and zeros). The proposed methodology tries to overcome some disadvantages associated with state of the art approaches, namely the size of corpus needed for training the model, while presenting comparable evaluations in several Natural Language Processing tasks.

Results show that high dimensionality sparse binary vectors representations, obtained from a very limited amount of training data, achieve comparable performances in similarity and categorization intrinsic tasks, whereas in analogy tasks good results are obtained only for nouns categories. Our embeddings outperformed eight state of the art word embeddings in word similarity tasks, and two word embeddings in categorization tasks.

Keywords: Word Embedding, Distributional Semantic Model, Text Clustering, Binary Sparse Vectors, Neural Networks

Contents

List of Tables

List of Figures

Abbreviations

HTM	Hierarchical Temporal Memory
LSA	Latent Semantic Analysis
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NMT	Neural Machine Translation
PMI	Pointwise Mutual Information
SFT	Semantic Folding Theory
SDR	Sparse Distributed Representations
SOM	Self-Organizing Map
SVD	Singular Value Decomposition
TF-IDF	Term Frequency – Inverse Document Frequency
VSM	Vector Space Model

1 Introduction

The master thesis research area is on the field of distributional semantics and aims to explore an alternative approach for word representations in the semantic vector space. The proposed methodology goal is to derive sparse distributed representations (SDR) as word embeddings. The evaluation of the proposed methodology is made by testing the resulting word embeddings in several Natural Language Processing tasks, and comparing the results with state of the art dense vector representations, such as Word2vec [53], FastText [9] and Glove [60].

A methodology for obtaining a binary sparse distributed representation for each word is herein proposed. This representation shares the main idea of a word embedding, which was first formulated by Firth in 1957: "a word is characterized by the company it keeps" [24].

The proposed method for deriving sparse distributed representations is inspired on Francisco Webber's work on the Semantic Folding Theory [81]. His work has the theoretical roots based on Hierarchical Temporal Memory (HTM) concepts described on the book "On Intelligence" [28], where is proposed a neuron model for machine intelligence inspired by real biological characteristics of the brain.

The properties of the derived sparse binary representations are discussed and compared with state of the art dense vector representations. By applying simple evaluation metrics, is possible to compare the proposed methodology and resulting word vectors with state of the art word embeddings, in the context of intrinsic natural language processing tasks (e.g. word similarity, word analogy and word categorization).

1.1 Context and Motivation

Typically, state of the art word embeddings are dense vector representations for each word in a given vocabulary. They are learned from large volumes of unstructured text data and are widely used in Natural Language Processing (NLP). The theory behind word embeddings is related with the distributional hypothesis, which states that the words that occur in the same context tend to have similar meanings. In practice these vector representations encode the semantic meaning of words, such as similar semantic words have close vector representations. Hence, word embeddings have a broad range of applications in Natural Language Processing, from simple word similarity, analogy or categorization tasks to complex document level tasks (e.g. classification, summarization, translation, intent detection). In the latter case, "word embeddings are often used as the first data processing layer in a deep learning model [82]" while in the former a simple distance metric (e.g. cosine distance) can be used.

State of the art algorithms for deriving word embeddings language models are mainly based on artificial neural networks. These word embeddings are typically dense vector representations, which require very large training times and corpus size in order to produce an acceptable word vector representation.

Training word embeddings on small size datasets does not produce good results, therefore "investigators have to use pre-trained word vectors such as Word2Vec and GloVe, which may not be the best fit for their data [63]".

Another shortcoming of most word embeddings is the fact that "they assume each word preserves a single vector, which is problematic due to homonymy and polysemy [44]". In practice these methods do not allow the dense vector representation to capture all the contextual meaning of a word.

The motivation for this work arises from the intent of proposing an alternative methodology to the state of the art artificial neural network word embeddings. This work proposes a new approach trying to reject the idea that huge volumes of data beats better algorithms. Although a difficult challenge, our aim is to propose an alternative methodology capable of having good performance in intrinsic Natural Processing Language tasks (e.g. word analogy, word similarity, word categorization), while overcoming the disadvantages of existing word embeddings, mentioned above.

1.2 Research Questions and Objectives

This work tries to answer three research questions. The first one addresses the feasibility of using text clustering techniques and concepts brought by semantic folding theory to produce word embeddings. Semantic folding proposes a distinct approach to derive word embeddings, when compared to state of the art methodologies. In contrast with neural network approaches, it preconizes document clustering as the fundamental method for deriving word embeddings. Another important difference is that the resulting word embeddings are binary sparse vectors in opposition with the short and dense state of the art embeddings. We propose a methodology for deriving binary sparse embeddings, address the feasibility of such a process and the advantages of the derived word embeddings in terms of memory size needed to store these representations, when compared to the traditional dense vector representations. This constitutes the first contribution of this work.

The second research question and contribution of this work, deals with the size of training material and tries to draw conclusions on the possibility of deriving word representations with less training material, allowing a compromise between corpora size and performance in NLP tasks.

The third question, regards the quality of the resulting word embeddings based on binary sparse vector representations when compared with state of the art dense representations. Are the proposed vector representations capable of achieving comparable results in several Natural Language Processing tasks when compared with state of the art word embeddings? To answer this question we evaluate the proposed word vector representations in several intrinsic NLP tasks, comparing the performance with several state of the art word embeddings.

Aligned with our research questions are the objectives this research work aims to achieve. We defined three main objectives that we consider milestones towards answering our three research questions.

The first objective is to propose a methodology for representing words in the semantic vector space by binary sparse vectors, mapping every word to a point in a vector space. Therefore, each word represented in the corpus will have a corresponding vector of some predefined dimensionality following the idea of the distributional hypotheses. By means of the proposed methodology words with similar meanings will have closer vector representations.

The second objective is to use the derived word vector representations, formed by large binary sparse vectors, in several Natural Language Processing tasks and compare those results with publicly available state of the art word embeddings.

The third objective is to compare the resources used to produce these vector representations, in terms of learning material size. If vector representations can be derived with less training material, it could be an advantage for deriving domain specific word embeddings, or if less raw material is available, as is the case with languages not so disseminated across the globe or not so used in the digital world.

1.3 Research Methodology

The methodology used in this research is based on the Design Science Research (DSR) model as all the work to solve the proposed research questions lies around the development and evaluation process of an innovative artifact. The building process of the artifact is defined in order to produce a rigorous, coherent, consistent and formally defined artifact. The artifact is thereafter properly evaluated in order to assess his utility in the context of the defined problem and research questions.

Adopting the DSR model proposed by Peffers et al. [59] this work is composed by the sequential phases depicted in Figure 1:

Figure 1 – Sequential Phases adopted

This is a problem-centered approach where the research work is originated by the identification of a problem. As defined in the previous sections, this research identifies the need for an alternative unsupervised methodology to derive word representations with less training material in order to address NLP tasks in a more efficient manner and achieving comparable results with state of the art methods.

The design and development phase produces four artifacts:

1. Constructs in which problems and solutions are defined;
2. Models to represent the real world situation;
3. Methods for solving the problem;
4. Instantiations of the previous three artifacts with the goal of demonstrating feasibility, assessment and suitability for the intended purpose.

An agile methodology was used in the development phase with the goal to instantiate a working prototype earlier in time, from where further improvements, corrections and fine tuning can be worked out.

The demonstration and evaluation phase were accomplished simultaneously. The main objective of this phase is to measure the performance of the proposed solution in several natural language tasks and compare it with state of the art language models.

1.4 Structure of the Dissertation

This dissertation presents the research work done as part of this master thesis. The document is organized as follows:

Chapter 2 provide the literature review of relevant research works, clarifying the main concepts and giving the theoretical background needed for contextualizing this work.

Chapter 3 is divided in two major sections, where the first section describes the constructs in which problems and solutions are defined, along with the required functionality for the artifact, while the second section details the implementation

Chapter 4 is dedicated to the evaluation of the word embeddings derived from the proposed methodology. The details and scope of the evaluation methodology are presented and the chapter concludes with the results of this work.

Chapter 5 presents the conclusions of this work and points possible directions for additional research work in the context of this research topic.

2 Literature Review

The following sections provide theoretical background obtained from other research works in the area of word vector representation. The foundational knowledge needed to frame the research questions and to understand the objectives of this work is also provided.

This chapter proceeds with the clarification of the main theoretical concepts used throughout this work. Theoretical and historic background is provided, in order to address the main theories and works that provided the foundations for more recent approaches. Four mathematical processing steps for vector space model generation are described: frequencies calculation, weighting, dimensionality reduction and similarity calculation.

Afterwards, we review several approaches, capable of producing state of the art results in several NLP tasks, having in common word embeddings has the base element of representing text.

Finally we address the concepts, theories and past works that support the ideas put to practice in this thesis, namely sparse distributed representations and semantic folding theory.

2.1 Distributional Semantic Model

The work done in this thesis has his foundations on distributional semantics theory and follows the assumption that the meaning of a word can be derived by the words that appear in his surroundings. The main idea of the distributional hypothesis is that words with similar distributional properties have similar meanings. Zellig Harris (1909-1992) with his work on distributional theory [27] was the main advocate of this hypothesis. Throughout his work we can find a number of texts that emphasize the relation between the meaning of words and the contexts they appear, of which “Words with similar meanings will appear in similar contexts” [27] is an example. Firth in 1957, with his studies in Linguistic Analysis, reinforced and popularized the distributional semantics theory [24]. His phrase "A word is characterized by the company it keeps" emphasizes the theory’s general idea.

An example that demonstrates the validity of the distributional hypothesis can be seen in the sentence: “…because of severe bacterial infections he took penicillin, and within days had a remarkable recovery”. Even if we do not know the meaning of “penicillin” we can presume with confidence that is a kind of drug used to treat infections, just knowing the surrounding words.

The Vector Space Model (VSM) was introduced by Salton and colleagues in 1975 [71], and the novelty of VSM was to use frequencies in a corpus of text as a clue for discovering semantic

information [79]. According to VSM, also known as the bag of words, “The frequencies of words in a document tend to indicate the relevance of the document to a query.” [71]. Turney and Pantel [79], states that “If documents and pseudo-documents (queries) have similar column vectors in a term–document matrix, then they tend to have similar meanings”.

Hinrich Schutze [74], refers the computational language model of meaning as the “word space model”, and defines it as “Vector similarity is the only information present in Word Space: semantically related words are close, unrelated words are distant”.

Sahlgren [69], states that “The general idea behind the distributional hypothesis seems clear enough: there is a correlation between distributional similarity and meaning similarity, which allows us to utilize the former in order to estimate the latter”. This author recovers the view on *words meaning* from Swiss linguist Fernand de Sausurre (1857-1913) to specify what kind of distributional information we should look for and what would be the differences in meaning if we apply two different models of word distribution: syntagmatic and paradigmatic. A syntagmatic relation between two words indicates that these two words co-occur, whereas in a paradigmatic relation the two words share its neighbors. Hence, Sahlgren defined the Refined Distributional Hypothesis:

“A distributional model accumulated from co-occurrence information contains syntagmatic relations between words, while a word-space model accumulated from information about shared neighbors contains paradigmatic relations between words” [69].

In order to apply any computational process to language is essential to transform words, as the basic units of language, into its numerical vector representation. As we will describe, several approaches are proposed to transform words into vectors, and at the same time build such vectors so that words with similar meanings are closer in a *N* dimensional space.

Turney and Pantel [79], with their work “From frequency to meaning: Vector space models of semantics”, systematize the linguistic processing for vector space models, considering three stages of pre-processing for transforming text into vectors. First, the raw text is tokenized, and a decision as to be made about what constitutes a term and how to extract terms from raw text. After tokenization, the raw text is normalized, converting superficially different strings of characters to the same form (e.g., hour, Hour, hours, and Hours could all be normalized to hour). Finally, raw text can be annotated, to mark identical tokens as being different (e.g., act as a verb could be annotated as act/VB and act as a noun could be annotated as act/NN.

After pre-processing the corpus, mathematical processing is needed to generate a matrix of vectors, also known as context-vectors, according to the VSM. Turney and Pantel [79] referring to Lowe [46] enumerates the four mathematical processing steps for word–context VSMs: calculate the frequencies, transform the raw frequency counts, smooth the space (dimensionality reduction), then calculate the similarities. The following paragraphs are intended to clarify those steps.

The goal of calculating frequencies is to build a frequency matrix where the counts of each element (term, word, words pair) are determined for a given context (e.g. document). Several types of frequency matrix can be derived, of which two kinds are mentioned; sparse term-document matrices represent documents in columns and terms in rows, and dense word-word matrices representing co-occurrences for each word pairs, with the count of each word, represented in the matrix diagonal (Figure 2). Term-document matrixes are implementations of context-vectors according to the bag of word hypothesis preconized by Salton and colleagues.

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
against	0	0	0	1	0	0	3	2	3	0
age	0	0	0	1	0	3	1	0	4	0
agent	0	0	0	0	0	0	0	0	0	0
ages	0	0	0	0	0	2	0	0	0	0
ago	0	0	0	2	0	0	0	0	3	0
agree	0	1	0	0	0	0	0	0	0	0
ahead	0	0	0	1	0	0	0	0	0	0
ain't	0	0	0	0	0	0	0	0	0	0
air	0	0	0	0	0	0	0	0	0	0
aka	0	0	0	1	0	0	0	0	0	0

	against	age	agent	ages	ago	agree	ahead	ain.t	air	aka	al
against	2003	90	39	20	88	57	33	15	58	22	24
age	90	1492	14	39	71	38	12	4	18	4	39
agent	39	14	507	2	21	5	10	3	9	8	25
ages	20	39	2	290	32	5	4	3	6	1	6
ago	88	71	21	32	1164	37	25	11	34	11	38
agree	57	38	5	5	37	627	12	2	16	19	14
ahead	33	12	10	4	25	12	429	4	12	10	7
ain't	15	4	3	3	11	2	4	166	0	3	3
air	58	18	9	6	34	16	12	0	746	5	11
aka	22	4	8	1	11	19	10	3	5	261	9
al	24	39	25	6	38	14	7	3	11	9	861

Figure 2 – Examples of term-document matrix and word-word matrix (source [84])

As stated by Turney and Pantel [79], transforming the raw frequency counts, or weighting "is to give more weight to surprising events and less weight to expected events". By other words, in information theory, a surprising event has higher information content than an expected event (Shannon [76]). According to Sahlgren [68] and Turney & Pantel [79], the most common method of weighting in information retrieval is TF-IDF (term frequency × inverse document frequency). Lunh [47], defines the term frequency (TF) as the number of times a word appears in a document, whereas Sparck Jones [77] defined the inverse document frequency (IDF) as a way of giving more weight to rare terms that are more discriminative. According to Shannon [76], those rare terms have higher information content in information theory. The mathematical definition of IDF is shown below, where *N* is the total number of documents in the collection and df_i is the number of documents in which a term occurs. The fewer documents a term occur the higher is his weight, or importance in terms of information gain, and vice-versa. The log function is applied to squash the measure, because of the usually high number of documents in many collections.

$$idf_i = \log(\frac{N}{df_i}) \qquad (2.1)$$

The TF-IDF weighting scheme is the combination of term frequency with inverse document frequency, where the weight of word i in document j is given by w_{ij}:

$$w_{ij} = tf_{ij} \; . \; idf_{ij} \qquad (2.2)$$

Pointwise Mutual Information (PMI) is an alternative to TF-IDF weighting scheme. Proposed by Church and Hanks [14] [15] is a measure of how often two events occur, compared with what we would expect if they were independent. Considering co-occurrence vectors the PMI between a target word *w* and a context word *c* is:

$$PMI(w,c) = log_2 \frac{P(w,c)}{P(w)P(c)} \quad (2.3)$$

The numerator is a measure of the probability that we observe two words together and the denominator is the probability we could expect the same two words co-occur assuming they each occurred independently. Thus, the higher the value of PMI the higher the probability of two words co-occur than we expect by chance [37]. To account for very small values of probability and the consequent difficulty in evaluate this levels of "unrelatedness", PPMI replaces all negative PMI values with zero [14] [18] [56].

$$PPMI(w,c) = \max(log_2 \frac{P(w,c)}{P(w)P(c)}, 0) \quad (2.4)$$

Pantel and Lin [57] [58] consider that PMI works well for both word–context matrices and for term–document matrices. Bullinaria and Levy [11] demonstrated that PPMI performs better than a wide variety of other weighting approaches when measuring semantic similarity with word– context matrices.

The third mathematical process step for generating vector space models, capable of representing terms with similar meaning closer in the vector space is dimensionality reduction also known as smoothing. Vector space models, due to the large number of document collections involved in building such models, tend to have a very high dimensionality, as a consequence of the number of different terms used in the process. This high dimensionality hinders scalability and efficiency of the algorithms that will process those matrices models. Sahlgren [68], states that "This presents us with the following delicate dilemma: on the one hand, we need as much data as we can get our hands on in order to build a truthful model of language use; on the other hand, we want to use as little data as possible because our algorithms will become computationally prohibitive otherwise". Sahlgren [68], also draws attention for the problem of data sparseness. Only a small fraction of the co-occurrence matrix will have non-zero values, because "the vast majority of words only occur in a very limited number of contexts". The general Zipf's law [84] represents the generalization of this specific phenomenon.

Jurafsky and Martin [37] also write about the advantages of short and dense vectors, like being easier to include as features in machine learning systems, may generalize better and help avoid overfitting, and they may do a better job of capturing synonym than sparse vectors, because synonym words in sparse vectors tend to be represented by distinct dimensions. Nonetheless, for the same authors, the use of dense vectors is not always the right approach, depending on the applications. The mentioned advantages will have their contradictory counterpart when we approach the semantic folding theory, in which our work is based.

In view of the fact that dimensionality reduction techniques play an important role in improving either the running time of the algorithms and the quality of the context vectors, in the context of vector space models, we dedicate the following paragraphs to mention the main algorithms for co-occurrence matrix smoothing.

The simplest forms of dimensionality reduction involve operations like filtering out words in the co-occurrence matrix and documents based on either linguistic or statistical criteria [68]. Lin [42] applied this technique by keeping only the context-word dimensions with a PMI above a conservative threshold and setting the others to zero, and showed that the number of comparisons needed to compare vectors greatly decreases while losing little precision in the similarity score between the top-200 most similar words of every word [79].

Deerwester and colleagues [19], used a method called Singular Value Decomposition (SVD) for generating dense vectors. SVD belongs to a family of methods that can approximate an N-dimensional dataset using fewer dimensions, including Principal Component Analysis (PCA) and Factor Analysis [37]. The general idea is to find the most important dimensions of the data, being those dimensions along which the data varies the most [37]. SVD applied to document similarity is called Latent Semantic Indexing (LSI), but it is called Latent Semantic Analysis (LSA) when applied to word similarity [79].

A textual explanation of the technique is given by Sahlgren [68] when he says that "SVD is a matrix factorization technique that decomposes, or factorizes, the original matrix into several (three when using SVD) smaller matrices, which can be multiplied to reproduce the original one. These smaller matrices contain the linearly independent factors of the original matrix (in the case of SVD, they are called "singular vectors" and "singular values"). If the smallest factors are disregarded when multiplying the smaller matrices, the result will be an approximation of the original co-occurrence matrix".

For a better understanding, Jurafsky and Martin [37] give a more formal definition of the method and of the linear algebra involved. SVD factorizes any such rectangular term-document matrix X

$$X = |V| \,.\, c \qquad (2.5)$$

, into the product of three matrices W, Σ, and C^T (Figure 3). In the |V|×m matrix W, each of the w rows still represents a word, but the columns do not; each column now represents one of m dimensions in a latent space, such that the m column vectors are orthogonal to each other and the columns are ordered by the amount of variance in the original dataset each accounts for. The number of such dimensions m is the rank of X (the rank of a matrix is the number of linearly independent rows). Σ is a diagonal $m \times m$ matrix, with singular values along the diagonal, expressing the importance of each dimension. The $m \times c$ matrix C^T still represents documents or contexts, but each row now represents one of the new latent dimensions and the m row vectors are orthogonal to each other. By using only the first k dimensions, of W, Σ, and C instead of all m dimensions, the product of these three matrices becomes a least-squares approximation to the original X. Since the first dimensions encode the most variance, one way to view the reconstruction is thus as modeling the most important information in the original dataset [37].

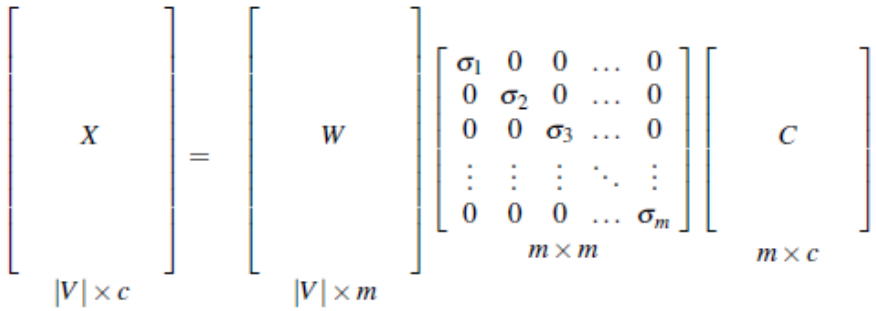

Figure 3 - SVD factors a matrix X into a product of three matrices, W, Σ, and C (source [5])

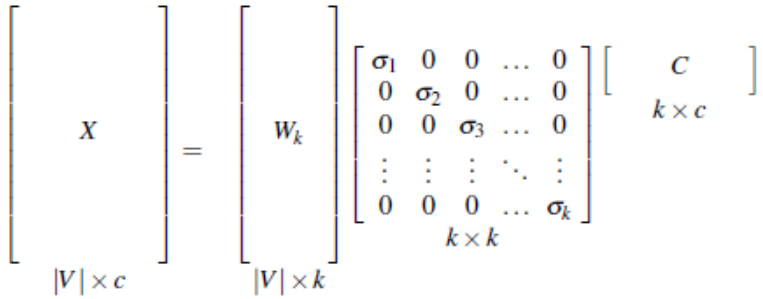

Figure 4 – Truncated SVD using just the k top dimensions of the product matrices (source [37])

The preferred weighting scheme for LSA applies the multiplication of two weights called the local and global weights for each cell (*i, j*), considering term *i* in document *j*.

The local weight of each term *i* is its log frequency:

$$\log f(i,j) + 1 \qquad (2.6)$$

The global weight of term *i* is a version of its entropy. The whole formula for weighting the co-occurrences according to Dumais [21] is:

$$f_{ij} = (\log(TF_{ij}) + 1) \ . \ (1 - (\sum_j \frac{p_{ij} \log p_{ij}}{\log D})) \qquad (2.7)$$

, where *D* is the total number of documents in the collection, $p_{ij=}{}^{tf_{ij}}/_{f_i}$ and f_i is the frequency of term *i* in the whole document collection.

As mentioned above, SVD can be applied to term-document matrices (as in the LSA algorithm) or to word-word or word-context matrix, an idea proposed by Schutze [75]. Hinrich Schütze models based on word co-occurrences, highly influenced a different implementation of the word space named Hyperspace Analogue to Language (HAL) proposed by Lund and colleagues [48]. HAL idea is to build distributional profiles based on which other words surround them [69]. HAL uses a words-by-words co-occurrence matrix, which is populated by counting word co-occurrences within a directional context window of 10 words wide. The co-occurrences are weighted with the distance between the words, so that words that occur next to each other get the highest weight, and words that occur on opposite sides of the context window get the lowest weight. The result of this operation is a directional co-occurrence matrix in which the rows and

the columns represent co-occurrence counts in different directions [68]. Then each row-column pair is concatenated to produce a very high dimensional context matrix, which is subsequently reduced, in their dimensionality, by computing the variances of the row and column vectors for each word, and discarding the elements with lowest variance, leaving only the 100 to 200 most variant vector elements [68].

Since the work of Deerwester and colleagues [20], subsequent research has discovered many alternative matrix smoothing processes, such as Nonnegative Matrix Factorization (NMF) [41], Probabilistic Latent Semantic Indexing (PLSI) [32], Iterative Scaling (IS) [3], Kernel Principal Components Analysis (KPCA) [73], Latent Dirichlet Allocation (LDA) [8], and Discrete Component Analysis (DCA) [13].

Regarding comparing the vectors (the fourth and final step for building vector space models) Turney and Pantel [79], state that "The most popular way to measure the similarity of two frequency vectors (raw or weighted) is to take their cosine". Let x and y be two vectors, each with n elements. The cosine of the angle θ between x and y is given by the following formula:

$$\cos(x, y) = \frac{\sum_{i=1}^{n} x_i . y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \cdot \sqrt{\sum_{i=1}^{n} y_i^2}} \qquad (2.8)$$

$$= \frac{x \,.\, y}{\sqrt{x.x} \,.\sqrt{y.y}} \qquad (2.9)$$

$$= \frac{x \,.\, y}{\|x\| \, \|y\|} \qquad (2.10)$$

Cosine captures the idea that the length of the vectors is irrelevant; the important thing is the angle between the vectors [79]. The cosine ranges from −1 when the vectors point in opposite directions (θ is 180 degrees) to +1 when they point in the same direction (θ is 0 degrees). When the vectors are orthogonal (θ is 90 degrees), the cosine is zero [79]. A measure of distance between two vectors can be converted to a measure of similarity by inversion or subtraction [79], assuming the following formulas:

$$sim(x, y) = 1/dist(x, y) \qquad (2.11)$$

$$sim(x, y) = 1 - dist(x, y) \qquad (2.12)$$

Several other similarity measures have been proposed: Euclidean distance, Manhattan distance, Jaccard Coefficient, Pearson Correlation Coefficient, Kullback-Leibler Divergenge and Similarity Measure for Text Processing (SMTP) [43] to name a few. Several studies [12][33] compared the effectiveness of these measures.

The word vector space semantic model is completely derived in an unsupervised manner, trying to capture the words meanings by looking to the distributional patterns in text. Sahlgren [68], underlines this idea:

"What makes the word-space model unique in comparison with other geometrical models of meaning is that the space is constructed with no human intervention, and with no a priori knowledge or constraints about meaning similarities. In the word-space model, the similarities between words are automatically extracted from language data by looking at empirical evidence of real language use".

2.2 Neural Networks-Based Semantic Models

Neural networks can be used for learning the representations of words as vectors, also known as "word embeddings". A word embedding method discovers distributed representations of words; these representations capture the semantic similarity between the words and reflect a variety of other linguistic regularities [67][7][55]. The main idea behind a word embedding is aligned with the distributional hypothesis theory and concepts previously described, of representing words with similar meaning with similar vector representations. The main difference in that the learned vectors derived from word embedding techniques are dense representations in contrast with the vectors representations based on counting words occurrences. Adopting the explanation given by Jurafsky and Martin [37]:

"The intuition is that words with similar meanings often occur near each other in texts. The neural models therefore learn an embedding by starting with a random vector and then iteratively shifting a word's embeddings to be more like the embeddings of neighboring words, and less like the embeddings of words that don't occur nearby".

Several authors state that word embeddings have been successfully used for analyzing language [7][53][54][60]. Given the importance of word embeddings for many NLP downstream tasks, we proceed with a historical review and clarification of the main models involved in learning word embeddings.

Bengio and colleagues in 2003 [6] pioneered the use of artificial neural networks for learning distributed representations of words. Their idea was to fight the "curse of dimensionality" through the following approach:

1. associate with each word in the vocabulary a distributed word feature vector (a real-valued vector in R^m),

2. express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneously the word feature vectors and the parameters of that probability function.

The classic neural language model proposed by Bengio et al. [6] in 2003 consists of a one-hidden layer feed-forward neural network that predicts the next word in a sequence as shown in Figure 5.

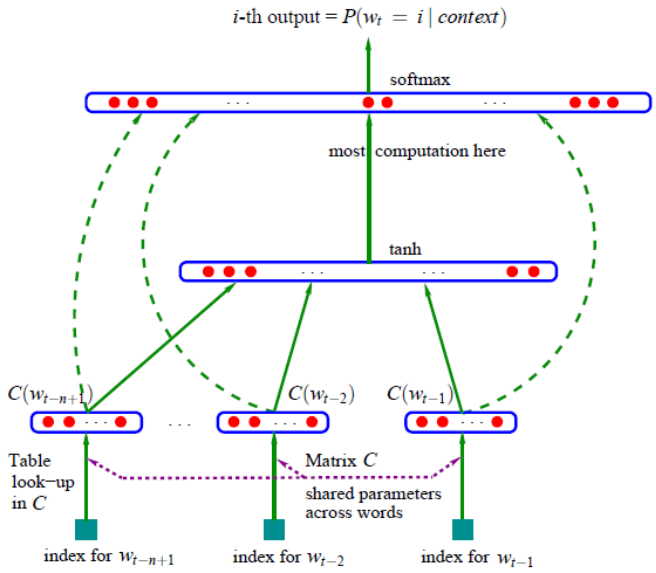

Figure 5 – One-hidden layer feed-forward neural network (source [6])

The objective is to learn a good model that maximizes an objective function that yields the probability of a word given the previous one:

$$f(w_t, \dots, w_{t-n+1}) = \hat{P}(w_t | w_1^{t-1}) \qquad (2.13)$$

The training of the model is achieved by looking for θ that maximizes the training corpus penalized log-likelihood L, where $R(\theta)$ is a regularization term [6]:

$$L = \frac{1}{T} \sum_t log f(w_t, w_{t-1}, \dots, w_{t-n+1}; \theta) + R(\theta) \qquad (2.14)$$

In practice, the neural network computes the following function, with a softmax output layer, which guarantees positive probabilities summing to 1.

$$\hat{P}(w_t | w_{t-1}, \dots, w_{t-n-1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}} \qquad (2.15)$$

, where y_i are the un-normalized log-probabilities for each output word i [6].

The bottleneck of this model is the softmax computation, as it is proportional to the number of words in the corpus which is typically on the order of hundreds of thousands.

In 2008 Collobert and Weston, with their work “A unified architecture for natural language processing” [16], demonstrated the power of using pre-trained word embeddings in many well-known NLP tasks including part-of-speech tagging, chunking, named-entity recognition, learning a language model and the task of semantic role-labeling. The method proposed by Collobert and Weston employs an alternative objective function to avoid computing the expensive softmax. Rather than the cross-entropy criterion of Bengio et al. [6], which maximizes the probability of the next word given the previous words, Collobert and Weston train a network to output a higher score f_θ for a correct word sequence than for an incorrect one.

They consider a window approach network, with parameter θ, which outputs a score $f_\theta(x)$ given a window of text $x = [w]_1^{d_{win}}$ [17]. Then they minimize the pairwise ranking criterion with respect to θ:

$$J_\theta = \sum_{x \in X} \sum_{w \in V} max\{0, 1 - f_\theta(x) + f_\theta(x^{(w)})\} \qquad (2.16)$$

In 2013 Mikolov et al. [53], proposed two learning methods for generating dense word embeddings models: skip-gram and continuous bag of words. These two methods along with the Word2Vec software gained a lot of traction, and provide state of the art word embeddings [25]. Word2Vec is an implementation of the two proposed methods that learns word embeddings by training a shallow neural network to predict neighboring words. These word embeddings are, simultaneously, good at predicting neighboring words and at representing similarity [37]. The main benefits of these two architectures for learning word embeddings when compared to Bengio’s and Collobert and Weston approaches are: they do not consider a computationally expensive hidden layer and they allow the language model to take additional context into account.

The skip-gram model, depicted in Figure 6, predicts each neighboring word in a context window of 4 words from the current center word. In this case the context is $[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]$ and we are predicting each of these words from word w_j [37].

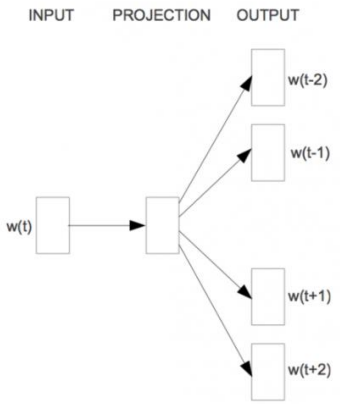

Figure 6 - Skip-gram (source [53])

In practice the skip-gram model learns two separate embeddings, encoded in two matrices, for each word *w*: the word embedding *v*, represented by matrix *W* and the context embedding *c*, represented by the context matrix *C* (Figure 7).

The skip-gram computes the probability $p(w_k|\, w_j)$ by taking the dot product between the word vector for $j(v_j)$ and the context vector for $k(c_k)$, and turning this dot product $v_j . c_k$ into a probability by passing it through a softmax function [37].

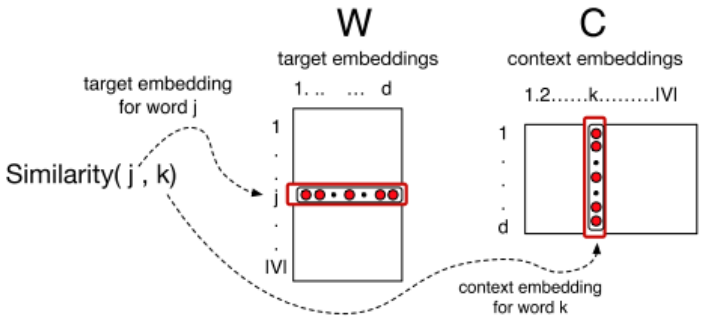

Figure 7 – Similarity function for selecting out a target vector v_j from W, and a context vector c_k from C (source [37])

The mathematical formulation for the skip-gram model is thus: considering a given corpus of words *w*, their contexts *c*, and the conditional probabilities $p(c|w)$ on a given a corpus T, the goal is to set the parameters θ of $p(c|w\,;\,\theta)$ so as to maximize the corpus probability [25]:

$$\arg\max_{\theta} \prod_{w\,\epsilon\,T}\left[\prod_{c\,\epsilon\,C\,(w)} p(c|w;\theta)\right] \qquad (2.17)$$

Following neural-network language models literature, we can use soft-max function to model the conditional probability:

$$p(c|w;\theta) = \frac{e^{v_c \cdot v_w}}{\sum_{c' \in C} e^{v_{c'} \cdot v_w}} \qquad (2.18)$$

, where v_c and $v_w \in$ Rd are vector representations for *c* and *w* respectively, and *C* is the set of all available contexts [25].

In summary, the skip-gram model learns two word embedding by iteratively making the embeddings for a word more like the embeddings of its neighbors (maximizing the numerator) and less like the embeddings of other words (minimizing the denominator) [37].

As we saw with the method proposed by Bengio et al [6], this version of the algorithm has the same problem with the softmax computation, as the time to compute the denominator can be very expensive. For each word w_t, the denominator requires computing the dot product with all other words [37].

To overcome this limitation, Mikolov et al. [54] present the negative-sampling approach as a more efficient way of deriving word embeddings [25]. The main idea is that, in the training phase, the

algorithm walks through the corpus, at each target word choosing the surrounding context words as positive examples, and for each positive example also choosing *k* noise samples or negative samples: non-neighbor words [37]. The goal will be to move the embeddings toward the neighbor words and away from the noise words. The learning objective for one word/context pair (w, c) is:

$$log\sigma\ (c.w) + \sum_{i=1}^{k} E_{w_i \sim p(w)}[log\sigma(-w_i.w)] \qquad (2.19)$$

, and can be expressed as a way to maximize the dot product of the word with the actual context words, and minimize the dot products of the word with the k negative sampled non-neighbor words. The learning algorithm starts with randomly initialized *W* and *C* matrices, and then walks through the training corpus moving *W* and *C* so as to maximize the objective. An algorithm like stochastic gradient descent is used to iteratively shift each value so as to maximize the objective, using error backpropagation to propagate the gradient back through the network [37].

The Continuous Bag of Words (CBOW) model (Figure 8) is also based on a predictive model, but this time predicting the current word w_t from the context window of *n* words around it, e.g. for *n*=2 the context is $[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]$. While CBOW and skip-gram are similar algorithms and produce similar embeddings, they do have slightly different behavior, and often one of them will turn out to be the better choice for any particular NLP task [37].

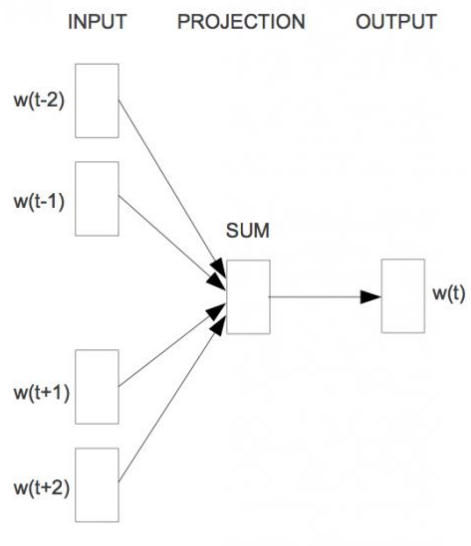

Figure 8 - Continuous bag-of-words (source [53])

Another advantage of the methods proposed by Mikolov et al. [55], is that it introduced a new evaluation scheme based on word analogies that probes the finer structure of the word vector space by examining not the scalar distance between word vectors, but rather their various dimensions of difference. As a consequence it allows to derive analogies encoded in the word vectors that make the vector space, simply by doing arithmetic operations on the vectors itself. For example *vec("king") – vec("man") + vec("woman") =~ vec("queen")*.

Piotr Bojanowski et al [9] proposed an extension to Mikolov's work by considering sub word information for deriving word vectors. The approach is based on the skip-gram model but the vector representations are associated to each character n-grams and the words vector representations are the sum of these representations [9]. This method besides being faster to train, outperformed state-of-the-art methods in analogy and similarity tasks. The original implementation of this method is named *FastText* and is publicly available.

Pennington et al. [60] proposes a model called GloVe that combines the advantages of statistical (e.g. LSA) and neural network (e.g. skip-gram) methods. Pennington et al. argues that:

"*While methods like LSA efficiently leverage statistical information, they do relatively poorly on the word analogy task, indicating a sub-optimal vector space structure. Methods like skip-gram may do better on the analogy task, but they poorly utilize the statistics of the corpus since they train on separate local context windows instead of on global co-occurrence counts.*"

GloVe model benefits from counting data while simultaneously capturing the meaningful linear substructures prevalent in recent log-bilinear prediction-based methods like Word2Vec" [60]. In practice, GloVe starts by a very sparse word-word co-occurrence matrix, then transforms the raw integer counts into a matrix where the co-occurrences are weighted based on the distance between the word co-occurrence, a log is applied to the weighted co-occurrences and finally the matrix is factorized to produce the final word vector representations. This final matrix when factorized using the particular stochastic gradient descent algorithm of Word2Vec, yields out the target and context matrices derived by this model.

Although GloVe model does not belong to the group of shallow neural networks, it is an important contribution to the semantic vector space research field, moreover the resulting matrix is closely related to the target and context embeddings produced by Mikolov's [53] model.

Word vector representations derived from neural networks-based semantic models have been used to initialize the word vectors input layer for deep learning models with gains in performance over random word vector initialization in several downstream tasks [52].

Alternatively, deep learning architectures with very different objective functions are by itself capable of deriving word embeddings with good results in several NLP tasks. Hill et al. [31], showed that Neural Machine Translation (NMT) models "… are not only a potential new direction for machine translation, but are also an effective means of learning word embeddings". Hill and colleagues trained two different NMT models based on recurrent neural networks (RNN) and concluded that word embeddings derived for both languages used for training the model may have particularly desirable properties and appear to be "more orientated towards conceptual similarity than those of monolingual models".

Luong et al. [49], applied a RNN architecture to derive word embeddings representations by using morphological sub word information. This method outperformed publicly available embeddings on word similarity tasks across many datasets, and proved to be more effective specially on modeling rare and complex words.

2.3 Semantic Folding Theory

All the theoretical framework used in this work is based on the work done by Francisco de Sousa Webber, and his Semantic Folding Theory (SFT) [81]. This theory offers an alternative approach for the representation of language semantics and in contrast with the methods mentioned thus far, it is not based on any statistical methods. In particular, Webber mentions that:

"Semantic Folding Theory (SFT) is an attempt to develop an alternative computational theory for the processing of language data. While nearly all current methods of processing natural language based on its meaning use word statistics, in some form or other, Semantic Folding uses a neuroscience-rooted mechanism of distributional semantics". [81]

SFT is built around a theoretical computational model, inspired by the biological characteristics of the human cortex, developed by Jeff Hawkins, and known as Hierarquical Temporal Memory (HTM). HTM defines a deep learning model which attempts to be comparable with the higher level structures and functionality of the brain [45]. One of the premises of HTM is that the fundamental form of information representation in the brain is by means of sparse distributed representations (SDR) [29]. Hence, according to SFT words and text are represented by very large sparsely filled binary vectors and every bit of information in this SDR encodes a specific semantic feature, meaning that each SDR is therefore a vector in the semantic space [81].

According to Webber using SDR's for encoding word and text semantics has the main advantage that "it allows any data-items to be directly compared" just by using boolean operators and similarity functions. Several other advantages of SDR, when compared with dense vectors representations, are highlighted by Webber [81]:

- Ability to store semantic meaning in every bit of the SDR;
- Very high compression rate by storing only the positions of the set bits;
- Fault tolerance when several bits are discarded or shifted, the overall semantics are preserved;
- Union of several SDRs preserves the information of each original constituent allowing for comparison with a new unseen SDR.

SFT proposes a novel approach to the representational problem, where each word or sentence is represented by a 2-dimensional sparse binary vector, allowing to solve several NLP problems by applying Boolean operators and a similarity function (e.g. Euclidean distance). Simultaneously, "Many practical problems of statistical NLP systems, like the high cost of computation, the fundamental incongruity of precision and recall, the complex tuning procedures etc., can be elegantly overcome by applying Semantic Folding" [81].

The mechanism for producing 2-dimensional word SDRs is in accordance with the distributional hypothesis. Every word is represented by all the contexts (or special case scenarios in SFT nomenclature) that it appears. The novelty of this approach is in a mechanism denominated

Semantic Folding where each context (text snippet) is positioned within a 2D-area in a way that semantically close contexts are placed near each other and dissimilar contexts are placed far apart in the 2D area. The topological 2D layer built from all the contexts extracted from the training corpus is the distributional reference for encoding word SDR (or semantic fingerprint) and associates a coordinate pair to every context.

Figure 9 – Semantic folding (source [81])

This 2D map is then used to produce a binary representation for every word by assigning “1” to all coordinates (contexts) in which the word appears and “0” for all other positons on the map.

According to Webber [81], the steps for the Semantic Folding process are:

1. *Definition of a reference text corpus of documents that represents the Semantic Universe the system is supposed to work in. The system will know all vocabulary and its practical use as it occurs in this Language Definition Corpus (LDC).*
2. *Every document from the LDC is cut into text snippets with each snippet representing a single context.*
3. *The reference collection snippets are distributed over a 2D matrix (e.g. 128x128 bits) in a way that snippets with similar topics (that share many common words) are placed closer to each other on the map, and snippets with different topics (few common words) are placed more distantly to each other on the map. This produces a 2D semantic map;*
4. *In the next step, a list of every word contained in the reference corpus is created;*
5. *By going down this list word by word, all the contexts a word occurs in are set to “1” in the corresponding bit-position of a 2D mapped vector. This produces a large, binary, very sparsely filled vector for each word. This vector is called the Semantic Fingerprint of the word. The structure of the 2D map (the Semantic Universe) is therefore folded into each representation of a word (Semantic Fingerprint). The list of words with their fingerprints is stored in a database that is indexed to allow for fast matching*.

This method results in similar word SDRs for similar words that appear in the same contexts. The level of similarity can be measured by similarity/distance metrics (e.g. overlap between SDRs, Euclidean distance, cosine similarity, Jaccard similarity).

It is important to note that the size of the generated text snippets determines the extent that each word is associated with other concepts [81].

By aggregating each atomic word SDR that are part of a document, a text SDR (document fingerprints) can be created. Then the aggregated fingerprint is sparsified by maintaining only the "1" bits in the 2D map coordinates where the matching between word SDRs is over a given threshold (Figure 10).

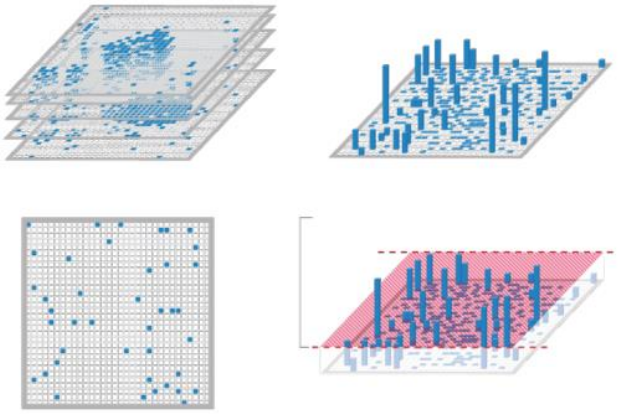

Figure 10 – Aggregation and sparsification resulting in a Text SDR (source [81])

In his paper [81], Webber refers that all semantic fingerprints are homologous (same size and feature space), hence they can be used directly in boolean expressions. Figure 11 shows an example of an arithmetic operations with SDR's that demonstrates the semantic associations between words.

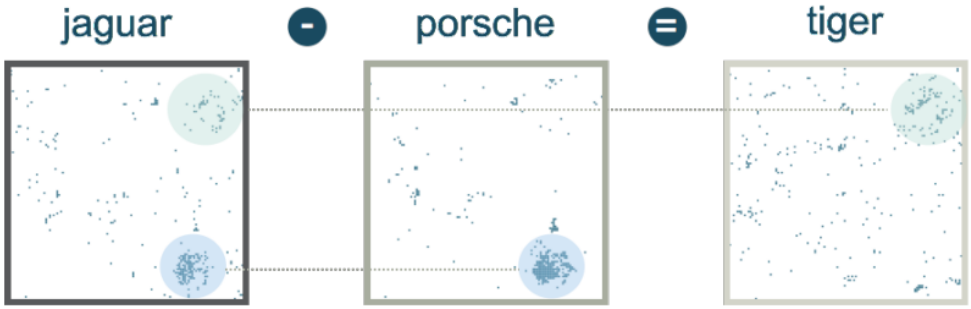

Figure 11 – Computing with word meanings (source [81])

Webber refers that by applying simple and efficient similarity measures, like number of bits that overlap between SDRs, several NLP tasks can be accomplished, namely document classification, searching documents, keyword extraction and semantic slicing.

Another benefit from the SFT approach is, according to Webber, similarity of fingerprints across different languages if aligned semantic spaces are used.

3 Towards Word Sparse Distributed Representations

The motivation of this chapter is to detail the conception and implementation of an artifact able to produce word sparse vector representations, aligned with the objectives defined for this work.

The main contributions are a methodology capable of producing word embeddings, in an unsupervised manner, with comparable performance in several NLP tasks to state of the art word embeddings. In the process, other contributions are given such as a systematic analysis on the influence that input parameters have over the efficiency and performance of the resulting model.

The first section of this chapter describe the constructs in which problems and solutions are defined, along with the required functionality for the artifact. The second section explains the proposed methodology for solving the research questions, including the artifact architecture and the way the resources communicate and integrate with each other.

3.1 Theoretical Framework

The aim of this section is to build the theoretical background from where the technical solutions to achieve this work's objectives were derived. For a given practical problem one or more approaches were chosen, and here we justify those options based on constructs.

At this moment, it is important to remember the objectives and research questions of this work. Regarding the objectives we seek an alternative methodology to derive word embeddings capable of achieving good results in downstream NLP task. Besides a new methodology we aim to produced and explore word embeddings with binary sparse vector representations in contrast with the dense vector representations obtained from state of the art methodologies. In terms of research questions we intend to answer questions regarding the feasibility of obtain good performant binary sparse vector representations with less training material when compared with state of the art methodologies.

As we recall from previous chapters word embeddings are word vector representations in which words with similar meanings are closer in a *N* dimensional vector space. With this goal in mind we built a model for transforming text into word embeddings, with different vector properties from the traditional and state-of-the-art word embeddings, namely higher dimensionality and sparsity.

The following sections of this chapter conceptually describe each step of the proposed model towards the unsupervised generation of word embeddings as sparse distributed representations, from a given corpus. Thus, the theoretical approach can be applied in practice to any corpus, ensuring at the end of the process the generation of sparse vector distributional representations of words – word embeddings.

3.1.1 Model Conception

Before the detailed description of all components of our work, a higher level model conception for the proposed artifact and implementation is given. The higher level model conception purpose is to give an idea of the main components and steps involved in our methodology. All the components of the model are then further described in the next sections.

As depicted in Figure 12 the main components, inputs and outputs from each module of our method are the following:

1. Pre-processing: Corpus -> Clean raw text;
2. Document definition: Clean raw text -> Clean documents;
3. Document representation: Clean documents -> Document vectors;
4. Document clustering: Document vectors -> Clustered documents;

5. Word representation: Clustered documents -> Raw word embeddings;
6. Binary sparse representation: Raw word embeddings -> Binary sparse distributed representations.

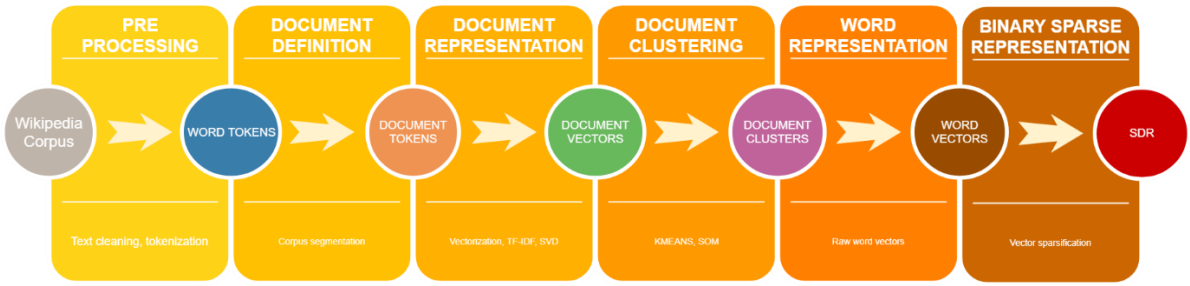

Figure 12 – Model Conception

3.1.2 Word Tokens

Given a textual dataset from which the unsupervised learning of the word embeddings will be performed, the first step of the proposed process is to obtain word tokens from the corpus dataset. Following the orientations from Turney and Pantel [79], that systematize the linguistic processing for transforming text into vectors, we start by defining what should constitute an appropriate word token suitable for word embedding generation.

We consider a word token any atomic unit of text that possesses an intrinsic meaning, typically any individual word, except stop words.

A data cleaning step allows the removal of all unmeaningful tokens like stop words, punctuation and numeric tokens.

3.1.3 From Word Tokens to Documents

Considering the distributional semantic model, words only have meaning in a particular given context, which is provided by the surrounding words. Thus an important step towards our goal is the definition of what constitutes a context. In that sense our proposal is to consider a context, any number of sentences enclosed in a given paragraph of text. This way we hope to achieve a reasonable balance between the number of words of a given piece of text and the specific meaning it encloses, while not generalizing too much the overall meaning or have many possible meanings. If we have considered a chapter or even an entire book as the context of a given word, too many concepts and meanings would be taken into account for word context definition, while

if considering a sentence or even a part of a sentence (e.g. between two commas) the information for encapsulating a meaning or an idea would not be enough.

This approach allows to separate homonyms into different contexts as the same word applied with a different meaning tends to be present in different paragraphs.

Considering only one meaningful topic context is key for capturing the semantics of a given word present in it. In our nomenclature those context text paragraphs are denominated "*documents*".

3.1.4 Document Vector Representation

Documents, as the atomic units of text used for defining the meaning of a word must be mathematically processed in order to transform documents into a matrix of vectors. These are the context vectors according to the Vector Space Model which are ready to undergone a series of mathematical processing steps. The proposed approach was inspired in the methodology defined by Lowe [46] that defines four mathematical processing steps for word–context VSMs, namely: calculate frequencies, transform the raw frequency counts, dimensionality reduction, and calculate similarities.

First, a sparse term-document matrix was derived by counting the occurrences of each word token in a document. Then, we transform the raw frequency counts in order to, and paraphrasing Turney and Pantel [79], "… give more weight to surprising events and less weight to expected events". Words that are rare across all documents, are more discriminative and have higher information content [76] than frequent words. Obvious examples are stop words (e.g. "the", "a", "or", "and") which are very frequent and to not carry any specific information, and on the other side of the spectrum a rare word like "*bacillus*" encloses much more information about the topic being covered.

By applying the method TF-IDF to the term-document frequency matrix yields another matrix weighted by the informational content of each word. A more discriminative word (with a high TF-IDF) is represented in a *N* dimensional space context vector with a higher value in the corresponding dimension and *vice-versa*. The main idea is to valorize words that have considerable informative content, so that in another downstream step, where document clustering is performed, a good performance can be achieved.

Then, dimensionality reduction is applied to the weighted term-document matrix as a way to improve algorithm performance and scalability in downstream processing tasks. Another benefits of dimensionality reduction, and thus working with short dense vectors, as seen in the literature review chapter, is that they may generalize better, avoiding overfitting.

From the available dimensionality reduction technics Singular Value Decomposition (SVD) was chosen. This method, as previously seen, decomposes the original matrix into three smaller

matrices, which contain the linearly independent factors of the original matrix. Disregarding the smallest factors when multiplying the smaller matrices the result will be a least-squares approximation of the original matrix with fewer dimensions. The idea is to consider only the latent dimensions present in the three smaller matrices that encode the most variance and thus represent the most important information in the original dataset. For a deeper explanation of SVD please refer to literature review.

3.1.5 Document Clustering

An important component of the proposed methodology is the clustering of text documents, in order to group similar texts based on a predefined distance metric (e.g. cosine distance). Our intent is to arrange our documents, so that documents with similar semantic meanings are grouped in the same cluster. According to [51], clustering algorithms "create clusters that are coherent internally, but clearly different from each other." In other words, clustering algorithms take the vector representation of each text document considering the vector space model and create coherent groups of documents in the sense that document vectors put in the same cluster should be closer, probably having more common words than documents put on another cluster. Hence, the distance measure applied on a given cluster algorithm greatly influences the result of the clustering.

Clustering algorithms, as a form of unsupervised learning can be distinct based on the type of clusters produced. As stated in [51], flat clustering creates clusters without any explicit structure that relate clusters to each other. Hard and soft clustering algorithms make another distinction, where the former assign only one cluster to each sample, whereas in the later each sample is assigned to a distribution over all clusters.

One of the difficulties when applying cluster algorithms is that the number of clusters must be known in advance, as this is an input parameter to this type of algorithms.

Two clustering algorithms were used in our methodology and compared, namely K-Means and Self Organizing Map. The main idea, is to apply a form of quantization to the input documents represented in the vector space, allowing similar documents to be assigned to the same discretized vector representation, in this case the vector representation's specific cluster. The two studied algorithms produce flat or disjoint clusters, meaning that by using distance measurements, each document is assigned to only one cluster.

3.1.5.1 K-Means

K-means algorithm divides the input documents into *k* clusters, by iteratively updating the *k* reference centroids. The *k* reference centroids position is calculated in each iteration as being the

mean of the vectors that are closer to each *k* centroid. This algorithm has the advantage of quickly converging to a local optimum but has the downside of being very much dependent on the initial choice of the *k* reference centroids.

The computation complexity of K-Means is $O(n\ .\ k\ .\ l)$ where *n* is the number of samples, *k* is the number of clusters and *I* is the number of iterations.

According to [51], K-Means is a flat clustering algorithm whose objective is to minimize the average squared Euclidean distance of documents to their cluster centers where a cluster center is defined as the mean of all documents in a cluster.

$$\vec{\mu}(w) = \frac{1}{|\omega|}\sum_{\vec{x}\in\omega}\vec{x} \qquad (3.1)$$

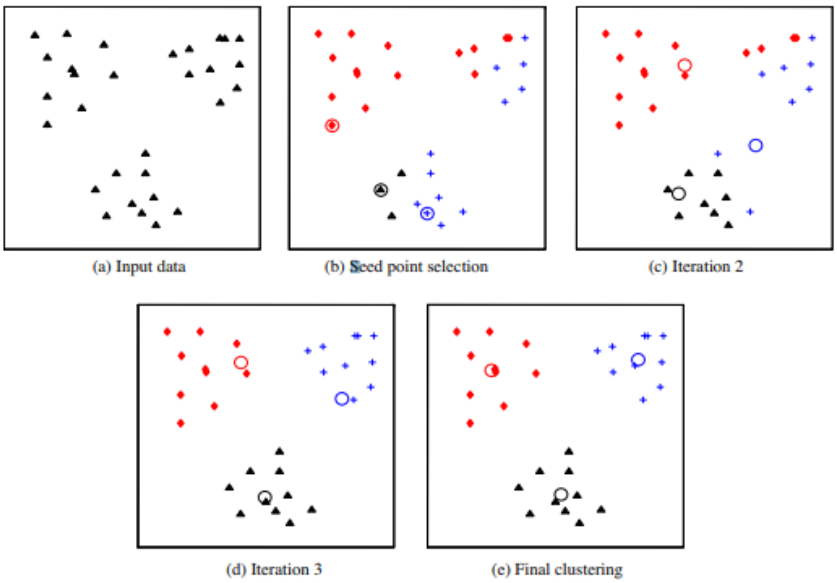

Figure 13 – Ilustration of K-Means algorithm (source [35])

3.1.5.2 Self-Organizing Maps

The other clustering approach used by this work is the Self-Organizing Map (SOM) algorithm. According to [65], the computation complexity of SOM algorithm is $O\ (n\ .\ c)$ where *n* is the input vector size and *c* is the number of document presentation cycles.

SOM algorithm was introduced by Teuvo Kohonen in 1982 [40]. Dino Isa et al [34] refer many applications of the SOM algorithm including “data mining, visualization of complex data, image processing, speech recognition, process control, diagnostics in industry and medicine and natural language processing”.

SOM is a type of artificial neural network but instead of an error correction approach as the method for learning the features of input data they apply competitive learning by using a neighborhood function to preserve the topological properties of the input space. The algorithm

produces a low-dimensional (typically two dimensions) discretized representation of the input space called a map that preserves the topological relation between the input data, simplifying visualization and interpretation.

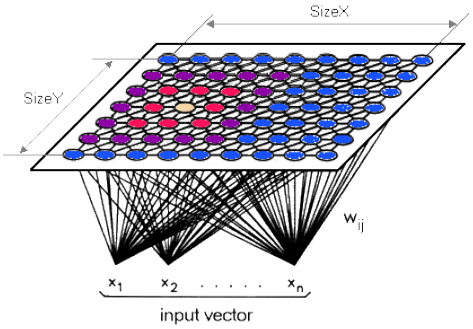

Figure 14 – SOM model (source: Sachin Joglekar's blog[1])

Adopting the description from [34], the SOM algorithm can be described in the following manner:

- The weight vectors of the SOM are initialized randomly or by principal component initialization;
- The SOM is trained iteratively. In each training step, a sample vector from the input data set is chosen randomly and the distance between this vector and all the weight vectors of the SOM, is calculated by using a Euclidean distance measure;
- The neuron with the weight vector which is closest to the input vector is called the Best Matching Unit (BMU). The distance between that vector and weight vectors, is computed using a distance measure, typically Euclidean distance;
- After the BMU is found, the weight vectors of the SOM are updated so that the BMU is moved closer to the input vector in the input space;
- The topological neighbors of the BMU are treated similarly. The update rule for the weight vector of *i* is:

$$x_i(t+1) = m_i(t) + \alpha(t)h_{ci}(t)[x(t) - m_i(t)] \qquad (3.2)$$

    - where x(t) is a vector which is randomly drawn from the input data set, and function a(t) is the learning rate and t denotes time. The function $h_{ci}(t)$ is the neighborhood kernel around the winner unit *c*.
    - Learning parameter is selected between 0.0 and 0.9
- The training steps will be in the range of 100 000 epochs in order to obtain a trained map.

Several research studies have shown that SOM are an effective approach for organizing text data, by considering the semantic similarities between text fragments. Samuel Kaski et al proposed the creation of word category maps for organizing vast collections of text documents.

[1] https://codesachin.wordpress.com

Their method proved to be effective in organizing documents on a map produced by a SOM approach [39].

H. Ritter and T. Kohonen [64] demonstrated the formation of semantic topographic word maps (Figure 15) where the semantic relationships in the data are reflected by their relative distances in the map. They hypothesized that SOM are effective not only on clustering and visualization of high dimensional data, but “they are also directly able to create in an unsupervised process topographical representations of semantic, nonmetric relationships implicit in linguistic data”.

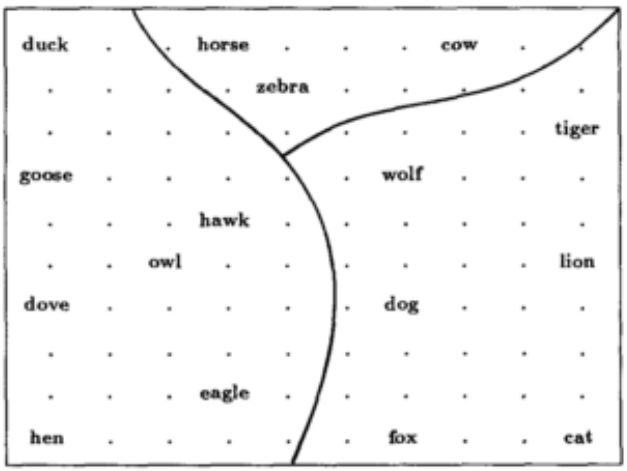

Figure 15 - Self-organizing semantic maps (source [64])

As previously mentioned, the Self-Organizing Map algorithm produces a 2-dimensional discretized representation of the input documents preserving the topological relations between the documents, meaning that the level of semantic similarity between documents is proportional to the closeness of the documents position in the 2- dimensional grid. This topological preserving property distinguishes this approach from K-Means and enables the use of distance measures based on the probability distribution over a 2-dimensional region, which is a measure that considers the topological properties in a 2-dimensional space.

Figure 16 shows the clustering of a dataset of text poems where the color represents the author. Each poem is associated with a cell in the 2 dimensional grid, and is possible to see that poems from the same author, possibly with similar words and semantics, are closer in the 2-dimensional space.

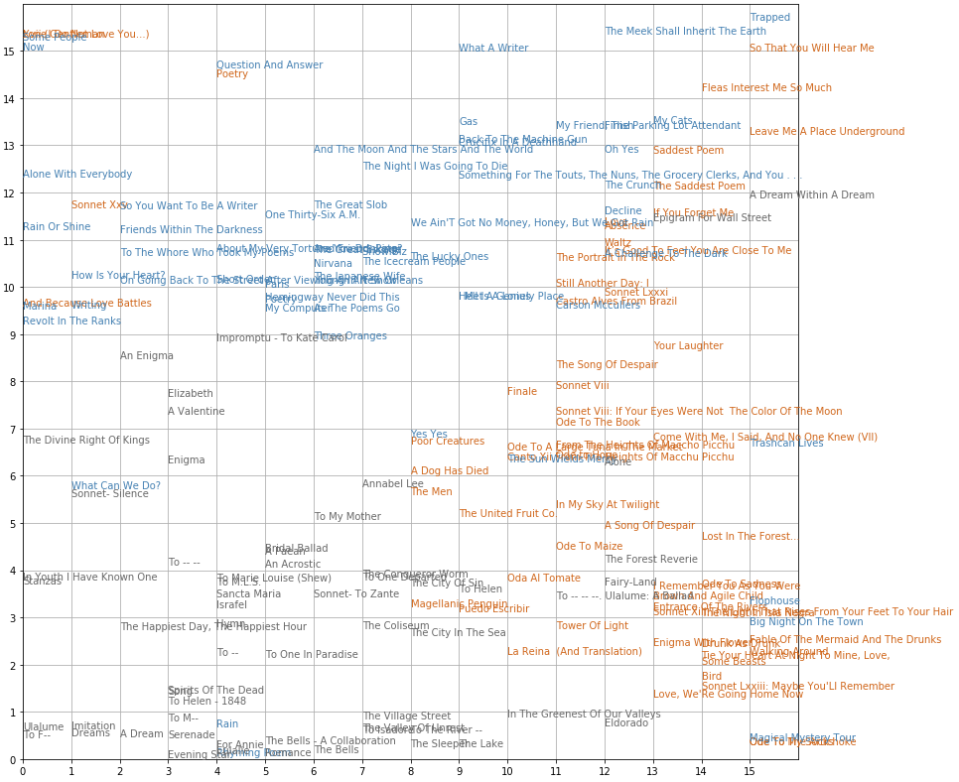

Figure 16 – Self-Organizing Map of Poems, adapted from [80]

3.1.6 From Document Clusters to Word Embeddings

At this point we have all the documents divided into similar groups. Based on distributional concepts theory, that considers that a correlation exists between words distribution and words meaning (similar words tend to appear next to each other), we assume that after documents are clustered into similar groups we are in conditions to derive a first raw word vector representation that encapsulates the concepts of the vector space model.

Inspired by semantic folding theory methodology for deriving binary sparse distributed word representations from clusters of document vectors, we developed an algorithm capable of producing a first word vector raw representation.

The dimensionality of the word vector representation is the same as the number of clusters *n* obtained in the previous clustering step. In the studied clustering methods this value is a predefined entry variable for the cluster algorithm. If *n* clusters were defined the word vector space would be in R^n. Assuming that the documents were clustered in *k* clusters, the pseudo algorithm for generating the vector representation of word *w* is as follows:

1. Identify all documents where the *w* is present;
2. Create an all zeros base vector (V_0) representation of dimensionality *k;*
3. For each document *d* identified in step 1:
    a. Calculate the number of occurrences of *w* in *d;*
    b. Find the cluster k_n closer in Euclidean distance to *d*;
    c. Increment V_0 dimension value corresponding to the cluster position by the value obtained in step 3a;

The described algorithm creates a *k* dimensional word vector representation that depending on the number of predefined cluster will be more or less sparse. For the same set of documents, increasing the number of clusters will generate sparser vectors, as the documents in which that word appears will be clustered in a higher dimensionality space.

3.1.7 Binary Sparse Vectors

Our initial goal is to represent words as binary and sparse vector representations in the vector space. Two factors have direct influence in the level of sparseness of the raw word vector: the first is the number of clusters defined in the document clustering phase and the second is the

number of occurrences of each word across all documents. As the level of sparseness in the vector can dramatically influence the experiment results, we assess the influence of this two variables in the quality of the sparse embeddings and try to reach the optimum value for them. The results obtained for similarity and analogy tasks, while varying these two variables are reported in the “Experiments and Results” section.

The raw vector representation obtained until this phase of the process is composed by a vector of integer values where each vector dimension value equals the sum of occurrences of a word in all documents grouped in a given cluster, considering that the number of clusters equals the dimensionality of the raw word vector.

For transforming this raw representation into a binary sparse representation we implemented an algorithm that takes as inputs the raw vector representation and the desired level of sparseness. The algorithm is as follows:

1. Considering vector space dimensionality *n* and percentage of sparseness *s*, obtain the maximum number of dimensions with 1 values: $S = (n \,.\, s)$;
2. Compute the histogram of the input vector representation;
3. Starting from the histogram bin with the higher range of values, calculate the histogram bins whose accumulated number of elements, do not exceed the assigned sparseness *S*;
4. Create a new vector with the same dimensionality of the input vector and all dimension values equal to 0; assign number 1 to all vector dimensions whose values are within the bins found in 3.

3.2 Implementation

The motivation of this section is to expose the implementation details of the proposed methodology. Supported in the conceptual model described above we propose a methodology for deriving word embeddings in an unsupervised way. All aspects concerning the practical implementation are explained in order to demonstrate feasibility, assessment and suitability for the intended purpose.

Given the proposed model, an important effort of our work was to optimize as many input variables as possible, in order to obtain good results. Our approach was to understand how each input model variable impacts the quality of the resulting word embeddings. Hence, throughout this chapter we assess the influence of input variables such as the size of corpus data, the type of word tokens, the vectorization algorithms, the cluster algorithms and the level of sparseness in the word vector representations.

We defined an assessment methodology that can be described as follows:

- For each assessed input variable:
    - Experiment several values between a given predefined range;
    - Maintain unchanged all other input variables, where each value is the mean of the given predefined range (except for corpus size where the lower limit range of 2 Wikipedia dump files were used).

Table 1 shows the summary of all the performed assessments, while each one of these are discussed in the sections below.

Table 1 – Quality assesements experiments

Variable	Range	Fixed value
Minimum document length	100; 200; 300; 400; 500	300
Number of Wikipedia dump files	2 ,3 ,4 ,5	2
Type of tokens	STEMM; WORDS	WORDS
Number of tokens in TF-IDF	1000, 3000, 5000, 7000, 9000	5000
Dimensionality after SVD	100, 200, 300, 400, 500, 600, 700	400
Clustering algorithm	KMEANS, SOM	KMEANS
Number of clusters	49, 100, 625, 4096, 16384	625
Vector Sparsity	0.01, 0.02, 0.03, 0.04	0.02

Our final goal is to produce sparse word embeddings with good performance across a variety of tasks, however for this kind of assessment we only tested against a small word similarity dataset,

namely RG-65[2] [66]. RG-65 dataset, besides the fact that it has been used extensively by the research community, is a small dataset composed of only 65 pairs of words, which is a great advantage as it allows expedite evaluation of word embeddings. For this reason we used this similarity dataset as a quality indicator, and assumed that the word embeddings tested against this specific task and dataset could have similar performances in other NLP tasks.

Thus, all variable assessments described in the following sections use, as quality indicator score, the Spearman correlation between cosine similarity of our derived word vector representations and the human labeled similarity of word pairs, considering RG-65 dataset.

3.2.1 Corpus

A large corpus or body of texts is the building block for the unsupervised generation of word embeddings. In this work we used Wikipedia as the source of our corpus. As our evaluation methodology considers generic benchmark datasets, in the sense that they are composed by words not belonging to a specific domain, we chose a generic corpus as well. Another important aspects for selecting Wikipedia were dimension and ease of obtaining the data.

The corpus was downloaded using https://dumps.wikimedia.org/enwiki/, specifically the snapshot copy of the English Wikipedia for January 2018. This Wikipedia snapshot size is about 14 GB, comprising 54 compressed xml dump files.

Next we transformed the information present in the Wikipedia compressed dump files from xml to *json* format. The result is a series of files, each containing a list of records in *json* format. Each *json* record is relative to one Wikipedia article and contains the following keys: *id, url, title* and *text*. An example of such a *json* record is:

“{"id": "239448", "url": "https://en.wikipedia.org/wiki?curid=239448", "title": "Eulipotyphla", "text": "Eulipotyphla\n\nEulipotyphla (\"truly fat and blind\") is an order of mammals suggested by molecular methods of phylogenetic reconstruction, and includes the laurasiatherian members of the now-invalid polyphyletic order Lipotyphla, but not the afrotherian members (tenrecs and golden moles, now in their own order Afrosoricida). Lipotyphla in turn had been derived by removing a number of groups from Insectivora, the previously used wastebasket taxon.\n\nThus, Eulipotyphla comprises the hedgehogs and gymnures (family Erinaceidae, formerly also the order Erinaceomorpha), solenodons (family Solenodontidae), the desmans, moles, and shrew-like moles (family Talpidae) and true shrews (family Soricidae). True shrews, talpids and solenodons were formerly grouped in the clade Soricomorpha; however, Soricomorpha has been found to be

[2] https://aclweb.org/aclwiki/RG-65_Test_Collection_(State_of_the_art)

paraphyletic, since erinaceids are the sister group of shrews.\n\nFamily-level cladogram of extant eulipotyphlan relationships, following Roca et al. and Brace et al.:\n"}"

The raw text used in our work is the part under the "*text*" *json* key and a simple comparison between this data and the actual Wikipedia page data shows that we are only considering text present in complete and regular sentences, meaning we are not considering text present in the Wikipedia page in the form of text structures like lists, sub-list, image captions or references. As we want to capture the meaning of words in a given sentence or group of sentences, the fact that these kind of text structures are not taken into account benefits our intent.

Another adopted definition in our work was the concept of document and the strategy to switch from Wikipedia Corpus to individual documents. We considered a document a peace of text limited by two or more line breaks. Considering the article above it will result in three individual documents.

According to Wikipedia official statistics[3], as of 12 October 2018, there are 5 730 629 articles in the English Wikipedia, with over 3.6 billion words (14.1 GB in compressed form). Given the large volume of data, our approach was to process and test our methodology with several and increasing sizes of corpus data. Simultaneously, this approach allows a better understanding on the impact of the corpus size on the final quality of the word embeddings, as one of the defined objectives of this work is to produce good quality embeddings with less training data.

The first assessment for optimizing the performance of the proposed model targeted the size of the corpus used as input. *A priori* the amount of text necessary for capturing word meanings should be a crucial factor in the outcome of our model, thereafter we assessed the influence of the number of Wikipedia page articles on the quality of the vector representations. Table 2 shows the number of Wikipedia file dumps, the cumulative number of Wikipedia articles and the number of words used for assessing the effect of corpus size in the quality of the embeddings. Last column indicates the number of words effectively supplied to our model, as pre-processing substantially reduces the total number of words, by removing stop words and other non-informative tokens.

[3] https://en.wikipedia.org/wiki/Wikipedia:Statistics

Table 2 – Wikipedia dump files used in assessment methodology

Number of Wikipedia file dumps	Number of Wikipedia articles	Number of words in source articles	Number of words after pre-processing
2	56 916	127 946 312	33 688 669
3	122 466	197 004 766	54 256 685
4	171 600	255 799 658	76 309 169
5	236 631	317 690 816	99 030 129

As expected Figure 17 depicts an increase in the quality indicator score for both clustering algorithms. As the number of Wikipedia articles increase, so the frequency of any word in the corpus increases, which enhances the proposed model, as it considers more semantic contexts (documents) for defining the word vector representation.

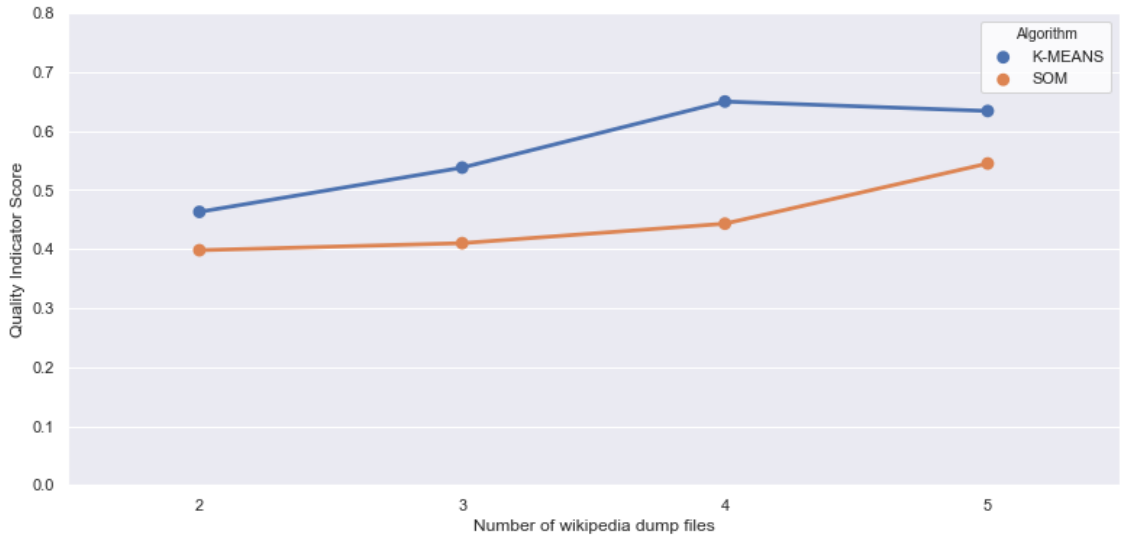

Figure 17 - Quality indicator score for increasing number of Wikipedia dump files

One of the objectives of this work is to produce word vector representations with less training material, with comparable performance in natural language processing intrinsic tasks. Hence, our implementation tries to encounter a good balance between corpus size and quality of the resulting representations.

Another potentially important variable in the proposed model is document length, in terms of number of words present in each document, as it directly influences vector representation and document clustering algorithms.

Considering the word similarity score for RG-65 dataset as an indicator of word embedding quality, Figure 18 shows the relation between this indicator with different lower limit document lengths (from 100 to 700 characters). It is possible to observe a decrease in the quality indicator with the increase of minimum document length. Considering all documents with more than 200

characters yield the best results with both K-Means and SOM cluster algorithms. This analysis should be considered with caution, as our test assessments considers a very limited size of Wikipedia articles (56 916), and if we further reduce this small corpus by considering only documents above 600 or 700 characters we end up with a very small corpus that should be insufficient for deriving good word embeddings.

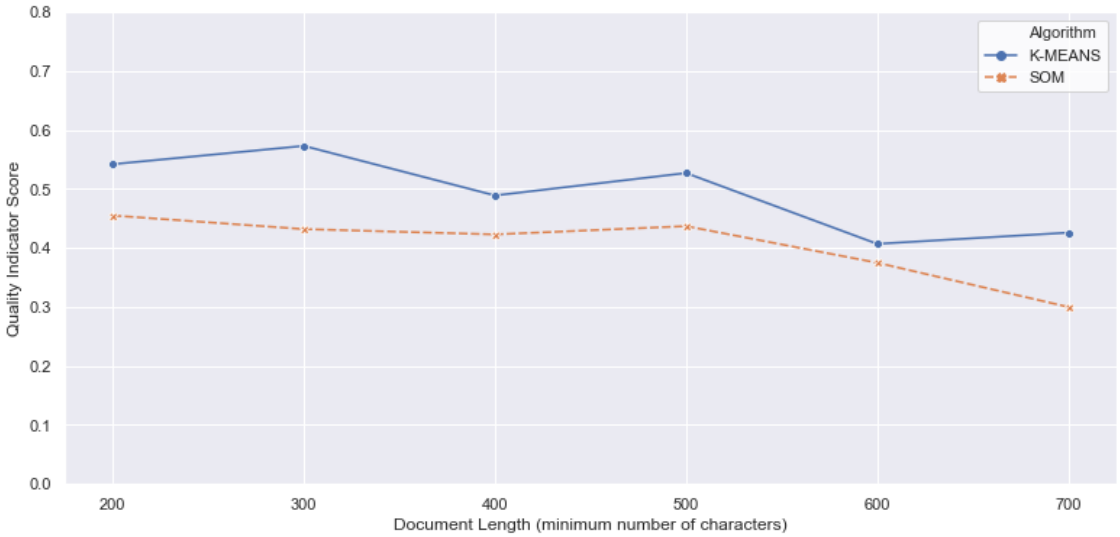

Figure 18 – Quality indicator score for increasing document lengths

3.2.2 Pre-processing

The intent of the preprocessing step is to achieve clean and normalized documents able to be transformed in a *N* dimensional vector representation. We compared two distinct document vector representations, resulting from different pre-processing methods.

The first approach considers all distinct words as they appear in the raw text, while the second includes a normalization step and takes into consideration only the stemming form of the words. Stemming is the process of reducing a word to his stem form, transforming several lexemes with the same root to only one token. For instance, the words "*work, works, working, worked*" will be represented by the same token "*work*".

In order to obtain the two distinct document versions, different pre-processing steps were applied. Documents composed of raw word tokens results from document cleaning only, while documents composed of word stem tokens results from document cleaning followed by document normalization, as is described:

1. Cleaning:
    a. Removal of all unmeaningful tokens: stop words, punctuation and numeric tokens.

2. Normalization:
    a. Obtaining the stem form of each word (e.g. worked -> work; bigger -> big)

Given the distinct nature of the two document representations we assessed which one could obtain better results. Therefore, following our input variable assessment methodology, we measured our word embedding quality indicator while varying the type of document tokens: word tokens and stem tokens. The results of this assessment are represented in Figure 19, indicating better results when stem tokens are used, in accordance with the assumption that word stems improve document representation in vectorization step.

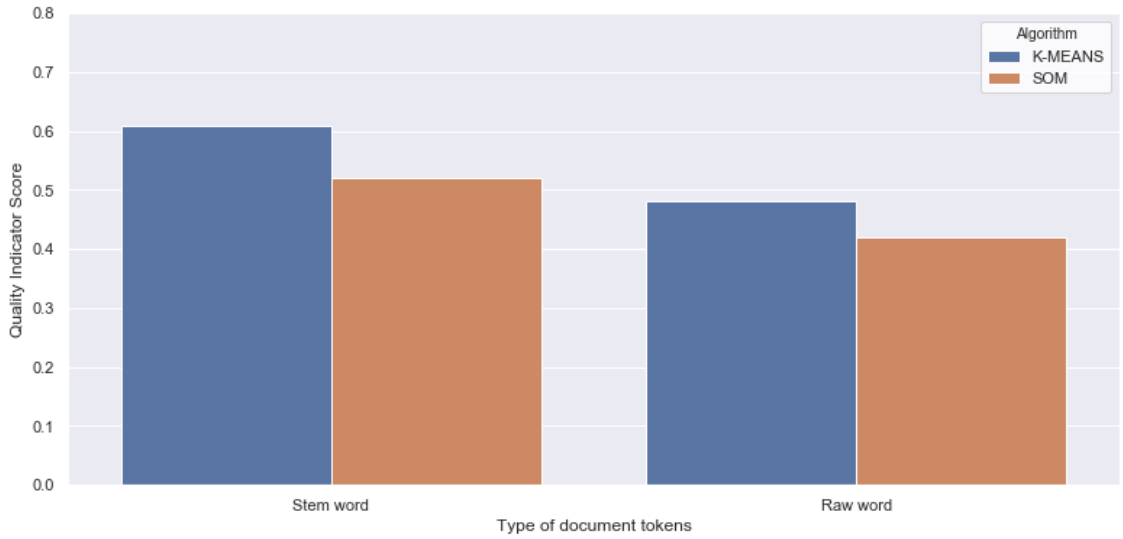

Figure 19 - Quality indicator score for different document tokens versions

At the final of this step, two distinct document versions containing two different types of tokens are ready for being transformed into document-term sparse matrices.

3.2.3 Vectorization

Vectorization allows the representation of each pre-processed document in a *N* dimensional space, so that it can be further processed by computing algorithms. As previously stated, our methodology involves three mathematical processing steps: calculate the frequencies, transform the raw frequency counts and reduce the dimensionality of the vector representations.

The first two steps were implemented by using TF-IDF vectorizer algorithm, which in practice applies a simple word count vectorizer followed by a transformation based on TF-IDF weighting. This weighting method allows more distinctive words (less common words) to express a higher value in the resulting document term matrix.

Recalling the previous step, two different versions of each document were presented to TF-IDF algorithm: documents with cleaned raw words and documents with clean word stems.

By default the algorithm yields a document term matrix (*n x m*) where typically both *n* and *m* are very large numbers. As all word tokens (raw word or stem word) are considered, each document will be represented by a vector of dimensionality equal to the number of distinct tokens present in the corpus. Working with very large matrices where the vector representations for documents and words have very large dimensionality is not practical for operational purposes, as most algorithms needs to load matrix data in memory.

In order to reduce the number of dimensions for document vectors representations we first set the maximum number of features (tokens) considered for building the vocabulary in the TF-IDF vectorizer algorithm. If specified, this parameter defines the number of tokens considered for document representation as the top tokens ordered by term frequency across the corpus. Another approach is to set another pair of input parameters to the TF-IDF vectorizer algorithm that sets the percentage of ignored tokens above and below a defined threshold in terms of frequency (e.g. neglect all tokens present in more than 80% of documents and in less than 2% of documents). In this way it is possible to eliminate simultaneously rare and very common words.

We applied our assessing methodology in order to understand the effect of the number of considered features on the quality of the resulting embeddings. Figure 20 shows the results, where we varied the number of features (and consequently the dimensionality of the document representation) between 1000 and 9000 and registered our embedding quality indicator. As expected, the results shows a tendency for better results with higher number of features. The document vector representation benefits when more features (tokens) are represented. At the limit, if we have considered all distinct tokens/features present in the corpus we would have obtained a document vector representation with dimensionality equal to the vocabulary size (in the order of hundreds of thousands). In this case, the document representation would be impractical to work in view of computing memory constraints. Our approach was to select a reasonable number of features considering these two competing variables: dimensionality and reliable document vector representation.

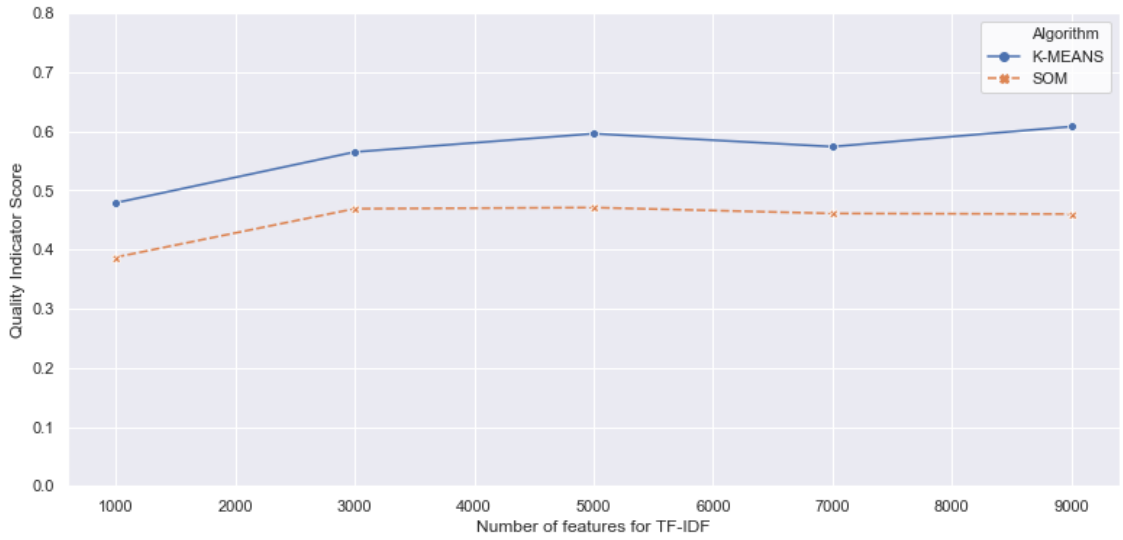

Figure 20 - Quality indicator score for different document vector dimensionalities

Following TF-IDF vectorization we further applied dimensionality reduction to the document vector representations, namely Singular Value Decomposition (SVD). Basically, it allows to represent a dataset using fewer dimensions by finding the most important dimensions of the data. In theory, finding the latent dimensions can benefit the document vector representation, as by encoding some information about other words (covariant dimensions) present in the same document it considers not only a count based model but a sense of the accompanying words in the document. It is important to note that, at this phase, applying SVD to our document term matrix is possible in computational terms because we considered a document vector representation with a limited dimensionality representation, otherwise it would be unfeasible considering computing memory and time constraints. Further reducing the document dimensionality allows for considerable improvements in clustering algorithm performance in following step of the methodology.

To understand the effect of singular value decomposition components on the quality indicator score, we varied this model parameter between 100 and 700 and registered the results present in Figure 21. The effect of different SVD components on quality score is not conclusive, however a value close to 400 seems to benefit the quality of the word representation on both clustering algorithms.

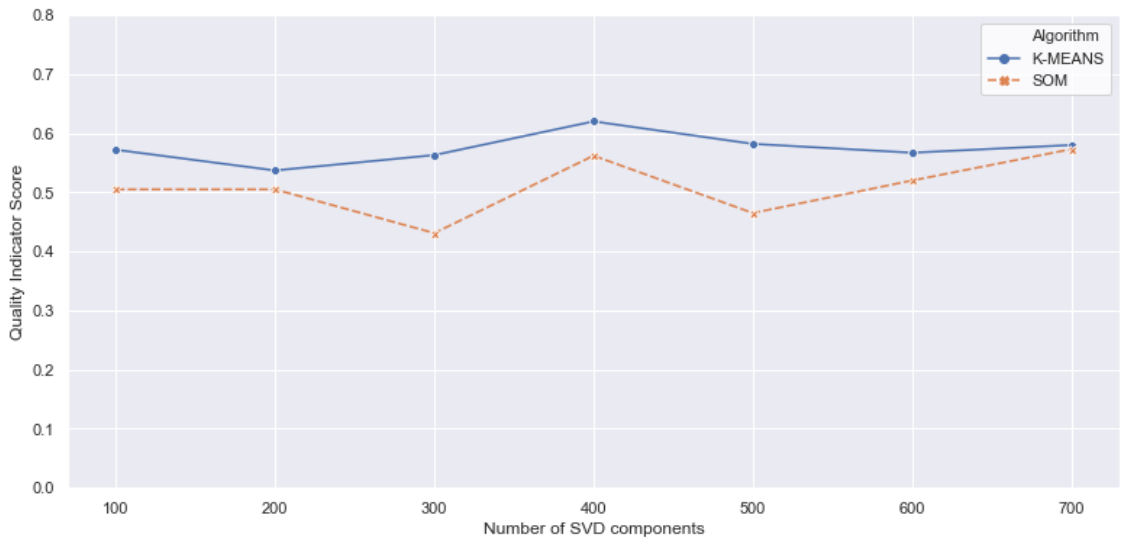

Figure 21 - Quality indicator score for different SVD components

3.2.4 Clustering

A crucial step on our methodology is being able to cluster the vector document representations in a way that semantic similar documents are clustered together. The clustering documents codebook, in which each document is assigned a specific cluster, will serve as basis for our first sparse vector representation, as will be detailed in the next section.

Considering that our document vectors are obtained via algorithms that allow dimensionality reduction, namely TF-IDF followed by SVD, the clustering algorithm is able to efficiently process a large number of document representations. Given the potential very large number of documents we needed to cluster, the clustering algorithms evaluated in this work should be efficient regarding memory and computer processing constraints. Another important criteria for cluster algorithm selection were the properties of the resulting clusters. Regarding this particular aspect, selected algorithms produce distinct cluster representation, as with K-Means clusters is not possible to establish any topological or closeness relation between clusters, whereas Self-Organizing maps produces clusters that have topological relations, meaning that documents assigned to clusters closer in the 2-dimensional SOM grid are more similar than documents assigned to clusters far apart.

Clustering algorithms are unsupervised technics that need to know, in advance, the number of clusters in which the samples will be organized. This input parameter is determinant for the final outcome of our model and will define the dimensionality of our word embeddings. As will be explained in the next section, a very large number of clusters will produce word embeddings with a very high dimensionality, and *vice versa*. As our model proposes to use binary sparse vector representations, the well-known disadvantages of working with high dimensionality vectors can be mitigated by using compressed sparse row matrices (CSR). This type of data structure allows efficient arithmetic operations and vector products, besides the important fact that serializing CSR to disk results in orders of magnitude lower file sizes.

The assessing methodology of the number of predefined clusters is represented in Figure 22. A different behavior is observed for K-Means and SOM clustering algorithms; SOM algorithm exhibits a more stable response to changing number of clusters, whereas K-Means shows a bigger variation on the quality indicator for different number of clusters. Less than 10 000 clusters seem to benefit quality when using K-Means algorithm, whereas for SOM when using a number of clusters around 15 000 better results are obtained.

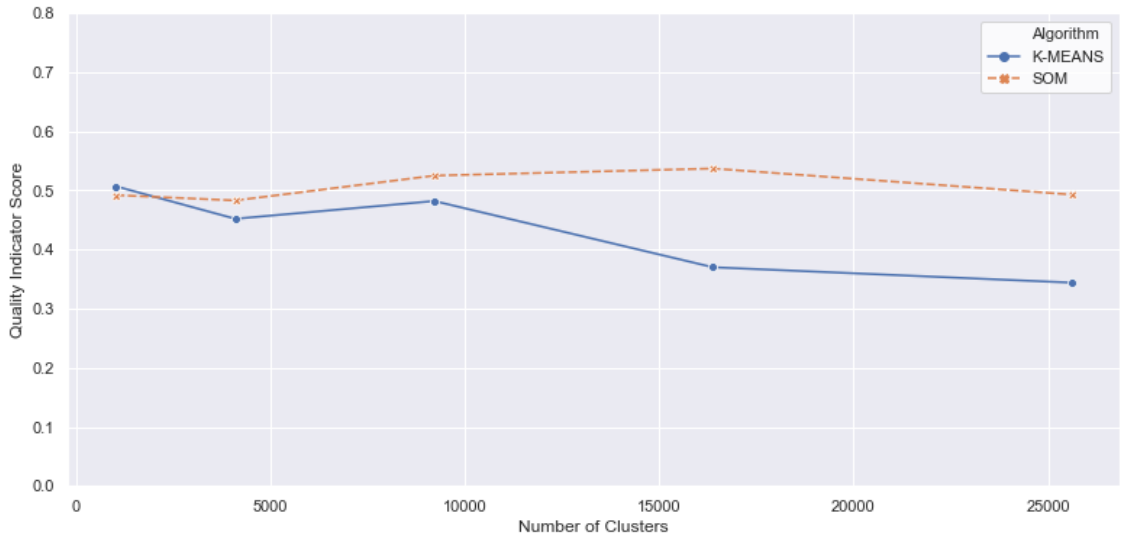

Figure 22 - Quality indicator score for different predefined clusters

3.2.5 Raw Word Vectors

At this phase of our work we are able to produce the first word vector representation. The output of the previous step is a clustering codebook where each pre-processed document is assigned to a specific cluster, accordingly to a given clustering algorithm. In other words, all the documents are distributed across a given number of clusters. As will be explained, the number of clusters defines the dimensionality of our word embeddings.

To shift from document clusters to word representations, firstly we implemented an algorithm that, for each word in the vocabulary, computes:

1. All the documents where that word occurs;
2. For each document identified in 1, identify the assigned cluster.

With this information we produced the word vector representation considering each cluster position as a single dimension of the word vector, where the value of each dimension is obtained counting the number of occurrences that word occurs in the documents assigned to that specific dimension/cluster. Figure 23 depicts the algorithm used for obtaining the word vector representations from document clusters. It shows how the word “apple” appearing in 4 hypothetic documents is transformed in a vector representation in space R^4. In this example, the final vector for word apple would be: $\vec{v} = [2,0,1,1]$.

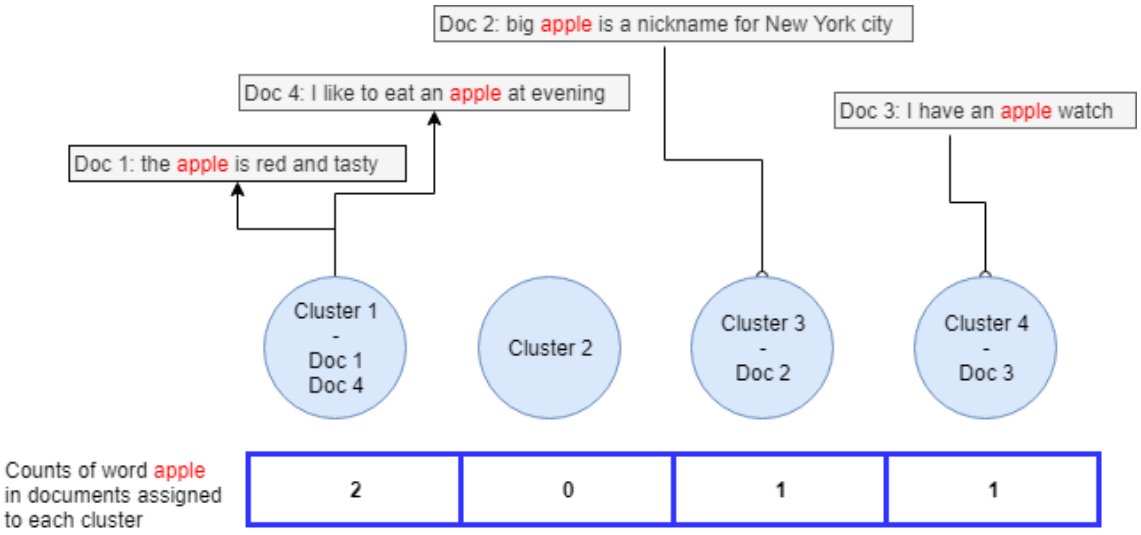

Figure 23 – From clusters to word vector representations

In order to implement the described method we calculated a large dictionary where for each key, word token in this case, we have a list of all document indexes and respective number of occurrences of that key in each document. An example for the word “car”, of the truncated dictionary representation would be:

car: [{'idx': 7, 'counts': 2}, {'idx': 91, 'counts': 1}, {'idx': 531, 'counts': 1}, {'idx': 680, 'counts': 1}, {'idx': 1075, 'counts': 1}, {'idx': 2158, 'counts': 1}, {'idx': 2622, 'counts': 1}, {'idx': 2868, 'counts': 1}, {'idx': 3100, 'counts': 3}, {'idx': 3148, 'counts': 1}, {'idx': 3326, 'counts': 1}, {'idx': 3432, 'counts': 2}, {'idx': 4197, 'counts': 1}, {'idx': 4812, 'counts': 1}, {'idx': 5083, 'counts': 2}, {'idx': 5134, 'counts': 1}, {'idx':

5800, 'counts': 1}, {'idx': 5817, 'counts': 1}, {'idx': 5961, 'counts': 1}, {'idx': 6020, 'counts': 1}, {'idx': 6026, 'counts': 1}, {'idx': 6040, 'counts': 1}, {'idx': 6164, 'counts': 1}, {'idx': 6287, 'counts': 1}, {'idx': 6574, 'counts': 1}, {'idx': 6742, 'counts' …

, where “idx” represents the index/reference of a given document.

This constitutes the database where for every word present in the corpus the documents and number of occurrences in each document, where the word is present, are given. As the size of the used Corpus grows, this database can easily grow to values that are difficult to manage, especially if all data needs to be serialized, deserialized and finally loaded into memory. As an example, a database dictionary of only five Wikipedia dump files, equivalent to about 8% of all Wikipedia, has a python pickle[4] serialized file size of 1.8 GB. For increasing performance, we divided the database file in several python pickle serialized files. Before dictionary database creation, another auxiliary data structure was produced in order to facilitate the process, essentially a list with the individual counts of every word for each document. An example for document with index 200 is shown:

Counter({'idx': 200, 'the': 14, 'and': 2, 'routes': 2, 'century': 2, 'with': 2, 'trade': 2, 'emergence': 1, 'key': 1, 'nile': 1, 'end': 1, 'these': 1, 'inhabitants': 1, 'west': 1, 'africa': 1, 'portions': 1, 'were': 1, 'nigeria': 1, 'remained': 1, 'its': 1, 'influence': 1, 'communities': 1, 'organized': 1, 'after': 1, 'scattered': 1, 'western': 1, 'sudan': 1, 'sahara': 1, 'communication': 1, 'linked': 1, 'way': 1, 'earlier': 1, 'encroaching': 1, 'that': 1, 'millennium': 1, 'made': 1, 'adjusting': 1, 'date': 1, 'trans': 1, 'third': 1, 'began': 1, 'widely': 1, 'time': 1, 'upper': 1, 'sahelian': 1, 'when': 1, 'south': 1, 'desert': 1, 'mediterranean': 1, 'avenues': 1, 'carthage': 1, 'much': 1, 'until': 1, 'since': 1, 'open': 1, 'prehistoric': 1, 'saharan': 1, 'establishing': 1, 'from': 1, 'islam': 1, 'desiccation': 1, 'same': 1, 'into': 1, 'cultural': 1})

3.2.6 Sparse Binary Vectors

The initial goal is to produce a sparse binary word vectors representations. This section explains the transformation from the word vector obtained in the previous step to a sparse binary vector representation.

The word vector obtained previously is of dimensionality *N*, equal to the number of predefined clusters in which all the pre-processed documents were clustered. The value of each dimension is obtained by counting the number of occurrences a specific token occurs in all documents assigned to that specific dimension/cluster. Figure 24 depicts an example of a word vector representation with 625 dimensions.

[4] https://docs.python.org/3/library/pickle.html

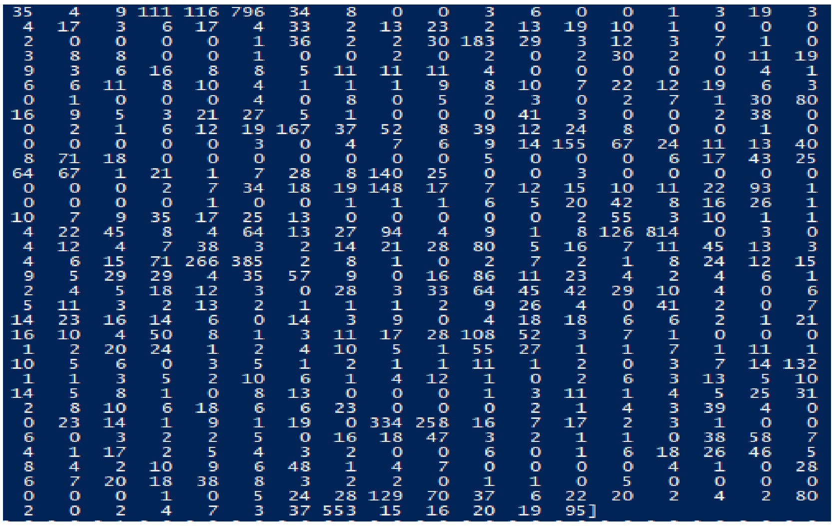

Figure 24 – Raw word vector representation with dimensionality 625

A python algorithm was developed to transform this vector representation into a sparse binary representation composed only by values 0 and 1, where the number of dimensions with 1's values is a small fraction of the vector dimensionality. The transformation of the vector shown in Figure 24 into a sparse binary vector representation is presented in Figure 25. As can be seen by comparing the two pictures only the higher values in the raw vector representation V in R^{625} produced 1 values in the sparse binary representation (e.g. $V_6 = 796$, $V_{620} = 553$).

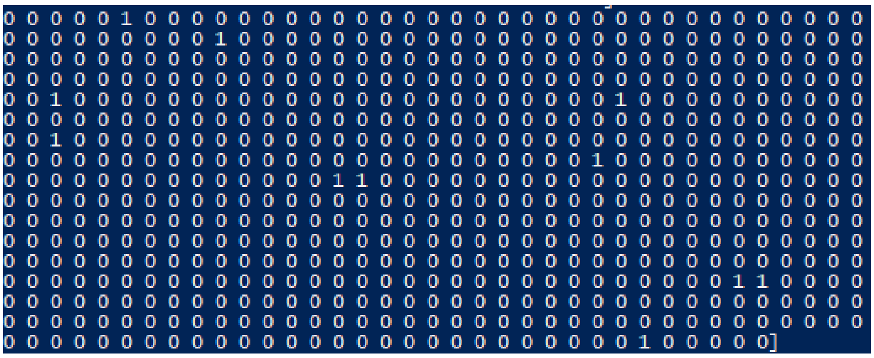

Figure 25 –Sparse binary word vector representation with 625 dimensions

The lower the percentage of 1 values in the binary vector the higher is the level of sparsity. The level of vector sparsity has a direct impact on the quality of the word embedding, thus we examined how the quality indicator is influenced by this parameter. Figure 26 suggests that the quality of word embeddings increases when that level of vector sparsity increases. The best results are obtained within the range of 98% to 99% of sparsity level, which means that only 1 to 2% of vector dimensions have values equal to 1.

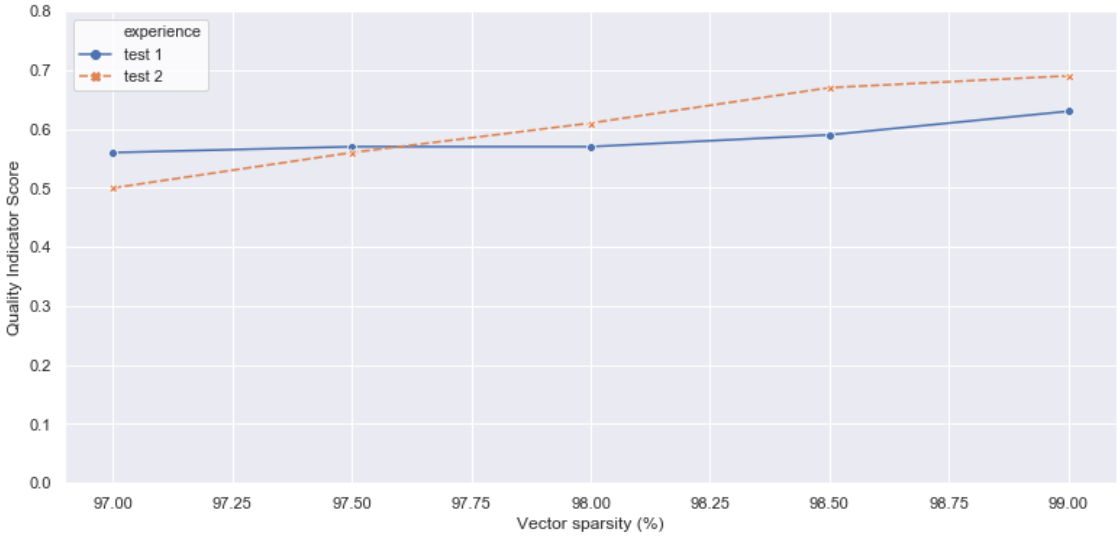

Figure 26 - Quality indicator score for different levels of vector sparsity

3.2.7 Word Embeddings Generation

The final step generates a reasonable number of word embeddings, one for each word present in the training corpus vocabulary. We built 100 000 word embeddings for the most frequents words in the corpus, reducing the probability of out of vocabulary words (OOV) in the following evaluation phase.

3.2.8 Instantiation

Considering the implementation steps described above and the performed assessments for optimizing the model parameters we proceed with the instantiation of our methodology. Two different instantiations, resulting from parameter optimization and knowledge from our assessment methodology were considered. Table 3 shows the details of the generated word embeddings, identified henceforward by the following designation: We-KM-8464, reflecting the K-Means clustering algorithm used in the process and 8464 as the final vector dimensionality.

Important to note that Self-Organizing Map clustering approach is absent for our chosen final instantiations. The reason lies in the experiments performed during our parameter assessment and optimization. We found that K-Means clustering achieves better or similar results than SOM for 10 000 or less number of clusters, with one considerable advantage, namely performance. In fact, the labels property of the K-Means algorithm implementation allows the expedite retrieval of the clusters assigned to all documents, when performing the raw word vectors generation. With

K-Means, all the clusters assignments for the documents are stored in a dictionary-like structure at the same time the clustering is performed, whereas SOM algorithm does not store this information and needs to find each assigned cluster to every document after the clustering process, on a computational time expensive process.

Table 3 –Word Embeddings variations

Parameter name	Parameter value	Word Embedding
Minimum Document length	300	We-KM-8464
Number Wikipedia articles dumps	5	
Type of tokens	Raw words	
Number of tokens in TF-IDF	10000	
Dimensionality in SVD	200	
Clustering Algorithm	K-Means	
Number of clusters	8464	
Sparsity	1%	

The word embeddings exhibits the properties presented in Table 4. The reduced file size of the embeddings, when compared with state of the art embeddings (Table 5), is due to the sparse binary vector structure of the embedding. Sparse binary vectors only need to store the position of the vector dimensions where the value is 1 (all other dimensions will be 0), strongly reducing the amount of memory needed for storing the vector information. In this work we used scipy sparse csr_matrix[5] class to store the vector information.

The training corpus used for deriving our word embeddings is composed of 100 million words, much less that the training corpus used for deriving the state of the art embeddings, as we describe later on this work. In fact, this value is 2.7% the number of words used for training publicly available FastText word embeddings, which uses the all Wikipedia composed of 3.6 billion words. Using such a small training corpus is aligned with our research objectives, as we want to obtain good quality word embeddings with considerable less training material.

Table 4 – Generated word embeddings properties

Word Embeddings	Vector Dimension	Vocabulary	Number words (millions)	Corpus	Size
we-km-8464	8464	100 000	100	Wikipedia	10 MB

[5] https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html

Considering the described methodology, Figure 27 summarizes on a single picture the implemented artifact. It considers a series of sequential steps that constitute a data pipeline, assuring the unsupervised transformation of text data from Wikipedia dump files to word embeddings. In practice it can be applied to other corpus, such as twitter data, or other domain specific corpus.

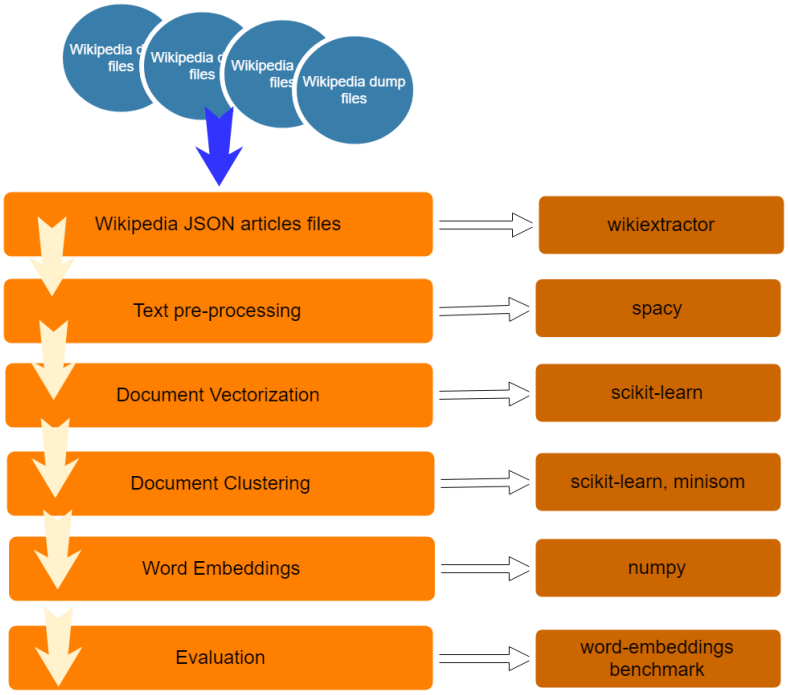

Figure 27 – Data pipeline implementation

In the scope of this work, *sparse-nlp*[6] library was developed and made publicly available as the result of the implementation effort done in the scope of this research work. The library is written in python version 3.6 programming language and is built on top of several other python libraries. The main modules and dependencies are:

1. Wikipedia dump files processing: wikiextractor[7]
2. Data pre-processing: pandas[8], nltk[9] , regular expressions
3. Document vectorization: scikit-learn[10]
4. Document clustering: scikit-learn, minisom[11] [80]

[6] https://github.com/avsilva/sparse-nlp

[7] http://attardi.github.io/wikiextractor/

[8] https://pandas.pydata.org/

[9] https://www.nltk.org/

[10] http://scikit-learn.org/

[11] https://github.com/JustGlowing/minisom

5. Word embeddings formation: numpy[12], nltk
6. Evaluation: word-embeddings-benchmarks[13], scipy[14]

The evaluation methodology, along with the software libraries used in that particular phase of this work is described in the next chapter.

[12] http://www.numpy.org/

[13] https://github.com/kudkudak/word-embeddings-benchmarks/wiki

[14] https://www.scipy.org/

4 Demonstration and Evaluation

The evaluation applied in this work is based on the methodology proposed by Stanisław Jastrzebski et al. [36] in their work “How to evaluate word embeddings? On importance of data efficiency and simple supervised tasks”.

The main criteria for selecting this evaluation methodology are expressed in four items present in the referred benchmark framework:

- Comparison with a considerable range of state of the art word embeddings, considering representatives of several architectures (e.g. deep learning, shallow neural networks)
- Evaluation of simple NLP intrinsic unsupervised tasks focused on word embeddings (e.g. word analogy, word similarity, word categorization);
- Evaluation on different dataset sizes;
- Possibility to reproduce and verify all the benchmark evaluation results.

The availability of all benchmark evaluation code, allowing the expedite verification, in situ, of all evaluation results, including the download of state of the art word embeddings and benchmark datasets, was a strong factor that favor the selection of this methodology.

Hence, from the base methodology, referred above, a subset of evaluation items were selected following the objectives proposed for this work.

4.1 Word Embeddings

The adapted evaluation methodology compares the sparse distributed vector representations derived by this work with 20 state of the art word embeddings. Table 5 describes their main properties considering vector dimension, number of words present in the derived embedding (vocabulary), number of tokens in the training corpus and corpus.

Table 5 – Word Embeddings properties

Word Embeddings	Vector Dimension	Number tokens (billions)	Corpus	Size
GloVe-6B-300D	300	6	Wikipedia + GigaWord	989 MB
GloVe-6B-200D	200			661 MB
GloVe-6B-100D	100			331 MB
GloVe-6B-50D	50			163 MB
GloVe-42B-300D	300	42	Common Crawl	4.67 GB
GloVe-27B-200	200	27	Twitter	1.91 GB
GloVe-27B-100	100			974 MB
GloVe-27B-50	50			487 MB
GloVe-27B-25	25			245 MB
FastText	300	3.6	Wikipedia	6.44 GB
LexVec	300	58	Common Crawl	2.1 GB
SG Google News	300	100	Google News	3.64 GB
NMT EN->FR[15]	--	--	--	--
NMT EN->DE[16]	--	--	--	--
HDC 300D	300	3.6	Wikipedia	1.04 GB
HDC 100D	100			363 MB
HDC 50D	50			180 MB
PDC 300D	300			1.03 GB
PDC 100D	100			353 MB
PDC 50D	50			175 MB

GloVe embeddings [60] combine the advantages of count-based matrix factorization and window predict neural network methods and are represented by nine versions of different dimensions and training corpus.

Word2Vec [53] learns word embeddings by training a shallow neural network to predict neighboring words, without considering a computationally expensive hidden layer and allowing the language model to take additional context into account.

FastText [9] is an extension of Word2Vec that considers sub word information for deriving word vectors. Based on the skip gram model, the vector representations are associated to each character n-grams and the words vector representations are the sum of these representations.

[15] It was not possible to obtain the word embeddings and verify their properties

[16] It was not possible to obtain the word embeddings and verify their properties

LexVec [70] is a method for factorizing PPMI matrices that uses skip-gram with negative sampling and stochastic gradient descent to minimize a loss function that weights frequent co-occurrences heavily but also takes into account negative co-occurrence.

Neural Translation Machine (NMT) [31] learns word embeddings by training a model, based on recurrent neural networks (RNN) that aims to translate a sequence of words from one language to another.

Hierarchical Document Context (HDC) and Parallel Document Context (PDC) [78] methods obtain word embeddings by jointly modeling syntagmatic and paradigmatic relations, augmenting the representation of words due to the mutual enhancement between these two types of relations. PDC is an extension of CBOW model whereas HDC explores the skip-gram model.

4.2 Evaluation Tasks and Datasets

Regarding NLP tasks, Schnabel and Labutov [72] divides the evaluation methods of word embeddings into two groups: extrinsic evaluation methods (e.g. part-of-speech tagging) and intrinsic evaluation methods that directly test embeddings for preserving syntactic of semantic relations, like word similarity (WS) or word analogy (WA).

The evaluation methodology applied in this work targets three intrinsic evaluation tasks:

- Word Similarity - given a pair of words estimate the similarity score between them;
- Word Analogy - given one pair of words with a semantic relation and a third word, predict the fourth word, such as the semantic relation between third and fourth words is equivalent to the semantic relation between first and second words;
- Word Categorization - given a dataset of words group similar semantic words.

For each kind of NLP intrinsic task used for word embeddings evaluation, one or more publicly available benchmark datasets were used.

Analogy datasets are composed of quadruples (two pairs of words in a specific relation, e.g. king - queen, man - woman). Similarity datasets are composed of pairs of words that are assigned a rank similarity value by human annotators. Categorization datasets are composed of several text files, where the content of each file is a set of words related to the same subject or category (e.g. trees, animals, cities, crimes).

All the datasets used in the evaluation methodology, considering the three different types of NLP task are present in Table 6.

Table 6 – Datasets for word vector evaluation

Dataset	NLP Task
MEN	Word Similarity
MTurk	
RG65	
RW	
SimLex999	
WS353	
WS353R	
WS353S	
Google	Word Analogy
MSR	
AP	Word categorization
BLESS	
Battig	
ESSLI_1a	
ESSLI_2b	
ESSLI_2c	

These datasets define a benchmark baseline suitable for comparing results obtained by the proposed methodology. Another benefit of this approach is that it allows the evaluation of word similarity and word analogy tasks using an unsupervised approach, where target word will be predicted by a simple distance measure calculation (e.g. cosine similarity).

4.2.1 Similarity

The MEN Test Collection [10] contains two sets of English word pairs (one for training and one for testing) together with human-assigned similarity judgments, obtained by crowdsourcing using Amazon Mechanical Turk via the CrowdFlower interface. The collection can be used to train and/or test computer algorithms implementing semantic similarity and relatedness measures.

The MTurk dataset [62] is composed by Human labeled examples of word semantic relatedness that considers patterns of word usage over time. Each pair of words was evaluated by 10 people on a scale of 1-5.

RG-65 or Rubenstein & Goodenough dataset [66] is composed by 65 word pairs where similarity of each pair is scored according to a scale from 0 to 4. The similarity values in the dataset are the means of judgments made by 51 subjects.

RW or rare words dataset [49] is composed of 2034 word pairs that are relatively rare where similarity score of each pair is given by humans. The dataset construction methodology first selects a list of rare words, then for each of the rare words, finds another word (not necessarily rare) to form a pair and finally collect human judgments on how similar each pair is.

SimLex999 [30] is a gold standard resource for the evaluation of models that learn the meaning of words and concepts. It provides a way of measuring how well models capture similarity, rather than relatedness or association. Experiments made by the authors of this dataset indicate that this is a challenging dataset for language models that infer connections between words from their co-occurrence in corpora, which essentially reflects relatedness not similarity.

WordSim353 (WS353) [23] [1] is a test collection composed by 353 word pairs that can be divided in two subsets: WS353S for measuring word similarity and WS353R for measuring relatedness. For each word pair a human-assigned similarity score is given.

4.2.2 Analogy

Google word analogy dataset [53] tests both semantic and syntactic analogies. It is composed by 19 544 analogy pairs (8 869 semantic and 10 675 syntactic) corresponding to 14 types of relations.

MSR [55] is composed of 8 000 analogy pairs of words for testing morphological questions in the form of "a is to b as c is to", testing:

- Base/comparative/superlative forms of adjectives;
- Singular/plural forms of common nouns;
- Possessive/non-possessive forms of common nouns;
- Base, past and 3rd person present tense forms of verbs.

4.2.3 Categorization

AP dataset [2] is, according to its authors, a balanced dataset with respect to three factors: class type, frequency, and ambiguity. It aims to be a balanced dataset as to ambiguity, estimated on the basis of the number of senses in WordNet.

BLESS dataset [4] includes 200 concrete nouns (100 animate and 100 inanimate nouns) from different classes (e.g. tools, clothing, vehicles, animals, etc.) designed for the evaluation of distributional semantic models.

Battig dataset [5] comprises a ranked list of 5 231 words listed in 56 taxonomic categories. People were asked to list as many exemplars of a given category in 30 seconds after which time the next category name was presented. Examples of taxonomic categories are: bird, color, country, disease, fish and fruit.

ESSLI dataset is divided in three sub-datasets. The ESSLI_2c consists of 45 verbs, belonging to nine semantic classes. The ESSLI_2b is formed by 40 abstract nouns extracted from the MRC Psycholinguistic Database and classified into three classes: highly, low and medium abstract nouns. The third sub-dataset, ESSLI_1a, consists of 44 concrete nouns, belonging to six semantic categories. The goal of the three sub-datasets is to group words into semantic categories.

4.3 Evaluation Metrics

Depending on the type of NLP task being evaluated, one of three different evaluation methods is applied.

For word similarity tasks, cosine similarity is used, where similarity between two vectors is calculated by their cosine similarity:

$$similarity = (\cos(\overrightarrow{v1}, \overrightarrow{v2})) \qquad (4.1)$$

Then, Spearman correlation between cosine similarity of the word embeddings and human rated similarity of word pairs is calculated to score the similarity task.

In word analogy tasks, an embedding is evaluated for its ability to infer the fourth word out from the first three. 3COSADD is the method used for solving these type of word analogies:

$$3\text{COSADD} = (arg\max \vec{v} \in V\, cos\,(\overrightarrow{v}, \overrightarrow{v2} - \overrightarrow{v1} + \vec{v}3) \qquad (4.2)$$

For word categorization tasks, purity evaluation measure is used. Briefly, purity assigns each cluster to the class which is most frequent in the cluster; then the accuracy of the clustering process is calculated by counting the number of correctly assigned samples and dividing by the total number of samples. From [51], the equation is shown below

$$purity\ (\Omega, \mathbb{C}) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j| \qquad (4.3)$$

, where $\Omega = \{\omega_1, \omega_2, \ldots \omega_k\}$ is the set of clusters and $\mathbb{C} = \{c_1, c_2, \ldots c_j\}$ is the set of classes. A perfect clustering has a purity value of 1, and a value close to 0 when clustering is poor.

4.4 Experiments and Results

As described, the evaluation methodology considers datasets and word embeddings that are publicly available. All the results presented in this work are reproducible by running a python script adapted from the public code repository[17] provided by Stanisław Jastrzebski. For the three evaluation tasks we compared the results obtained by our word embeddings (We-KM-8464) with 20 state of the art word embeddings. Considering that these 20 word embeddings are obtained with a training corpus of several orders of magnitude bigger than the corpus used on our methodology, we derived Word2Vec embeddings considering the same training corpus (five Wikipedia dump files composed of 100 million words) used for deriving our word embeddings. Thus, we obtained 100 dimensions word embeddings trained using gensim[18] with skip-gram algorithm with negative sampling. Thereafter, in all the evaluation tasks performed, we include these word embeddings, identified by **Word2vec-100-wiki5**. Our results can be better evaluated when analyzed with Word2vec-100-wiki5, because both are trained on the same training corpus.

4.4.1 Word Similarity

The results for the word similarity evaluation task are presented in Table 7.

Table 7 – Word Similarity evaluation results

	MEN	MTurk	RG65	RW	SimLex 999	WS353	WS353R	WS353S	Average results
PDC dim=300	0.773	0.672	0.790	0.455	0.427	0.721	0.641	0.789	0.659
FastText	0.763	0.679	0.799	0.479	0.380	0.705	0.655	0.753	0.651

[17] https://github.com/kudkudak/word-embeddings-benchmarks

[18] https://radimrehurek.com/gensim/

SG GoogleNews (word2vec)	0.741	0.670	0.761	0.471	0.442	0.700	0.635	0.772	0.649
HDC dim=300	0.760	0.655	0.806	0.438	0.407	0.677	0.581	0.787	0.639
PDC dim=100	0.755	0.710	0.774	0.421	0.361	0.690	0.606	0.779	0.637
LexVec which="commoncrawl-W+C"	0.809	0.712	0.765	0.478	0.419	0.647	0.571	0.756	0.631
PDC dim=50	0.720	0.700	0.763	0.390	0.309	0.637	0.543	0.741	0.600
HDC dim=100	0.738	0.648	0.804	0.388	0.324	0.617	0.523	0.753	0.599
Word2vec-100-wiki5	**0.697**	**0.669**	**0.756**	**0.332**	**0.340**	**0.617**	**0.518**	**0.737**	**0.583**
GloVe dim=300 corpus=common-crawl-42B	0.736	0.645	0.817	0.376	0.374	0.553	0.473	0.669	0.580
GloVe dim=300 corpus=wiki-6B	0.737	0.633	0.77	0.359	0.371	0.522	0.446	0.653	0.561
HDC dim=50	0.708	0.649	0.723	0.361	0.281	0.575	0.472	0.713	0.560
GloVe dim=200 corpus=wiki-6B	0.710	0.620	0.713	0.331	0.340	0.489	0.418	0.615	0.530
GloVe dim=100 corpus=wiki-6B	0.681	0.619	0.676	0.310	0.298	0.451	0.380	0.587	0.500
We-KM-8464	**0.618**	**0.548**	**0.638**	**0.117**	**0.160**	**0.578**	**0.595**	**0.583**	**0.480**
NMT which=FR	0.492	0.464	0.590	0.301	0.460	0.488	0.444	0.572	0.476
NMT which=DE	0.492	0.464	0.590	0.301	0.460	0.488	0.444	0.572	0.476
GloVe dim=50 corpus=wiki-6B	0.652	0.619	0.595	0.285	0.265	0.419	0.348	0.554	0.467
GloVe dim=200 corpus=twitter-27B	0.594	0.555	0.698	0.197	0.130	0.451	0.373	0.59	0.449
GloVe dim=100 corpus=twitter-27B	0.577	0.559	0.677	0.21	0.122	0.442	0.364	0.592	0.443

GloVe dim=50 corpus=twitter-27B	0.531	0.515	0.574	0.196	0.098	0.392	0.325	0.54	0.396
GloVe dim=25 corpus=twitter-27B	0.444	0.481	0.503	0.173	0.073	0.307	0.235	0.458	0.334

Word embeddings are ordered by the average result considering all similarity datasets, value that is shown on the last column of Table 7. Our embeddings present comparable results with state of the art word embeddings and occupy the 15º position considering the average results across all datasets.

Comparing with Word2vec-100-wiki5, the obtained results are 10 points below that specific word embedding. Even though the results are similar our methodology did not outperform Word2Vec algorithm using the same training corpus. Only with WS353R dataset our results outperformed Word2vec-100-wiki5. In fact, considering WS353R relatedness dataset only four state of the art embeddings achieved better results.

The worst results were obtained with RW and SimLex999, positioning our word embeddings in the bottom of the rank considering these datasets.

Nevertheless, taking into account our experiments with Word2vec-100-wiki5, and comparing the results with word2vec-SG GoogleNews, if state of the art word embeddings were trained with 100 million words, as done on our work, the rank position of We-KM-8464 would be considerably higher.

4.4.2 Word Analogy

Word analogy evaluation results are presented in Table 8.

Table 8 – Word Analogy evaluation results

	Google	**MSR**	**Average Results**
GloVe dim=300 corpus=common-crawl-42B	0.750	0.702	0.726
PDC dim=300	0.748	0.596	0.672
GloVe dim=300 corpus=wiki-6B	0.718	0.616	0.667

LexVec which="commoncrawl-W+C"	0.710	0.601	0.656
HDC dim=300	0.731	0.564	0.648
GloVe dim=200 corpus=wiki-6B	0.698	0.596	0.647
PDC dim=100	0.704	0.543	0.624
GloVe dim=100 corpus=wiki-6B	0.632	0.551	0.592
FastText	0.655	0.521	0.588
HDC dim=100	0.667	0.497	0.582
SG GoogleNews (word2vec)	0.402	0.712	0.557
GloVe dim=200 corpus=twitter-27B	0.534	0.503	0.519
Word2vec-100-wiki5	**0.524**	**0.439**	**0.481**
PDC dim=50	0.579	0.369	0.474
HDC dim=50	0.534	0.347	0.441
GloVe dim=100 corpus=twitter-27B	0.429	0.428	0.429
GloVe dim=50 corpus=twitter-27B	0.260	0.271	0.409
NMT which=FR	0.212	0.434	0.323
NMT which=DE	0.212	0.434	0.323
GloVe dim=50 corpus=wiki-6B	0.462	0.356	0.266
We-KM-8464	**0.176**	**0.180**	**0.178**
GloVe dim=25 corpus=twitter-27B	0.111	0.116	0.114

Word embeddings are ordered by the average result considering all analogy datasets. The results shows that for analogy tasks our methodology does not achieve comparable results, even though they are better than GloVe dim=25 embeddings trained on twitter corpus.

Word2vec-100-wiki5 achieves 0.481 points whereas our embeddings only achieve an average result of 0.178. A closer look on the analogy results by category helps to explain the not so good performance results. We observe very unbalanced results by category for both Google (Table 9) and MSR (Table 10) datasets. For Google dataset, whereas reasonable scores where obtained

for categories like capital-common-countries and gram6-nationality-adjective, the majority of the classes performed poorly. The same is true for MSR dataset, where we can distinct good results for nouns categories and poor results for adjectives and verbs.

Table 9 – Scores by category for Google analogy dataset

Analogy Category	**Score for We-KM-8464**
capital-common-countries	0.227
capital-world	0.154
city-in-state	0.074
currency	0.009
family	0.138
gram1-adjective-to-adverb	0.021
gram2-opposite	0.000
gram3-comparative	0.180
gram4-superlative	0.004
gram5-present-participle	0.167
gram6-nationality-adjective	0.443
gram7-past-tense	0.260
gram8-plural	0.515
gram9-plural-verbs	0.160

Table 10 - Scores by category for MSR analogy dataset

Analogy Category	Score for We-KM-8464
Adjective -> Adjective,comparative	0.058
Adjective -> Adjective,superlative	0.004
Adjective, comparative -> Adjective	0.072
Adjective, comparative -> Adjective, superlative	0.018
Adjective, superlative -> Adjective	0.020
Adjective, superlative ->Adjective, comparative	0.032
Noun, singular -> nnpos	0.354
Noun, singular -> Noun, plural	0.586
Noun Possessive ending -> Noun, singular	0.422
Noun, plural -> Noun, singular	0.540
Verb, base form -> Verb, past tense	0.132
Verb, base form ->_vbz	0.168
Verb, past tense -> Verb, base form	0.168

Verb, past tense -> Verb, 3rd person singular present	0.006
Verb, 3rd person singular presente-> Verb, base form	0.284
Verb, 3rd person singular presente -> Verb, past tense	0.018

4.4.3 Word Categorization

Word categorization evaluation results are presented in Table 11.

Table 11 – Word Categorization evaluation results

	AP	BLESS	Battig	ESSLI_1a	ESSLI_2b	ESSLI_2c	Average Results
FastText	0.654	0.845	0.438	0.772	0.75	0.666	0.687
GloVe dim=300 corpus=wiki-6B	0.637	0.820	0.41	0.773	0.825	0.644	0.685
LexVec which="commoncrawl-W+C"	0.612	0.795	0.438	0.818	0.750	0.667	0.680
HDC dim=300	0.632	0.815	0.432	0.773	0.750	0.644	0.674
SG GoogleNews (word2vec)	0.649	0.795	0.406	0.750	0.800	0.644	0.674
HDC dim=100	0.619	0.825	0.432	0.773	0.750	0.622	0.670
PDC dim=300	0.639	0.805	0.431	0.773	0.725	0.644	0.670
GloVe dim=200 corpus=wiki-6B	0.634	0.810	0.423	0.773	0.725	0.622	0.665
GloVe dim=300 corpus=common-crawl-42B	0.622	0.785	0.451	0.795	0.75	0.578	0.664
GloVe dim=100 corpus=wiki-6B	0.644	0.780	0.435	0.705	0.750	0.644	0.660
PDC dim=100	0.632	0.760	0.431	0.727	0.750	0.622	0.654

GloVe dim=50 corpus=wiki-6B	0.634	0.725	0.391	0.773	0.750	0.600	0.646
Word2vec-100-wiki5	**0.644**	**0.710**	**0.422**	**0.704**	**0.775**	**0.60**	**0.642**
PDC dim=50	0.617	0.760	0.426	0.682	0.750	0.556	0.632
HDC dim=50	0.555	0.730	0.429	0.705	0.775	0.578	0.629
GloVe dim=200 corpus=twitter-27B	0.515	0.690	0.326	0.773	0.700	0.578	0.597
GloVe dim=100 corpus=twitter-27B	0.500	0.675	0.315	0.727	0.675	0.60	0.582
GloVe dim=50 corpus=twitter-27B	0.458	0.665	0.308	0.705	0.675	0.511	0.554
GloVe dim=25 corpus=twitter-27B	0.453	0.545	0.267	0.659	0.700	0.489	0.519
We-KM-8464	**0.440**	**0.550**	**0.275**	**0.680**	**0.525**	**0.489**	**0.493**
NMT which=FR	0.420	0.445	0.165	0.568	0.700	0.644	0.490
NMT which=DE	0.415	0.445	0.165	0.568	0.700	0.622	0.486

Word embeddings in Table 11 are ordered by the average result considering all categorization datasets. Our embeddings present comparable results with state of the art word embeddings, nevertheless they only achieved better results than NMT approaches, considering the average results across all datasets.

Considering average results, Word2vec-100-wiki5 achieved a score of 0.642 which is considerable better than 0.493 achieved by our word embeddings, repeating the results obtained for similarity and analogy.

5 Conclusions and Future Work

Our first research question addressed the feasibility of using text clustering techniques and concepts brought by semantic folding theory to produce word embeddings. Our work shows that our methodology based on document clustering and semantic folding theory achieves comparable results in intrinsic Natural Language Processing tasks with state of the art word embeddings. The results obtained in similarity and categorization word evaluation tasks are comparable with 20 state of the art word embeddings outperforming some of them. Our embeddings outperformed eight state of the art word embeddings in word similarity tasks, and two word embeddings in categorization tasks. Regarding analogy tasks our methodology does not achieve good results for the majority of the analogy categories, even though they outperform one state of the art word embedding. Interestingly, we can observe that good results were obtained for nouns categories and poor results for adjectives and verbs.

These results should be analyzed considering that our word embeddings were obtained using 2% or less training material than all other embeddings. To address this important factor we compared the results with Word2Vec embeddings trained with the same corpus used on our work (100 million words). Even though Word2Vec performed better, the results are considerable closer to the ones achieved by our word embeddings. Thus, we can extrapolate with a reasonable amount of confidence that the classification obtained by our approach would be better if we compared with the same 20 state of the art word embeddings trained with 100 million words.

With the above consideration in mind, our second research question addresses the size of the training material, as our proposal is to derive acceptable word vector representations with less training material. We conclude that our methodology is able to achieve comparable results with 20 state of the art word embeddings trained with considerable bigger volumes of training material.

Word vector representations resulting from our methodology are sparse binary vectors in contrast with state of the art dense vector representations with dimensionality varying from 25 to 300. Even though our word vector representations have 8464 dimensions, they have a much smaller memory size. Our word embeddings contains 100 000 word vectors and need 10 MB of computing memory, in contrast with hundreds of MB needed to store state of the art dense vector representations.

This work, while trying to provide an alternative methodology to produce word embeddings, suggests further research directions for future works.

One research possibility is the study of other types of clustering algorithms that could improve the performance obtained by K-Means or Self-Organizing Map. The success of our methodology

strongly depends on an effective way of clustering documents based on their semantic similarity. Agglomerative clustering and spectral clustering technics, or algorithms like Birch [83] or DBSCAN [22] have been applied to text documents and can be a valid substitute for the clustering algorithms used in this work.

Another research idea is applying other forms of dimensionality reduction techniques to the document-term matrix in order to improve the clustering process. A possibility is using T-SNE [50] as a method to project all documents to a 2-dimensional space, before applying a clustering algorithm. This approach has the advantage of enabling visual inspection of the documents in a 2 dimensional space.

Further enhancements can be applied to our methodology if documents are already labeled according to their category. In this case, human assignment of documents to a predefined set of cluster would improve clustering performance. Even if the number of human assigned labels is much less than the number of clusters applied in the methodology, it enables a top down agglomerative clustering, where the top clusters are the human assigned labels.

Another area of research that would extend our work is a methodology for deriving binary sparse document representations from our binary sparse word representations embeddings. A simple approach would be composing the document vector by adding all word vectors that compose the document and applying a transformation to the final vector for keeping the same level of sparsity on the resulting document vector.

The obtained document vector would allow to broaden the range of Natural Language Processing tasks that could be evaluated. Supervised text categorization with model fitting using different classifiers is an example of such kind of NLP tasks.

Finally, experimentation with larger sizes of training corpus would allow comparison with state of the art word embeddings based on the same premises and assess the performance of our model with larger training corpus.

6 Bibliography

[1] Agirre, E., Alfonseca, E., Hall, K., Kravalova, J., Pasca, M. and Soroa, A. (2009). A Study on Similarity and Relatedness Using Distributional and WordNet-based Approaches, In Proceedings of NAACL-HLT

[2] Almuhareb et al., Almuhareb, A. and Poesio, M. (2005). Concept learning and categorization from the web. In Proceedings of CogSci, pages 103–108, Stresa, 2005

[3] Ando, R. K. (2000). Latent semantic space: Iterative scaling improves precision of inter-document similarity measurement. In Proceedings of the 23rd Annual ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-2000), pp. 216–223.

[4] Baroni, M. and Lenci, A. (2011). How we BLESSed distributional semantic evaluation. In Proceedings of the GEMS 2011 workshop on Geometric Models of Natural Language Semantics, EMNLP 2011

[5] Battig, W. F. and Montague, W. E. (1968). Category norms for verbal items in 56 categories: A replication and extension of the Connecticut norms using University of Maryland and Illinois students(Tech. Rep.) University of Colorado, Boulder, CO

[6] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. Journal of machine learning research, 3(Feb), 1137–1155.

[7] Bengio, Y., Schwenk, H., Senécal, J.-S., Morin, F., and Gauvain, J.-L. (2006). Neural probabilistic language models. In Innovations in Machine Learning, pages 137–186. Springer

[8] Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet allocation. Journal of Machine Learning Research, 3(5), 993–1022.

[9] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching Word Vectors with Subword Information. arXiv preprint arXiv:1607.04606

[10] Bruni, E., Tran, N. K. and Baroni, M. (2013). Multimodal distributional semantics. Journal of Artificial Intelligence Research. In press; http://clic.cimec.unitn.it/ marco/publications/mmds-jair.pdf

[11] Bullinaria, J. A. and Levy, J. P. (2007). Extracting semantic representations from word co-occurrence statistics: A computational study. Behavior research methods, 39(3), 510–526

[12] Bullinaria, J. A. and Levy, J. P. (2012). Extracting semantic representations from word co-occurrence statistics: stop-lists, stemming, and svd. Behavior research methods, 44(3), 890–907.

[13] Buntine, W., & Jakulin, A. (2006). Discrete component analysis. In Subspace, Latent Structure and Feature Selection: Statistical and Optimization Perspectives Workshop at SLSFS 2005, pp. 1–33, Bohinj, Slovenia. Springer

[14] Church, K. W. and Hanks, P. (1989). Word association norms, mutual information, and lexicography. In ACL-89, Vancouver, B.C., pp. 76–83.

[15] Church, K. W. and Hanks, P. (1990). Word association norms, mutual information, and lexicography. Computational Linguistics, 16(1), 22–29.

[16] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: deep neural networks with multitask learning. In Proceedings of the 25th international conference on Machine learning, pages 160–167.

[17] Collobert, R., et al. (2011). Natural language processing (almost) from scratch. J. Mach. Learn. Res. 12, 2493–2537

[18] Dagan, I., Marcus, S., and Markovitch, S. (1993). Contextual word similarity and estimation from sparse data. In ACL-93, Columbus, Ohio, pp. 164–171.

[19] Deerwester, S. C., Dumais, S. T., Furnas, G. W., Harshman, R. A., Landauer, T. K., Lochbaum, K. E., and Streeter, L. (1988). Computer information retrieval using latent semantic structure: Us patent 4,839,853.

[20] Deerwester, S., Dumais, S., Furnas, G., Landauer, T., & Harshman, R. (1990). Indexing by latent semantic analysis. Journal of the Society for Information Science, 41(6), 391–407.

[21] Dumais, S.(1993). Lsi meets trec: A status report. In D. Harman (Ed.), Proceedings of the first Text REtrieval Conference, TREC1 (pp. 137–152).

[22] Ester, M., Kriegel, H., Sander, J. and Xiaowei, X. (1996). Simoudis, Evangelos; Han, Jiawei; Fayyad, Usama M., eds. A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226–231

[23] Finkelstein, L., Gabrilovich, E., Matias, Y., Rivlin, E., Solan, Z., Wolfman, G. and Ruppin, E. (2001). Placing search in context: The concept revisited. In Proceedings of the 10th international conference on World Wide Web, pages 406–414. ACM

[24] Firth, J.R. (1957). A synopsis of linguistic theory 1930-1955. Studies in Linguistic Analysis. Oxford: Philological Society: 1–32. Reprinted in F.R. Palmer, ed. (1968). Selected Papers of J.R. Firth 1952-1959. London: Longman

[25] Goldberg, Y., & Levy, O. (2014). word2vec Explained: Deriving Mikolov et al.'s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722 http://arxiv.org/abs/1402.3722v1

[26] Goodman, B. and Flaxman, S. (2016). European Union regulations on algorithmic decision-making and a "right to explanation"

[27] Harris, Z. S. (1954). Distributional structure. Word, 10(2-3):146–162.

[28] Hawkins, J. (2004). On Intelligence. Times Books

[29] Hawkins, J. et al, (2016). Biological and Machine Intelligence. Release 0.4. Accessed at https://numenta.com/biological-and-machine-intelligence/.

[30] Hill, F. and Korhonen, A. (2014). Simlex-999: Evaluating semantic models with (genuine) similarity estimation. arXiv preprint arXiv:1408.3456

[31] Hill, F., Cho, K., Jean, S., Devin, C., Bengio, Y. (2014). Embedding Word Similarity with Neural Machine Translation. Workshop Paper at ICLR 2015

[32] Hofmann, T. (1999). Probabilistic latent semantic indexing. In SIGIR-99, Berkeley, CA.

[33] Huang, A. (2008). Similarity measures for text document clustering. In New Zealand Computer Science Research Student Conference, pp. 49–56.

[34] Isa, D., Kallimani, V. P., Lee, L. H., (2008). Using Self Organizing Map for Clustering of Text Documents, Elsever, Expert System with Applications

[35] Jain, A. K. (2010). Data clustering: 50 years beyond K-means. Pattern Recognition Letters, vol. 31, no. 8, pp. 651 – 666, 2010

[36] Jastrzebski, S., Leśniak, D., and Czarnecki, W. M. (2017). How to evaluate word embeddings? on importance of data efficiency and simple supervised tasks. arXiv preprint arXiv:1702.02170.

[37] Jurafsky, D., Martin, J. H. (2016). An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. University of Colorado at Boulder.

[38] Jurgens, D. A. et al., Measuring degrees of relational similarity. In SEM 2012: The First Joint Conference on Lexical and Computational Semantics, 2012

[39] Kaski, S., Honkela, T., Lagus, K., Kohonen, T. (1998) WEBSOM - Self-organizing maps of document collections. Elsevier, Neurocomputing

[40] Kohonen, T. (1982). Self-Organized Formation of Topologically Correct Feature Maps. Biological Cybernetics

[41] Lee, D. D. and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. Nature, 401(6755), 788–791.

[42] Lin, D. (1998). Automatic retrieval and clustering of similar words. In COLING/ACL-98, Montreal, pp. 768–774.

[43] Lin, Y., Jiang, J., Lee, S. (2014). A Similarity Measure for Text Classification and Clustering, in Knowledge and Data Engineering, IEEE Transactions on , vol.26, no.7, pp.1575-1590.

[44] Liu, Y., Liu, Z., Chua, T. and Sun, M. (2015). Topical Word Embeddings. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, pages 2418–2424

[45] Lomonaco, V. (2015). Deep Learning for Computer Vision: A comparison between Convolutional Neural Networks and Hierarchical Temporal Memories on object recognition tasks. Master Degree in Computer Science, University of Bologna.

[46] Lowe, W. (2001). Towards a theory of semantic space. In Proceedings of the Twenty-first Annual Conference of the Cognitive Science Society, pp. 576–581.

[47] Luhn, Hans Peter (1957). A Statistical Approach to Mechanized Encoding and Searching of Literary Information. IBM Journal of Research and Development.

[48] Lund, K., Burgess, C., & Atchley, R. (1995). Semantic and associative priming in high-dimensional semantic space. In Proceedings of the 17th Annual Confer- ence of the Cognitive Science Society, CogSci'95 (pp. 660–665). Erlbaum

[49] Luong, M., Socher, R., Manning, C. (2013). Better Word Representations with Recursive Neural Networks for Morphology. CoNLL-2013

[50] Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. J. Mach. Learn. Research 9, 2579–2605.

[51] Manning, D., Raghavan, P. and Schütze, H. (2008). Introduction to Information Retrieval, Cambridge University Press.

[52] McCann, B., Bradbury, J., Xiong, C. et al. (2017). Learned in Translation: Contextualized Word Vectors, Advances in Neural Information Processing Systems 30 (NIPS 2017)

[53] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. ICLR Workshop Proceedings. arXiv:1301.3781

[54] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In Neural Information Processing Systems, pages 3111–3119.

[55] Mikolov, T., Yih, W.-T. a., and Zweig, G. (2013c). Linguistic regularities in continuous space word representations. In HLT-NAACL, pages 746–751.

[56] Niwa, Y. and Nitta, Y. (1994). Co-occurrence vectors from corpora vs. distance vectors from dictionaries. In ACL-94, pp. 304–309.

[57] Pantel, P., & Lin, D. (2002a). Discovering word senses from text. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 613–619, Edmonton, Canada

[58] Pantel, P., & Lin, D. (2002b). Document clustering with committees. In Proceedings of the 25th Annual International ACM SIGIR Conference, pp. 199–206

[59] Peffers, K., Tuunanen, T., Rothenberger, M., and Chatterjee, S. (2008). A Design Science Research Methodology for Information Systems Research. Journal of MIS (24:3), pp. 45-77

[60] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In Conference on Empirical Methods on Natural Language Processing, volume 14, pages 1532–1543.

[61] Potts, C. (2013). Distributional approaches to word meanings. Stanford course. Ling 236/Psych 236c: Representations of meaning.

[62] Radinsky, K., Agichtein, E., Gabrilovich, E. and Markovitch, S. (2011). A word at a time: Computing word relatedness using temporal semantic analysis. In Proceedings of the 20th international conference on World wide web, pages 337–346. ACM

[63] Rezaeinia, S., Ghodsi, A., and Rahmani, R. (2017). Improving the Accuracy of Pre-trained Word Embeddings for Sentiment Analysis. In: arXiv:1711.08609.

[64] Ritter, H. and Kohonen, T. (1989). Self-organizing semantic maps. Springer, Biological cybernetics

[65] Roussinov, D. G. and Chen, H. (1998). A scalable self-organizing map algorithm for textual classification: a neural network approach to thesaurus generation. CC-AI, The Journal for the Integrated Study of Artificial Intelligence, Cognitive Science and Applied Epistemology, 15(1–2):81–11

[66] Rubenstein, H., & Goodenough, J. B. (1965). Contextual Correlates of Synonymy. Communications of the ACM, 8(10), 627-633

[67] Rumelhart, D. E., Hintont, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323:9.

[68] Sahlgren, M. (2006). The Word-Space Model Using distributional analysis to represent syntagmatic and paradigmatic relations between words in high-dimensional vector spaces. Doctoral Dissertation, Department of Linguistics, Stockholm University.

[69] Sahlgren, M. (2008). The distributional hypothesis. Italian Journal of Linguistics, 33–54.

[70] Salle, A., Idiart, M., and Villavicencio, A. (2016). Matrix factorization using window sampling and negative sampling for improved word representations. arXiv preprint arXiv:1606.00819

[71] Salton, G., Wong, A., and Yang, C. (1975). A vector space model for automatic indexing. Communications of the ACM 18, 11, 613–620. Also reprinted in [Sparck Jones and Willett 1997], pp. 273–280

[72] Schnabel, T., Labutov, I., Mimno, D. and Joachims, T. (2015). Evaluation methods for unsupervised word embeddings. In Proc. of EMNLP

[73] Scholkopf, B., Smola, A. J., & Muller, K.-R. (1997). Kernel principal component analysis. In Proceedings of the International Conference on Artificial Neural Networks (ICANN- 1997), pp. 583–588, Berlin.

[74] Schütze, H. (1993). Word Space. Advances in Neural Information Processing Systems 5. pp. 895–902.

[75] Schutze, H.(1992). Dimensions of meaning. In Proceedings of the 1992 ACM/IEEE Conference on Supercomputing, Supercomputing'92 (pp. 787–796). IEEE Computer

[76] Shannon, C. (1948). A mathematical theory of communication. Bell System Technical Journal, 27, 379–423, 623–656

[77] Spärck Jones, K. (1972). A Statistical Interpretation of Term Specificity and Its Application in Retrieval. Journal of Documentation

[78] Sun, F., Guo, J., Lan, Y., Xu, J. and Cheng, X. (2015). Learning word representations by jointly modeling syntagmatic and paradigmatic relations. In Proceedings of the 53rd Annual Annual Meeting of the Association for Computational Linguistics.

[79] Turney, P. D. and Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. Journal of Artificial Intelligence Research.

[80] Vettigli, G. MiniSom: minimalistic and NumPy-based implementation of the Self Organizing Map. https://github.com/JustGlowing/minisom

[81] Webber, F. (2015). Semantic folding theory and its application in semantic fingerprinting. arXiv preprint arXiv:1511.08855.

[82] Young, T., Hazarika, D., Poria, S. and Cambria, E. (2017). Recent trends in deep learning based natural language processing, in arXiv preprint arXiv:1708.02709.

[83] Zhang, T., Ramakrishnan, R. and Livny, M. (1996). "BIRCH: an efficient data clustering method for very large databases". Proceedings of the 1996 ACM SIGMOD international conference on Management of data - SIGMOD '96. pp. 103–114

[84] Zipf, G.(1949). Human behavior and the principle of least-effort. Cambridge, MA: Addison-Wesley.