UCLouvain

FACULTE DES SCIENCES

Faculté des sciences

Word Embeddings using Canonical Correlation Analysis

Author :William RUAN
Supervisors : Eugen PIRCALABELU
Readers : Johan SEGERS, Donatien HAINAUT
Academic year 2021-2022
Master in Data Science, Statistics Orientation

Contents

Chapter 1

Introduction

Machine learning is an umbrella term for a collection of techniques and tools that assist computers in learning and adapting on their own. It allows computers to develop judgement-based decision-making, closer to human rational thinking, as opposed to traditional rule-based algorithms. Machine learning and AI are forming an increasingly essential part of modern society and our day-to-day lives, under the form of simple things such as recommendations on applications like Spotify or Netflix, and more complex, critical things such as pedestrian recognition and safety measures in cars, and automatic medical diagnostics. As the field of machine learning kept expanding, more research went into different ways to bring machines closer to humans. What if a machine could understand emotions, learn our understanding of words and their underlying meaning, and create texts and speech on their own? This is the main domain of study of Natural Language Processing or NLP, which combines computer science, linguistics, and machine learning in order to create programs that are able to process natural human language. The most notorious applications of NLP are virtual personal assistants, such as Siri by Apple, Alexa by Amazon, and Google Virtual Assistant by Google (Hoy, 2018). These applications are advanced enough to recognise speech, and Google's AI even demonstrated its capacity to produce speech convincing enough to pass for a human.

The totality of human communication is a complex system comprised of patterns (words, body language, intonation, etc.) specifically constructed to convey meaning between people, that can be created and perceived through sound, vision, and even touch (in the case of braille). Even when just addressing words, the system is still considered complex. For instance, many word combinations can convey the same meaning, and words can have several meanings depending on contextual information. This presents a stark contrast between the language spoken by machines, which is rule-based, numerical and unambiguous. The bridge between both systems can be built with the aid of word embeddings.

As a matter of fact, word embedding models are algorithms that aim to mimic the words semantic and syntactic meaning through real-valued vectors. This thesis is grounded on a rather recent field, and aims to shed some light on the mathematical background of a specific statistical approach to this algorithm. In particular, we will present the Canonical Correlation Analysis (Hotelling, 1936) technique to transform words from texts to vectors, a format that can be understood by machines, and other machine learning systems.

This work is split into six chapters. In Chapter 2, we will discuss why word embeddings are useful within the field of NLP, as well as introduce various terms applied in the field. Chapter 3 will focus on dissecting the mathematical background of the statistical method, called Canonical Correlation Analysis (CCA) used to achieve word embeddings. One of its formulations is a Spectral decomposition of matrices using Singular Value Decomposition (Lay et al., 2016), which is the one we are interested in and which has received a lot of attention in the literature.

At this point it will still be unclear how to apply the described statistical method, generally used with numerical data, on text data. Chapter 4 will clarify how we transform text into a number format for this specific purpose, and demonstrate how to apply this mathematical tool on the newly transformed data in order to achieve the word embeddings. As a numerical motivation based on the work of Dhillon et al. (2015), we will present their results obtained by using the CCA-based method. Despite the favorable results, we will also showcase a technical limitation due to the non-positive definiteness of one of the matrix needed in the computation.

The next chapter, Chapter 5 briefly introduces two state-of-the-art models based on the work of Mikolov et al. (2013a). They will be used to compare how the older statistical approach to this problem fares against the modern neural network models, favored by the machine learning community.

As we need data to train the models, Chapter 6 will address the methods used to obtain articles about the Russo-Ukrainian War from online news outlets. It will also develop two schemes to clean the obtained raw data. Consequently, the text data will be shown to follow a certain statistical distribution.

Chapter 7 will feature an application of the word embedding models using the articles as a training set. We will first optimize the CCA approach, then compare it with the neural networks models, and test the ability of these models to capture the semantic meaning behind the words. As a last quantitative test, we will showcase

a few relations between pairs of words in the context of the Russo-Ukrainian War using the model that showed the best results.

Finally, we will close off by summarizing the results, the encountered limitations, and by mentioning some possible avenues for future work in Chapter 8.

Chapter 2

Goals of Word Embedding and Definitions

2.1 Motivations for the Use of Word Embedding

In the field of Natural Language Processing, the supervised learner deployed in different applications needs labeled data in order to train and create predictions of its own. When working with machine learning it is typical to use sample of data that reaches over 100 million entries. However, the amount of labeled data such as texts with a part-of-speech tag associated with every word, and tweets with a sentiment annotation, are limited compared to the vast amount of unlabeled data available.

In this day and age, information is omnipresent in digital form, and the amount of unlabeled text is way more than a single human could handle. A method that could take advantage of such data and supplement other applications was what the researchers working in NLP needed. This is exactly what Word Embedding is, it feeds on any text data. It is an unsupervised learner that maps each word of a text to a real-valued vector, such that words sharing a similar meaning have vectors that are close in the vector space, these vectors are named word embeddings or word vectors.

For example, if a model learns the terms "money" and "cash" from a text that mentions these words extensively and in similar contexts, then it should create embeddings for these two words such that they are relatively close in the vector space, and far from words that are different in meaning. The Figure 2.1, shows 9 word vectors corresponding to 9 different words. The model producing the word vectors, its parameters, and the training data will be introduced later in this work. As for now, this simple plot showcase the ability of the word embedding algorithms

to understand the word semantic meaning and group them according to their similarity.

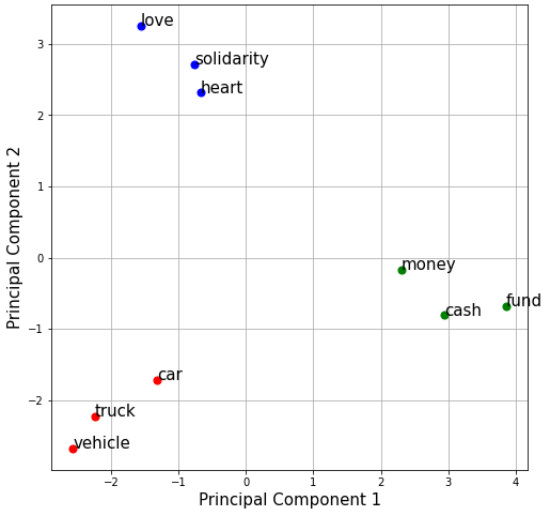


Figure 2.1: Plot of the PCA (Pearson, 1901) of 9 word vectors corresponding to 9 different words. The word vectors are from OSCCA (Dhillon et al., 2015) trained on the heavy corpus with the parameters $p = 50$, $q = 5$, $h = 2$ and $k = 200$.

Hence, the use of word embeddings has become increasingly popular in NLP, especially as a way to improve existing supervised learners dedicated towards a specific task such as sentiment classification, or even more complex systems such as Apple's virtual assitant, Siri, mentioned in the introduction. Moreover, word embeddings can also be used on their own, for example to infer intrinsically how the models perceive certain pair of words that one show interest in.

Before digging deeper into the process of these embedding models and their properties, let us get familiar with terminology within the NLP field.

2.2 Definitions

Let us define the vocabulary used in NLP and word embeddings. For the purpose of clarity, we will illustrate each definition with an example. Let the sentence "Red roses are red and red" be our not so poetic text example.

Definition 2.1. A word **type** is defined as a unique word in a text.

In this instance, the word **types** would be "Red", "roses", "are" and "and", each being unique. An important point to note, is that in NLP, word casing matters. For example, "Red" and "red" are considered as two different instances.

Definition 2.2. A word **token** is defined as each individual word in a text, regardless of how many times the same word occurs.

Thus, the **tokens** of the example sentence would be "Red", "roses", "are", "red", "and" and "red". Therefore, if a text is composed of 100 words, it has 100 **tokens**. But if some words are repeated, leaving say only 30 different unique words, it is said that the text has 30 word **types**.

Definition 2.3. The **vocabulary** refers to the set of unique words (word **types**) used in a document or in a set of documents. The vocabulary is also referred to as the word space.

In this manner, the set {Red, roses, are, and} is the **vocabulary** of our sentence.

Let us introduce some other vocabulary terms related this time to the cleaning of text data that we will use later in Chapter 6, which will handle the topic of webscraped data on the Daily Mail news articles.

Definition 2.4. In the context of NLP, a **corpus** indicates a collection of texts organized into a dataset. A corpus can be in the form of a novel, mails, newspapers or even tweets, etc.

Since our example contains only one sentence, we can consider the collection of texts, namely the **corpus**, as the sentence itself.

Definition 2.5. Stop words are words that need to be filtered out since they either do not imply any context information or do not provide useful 'information overall. These words are usually the most common words in a given language, such as prepositions or conjunctions, just to name a few. For instance, in English, "the" and "a" are words that do not contribute to the understanding of the content of a sentence nor carry any meaning in themselves. The set of stop words is arbitrary to one's choice and depends on the kind of task we are trying to achieve.

In our small example's case, the word "and" does not provide us useful information in understanding what the sentence is about. Thus, we can safely label "and" as a **stop word** and remove it. It yields the new sentence "Red roses are red red".

Definition 2.6. Another important process of our pre-processing of the corpus, is **lemmatization**. According to Plisson et al. (2004), it is the process of transforming the different forms of a word to one single normalized form, for example, reducing words such as "talks", "talking", or "talked" to the lemma "talk". Usually this process is achieved by removing or adding a suffix to the word endings.

By applying **lemmatization** to the example without stop words "Red roses are red red", it produces the new lemmatized sentence "Red rose be red red". "Roses"

and "are" have been reduced to their normalized forms, respectively "Rose" and "be".

Lastly, we will introduce Zipf's law which is used to characterize the distribution of words in any corpus:

Definition 2.7. As explained by Piantadosi (2014) and Yang (2013), Zipf's law describes words occurrences. It states that word occurrences follow a systematic frequency distribution, which is of the form:

$$f(r) \approx \frac{1}{r\sum_{i=1}^{n} \frac{1}{i}}, \tag{2.1}$$

with r the frequency rank of the word types and n the total number of tokens. The rank 1 word being defined as the most frequently used word, the rank 2 word being the second most used word, and so on. This would mean that the most frequent word ($r = 1$) would have a frequency proportional to $\frac{1}{\sum_{i=1}^{n} \frac{1}{i}}$, the second most frequent word would be proportional to $\frac{1}{2\sum_{i=1}^{n} \frac{1}{i}}$, and so on.

Thus, we have an empirical law between the rank of the word which is inversely proportional to its frequency. It can be easily observed for any corpus by plotting on a logarithmic graph the rank of the words against their frequencies.

Chapter 3

Canonical Correlation Analysis Review

Canonical Correlation Analysis (CCA) was introduced by Hotelling (1936), it is analog to PCA (Pearson, 1901), which produces linear combination of feature vectors in a data set, also known as Principal Components, in order to maximize the variance explained by the Principal components themselves. As for CCA, its goal is to maximize the correlation between linear transformations of two datasets. This process is achieved by taking linear combinations of the feature vectors inside one data set and finding another combination in the other data set such that their correlation is maximized. Both methods accomplish dimensionality reduction as well.

In this section, we will review CCA. To be more specific, we will display its derivation, introduce numerical approaches to solving it, namely, deterministic Singular Value Decomposition (SVD) or Randomized SVD. We will also discuss the parameters of these decomposition techniques and solve CCA using randomized SVD techniques.

3.1 Derivation of Canonical Correlation Analysis (CCA)

CCA is a two-view multivariate statistical method. Uurtio et al. (2017) claims that the variables of an observation can be partitioned into two sets that can be seen as the two views of the data.

First, let the views a and b be matrices $\mathbf{X}_a$ of size $n \times m_1$ and $\mathbf{X}_b$ of size $n \times m_2$. Then, define the vectors $x_i^a \in \mathbb{R}^{m_1}$ and $x_i^b \in \mathbb{R}^{m_2}$ for $i = 1, ..., n$ as the sets of empirical observations that are stored on row i of $\mathbf{X}_a$ and $\mathbf{X}_b$ respectively. Thereafter, define the vectors $a_j \in \mathbb{R}^n$ for $j = 1, ..., m_1$ as the variables of the n

observations that are stored on column j of $\mathbf{X}_a$, and define the vectors $b_k \in \mathbb{R}^n$ for $k = 1, ..., m_2$ as the variables of the n observations that are stored on column k of $\mathbf{X}_b$. In order to simplify notations, the column vectors, or variables, are assumed to be standardised to a mean of zero and a unit variance. The purpose of CCA is to identify the linear relationships between the matrices $\mathbf{X}_a$ and $\mathbf{X}_b$.

Definition 3.1. According to Hotelling (1936) and Bach and Jordan (2005), let $\phi_a \in \mathbb{R}^{m_1}$ and $\phi_b \in \mathbb{R}^{m_2}$ be linear transformation vectors. By performing projections on the datasets $\mathbf{X}_a$ and $\mathbf{X}_b$ using those vectors, we obtain new vectors called **canonical variables**,

$$\mathbf{X}_a\phi_a = z_a \in \mathbb{R}^n \qquad\qquad \mathbf{X}_b\phi_b = z_b \in \mathbb{R}^n.$$

These canonical variables are just images of the data matrices $\mathbf{X}_a$ and $\mathbf{X}_b$ into the space $\mathbb{R}^n$ obtained through projections. Thus, z_a and z_b are n-dimensional. One point to note is that this formulation of CCA by Hotelling (1936), Bach and Jordan (2005), and Dhillon et al. (2015) is not the only one. In the work of Uurtio et al. (2017), they describe the data matrices as the projections and the vectors $\phi_a \in \mathbb{R}^{m_1}$ and $\phi_b \in \mathbb{R}^{m_2}$ as position vectors on which we apply the projection. We prefer the first formulation as it is easier to interpret the results coming later in this paper.

Definition 3.2. Let x and y be two vectors of same dimension, then $\langle x, y\rangle$ is the Euclidian scalar product between x and y. Thus, we can define the Euclidean norm of x as, $\|x\| = \sqrt{\langle x, x\rangle}$.

Since the columns of the data matrices are assumed to be standardised to have an empirical mean of zero and a unit variance, we can define the different empirical covariance matrices using $\mathbf{X}_a$ and $\mathbf{X}_b$ as such: $\mathbf{\Sigma}_{ab} = \frac{1}{n}\mathbf{X}_a^t\mathbf{X}_b$, $\mathbf{\Sigma}_{aa} = \frac{1}{n}\mathbf{X}_a^t\mathbf{X}_a$, and $\mathbf{\Sigma}_{bb} = \frac{1}{n}\mathbf{X}_b^t\mathbf{X}_\mathbf{b}$. Now let us define the goal of CCA,

Definition 3.3. CCA's objective is to learn the linear transformations ϕ_a and ϕ_b, such that their pair of canonical variables z_a and z_b are maximally correlated. To put it another way, the aim of CCA is to find the solution ϕ_a^1 and ϕ_b^1 to this objective function,

$$\phi_a^1, \phi_b^1 = \arg\max_{\phi_a, \phi_b} \frac{\phi_a^t\mathbf{\Sigma}_{ab}\phi_b}{\sqrt{\phi_a^t\mathbf{\Sigma}_{aa}\phi_a}\sqrt{\phi_b^t\mathbf{\Sigma}_{bb}\phi_b}}. \tag{3.1}$$

Furthermore, we notice that

$$\begin{aligned}\phi_a^t\mathbf{\Sigma}_{ab}\phi_b &= \phi_a^t(\mathbf{X}_a^t\mathbf{X}_b)\phi_b \\ &= (\mathbf{X}_a\phi_a)^t\mathbf{X}_b\phi_b \\ &= z_a^t z_b \\ &= \langle z_a, z_b\rangle.\end{aligned}$$

Using the same logic, it can also be easily shown that $\phi_a^t \mathbf{\Sigma}_{aa} \phi_a = \langle z_a, z_a \rangle$ and $\phi_b^t \mathbf{\Sigma}_{bb} \phi_b = \langle z_b, z_b \rangle$. Thus, we can rewrite the objective Equation (3.1), as follows:

Definition 3.4. The goal of CCA is to find the minimum angle θ between the two canonical variables z_a and z_b. This is equivalent to finding,

$$\cos(\theta) = \max_{z_a, z_b} \frac{\langle z_a, z_b \rangle}{\|z_a\| \, \|z_b\|} \tag{3.2}$$

$$= \rho \tag{3.3}$$

also known as the **canonical correlation**.

There are 2 constraints of CCA on the mappings (Uurtio et al., 2017). First, the images z_a and z_b must be unit norm vectors, such that Equation (3.2) is scale invariant. Secondly the enclosing angle, θ, must belong to the interval $[0, \frac{\pi}{2}]$. Thus, the objective function is further simplified to:

$$\cos(\theta) = \max_{z_a, z_b} \langle z_a, z_b \rangle$$
$$s.t \|z_a\| = \|z_b\| = 1.$$

Therefore, CCA can be summarized as finding the projections ϕ_a and ϕ_b on $\mathbf{X_a}$ and $\mathbf{X_b}$, such that their images z_a and z_b on a $\mathbb{R}^n$ unit ball have a cosine of θ that is maximized. In other words, we have to minimize the angle θ between z_a and z_b.

The images z_a and z_b that result in the smallest angle, θ_1, determine the first canonical correlation which equals $\cos(\theta_1)$ (Uurtio et al., 2017). Thus, the smallest angle θ_1 is obtained through

$$z_a^1, z_b^1 = \arg\max_{z_a, z_b} \langle z_a, z_b \rangle,$$
$$s.t \|z_a\| = \|z_b\| = 1.$$

The solutions z_a^1 and z_b^1 to obtain the maximum are called the first pair of canonical variables.

Throughout the breakdown of CCA shown so far, the mentionned projections amount to two, the m_1-dimensional and m_2-dimensional projections to an n-dimensional space. But there are multiple more. As explained by Uurtio et al. (2017), the next pair of images z_a^2 and z_b^2, that has the second smallest enclosing angle θ_2, is constrained to be orthogonal to the first pair of canonical variables. This

process is continued in this way until no more pairs are found. Therefore, to be more concise, the angles $\theta_r \in [0, \frac{\pi}{2}]$ are recursively defined for $r = 1, ..., \min\{m_1, m_2\}$,

$$cos(\theta_r) = \max_{z_a, z_b} \langle z_a^r, z_b^r \rangle, \tag{3.4}$$

$$s.t \|z_a^r\| = \|z_b^r\| = 1, \tag{3.5}$$

subject to the additional constraints of

$$\langle z_a^i, z_a^j \rangle = \langle z_b^i, z_b^j \rangle = 0, \tag{3.6}$$

for $i, j = 1, ..., \min\{m_1, m_2\}$ for $i \neq j$.

Thus the number of canonical correlations, r, corresponds to the dimensionality of the CCA, which is the number of patterns that can be extracted from the pair of data matrices through CCA (Uurtio et al., 2017). With this, we can define the goal of CCA as finding r pair of linear transformations ϕ_a and ϕ_b, that satisfy Equation (3.4) and both constraints (3.5) and (3.6).

Furthermore, we can build and define the pair of linear transformation matrices $\mathbf{\Phi}_a \in \mathbb{R}^{m_1 \times r}$ and $\mathbf{\Phi}_b \in \mathbb{R}^{m_2 \times r}$ that maximizes the r correlations between the canonical variables. We form those matrices by respectively concatenating the r pairs of vectors $\phi_a \in \mathbb{R}^{m_1}$ and $\phi_b \in \mathbb{R}^{m_2}$ into $\mathbf{\Phi}_a$ and $\mathbf{\Phi}_b$. As a reminder, the column dimensions of the matrices $\mathbf{\Phi}_a$ and $\mathbf{\Phi}_b$ are based upon,

$$r \leq \min\{m_1, m_2\}, \tag{3.7}$$

with m_1 and m_2 the number of columns in the data matrices $\mathbf{X}_a \in \mathbb{R}^{n \times m_1}$ and $\mathbf{X}_b \in \mathbb{R}^{n \times m_2}$.

3.2 Singular Value Decomposition (SVD)

In this section, we will introduce briefly the deterministic SVD and the randomized SVD. The aim of this part is to explain this mathematical tool used to solve efficiently the CCA problem.

In theory, deterministic SVD is a correct analytical way to decompose the CCA equations. However, from the numerical point of view, for large matrices, any deterministic decomposition is bound to fail on a mid-range PC due to the lack of memory space.

3.2.1 Deterministic SVD

In order to clarify the Theorem that we are presenting below, let us first define an orthogonal matrix.

Definition 3.5. An orthogonal matrix $\mathbf{Q}$ is a real valued square matrix that satisfies, $\mathbf{Q}^t\mathbf{Q} = \mathbf{Q}\mathbf{Q}^t = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix sharing the same dimensions as $\mathbf{Q}$.

The deterministic Singular Value Decomposition will now be briefly introduced through the lens of Lay et al. (2016),

Theorem 3.6. *Let $\mathbf{A}$ be an $n \times m$ matrix with rank r. Then there exists a $n \times m$ matrix $\mathbf{\Lambda}$ of the form*

$$\mathbf{\Lambda} = \begin{bmatrix} \mathbf{D} & 0 \\ 0 & 0 \end{bmatrix} \tag{3.8}$$

where $\mathbf{D}$ is an $r \times r$ diagonal matrix for some $r \leq min\{n, m\}$, for which the diagonal entries in $\mathbf{D}$ are the first r singular values of $\mathbf{A}$, $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r \geq 0$, and there exist an $n \times n$ orthogonal matrix $\mathbf{U}$ and an $m \times m$ orthogonal matrix $\mathbf{V}$ such that, any factorization of the form

$$\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{V^t} \tag{3.9}$$

*with $\mathbf{U}$ and $\mathbf{V}$ orthogonal matrices, $\mathbf{\Lambda}$ as in Equation (3.8), and positive entries in $\mathbf{D}$, is called a **singular value decomposition** (or SVD) of $\mathbf{A}$. The columns of $\mathbf{U}$ in such a decomposition are called **left singular vectors** of $\mathbf{A}$, and the columns of $\mathbf{V}$ are called **right singular vectors** of $\mathbf{A}$.*

In practice, standard algorithms, can be applied in order to compute the full deterministic SVD. But for large matrices, it becomes very expensive in terms of memory and computation time. Instead of computing the full deterministic SVD, there exist another computation that is slightly more efficient. It is called the economy-sized SVD (Bai et al., 2000). Let $r \leq \min\{n, m\}$ be the rank of matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$, the economy-sized SVD of $\mathbf{A}$ is given by

$$\mathbf{A} = \mathbf{U}_r\mathbf{\Lambda}_r\mathbf{V}_r^t,$$

the matrices $\mathbf{U}_r \in \mathbb{R}^{n \times r}$ and $\mathbf{V}_r \in \mathbb{R}^{r \times m}$ contain only the first r columns of $\mathbf{U}$ and $\mathbf{V}$, and $\mathbf{\Lambda}_r \in \mathbb{R}^{r \times r}$ contains only the first r singular values from $\mathbf{\Lambda}$, which are non-zeros.

3.2.2 Randomized SVD

First, in order to provide clarification, let us define the 2-norm of a matrix as a matrix norm induced by vector norms.

Definition 3.7. The **2-norm** (Meyer, 2000) for matrices is defined by

$$\begin{aligned} \|\mathbf{A}\|_2 &= \max_{\|x\|=1} \|\mathbf{A}x\| \\ &= \sqrt{\lambda_{max}} \\ &= \sigma_{max} \end{aligned}$$

with $\mathbf{A} \in \mathbb{R}^{n \times m}$, $x \in \mathbb{R}^m$, λ_{max} is the largest eigenvalue of the matrix $\mathbf{A}^t\mathbf{A}$, σ_{max} is the largest singular value of $\mathbf{A}$ and $\|\cdot\|$ the Euclidean norm of vectors from Definition 3.2. Therefore, the 2-norm of a matrix $\mathbf{A}$ is the square root of the maximum eigenvalue of $\mathbf{A}^t\mathbf{A}$ (i.e, the largest singular value of $\mathbf{A}$).

As mentioned previously, a full deterministic SVD decomposition is often not possible to compute numerically, especially for large matrices, where the algorithm has to compute every non-zero singular value. This is also the case with the more efficient economy-sized SVD. One solution to this problem is the truncated SVD (TSVD) (Hansen, 1987). It provides an approximate decomposition of the original matrix. The idea behind this approximation is as follows: Let $\mathbf{A} \in \mathbb{R}^{n \times m}$, we wish that the 2-norm of the difference between the original matrix $\mathbf{A}$ and the approximated matrix obtained through decomposition, to be small. This is achieved by removing the components of the approximated matrix corresponding to the smallest singular values. These contributions to the 2-norm are the least important.

Definition 3.8. The **TSVD** of a matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ of rank r, is defined as the rank k matrix $\mathbf{A}_k \in \mathbb{R}^{n \times m}$, where $k \ll r$, and

$$\mathbf{A}_k \equiv \mathbf{U}_k \mathbf{\Lambda}_k \mathbf{V}_k^t,$$

with $\mathbf{\Lambda}_k = \text{diag}(\sigma_1, ..., \sigma_k) \in \mathbb{R}^{k \times k}$. Halko et al. (2011) call the dimension k, the **numerical rank** or the **target rank** of the matrix $\mathbf{A}$. The matrix of singular values $\mathbf{\Lambda}_k \in \mathbb{R}^{k \times k}$ from the TSVD correspond to the matrix $\mathbf{\Lambda}$ in the deterministic SVD from Equation (3.8) with the smallest $(\min\{n, m\} - k)$ singular values replaced by zeros (Hansen, 1987). Furthermore, $\mathbf{U}_k \in \mathbb{R}^{n \times k}$ and $\mathbf{V}_k^t \in \mathbb{R}^{k \times m}$ are the k columns vectors of $\mathbf{U}$ and the k row vectors of $\mathbf{V}^t$, such that they correspond to the k largest singular values of $\mathbf{\Lambda}_k$.

This truncated version of SVD let us express a low-rank approximation of the matrix $\mathbf{A}$. According to Halko et al. (2011), many low-rank matrix approximation techniques exist and most of them are expensive to compute. Thus, they suggest a method to perform a low-rank approximation of the full SVD, called randomized SVD (RSVD). In order to better understand RSVD, a vector basis and the range of a matrix should be defined according to Vitale (2017):

Definition 3.9. Let E be a subspace of $\mathbb{R}^n$ and let the vectors $e_1, ..., e_t \in \mathbb{R}^n$. The vectors $e_1, ..., e_t$ form a **basis** of E, if and only if the following conditions are satisfied:

- $e_1, ..., e_t \in E$,
- $e_1, ..., e_t$ are linearly independent,
- $\forall e \in E, \exists\ \alpha_1, ..., \alpha_t \in \mathbb{R}$ such that $e = \alpha_1 e_1 + ... + \alpha_t e_t$, can be uniquely written (spanning property).

Definition 3.10. The **range** of a matrix is another term for the column space of a matrix. Let $\mathbf{A} \in \mathbb{R}^{n \times m}$. Thus, every column of $\mathbf{A}$ is a vector from $\mathbb{R}^n$. Therefore, let the column space of $\mathbf{A} \in \mathbb{R}^{n \times m}$ be defined as,

$$\text{Col}(\mathbf{A}) = \text{subspace of } \mathbb{R}^n \text{ spanned by the columns of matrix } \mathbf{A}.$$

Let's present Halko et al. (2011)'s solution to the low-rank matrix approximation. Let $\mathbf{A} \in \mathbb{R}^{n \times m}$, the goal is to perform a low-rank approximation of the matrix $\mathbf{A}$. In order to do so, Halko et al. (2011) divide the process into two stages:

Stage A Compute an approximate vector basis for the range of $\mathbf{A}$. This is equivalent to constructing a low-dimensional matrix $\mathbf{Q} \in \mathbb{R}^{n \times l}$, with l orthogonal columns, that span the columns of $\mathbf{A}$. Formally,

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^t\mathbf{A}. \tag{3.10}$$

In order to compute $\mathbf{Q}$ efficiently, Halko et al. (2011) use randomized methods that they call "Randomized Range Finders".

Stage B Having computed the matrix $\mathbf{Q}$, which is much smaller than the original input matrix $\mathbf{A}$, it is then used to deterministically compute the SVD factorization.

Randomized Range Finders

Let us rewrite Equation (3.10) to fit the objective of the Randomized Range Finders of Stage A, which is to find a matrix $\mathbf{Q} \in \mathbb{R}^{n \times l}$ with a smaller number of columns than $\mathbf{A} \in \mathbb{R}^{n \times m}$ and a tolerance error $\epsilon > 0$, such that,

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^t\mathbf{A}\|_2 \leq \epsilon. \tag{3.11}$$

The idea is to first draw l Gaussian random vectors $\omega_i \in \mathbb{R}^m$, for $i = 1, 2, ..., l$ from a multivariate Gaussian distribution $N(0, \mathbf{I}_{m \times m})$, with $\mathbf{I}_{m \times m}$ the identity matrix of size $m \times m$. In particular, these vectors form a linearly independent set and any linear combination of these vectors is different from the null space of $\mathbf{A}$ (Halko et al., 2011). We then project them using the linear map $\mathbf{A}$:

$$y_i = \mathbf{A}\omega_i, i = 1, 2, ..., l.$$

Thus, the set $\{y_i \in \mathbb{R}^n : i = 1, 2, ..., l\}$ of sample vectors is also linearly independent and spans the range of $\mathbf{A}$. Intuitively this is equivalent to randomly sampling the range of $\mathbf{A}$. The last step would be to find an orthonormal basis of the sample vectors y_i, forming the $\mathbf{Q}$ we were looking for. This is done through QR factorization of $\mathbf{Y} \in \mathbb{R}^{n \times l}$, the matrix formed using the vectors y_i. Indeed, this factorization creates the orthonormal matrix $\mathbf{Q}$, which we were looking for, and an upper triangular matrix $\mathbf{R}$.

One thing that we did not mention so far is the definition of the parameter l. It is defined as $l = k + p$, with k the numerical rank of matrix $\mathbf{A}$, and p the oversampling parameter. According to Halko et al. (2011), the reason for adding the parameter l to k is due to numerical results. By adding a few more samples than the numerical rank k of $\mathbf{A}$, the matrix $\mathbf{Q}$ yields a smaller 2-norm in Equation (3.11). Furthermore, the authors claim that choosing $p > k$ will not yield significantly better results. A numerical justification to the use of the oversampling parameter p will be shown in Example 3.12.

In order to illustrate the scheme described above, Algorithm 1 is the implementation of the basic Randomized Range Finders from Halko et al. (2011).

Algorithm 1: Randomized Range Finders (RRF)

Input: A matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ and a sampling parameter $l = k + p$, with k the numerical rank of $\mathbf{A}$, and p the oversampling parameter.

Output: An orthonormal matrix $\mathbf{Q}$ of size $n \times l$ that approximates the range of $\mathbf{A}$.

1 Draw the vectors $\omega_i \in \mathbb{R}^m$ for $i = 1, 2, ..., l$ from a multivariate Gaussian distribution $N(0, \mathbf{I}_{m \times m})$. The l vectors ω_i are concatenated to form the column vectors of the matrix $\mathbf{\Omega} \in \mathbb{R}^{m \times l}$.

2 Compute the $n \times l$ matrix $\mathbf{Y} = \mathbf{A\Omega}$.

3 Generate an $n \times l$ matrix $\mathbf{Q}$ whose columns form an orthonormal basis for the range of $\mathbf{Y}$. This is done through thin QR factorization, $[\mathbf{Q}, \mathbf{R}] = \mathrm{QR}(\mathbf{Y})$.

The authors claim that for matrices whose singular values show some decay, the basic RRF works well. However, they also assert that in the case of an input matrix with slowly decaying singular values or flat singular values or simply a very large input matrix, this technique may produce poor results.

Randomized Power Iteration

Therefore, we will showcase a second implementation of the RRF fixing these issues and that produces better results. Algorithm 2 introduces the Randomized Power Iteration, described by Halko et al. (2011).

Algorithm 2: Randomized Power Iteration (RPI)

Input: A matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$, a sampling parameter $l = k + p$, with k the numerical rank of $\mathbf{A}$, p the oversampling parameter, and q the power iteration parameter.

Output: An orthonormal matrix $\mathbf{Q}$ of size $n \times l$ that approximates the range of $\mathbf{A}$.

1 Draw the vectors $\omega_i \in \mathbb{R}^m$ for $i = 1, 2, ..., l$ from a multivariate Gaussian distribution $N(0, \mathbf{I}_{m \times m})$. The l vectors ω_i are concatenated to form the column vectors of the matrix $\mathbf{\Omega} \in \mathbb{R}^{m \times l}$.

2 Compute the $n \times l$ matrix $\mathbf{Y}_0 = \mathbf{A\Omega}$ and its QR factorization, $[\mathbf{Q}_0, \mathbf{R}_0] = \mathrm{QR}(\mathbf{Y}_0)$.

3 **for** $j \leftarrow 1$ **to** q **do**

4 Compute $\tilde{\mathbf{Y}}_j = \mathbf{A}^t \mathbf{Q}_{j-1}$ and its QR factorization, $[\tilde{\mathbf{Q}}_j, \tilde{\mathbf{R}}_j] = \mathrm{QR}(\tilde{\mathbf{Y}}_j)$.

5 Compute $\mathbf{Y}_j = \mathbf{A}\tilde{\mathbf{Q}}_j$ and its QR factorization, $[\mathbf{Q}_j, \mathbf{R}_j] = \mathrm{QR}(\mathbf{Y}_j)$.

6 $\mathbf{Q} = \mathbf{Q}_q$, the matrix of size $n \times l$ whose columns form an orthonormal basis for the range of $\mathbf{Y}$.

The idea behind this process is that by applying the power iteration, the singular values decay quicker and in return will improve the ability of the matrix $\mathbf{Q}$ in covering the range of $\mathbf{Y}$ (Halko et al., 2011). A thing worth noting is that the RPI algorithm with a power iteration parameter $q = 0$ would just be the RRF.

Solution to the Randomized SVD

After obtaining the matrix $\mathbf{Q}$ of Stage A with either the simple RRF or the RPI, we can finally compute an approximation of the SVD of our matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$. Let us develop the Equation (3.10), $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^t\mathbf{A}$:

1 Form $\mathbf{B} = \mathbf{Q}^t\mathbf{A}$, using $\mathbf{Q}$ from Stage A.

2 Compute the economy-sized SVD of $\mathbf{B}$, i.e. $\mathbf{B} = \tilde{\mathbf{U}}_r\mathbf{\Lambda}_r\mathbf{V}_r^t$, with r the rank of matrix $\mathbf{A}$.

3 Set $\mathbf{U}_r = \mathbf{Q}\tilde{\mathbf{U}}_r$.

4 Return $\mathbf{U}_k, \mathbf{\Lambda}_k, \mathbf{V}_k^t$, where $k \ll r$ is the target rank.

Thus, one obtains $\mathbf{A} \approx \mathbf{Q}(\mathbf{Q}^t\mathbf{A}) = \mathbf{Q}(\tilde{\mathbf{U}}_r\mathbf{\Lambda}_r\mathbf{V}_r^t)$. By setting $\mathbf{U}_r = \mathbf{Q}_r\tilde{\mathbf{U}}_r$, we produce a low-rank approximation by keeping only the first k singular values from $\mathbf{\Lambda}_r$ and the corresponding singular vectors: $\mathbf{A} \approx \mathbf{U}_k\mathbf{\Lambda}_k\mathbf{V}_k^t$.

In order to demonstrate the ability and precision of these two methods in performing SVD, we will apply the deterministic SVD, RRF and RPI on a sparse matrix $\mathbf{G}$, with values randomly drawn from an Uniform distribution between 0 and 1, and compare their results.
One way to measure the quality of each method, is to measure their accuracy in recreating the original matrix. The higher the accuracy of the recreated matrix compared to the original matrix, the less information would be lost after performing SVD.
Therefore, we will, after performing SVD on $\mathbf{G}$, reconstruct it by multiplying its singular vectors and singular value matrices. In other words, perform one of the approximated SVD method on $\mathbf{G}$, it yields the singular vectors $\mathbf{U}_G$, $\mathbf{V}_G$ and the singular value matrix $\mathbf{\Lambda_G}$. Then, we construct $\mathbf{U}_G\mathbf{\Lambda}_G\mathbf{V}_G^t$ and compare the product based on each method to the original matrix $\mathbf{G}$. Let $\mathbf{G}_r$ be the matrix $\mathbf{G}$ rebuilt using the RRF and $\mathbf{G}_p$ be the matrix $\mathbf{G}$ rebuilt using the RPI.

To be able to measure the accuracy of the rebuilt matrices compared to the original, we need a metric. Let us define the Frobenius norm of matrices. Similar

to the vector norm (Definition 3.2) that is used to measure lengths of vectors, the Frobenius norm measure "sizes" of matrices.

Definition 3.11. The Frobenius norm (Ford, 2014) is defined as

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}^2},$$

for $\mathbf{A} \in \mathbb{R}^{n\times m}$.

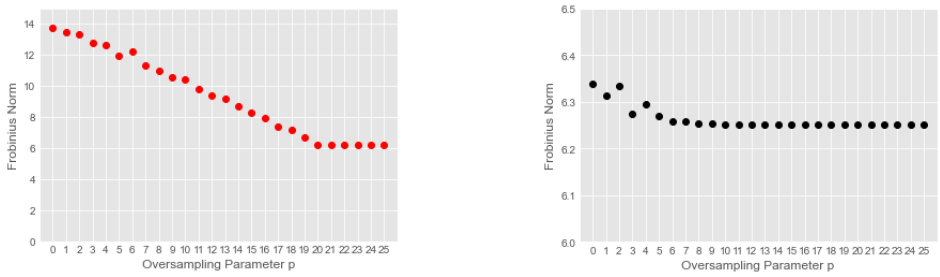
By calculating the Frobenius norm of the difference between the original matrix $\mathbf{G}$ and the reconstructed matrices, we can measure how "similar" they are. There are multiple other matrix norms and other methods to measure similarity between matrices, but in this case we will stick to the Frobenius norm due to the simplicity of its interpretation. To summarize, the approximated matrix yielding the smallest Frobenius norm would be the most similar to the original matrix.

Example 3.12. Let $\mathbf{G} \in \mathbb{R}^{300\times 200}$ be a sparse matrix (30% of its values are different from zero), the non-zero entries are drawn from an uniform distribution between 0 and 1. Following Theorem 3.6, we know that the rank of the matrix $\mathbf{G}$ is $r \leq \min\{300, 200\}$. Let the target rank $k \leq r$, be set to 180 and let p be the oversampling parameter. Adding those parameters together make up the sampling parameter $l = k + p$.

We apply the deterministic SVD, the RRF and the RPI on the matrix $\mathbf{G}$. Each of these methods yield singular vectors and singular value matrices for $\mathbf{G}$. Afterwards, we proceed to multiply them in order to create the matrices $\mathbf{G_r}$ and $\mathbf{G_p}$.

For the RRF and the RPI methods the oversampling parameter p was varied from 0 to 25, and in the case of the RPI, we further vary the power iteration parameter q from 0 to 20 too. The purpose would be to see the impact those parameters have on the Frobenius norm of the difference between the original matrix $\mathbf{G}$ and the reconstructed matrices, namely $\mathbf{G}_r$ and $\mathbf{G}_p$.

The Figure 3.1 shows two plots of the Frobinius norm between $\mathbf{G}$ and the reconstructed matrices $\mathbf{G}_r$ and $\mathbf{G}_p$ as a function of the oversampling parameter p.



(a) (b)

Figure 3.1: (a) Plot of the Frobenius norm between the original matrix $\mathbf{G}$ and $\mathbf{G}_r$ (obtained through Randomized Range Finders) as a function of the oversampling parameter p. (b) Plot of the Frobenius norm between the original matrix $\mathbf{G}$ and $\mathbf{G}_p$ (obtained through Randomized Power Iteration) as a function of the oversampling parameter p. Here, the power iteration parameter q was set to 5.

Clearly, we can observe that the oversampling parameter improves the results for both the RRF and the RPI, leaving us with a smaller Frobenius norm. In the case of the RRF, the behavior of the norm is almost linear as a function of p until it reaches a threshold. Here, it would be $p = 20$, further increasing the oversampling parameter does not improve the result anymore. On the other hand, for the RPI, the stagnation happens at a smaller threshold, around $p = 6$ the norm will not get any lower.

Now, let us take a look at the effect of the power iteration parameter q:

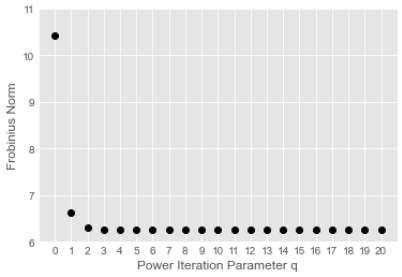


Figure 3.2: Plot of the Frobenius norm between the original matrix $\mathbf{G}$ and $\mathbf{G}_p$ (obtained through Randomized Power Iteration) as a function of the power iteration parameter q. Here, the oversampling parameter p was set to 10. The results are averaged over 5 different random seeds.

A similar conclusion can be reached for the behavior of the power iteration parameter: increasing it yields a $\mathbf{G}_p$ matrix more similar to $\mathbf{G}$. Moreover, unlike the oversampling parameter, a very small value of q is enough. As seen from the graph, a value larger than $q = 3$ does not reduce the norm any further.

Thus, from this small experiment, we can conclude that for the $\mathbf{G} \in \mathbb{R}^{300\times200}$ an oversampling parameter p of 20 is ideal for RRF, and 5 to 10 is enough for RPI. As for the power iteration, $q = 5$ is also enough to get the best results. Additionally, even with larger matrices and a threshold of $q = 5$, the results are comparable. In contrast, it is necessary to raise the value of the oversampling parameter; typically, $p \leq k$ is sufficient for satisfactory results.

More importantly, the Frobenius norm of the differences between the original matrix and the created matrices is still significant overall. However, in order to use the word embedding algorithms utilizing CCA, which will be introduced in the next chapter, from a computational perspective we need a low-rank approximation of the SVD decomposition.

In fact, employing the singular vectors to perform matrix multiplication is necessary to solve the algorithms. The singular vectors in either the deterministic or economy-sized SVD, however, have extremely high column dimensions. The RSVD formulation, on the other hand, produces singular vectors with smaller column dimensions, which is better for the methods demonstrated in Chapter 4.

3.3 Solving CCA with Singular Value Decomposition

One way, among multiple methods to solve the CCA problem, is to apply SVD. Let us start from Equation (3.1). The sample covariance matrices built from the matrices of multivariate data $\mathbf{X}_a \in \mathbb{R}^{n\times m_1}$ and $\mathbf{X}_b \in \mathbb{R}^{n\times m_2}$, with mean zero, defined in section 3.1 are

$$\mathbf{\Sigma}_{ab} = \frac{1}{n}\mathbf{X}_a^t\mathbf{X}_b \in \mathbb{R}^{m_1\times m_2} \tag{3.12}$$

$$\mathbf{\Sigma}_{aa} = \frac{1}{n}\mathbf{X}_a^t\mathbf{X}_a \in \mathbb{R}^{m_1\times m_1} \tag{3.13}$$

$$\mathbf{\Sigma}_{bb} = \frac{1}{n}\mathbf{X}_b^t\mathbf{X}_b \in \mathbb{R}^{m_2\times m_2}. \tag{3.14}$$

Moreover, the joint covariance matrix is as follows,

$$\mathbf{\Sigma} = \begin{pmatrix} \mathbf{\Sigma}_{aa} & \mathbf{\Sigma}_{ab} \\ \mathbf{\Sigma}_{ba} & \mathbf{\Sigma}_{bb} \end{pmatrix}. \tag{3.15}$$

Instead of finding a solution for the r number of canonical correlations ρ as described in Section 3.1, we can solve the CCA problem through SVD. Furthermore, let us define the following definitions and Theorem from Horn and Johnson (2013):

Definition 3.13. Let $\mathbf{A}$ be a real symmetric matrix of size $n \times n$. It is positive definite if:

$$x^t \mathbf{A} x > 0, \text{ for all non-zero } x \in \mathbb{R}^n$$

and it is positive semi-definite if:

$$x^t \mathbf{A} x \geq 0, \text{ for all non-zero } x \in \mathbb{R}^n.$$

Theorem 3.14. *Let* $\mathbf{A}$ *be a real symmetric matrix and positive semi-definite, then there is a unique symmetric positive semi-definite matrix* $\mathbf{B}$ *such that* $\mathbf{B}^2 = \mathbf{A}$.

Since, $\mathbf{\Sigma}_{aa}$ and $\mathbf{\Sigma}_{bb}$ are symmetric and positive semi-definite by definition of covariance matrices, we can apply Theorem 3.14:

$$\mathbf{\Sigma}_{aa} = \mathbf{\Sigma}_{aa}^{\frac{1}{2}} \mathbf{\Sigma}_{aa}^{\frac{1}{2}}$$
$$\mathbf{\Sigma}_{bb} = \mathbf{\Sigma}_{bb}^{\frac{1}{2}} \mathbf{\Sigma}_{bb}^{\frac{1}{2}}.$$

According to Ewerbring and Luk (1989) and Uurtio et al. (2017), by multiplying symmetrically on the joint covariance matrix (3.15), we get

$$\begin{pmatrix} \mathbf{\Sigma}_{aa}^{-\frac{1}{2}} & 0 \\ 0 & \mathbf{\Sigma}_{bb}^{-\frac{1}{2}} \end{pmatrix} \begin{pmatrix} \mathbf{\Sigma}_{aa} & \mathbf{\Sigma}_{ab} \\ \mathbf{\Sigma}_{ba} & \mathbf{\Sigma}_{bb} \end{pmatrix} \begin{pmatrix} \mathbf{\Sigma}_{aa}^{-\frac{1}{2}} & 0 \\ 0 & \mathbf{\Sigma}_{bb}^{-\frac{1}{2}} \end{pmatrix} = \begin{pmatrix} \mathbf{I}_q & \mathbf{\Sigma}_{aa}^{-\frac{1}{2}} \mathbf{\Sigma}_{ab} \mathbf{\Sigma}_{bb}^{-\frac{1}{2}} \\ \mathbf{\Sigma}_{bb}^{-\frac{1}{2}} \mathbf{\Sigma}_{ba} \mathbf{\Sigma}_{aa}^{-\frac{1}{2}} & \mathbf{I}_p \end{pmatrix}.$$

Thus, by Theorem 3.6, one can apply the deterministic SVD or any type of SVD algorithm (economy-sized SVD, RSVD using either RRF or RPI) on the product of the covariance matrices, namely $\mathbf{\Sigma}_{aa}^{-\frac{1}{2}} \mathbf{\Sigma}_{ab} \mathbf{\Sigma}_{bb}^{-\frac{1}{2}}$ of dimension $m_1 \times m_2$, and rank $r \leq \{m_1, m_2\}$. Let us apply RSVD,

$$\mathbf{\Sigma}_{aa}^{-\frac{1}{2}} \mathbf{\Sigma}_{ab} \mathbf{\Sigma}_{bb}^{-\frac{1}{2}} \approx \mathbf{U}_k \mathbf{\Lambda}_k \mathbf{V}_k^t, \tag{3.16}$$

with the columns of $\mathbf{U} \in \mathbb{R}^{m_1 \times k}$ and $\mathbf{V} \in \mathbb{R}^{m_2 \times k}$ being the left and right singular vectors respectively, and the diagonal matrix $\mathbf{\Lambda} \in \mathbb{R}^{k \times k}$ entries being the singular values, also known here as the canonical correlations, and the target rank $k \ll r$. This derivation shows that the CCA problem can be reduced to finding a SVD of the products of the covariance matrices in Equation (3.16). In the case of Dhillon et al. (2015), they used RPI with a power iteration parameter $q = 5$ to solve Equation (3.16).

Finally, the pair of linear transformation matrices $\mathbf{\Phi}_a \in \mathbb{R}^{m_1 \times k}$ and $\mathbf{\Phi}_b \in \mathbb{R}^{m_2 \times k}$, solution to our problem, is

$$\mathbf{\Phi}_a = \mathbf{\Sigma}_{aa}^{-\frac{1}{2}} \mathbf{U}_k \tag{3.17}$$
$$\mathbf{\Phi}_b = \mathbf{\Sigma}_{bb}^{-\frac{1}{2}} \mathbf{V}_k. \tag{3.18}$$

Since the SVD solution to the CCA problem requires fewer multiplications of matrices, it is preferred over other methods. It is further motivated by the fact that most matrices involved in this work will be large sparse matrices. Hence, the SVD formulation should reduce the computational complexity of the algorithms that will be shown in the next chapter (Dhillon et al., 2015). For a more detailed derivation of CCA using SVD as a solution, see Appendix B.

Furthermore the randomized solution is favored over the deterministic SVD or economy-sized SVD. This is due to the dimensions of the singular vectors being smaller as a result of the RSVD compared to the other SVD factorizations. RSVD achieves dimensionality reduction of the matrices $\mathbf{\Phi}_a$ and $\mathbf{\Phi}_b$.

To be more specific, we shall assume that SVD decomposition is applied on the product of the covariance matrices from Equation (3.16). Then, the singular vectors $\mathbf{U}$ and $\mathbf{V}$, have the following dimensions based on the type of SVD applied:

- Deterministic SVD, $\mathbf{U} \in \mathbb{R}^{m_1 \times m_1}$ and $\mathbf{V} \in \mathbb{R}^{m_2 \times m_2}$.
- Economy-sized SVD, $\mathbf{U}_r \in \mathbb{R}^{m_1 \times r}$ and $\mathbf{V}_r \in \mathbb{R}^{r \times m_2}$.
- Truncated or RSVD, $\mathbf{U}_k \in \mathbb{R}^{m_1 \times k}$ and $\mathbf{V}_k \in \mathbb{R}^{k \times m_2}$,

with the rank $r \leq \min\{m_1, m_2\}$, and target rank $k \ll r$. Thus, using RSVD to solve CCA let us freely choose k, the column dimensions of the position matrices (3.17) and (3.18). Therefore, whenever we imply SVD later in this paper, we will actually refer to the RSVD using either the RRF or the RPI.

Finally, one can define,

Definition 3.15. Let the data matrices $\mathbf{X_a}$ of size $n \times m_1$ and $\mathbf{X_b}$ of size $n \times m_2$ be constructed from the observations. Consequently, one can define the **CCA operation** as

$$(\mathbf{\Phi_a}, \mathbf{\Phi_b}) \equiv CCA(\mathbf{X_a}, \mathbf{X_b}), \tag{3.19}$$

with $(\mathbf{\Phi_a}, \mathbf{\Phi_b})$ the pair of linear transformation matrices.

Chapter 4

Spectral Word Embedding Models based on CCA

In recent years, various models of word embedding have emerged, usually being unsupervised learning methods that map words to vectors using the words' occurrence structure in unlabeled corpus of text. Some of these models are based upon neural networks such as in Mikolov et al. (2013a) or spectral learning (Matrix factorization) such as Eigenwords (Dhillon et al., 2015), just to name a few.

We present next, Spectral word embeddings based on CCA as described by Dhillon et al. (2015), "Eigenwords". Many state-of-the-art models suffer from different shortcomings. According to Dhillon et al. (2015), deep learning based models are slow due to the hidden layers. Some other successful spectral models, such as Latent Semantic Analysis or LSA (Landauer et al., 1998), one of the earliest embedding models that works by taking the SVD of the document-term matrix, and as a result achieves word embeddings, struggles to capture polysemy (a word having multiple meaning). Indeed, one embedding vector is created for each word type. Let us take, for instance, the word "bank"; a single embedding will be created by the bag-of-words model LSA, regardless if the context of the word suggests "a financial institution" or "a river bank" (Dhillon et al., 2015). Some other models even suffer from having bad scalability, the more incoming data the models receive, the higher the inability of the model to work under such quantity. Others struggle to learn rare words and need a huge amount of data.

According to Dhillon et al. (2015), Eigenwords is offered as a solution for these drawbacks, reasoning that Spectral word embedding models using CCA are fast, scalable, have a good sample complexity for rare words, and can generate context specific embeddings for word polysemy.

4.1 The Lexical Framework

Following Dhillon et al. (2015), let us explain briefly the idea behind CCA as a technique to induce word embeddings. As a reminder, one natural approach of operating CCA here, would be to firstly set two matrices: **W** representing the word types and **C** representing the context of the words. Then, we use CCA to project these matrices into a lower dimensional space, while maximizing the correlation between them. Thus, the new projection of **W** would additionally be a representation of those word types. Dhillon et al. (2015)'s goal was to estimate the lower dimensional representation of those word types.

Before digging deeper and estimating those vectors through CCA, let us formulate the problem at hand. Assume a documents or a set of document containing n tokens $\{t_1, ..., t_n\}$, each drawn from a vocabulary of v word types (word space of size v).

Definition 4.1. The **left and right contexts** of each token t_i, are respectively the h words to the left and right of that token, with h the size of the context, chosen arbitrarily.

For example, let us take our trusty lower-cased and lemmatized toy sentence from Section 2.2, "red rose be red red". The vocabulary set V, being the set of unique words (word types), in alphabetical order is

$$V = \{be, red, rose\}$$

of size $v = 3$. Furthermore, we have the set of tokens, T, of size $n = 5$,

$$T = \{red, rose, be, red, red\}.$$

Let the size of the context be $h = 2$. For the token t_3, represented by "be", the left context would be the $h = 2$ words to its left, namely "red" and "rose". Similarly, the right context are the $h = 2$ words to its right, "red" and "red".

Definition 4.2. Define $\mathbf{L}$ and $\mathbf{R} \in \mathbb{R}^{n \times vh}$ as the left and right context matrices of the tokens and $\mathbf{W} \in \mathbb{R}^{n \times v}$ as the matrix of the tokens themselves. These matrices keep counts of the words co-occurrences (words occurring in the same context) and occurrences.

To be more specific, in the matrix $\mathbf{W} \in \mathbb{R}^{n \times v}$, the rows are defined by the n tokens in the same order as they appear in the text corpus and the columns are defined by the v word types in alphabetical order. Therefore, to represent the presence of the j^{th} word type on the i^{th} position in a document, one sets the entry

$w_{ij} = 1$.
Going back to our simple toy example "red rose be red red", in order to represent the existence of the second word type ("red") in the fourth position, we set the entry $w_{42} = 1$. The full matrix of the tokens $\mathbf{W} \in \mathbb{R}^{5\times3}$, is obtained in such like manner,

$$\mathbf{W} = \begin{array}{c} \begin{array}{ccc} be & red & rose \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array} \begin{array}{c} red \\ rose \\ be \\ red \\ red \end{array} .$$

The matrices $\mathbf{L} \in \mathbb{R}^{n\times vh}$ and $\mathbf{R} \in \mathbb{R}^{n\times vh}$ are obtained in a similar fashion, but have columns for each word on each position in the context (Dhillon et al., 2015). The rows are defined by the n tokens in the same order as they appear in the text corpus similar to $\mathbf{W}$ but the columns are defined by the v word types multiplied by h.

These two matrices are not as clear to visualize, therefore let us take a look at the empty body of the $\mathbf{L}$ matrix, each increment of the value of h by one would enlarge the matrix $\mathbf{L}$ by a submatrix of size $n \times v$. It is equivalent to state that the number of columns of $\mathbf{L}$ are enlarged by the size of the vocabulary v each time h is incremented by one.

$$\begin{array}{c} \qquad\qquad v \times h \text{ word types from the vocabulary} \\ \qquad h = 1 \qquad\qquad\qquad\qquad h = 2 \\ \mathbf{L} = \left(\text{Word type left of token} \;\middle|\; \text{Word type } \textbf{two} \text{ words left of token} \right) n \text{ tokens.} \end{array}$$

In order to be more clear on how to construct and populate the matrix $\mathbf{L}$ and $\mathbf{R}$, let us first explain the procedure for the matrix $\mathbf{L}$. If $h = 1$, to represent the presence of the j^{th} word type on the position of the left context of the i^{th} token, one sets $l_{ij} = 1$. If $h = 2$, we set the entry $l_{ij} = 1$ as explained previously, and account for the presence of the second word type being two words away on the left context of the i^{th} token and set that entry to 1 too.

Let us illustrate this explanation with our previous example, "red rose be red red" and focus on the contexts of the middle token, "be".

Let $h = 2$ and, first, we need to set the entry for the first left context in the submatrix of $h = 1$. To do so, we express the presence of the word type $j = 3$ ("rose"), on the position of the left context of the third token $i = 3$ ("be"), by setting $l_{33} = 1$. Secondly, in the submatrix of $h = 2$, we need to also represent the second left context, which is two words away on the left of the chosen token $i = 3$ ("be"), the second context being the second word type $j = 2$ ("red"). To represent the second word type $j = 2$ ("red") in the submatrix of $h = 2$, one only needs to add $v = 3$ the size of the vocabulary to j. Thus, we set $l_{35} = 1$. One yields the full $\mathbf{L} \in \mathbb{R}^{5\times 6}$ matrix based on the concatenation of the two submatrices $h = 1$ and $h = 2$ as one after each other,

$$\mathbf{L} = \begin{array}{c} \quad \text{h=1} \qquad\qquad \text{h=2} \\ \begin{array}{ccc|ccc} \textit{be} & \textit{red} & \textit{rose} & \textit{be} & \textit{red} & \textit{rose} \end{array} \\ \left(\begin{array}{ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{array}\right) \begin{array}{c} \textit{red} \\ \textit{rose} \\ \textit{be} \\ \textit{red} \\ \textit{red} \end{array} \end{array}.$$

The matrix $\mathbf{R} \in \mathbb{R}^{5\times 6}$ is built in the same fashion but we consider the right contexts instead of the left.

Now, let us define the complete context matrix.

Definition 4.3. The complete context matrix $\mathbf{C} \in \mathbb{R}^{n\times 2vh}$, is the concatenation of the left and right context matrices one after each other,

$$\mathbf{C} = [\mathbf{L}, \mathbf{R}].$$

Lastly, using the same toy example, the matrix $\mathbf{C} \in \mathbb{R}^{5\times 12}$ is simply the concatenation of the matrices $\mathbf{L}$ and $\mathbf{R}$.

Therefore, the context belongs to a very high dimensional space. This is so, since for a vocabulary of size v, each of the $2h$ words in the combined contexts requires an indicator function of dimension v. The tokens themselves belong to an-v dimensional space of words (the vocabulary), which we want to project down to an k-dimensional space (Dhillon et al., 2015).

Thus, the matrices in (3.12) to (3.14), can be computed with $\mathbf{C}$ and $\mathbf{W}$. We

will define them as such:

$$\begin{aligned}\mathbf{\Sigma}_{wc} &= \mathbf{W}^t\mathbf{C} \\ \mathbf{\Sigma}_{cc} &= \mathbf{C}^t\mathbf{C} \\ \mathbf{\Sigma}_{ww} &= \mathbf{W}^t\mathbf{W}.\end{aligned}$$

To give meaning to these matrices, $\mathbf{\Sigma}_{wc} \in \mathbb{R}^{v\times 2vh}$ is the matrix representing the counts of how often each word w occurs in each context c, $\mathbf{\Sigma}_{cc} \in \mathbb{R}^{2vh\times 2vh}$ is the empirical covariance of the contexts matrix and $\mathbf{\Sigma}_{ww} \in \mathbb{R}^{v\times v}$ is the empirical word covariance matrix.

The above forms of the matrices are valid under the assumption that their sample means are equal to zero. This is indeed the setting in this thesis. It follows through a simple observation. According to Stratos et al. (2015), let $\mathbf{X}_a \in \mathbb{R}^{n\times m_1}$ and $\mathbf{X}_b \in \mathbb{R}^{n\times m_2}$ be two data matrices that have not been standardized, with their respective row vectors $x_i^a \in \mathbb{R}^{m_1}$ and $x_i^b \in \mathbb{R}^{m_2}$, for $i = 1, ..., n$. The sample cross-covariance matrix (Gubner, 2006; Stratos et al., 2015) has the following form:

$$\mathbf{\Sigma}_{ab} = \frac{1}{n}\sum_{i=1}^{n} x_i^a (x_i^b)^t - \frac{1}{n^2}\left(\sum_{i=1}^{n} x_i^a\right)\left(\sum_{i=1}^{n} x_i^b\right)^t. \tag{4.1}$$

As stated by Dhillon et al. (2015), since the distribution of words is described by Zipf's law, the second term is dominated by the first one, especially as the number of word-context n in a corpus increases. In the limit of n tending to infinity, the second term of Equation (4.1) converges to 0. Furthermore, the factor $\frac{1}{n}$ in (4.1), is discarded as the objective Equation (3.1) of CCA is scale invariant. Therefore, $\mathbf{\Sigma}_{wc}$ can be considered as an empirical cross-covariance matrix, and $\mathbf{\Sigma}_{cc}$ and $\mathbf{\Sigma}_{ww}$, as empirical covariance matrices.

Finally, as stated by Dhillon et al. (2015), by applying different CCA algorithms on either, the Left and Right context matrices, the complete context matrix $\mathbf{C}$ and the matrix of the tokens $\mathbf{W}$, one can find latent vectors that describe each v word types of the vocabulary. Those words that are distributionally similar in the sense that they have similar context, are expected to have similar state vectors.

Definition 4.4. By using the above formulation for the matrices describing the word space (size v) and by applying CCA on those matrices, one accomplishes the mapping from this same word space to a lower dimension space of size k. This mapping, $\mathbf{\Phi}_w$, is also called the **eigenword dictionary**. It associates for every word, a latent vector that retains the word's syntactic and semantic attributes (Dhillon et al., 2015).

4.2 One Step Canonical Correlation Analysis

The most simple application of CCA in order to estimate the eigenword dictionary is the One Step CCA (OSCCA) (Dhillon et al., 2015). It is achieved by taking the CCA between, $\mathbf{C}$ and $\mathbf{W}$ (defined above) for some text data.

Let a corpus be composed of n tokens and v word types, we can then form $\mathbf{W} \in \mathbb{R}^{n \times v}$ the matrix of the tokens and $\mathbf{C} \in \mathbb{R}^{n \times 2vh}$ the complete context matrix, with h the size of the context chosen arbitrarily. The OSCCA can be defined as such,

$$(\mathbf{\Phi}_w, \mathbf{\Phi}_c) \equiv CCA(\mathbf{W}, \mathbf{C}),$$

with $\mathbf{\Phi}_w \in \mathbb{R}^{v \times k}$ the eigenword dictionary and $\mathbf{\Phi}_c \in \mathbb{R}^{2hv \times k}$,

$$\mathbf{\Phi}_w = \mathbf{\Sigma}_{ww}^{-\frac{1}{2}} \mathbf{U}$$
$$\mathbf{\Phi}_c = \mathbf{\Sigma}_{cc}^{-\frac{1}{2}} \mathbf{V},$$

The matrices $\mathbf{U} \in \mathbb{R}^{v \times k}$ and $\mathbf{V} \in \mathbb{R}^{2hv \times k}$ are obtained through the SVD of the covariance matrices product,

$$\mathbf{\Sigma}_{ww}^{-\frac{1}{2}} \mathbf{\Sigma}_{wc} \mathbf{\Sigma}_{cc}^{-\frac{1}{2}} = \mathbf{U} \mathbf{\Lambda} \mathbf{V}^t \in \mathbb{R}^{v \times 2hv}, \tag{4.2}$$

with $\mathbf{\Lambda} \in \mathbb{R}^{k \times k}$ the matrix of the biggest k singular values (i.e. correlations). Furthermore, by using the square root factorizations on the covariance matrices, one gets

$$\mathbf{\Sigma}_{ww} = \mathbf{\Sigma}_{ww}^{\frac{1}{2}} \mathbf{\Sigma}_{ww}^{\frac{1}{2}}$$
$$\mathbf{\Sigma}_{cc} = \mathbf{\Sigma}_{cc}^{\frac{1}{2}} \mathbf{\Sigma}_{cc}^{\frac{1}{2}}.$$

Then, by taking their inverses [1], we get all the tools needed to compute the OSCCA and solve the equations above. The output of this algorithm is $\mathbf{\Phi_w} \in \mathbb{R}^{v \times k}$, the eigenword dictionary in the form of a matrix. It associates for each of the v words an k-dimensional vector.

The r and k dimensional spaces

One thing we want to clarify, is the purpose of the notation of r and k. Let us investigate all the different instances of r and k we used.

[1] Note that section 4.2 assumes $\mathbf{\Sigma}_{ww}$ and $\mathbf{\Sigma}_{cc}$ are positive definite such that their inverses exist. However in practice the construction suggested in section 4.1 does guarantee only the positive definiteness of the matrix $\mathbf{\Sigma}_{ww}$. We will show how to solve this problem in section 4.5.1.

First, we used r as the number of canonical correlations in Section 3.1. When one applies CCA on $\mathbf{W} \in \mathbb{R}^{n \times v}$ and $\mathbf{C} \in \mathbb{R}^{n \times 2vh}$, the number of canonical correlations obtained is,

$$\begin{aligned} & r \leq \min\{v, 2vh\} \\ & \Rightarrow r \leq v, \end{aligned}$$

according to Equation (3.7).

Secondly, we also labeled r as the number of singular values in the SVD decomposition in Section 3.2. Following Theorem 3.6 and the derivation of CCA using SVD, the r canonical correlations of the CCA between $\mathbf{W}$ and $\mathbf{C}$ are actually the r singular values of the SVD decomposition of Equation (4.2).

From this same Theorem (3.6) we know that the r singular values of the SVD of a matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ is also its rank $r \leq \min\{n, m\}$. Thus, the dimension of the triple product of Equation (4.2) being $v \times 2vh$, and r being the rank of the matrix on which we apply SVD, we get,

$$\begin{aligned} & r \leq \min\{v, 2vh\} \\ & \Rightarrow r \leq v. \end{aligned}$$

Lastly, k is defined as the target rank for a low-rank matrix approximation, such as RSVD. According to definition 3.8, $k \ll r \leq v$. Hence, we can set any positive integer we want to be k as long as it is not exceeding r, which is in and of itself smaller than v, the size of the corpus vocabulary. Therefore, by solving CCA with RSVD, we can choose k, and thus the dimension of the eigenword dictionary $\mathbf{\Phi}_w \in \mathbb{R}^{v \times k}$. It is in fact, a parameter of the OSCCA model. Thus, we have motivated the choice of the common notation of r and k.

4.3 Two Step Canonical Correlation Analysis

Now let us take a look through the lens of Dhillon et al. (2015) at an improved version of Canonical Correlation Analysis to create word embeddings, the Two Step CCA (TSCCA). The motivation behind this second procedure is that, OSCCA requires vast amounts of text data to perform well. As an illustration, some languages have limited amount of written literature, as a consequence of which, due to Zipf's law, rare words will be scarcely represented, and so OSCCA will create low-quality embeddings.

To remedy this problem, Dhillon et al. (2015) suggests a two-step version of OSCCA that, according to them, has a better sample complexity for rare words.

Instead of taking the CCA between $\mathbf{C}$ and $\mathbf{W}$, we split the process into two-steps. First, we apply CCA on the left and right context matrices and use the result to estimate the state $\mathbf{S}$, which we will define later in this Section. Once again, let a corpus of text be composed of n tokens, v word types and let h be the size of the context chosen arbitrarily. We can form the left and right context matrices, $\mathbf{L} \in \mathbb{R}^{n \times vh}$ and $\mathbf{R} \in \mathbb{R}^{n \times vh}$. Let the empirical covariance matrices $\mathbf{\Sigma}_{ll} \in \mathbb{R}^{vh \times vh}$, $\mathbf{\Sigma}_{rr} \in \mathbb{R}^{vh \times vh}$, and $\mathbf{\Sigma}_{lr} \in \mathbb{R}^{vh \times vh}$ be defined as,

$$\begin{aligned}\mathbf{\Sigma}_{ll} &= \mathbf{L}^t\mathbf{L} \\ \mathbf{\Sigma}_{rr} &= \mathbf{R}^t\mathbf{R} \\ \mathbf{\Sigma}_{lr} &= \mathbf{L}^t\mathbf{R}.\end{aligned}$$

Hence, one can perform the first CCA,

$$(\mathbf{\Phi}_l, \mathbf{\Phi}_r) \equiv CCA(\mathbf{L}, \mathbf{R}),$$

with $\mathbf{\Phi}_l \in \mathbb{R}^{vh \times k}$ and $\mathbf{\Phi}_r \in \mathbb{R}^{vh \times k}$,

$$\begin{aligned}\mathbf{\Phi}_l &= \mathbf{\Sigma}_{ll}^{-\frac{1}{2}}\mathbf{U}_1 \\ \mathbf{\Phi}_r &= \mathbf{\Sigma}_{rr}^{-\frac{1}{2}}\mathbf{V}_1.\end{aligned}$$

The matrices $\mathbf{U}_1 \in \mathbb{R}^{vh \times k}$ and $\mathbf{V}_1 \in \mathbb{R}^{vh \times k}$ are obtained from the SVD of the triple covariance matrices product,

$$\mathbf{\Sigma}_{ll}^{-\frac{1}{2}}\mathbf{\Sigma}_{lr}\mathbf{\Sigma}_{rr}^{-\frac{1}{2}} = \mathbf{U}_1\mathbf{\Lambda}_1\mathbf{V}_1^t \in \mathbb{R}^{vh \times vh}, \tag{4.3}$$

with $\mathbf{\Lambda}_1 \in \mathbb{R}^{k \times k}$ the matrix of the k largest singular values of Equation (4.3). In the same fashion, by using square root factorization on the covariance matrices, one gets

$$\begin{aligned}\mathbf{\Sigma}_{ll} &= \mathbf{\Sigma}_{ll}^{\frac{1}{2}}\mathbf{\Sigma}_{ll}^{\frac{1}{2}} \\ \mathbf{\Sigma}_{rr} &= \mathbf{\Sigma}_{rr}^{\frac{1}{2}}\mathbf{\Sigma}_{rr}^{\frac{1}{2}}.\end{aligned}$$

All that is left to do is to take the inverse of the factorizations [2]. Doing so, we can then compute the triple product (4.3), its SVD decomposition, and then $\mathbf{\Phi}_l$ and

[2]Similar to OSCCA, the construction of $\mathbf{\Sigma}_{ll}$ and $\mathbf{\Sigma}_{rr}$ does not guarantee positive definiteness. We show a solution to this problem in Section 4.5.1.

$\mathbf{\Phi}_r$.

Now let us estimate the matrix $\mathbf{S} \in \mathbb{R}^{n \times 2k}$. Dhillon et al. (2015) defines it as "the state matrix of all the tokens in the text corpus from their contexts". One constructs $\mathbf{S}$ as such,

$$\mathbf{S} = [\mathbf{L}\mathbf{\Phi}_l, \mathbf{R}\mathbf{\Phi}_r].$$

It is the concatenation of the product of the contexts and their respective lower dimensional estimations. The matrix $\mathbf{S}$ can contain redundant information, but the main purpose of this concatenation process is to not lose any information present exclusive to one side. Let next the empirical covariance matrix $\mathbf{\Sigma}_{ss} \in \mathbb{R}^{2k \times 2k}$ be defined as

$$\mathbf{\Sigma}_{ss} = \mathbf{S}^t\mathbf{S}.$$

Secondly, we take the CCA between $\mathbf{W}$ and $\mathbf{S}$,

$$(\mathbf{\Phi}_w, \mathbf{\Phi}_s) \equiv CCA(\mathbf{W}, \mathbf{S}),$$

with the eigenword dictionary $\mathbf{\Phi}_w \in \mathbb{R}^{v \times k}$ and $\mathbf{\Phi}_s \in \mathbb{R}^{2k \times k}$,

$$\mathbf{\Phi}_w = \mathbf{\Sigma}_{ww}^{-\frac{1}{2}}\mathbf{U}_2$$
$$\mathbf{\Phi}_s = \mathbf{\Sigma}_{ss}^{-\frac{1}{2}}\mathbf{V}_2.$$

In the same fashion as in the first step, the matrices $\mathbf{U}_2 \in \mathbb{R}^{v \times k}$ and $\mathbf{V}_2 \in \mathbb{R}^{2k \times k}$ are obtained through the SVD of the triple product of the covariance matrices,

$$\mathbf{\Sigma}_{ww}^{-\frac{1}{2}}\mathbf{\Sigma}_{ws}\mathbf{\Sigma}_{ss}^{-\frac{1}{2}} = \mathbf{U}_2\mathbf{\Lambda}_2\mathbf{V}_2^t \in \mathbb{R}^{v \times 2k}, \tag{4.4}$$

with $\mathbf{\Lambda}_2 \in \mathbb{R}^{k \times k}$ the matrix of the k largest singular values of the equation. Similarly, we compute the inverse square roots of $\mathbf{\Sigma}_{ww}$ and $\mathbf{\Sigma}_{ss}$ [3], and finally form $\mathbf{\Phi}_w \in \mathbb{R}^{v \times k}$, the eigenword dictionary.

The r and k dimensional space

Once again, using the same logic as in the OSCCA case, one can easily motivate the common notation of r and k. It is also a parameter of TSCCA, and it has the same restriction $k \ll r \leq v$.

[3] By construction, $\mathbf{\Sigma}_{ww}$ is guaranteed to be positive definite but not $\mathbf{\Sigma}_{ss}$. We will suggest an alternative to this problem in Section 4.5.2.

4.4 Numerical Motivation

So far we have only laid the theoretical background of OSCCA and TSCAA without further motivation on how these methods perform. To demonstrate the effectiveness of these two models, we shall take a short walk through the numerical results demonstrated by Dhillon et al. (2015) on two tests: word similarity task on WordSim-353 and part-of-speech (POS) tagging.

Word Similarity Task

First, Dhillon et al. (2015) trained their models, OSCCA, TSCCA and other models on Reuters RCV1 corpus (Rose et al., 2002). RCV1 contains newswire stories from Reuters from August 1996 to August 1997. They did not change the case, nor did they clean the corpus data. In order to fairly compare to other methods, they set $h = 2$ and the vocabulary to $100,000$ of the most frequent words. According to them, since Eigenwords algorithms are robust to the dimensionality reduction, they did not tune it and set $k = 200$. As for the other algorithms, they tuned it in order to use their best iterations.

Without going into too much detail as we will talk about word similarity task later in Section 7.1.3, the word similarity task is an evaluation method that compares pairs of word vectors to pairs of words from an evaluation data set. In this case, the dataset used was the **WordSim-353** (Finkelstein et al., 2001), which contains 353 pairs of nouns with a similarity and relatedness scores from 0 to 10, for each pair given by 13 to 16 human assessments.

Firstly, one requires the scores of the word pairs from the data set of human judgments and the cosine similarities (Faruqui et al., 2016) between the word embeddings of the corresponding pairs of words.
The cosine similarity between two word vectors a and b is defined by Faruqui et al. (2016) as:

$$\text{cosine}(a, b) = \frac{\langle a, b \rangle}{\|a\| \|b\|},$$

Secondly, one computes the Spearman correlation coefficient between the human scores and the cosine similarities. The goal of this test is to compare how well word embedding models, such as OSCCA or TSCCA, capture the different notions of similarity between words compared to human judgements.

Their results are summarized in the Table 4.1.

Model	$\rho \times 100$
PCA (Mikolov et al., 2013a)	30.25
Word2vec (SK) (Mikolov et al., 2013a)	42.73
Word2vec (CBOW) (Mikolov et al., 2013a)	42.97
Eigenwords (OSCCA) (Dhillon et al., 2015)	43.00
Eigenwords (TSCCA) (Dhillon et al., 2015)	44.85

Table 4.1: Table from Dhillon et al. (2015) showing some of the Spearman correlations between the human scores and the cosine similarities obtained through word embedding models.

As we can see, TSCCA has the best result, its correlation being the highest. All these embedding methods have correlations in comparable range apart from the Principal Component Analysis (PCA) method, which has shown the worst result. On another note, the PCA algorithm in the context of creating embeddings as described by Dhillon et al. (2015) is pretty straightforward. SVD is applied on $\mathbf{\Sigma}_{wc}$ in order to perform PCA. In doing so, a projection of the matrix $\mathbf{\Sigma}_{wc}$ in a lower dimensional space is obtained, which contains the word embeddings.

In general, the correlation values are not very high, ranging from 0.3 to 0.5. Therefore, we can conclude that the embeddings and the human scores show some moderate positive correlation in all methods. From this, we can assume that most of these models' results are satisfactory, but they somewhat struggle to capture the similarities between words as the human assessors see it. Not to mention the **WordSim-353** has only 353 samples, thus the results' representativity can be called into question.

Part-of-Speech Tagging (POS)

Part-of-speech tagging or POS tagging aims to correctly assign a part of speech tag to each token in a corpus. Each word type can only have one POS tag, such as noun, verb, adjective, etc. It is a supervised learner that is trained on a labeled corpus to predict the tags. To be more specific, these corpus are called treebanks, where each token is assigned a part of speech label. The idea behind this method is to use the already trained word embeddings from the unsupervised model and utilize them as features for the supervised learner. Therefore, in this evaluation scheme, word embeddings are considered "good" when it significantly enhances the ability of the supervised learner to predict tags.

For this specific task, Dhillon et al. (2015) use 5 word embedding models but we will only talk about 3 of them in the scope of this thesis: a simple PCA approach to word embeddings, OSCCA and TSCCA. In order to create the embedding vectors,

the embedding models have been trained in an unsupervised manner on the Wall Street Journal of the Penn Treebank (PTB). (Marcus et al., 1993). This dataset has a POS label associated to each token. It is worth noting that only 17 part-of-speech tags are considered for this test. The goal of their experiment was to see the improvement of the supervised learner when it utilized the word vectors, when the size of the input training data is increased. In order to do so, the vocabulary must stay fixed to ensure that the improvement is not due to an increase in word types but due to the training set size only. Therefore, they vary the unlabeled data from $5,000$ to $100,000$ tokens, and made sure that the vocabulary stays fixed. Their prior assumption is that TSCCA should perform better than OSCCA with smaller sample sizes and should produce similar results for a large training set.

As for the evaluation part, the data set has been randomly split into 80% word types in the training set and 20% word types in the test set, such that each set does not have any word types in common. Then, a multi-class logistic regression model, using the word embedding vectors as features, will learn from the training set to predict the part-of-speech label associated with every word type. Finally, they test the predictive power of the model on the remaining test set. They repeat this procedure 10 times and average it to assure the reliability of the results.
To summarize, in order to test word embedding models on the task of POS tagging, one needs to split the process in two distinct training. First, the unsupervised learning of the word embedding estimation vectors, and secondly the learning of a supervised model in order to test the accuracy of the embedding estimations.

Figure 4.1 shows a plot of the evolution of the accuracy when the word embedding models are given a bigger training set.

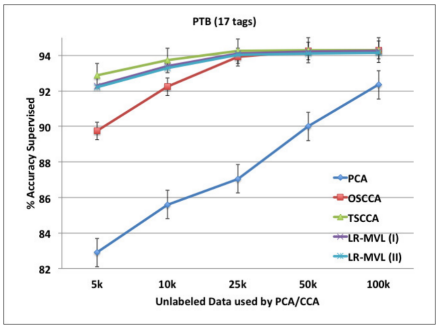


Figure 4.1: Plot from Dhillon et al. (2015) showing the evolution of the supervised model's averaged accuracy (averaged over 10 random splits) as a function of the number of tokens in the training corpus for different word embedding models.

This graph demonstrates that, regardless of the size of the unlabeled corpus, utilizing a simple PCA to produce word embeddings yields the worst accuracy. As

expected from prior assumption, OSCCA shows worse result than TSCCA, in the case of a corpus with fewer tokens. However, when reaching a threshold of a larger dataset, the accuracy improves until reaching results comparable to TSCCA.

4.5 Covariance Matrices in OSCCA and TSCCA

One recurrent problem occurring in the process of OSCCA and TSCCA is the non-positive definiteness of some sparse covariance matrices in order to use the algorithms. This issue is glossed over by Dhillon et al. (2015) in their appendix, and we could not find any relevant articles that mention their work and raise this issue.

4.5.1 Sparse Covariance Matrices

First, let us walk through some key equations where the non-positive definiteness of the sparse covariance matrices becomes an issue, namely, OSCCA's Equation (4.2) and TSCCA's Equation (4.3),

$$\boldsymbol{\Sigma}_{ww}^{-\frac{1}{2}}\boldsymbol{\Sigma}_{wc}\boldsymbol{\Sigma}_{cc}^{-\frac{1}{2}} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^t$$
$$\boldsymbol{\Sigma}_{ll}^{-\frac{1}{2}}\boldsymbol{\Sigma}_{lr}\boldsymbol{\Sigma}_{rr}^{-\frac{1}{2}} = \mathbf{U}_1\boldsymbol{\Lambda}_1\mathbf{V}_1^t.$$

Before applying SVD on either triple product of matrices, we need to individually compute each matrix on the left hand side for both equations. As for the matrix $\boldsymbol{\Sigma}_{ww}$, by construction, it is just the covariance matrix of word occurrences, it is a diagonal matrix with only positive entries on the diagonal. Therefore, it is a positive definite matrix and $\boldsymbol{\Sigma}_{ww}^{-\frac{1}{2}}$ can be formed easily by taking the inverse of the square root of each element on the diagonal of $\boldsymbol{\Sigma}_{ww}$.

The issue emerges for the empirical covariance matrices $\boldsymbol{\Sigma}_{cc}$, $\boldsymbol{\Sigma}_{ll}$ and $\boldsymbol{\Sigma}_{rr}$. Unlike $\boldsymbol{\Sigma}_{ww}$, these covariance matrices are not diagonal matrices with only non-zeros entries on the diagonal, and therefore one cannot apply the simple process described to form $\boldsymbol{\Sigma}_{ww}^{-\frac{1}{2}}$. By definition of covariance matrices, they are symmetric and positive semi-definite. Consequently, for these matrices to be invertible, we will need all their eigenvalues to be positive, which is equivalent to stating that positive semi-definite matrices are invertible if and only if they are positive definite.

First of all, we have to ask ourselves when an empirical covariance matrix $\mathbf{X}^t\mathbf{X}$ is positive definite and when it is not. We can show that:

Observation 4.5. Let $\mathbf{X}$ be a matrix of size $n \times m$, $\mathbf{X}^t\mathbf{X}$ is always at least semi-definite positive, and positive definite when $\mathbf{X}$ is a full rank matrix.

Proof. Let $v \in \mathbb{R}^m$ be non-trivial, $\mathbf{X} \in \mathbb{R}^{n \times m}$, and let us assume that $m \leq n$. If $\text{Rank}(\mathbf{X}) = m$, then $\mathbf{X}$ has full column rank. This means that $\mathbf{X}v$ is a linear combination of the columns of $\mathbf{X}$ with a vector v. Since $v \neq 0$ is non-trivial, $\mathbf{X}v$ is a non-trivial linear combination as well. Thus, we can define a vector $y = \mathbf{X}v \neq 0$. Accordingly, given any $v \neq 0$, we get

$$v^t \mathbf{X}^t \mathbf{X} v = (\mathbf{X}v)^t \mathbf{X}v = y^t y = \sum_i^n y_i^2 > 0$$

This is the implication for positive definiteness of a matrix. Hence, if $m \leq n$ and $\text{Rank}(\mathbf{X}) = m$, then the empirical matrix $\mathbf{X}^t\mathbf{X}$ is positive definite.
In the case of the matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$ not being full rank, the proof can be easily modified and we can show that any matrix of the form $\mathbf{X}^t\mathbf{X}$ is always positive semi-definite. □

Furthermore, let us say that $\mathbf{\Sigma}_{cc}$ is positive definite, and one could compute the inverse. However, there is nothing guaranteeing that the inverse of a sparse matrix is sparse as well. Therefore, the resulting $\mathbf{\Sigma}_{cc}^{-1}$ and $\mathbf{\Sigma}_{cc}^{-\frac{1}{2}}$ of size $2vh \times 2vh$ on top of being large matrices, can also be highly dense, making the triple product of Equation (4.2) burdensome on the memory and slow, which can easily result into a memory overflow error if the corpus data are large. The same issue applies to $\mathbf{\Sigma}_{ll}$ and $\mathbf{\Sigma}_{rr}$ in Equation (4.3) for the TSCCA procedure.

In the work of Dhillon et al. (2015), to bypass this issue in their code[4], they approximate $\mathbf{\Sigma}_{cc}$ to a diagonal covariance matrix and drop off all the values that are not on the diagonal and replace the zeros on the diagonal with a very small $\epsilon > 0$. By doing so, the matrix becomes positive definite, and $\mathbf{\Sigma}_{cc}^{-\frac{1}{2}}$ can be obtained by the simple process of inverting and taking the square root of each element on the now diagonal matrix. The major advantage of this process, is that it guarantees the sparseness of $\mathbf{\Sigma}_{cc}^{-\frac{1}{2}}$. They justify the validity of this approximation by assuming the covariance matrix can be approximated using a Taylor expansion, and the trade-off between the result's quality, the sparseness of the output matrix and the time taken to compute, is not worth adding the first order of the Taylor expansion (non-diagonal entries). Unfortunately, adding a very small $\epsilon > 0$ only on the zeros in the diagonal of a covariance matrix, is a procedure not present in the literature, or at least we could not find anything backing this regularization.

Another alternative would be to first approximate the covariance matrices by their diagonal (Tibshirani et al., 2003; Dudoit et al., 2002) and then regularize

[4]https://github.com/paramveerdhillon/swell

them by adding $\epsilon\mathbf{I}$, with $\epsilon > 0$ and $\mathbf{I}$ the identity matrix (Goodfellow et al., 2016). Thus, the regularized matrix is guaranteed to be invertible and sparse. One could also use Moore–Penrose's pseudo-inverse (Penrose, 1955) to invert the covariance matrices, as this method does not require the matrices to be approximated to their diagonals. However, applying the pseudo-inverse does not guarantee that the inverse matrix would be sparse. Moreover, in the case of a large corpus, if the matrices are not sparse, the products from Equations (4.2) and (4.3) would yield a memory overflow error. Thus, we can abandon the Moore–Penrose's pseudo-inverse for computing the inverses.

Let us take a look at the sparseness percentage of $\mathbf{\Sigma}_{cc}$, $\mathbf{\Sigma}_{ll}$ and $\mathbf{\Sigma}_{rr}$. These matrices are composed entirely of zeros and positive integers. We will look into the percentage of non-zero entries for the whole matrix, inside the diagonal and outside the diagonal. For this purpose, we shall take a corpus larger than our trusty toy sentence. Instead, we are using a sample input file[5] from SWELL, the directory containing the original codes for Eigenwords algorithms. This corpus contains 419 tokens and 219 word types. The following Table 4.2 shows the sparseness percentage for the sparse covariance matrices of interest.

Matrix	%non-zeros globally	%non-zeros on diag	%non-zeros outside diag
$\mathbf{\Sigma}_{cc}$	0.66	94.98	0.55
$\mathbf{\Sigma}_{ll}$	0.60	96.80	0.38
$\mathbf{\Sigma}_{rr}$	0.59	93.15	0.38

Table 4.2: Table showing the percentage of non-zeros entries for the matrices $\mathbf{\Sigma}_{cc}$, $\mathbf{\Sigma}_{ll}$ and $\mathbf{\Sigma}_{rr}$.

From this table, we can note that the covariance matrices are very sparse, not even 1% of the values being positive integers. The diagonal part of these matrices, on the other hand, are not sparse. They hold a percentage of non-zero values ranging from 93% to 97%. Although most of the diagonal is comprised of positive integers, it does not contain half of the positive integers of the whole matrix. Indeed, $\mathbf{\Sigma}_{cc}$, has a percentage of 0.66% of non-zero entries, with 0.55% non-zero entries outside its diagonal. If we approximate $\mathbf{\Sigma}_{cc}$ by its diagonal, we will lose those 0.55% of positive entries, and thus lose that much information from the original matrix. The same can be said for $\mathbf{\Sigma}_{ll}$ and $\mathbf{\Sigma}_{rr}$. Therefore, we must ask ourselves if one can achieve decent results for OSCCA and TSCCA, by applying the diagonal approximation to the covariance matrices, at the cost of discarding more than half of the non-zero entries.

[5]https://github.com/paramveerdhillon/swell/blob/master/Input_Files/1.txt

To answer this question, it has been shown that approximating the covariance matrix to a diagonal matrix can provide good results (Tibshirani et al., 2003; Dudoit et al., 2002). Additionally, this method guarantees a sparse square root of the covariance matrices, which is crucial for the computation of OSCCA and TSCCA, and is therefore more time and memory efficient. This is especially invaluable in the case of a large input corpus.

Therefore, in this paper, the sparse covariance matrices $\mathbf{\Sigma}_{cc}$, $\mathbf{\Sigma}_{ll}$ and $\mathbf{\Sigma}_{rr}$ are all replaced by their diagonal values, plus the matrix $\epsilon\mathbf{I}$ in order to regularize them and ensure their positive definiteness [6]. Thus, the triple products of Equations (4.2) and (4.3), can be estimated efficiently without significant loss of information before applying SVD.

4.5.2 Matrix $\mathbf{\Sigma}_{ss}$

Similarly, a second issue comes in the form of computing the inverse square root of the matrix $\mathbf{\Sigma}_{ss}$ in Equation (4.4),

$$\mathbf{\Sigma}_{ww}^{-\frac{1}{2}}\mathbf{\Sigma}_{ws}\mathbf{\Sigma}_{ss}^{-\frac{1}{2}} = \mathbf{U}_2\mathbf{\Lambda}_2\mathbf{V}_2^t.$$

The main difference between this covariance matrix and the ones seen in the previous section, is the fact that $\mathbf{\Sigma}_{ss}$ is a matrix of size $2k \times 2k$. The size of this matrix is what changes the presented issue, as it is comparatively small. Indeed, the following constraint for TSCCA from Section 4.3, $k \ll r \leq v$, lets us choose k arbitrarily as long as it is smaller than the rank of the matrix in (4.4), which is also smaller than the size of the vocabulary. According to Dhillon et al. (2015), for a large corpus, even $k = 200$ is reasonable in order to obtain great results.

One would think that applying the simple regularization process $(\mathbf{\Sigma}_{ss} + \epsilon\mathbf{I})$, with a small ϵ would be enough, since it should result in a positive definite matrix. We could therefore simply compute the inverse square root without going through any further approximations, as this matrix is fairly small compared to the other sparse covariance matrices. Unfortunately, it is not that evident. Let us explain why this matrix is problematic. First, the matrix $\mathbf{\Sigma}_{ss} = \mathbf{S}^t\mathbf{S}$, is positive semi-definite according to Observation 4.5. In fact, we regularize it with the following operation $\tilde{\mathbf{\Sigma}}_{ss} = (\mathbf{\Sigma}_{ss} + \epsilon\mathbf{I})$. Therefore, the regularized form is a full rank matrix on top of being positive semi-definite, and according to Observation 4.5, the regularized form

[6] We prefer this method over Dhillon's choice of adding a small $\epsilon > 0$ only on the zeros on the diagonal, as there are definite references in the literature to this solution. The numerical results are about the same for the OSCCA and TSCCA algorithms.

becomes positive definite. Thus, we can invert it. The inverse of a positive definite matrix is also positive definite:

Proof. If $\tilde{\mathbf{\Sigma}}_{ss} = \mathbf{\Sigma}_{ss} + \epsilon\mathbf{I}$ is positive definite, then $\tilde{\mathbf{\Sigma}}_{ss}$ is invertible. We shall define $y = \tilde{\mathbf{\Sigma}}_{ss}x$, for $x \in \mathbb{R}^{2k}\backslash\{0\}$, and since $\tilde{\mathbf{\Sigma}}_{ss}$ is symmetric,

$$\begin{aligned} y^t\tilde{\mathbf{\Sigma}}_{ss}^{-1}y &= x^t\tilde{\mathbf{\Sigma}}_{ss}^t\tilde{\mathbf{\Sigma}}_{ss}^{-1}\tilde{\mathbf{\Sigma}}_{ss}x \\ &= x^t\tilde{\mathbf{\Sigma}}_{ss}x > 0, \end{aligned}$$

which is the implication for a positive definite matrix. □

So, theoretically $\tilde{\mathbf{\Sigma}}_{ss}^{-1}$ is positive definite, and we can apply Theorem 3.14 to compute its square root. However, this is not the case. Let us use an example to illustrate the problem at hand.

Example 4.6. Let us take the same corpus utilized in Section 4.5.1 from the SWELL directory[7]. It contains 419 tokens and 219 word types, and we construct $\mathbf{\Sigma}ss$ from the first step of TSCCA. Let us dissect the obtained matrix to demonstrate the shortcomings of the TSCCA algorithm.
First, by using the numpy library[8], we computed the rank and eigenvalues of $\tilde{\mathbf{\Sigma}}_{ss}$. We obtained that while it is full rank, it has positive and negative eigenvalues, thus it is not positive definite nor positive semi-definite. This contradicts the theory formulated above. Therefore, we also computed the rank and eigenvalues with SciPy's library and by using the Matlab software, however, the results were still the same. By exploring the entries of $\tilde{\mathbf{\Sigma}}_{ss}$, we discovered that it is a matrix formed by 4 block diagonal matrices, where the values outside the diagonal of a block matrix are infinitesimally close to zero.

In fact, when we tried to compute the full inverse square root of $\tilde{\mathbf{\Sigma}}_{ss}$, we obtained a complex-valued matrix, which cannot be used for Eigenwords algorithms. Indeed, by definition of word embeddings, they are real valued. We explored a number of techniques to compute a real-valued approximation of the inverse square root of $\mathbf{\Sigma}_{ss}$ or $\tilde{\mathbf{\Sigma}}_{ss}$, but none of them, not even Moore-Penrose pseudo-inverse or any SVD decomposition, could solve this issue.

Therefore, let us take a step back and try to develop the covariance matrix $\mathbf{\Sigma}_{ss}$, using the matrix $\mathbf{S} = [\mathbf{L}\mathbf{\Phi}_l, \mathbf{R}\mathbf{\Phi}_r]$. We are studying this matrix to gain insight into

[7] https://github.com/paramveerdhillon/swell/blob/master/Input$_{F}iles/1.txt$

[8] https://numpy.org/doc/stable/

why it is not positive definite. Hence, by using the definitions of Section 4.3, with $\mathbf{\Phi}_l = \mathbf{\Sigma}_{ll}^{-\frac{1}{2}}\mathbf{U}_1$, and $\mathbf{\Phi}_r = \mathbf{\Sigma}_{rr}^{-\frac{1}{2}}\mathbf{V}_1$, and by plugging in Equation (4.3), we have that

$$\begin{aligned}
\mathbf{\Sigma}_{ss} &= \mathbf{S}^t\mathbf{S} \\
&= [\mathbf{L}\mathbf{\Phi}_l, \mathbf{R}\mathbf{\Phi}_r]^t[\mathbf{L}\mathbf{\Phi}_l, \mathbf{R}\mathbf{\Phi}_r] \\
&= \begin{pmatrix} \mathbf{\Phi}_l^t\mathbf{L}^t\mathbf{L}\mathbf{\Phi}_l & \mathbf{\Phi}_l^t\mathbf{L}^t\mathbf{R}\mathbf{\Phi}_r \\ \mathbf{\Phi}_r^t\mathbf{R}^t\mathbf{L}\mathbf{\Phi}_l & \mathbf{\Phi}_r^t\mathbf{R}^t\mathbf{R}\mathbf{\Phi}_r \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{U}_1^t(\mathbf{\Sigma}_{ll}^{-\frac{1}{2}})^t\mathbf{\Sigma}_{ll}\mathbf{\Sigma}_{ll}^{-\frac{1}{2}}\mathbf{U}_1 & \mathbf{U}_1^t(\mathbf{\Sigma}_{ll}^{-\frac{1}{2}})^t\mathbf{\Sigma}_{lr}\mathbf{\Sigma}_{rr}^{-\frac{1}{2}}\mathbf{V}_1 \\ \mathbf{V}_1^t(\mathbf{\Sigma}_{rr}^{-\frac{1}{2}})^t\mathbf{\Sigma}_{rl}\mathbf{\Sigma}_{ll}^{-\frac{1}{2}}\mathbf{U}_1 & \mathbf{V}_1^t(\mathbf{\Sigma}_{rr}^{-\frac{1}{2}})^t\mathbf{\Sigma}_{rr}\mathbf{\Sigma}_{rr}^{-\frac{1}{2}}\mathbf{V}_1 \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{I}_{k\times k} & \mathbf{\Lambda}_1 \\ \mathbf{\Lambda}_1 & \mathbf{I}_{k\times k} \end{pmatrix},
\end{aligned}$$

with the orthogonal matrices $\mathbf{U}_1$ and $\mathbf{V}_1$, which are singular vectors, and $\mathbf{\Lambda}_1$ the matrix of singular values of Equation (4.3).

By comparing the matrix $\mathbf{\Sigma}_{ss}$ from the theoretical development above to the one in Example 4.6, both matrices match. Indeed, in Example 4.6 the diagonal values of $\mathbf{\Sigma}_{ss}$ are close to 1, while the values outside of the diagonal of the submatrix $\mathbf{I}_{k\times k}$ are infinitesimally close to 0. In practice, the exact identity matrix is not obtained due to the applied RSVD approximation. As for the matrix $\mathbf{\Lambda}_1$, it also corresponds to the results obtained in the same example, with values outside the diagonal infinitesimally close to 0, due to RSVD. In fact, we obtained for this matrix in the example, that the values on the diagonal range from 1 to 2.7, such that they are decaying:

$$\mathbf{\Lambda}_1 = \text{diag}(\sigma_1, ..., \sigma_k),$$

with $\sigma_1 \geq ... \geq \sigma_k \geq 0$, the singular values.

Let us define the Schur complement and the condition for positive definiteness, (Horn and Zhang, 2005).

Theorem 4.7. *Let* $\mathbf{X}$ *be a real-valued symmetric matrix given by*

$$\mathbf{X} = \begin{pmatrix} \mathbf{C} & \mathbf{D} \\ \mathbf{D}^t & \mathbf{E} \end{pmatrix}$$

in which $\mathbf{C}$ *is square and invertible. We have that* $\mathbf{X}$ *is positive definite if and only if both* $\mathbf{C}$ *is positive definite, and the Schur complement of the block* $\mathbf{C}$ *of the matrix* $\mathbf{X}$*, defined as* $\mathbf{X}/\mathbf{C} = \mathbf{E} - \mathbf{D}^t\mathbf{C}^{-1}\mathbf{D}$ *is positive definite.*

Thus, let us apply Theorem 4.7 on matrix $\mathbf{\Sigma}_{ss}$. We have that $\mathbf{I}_{k\times k}$ is positive definite since it is the identity matrix, however, the Schur complement

$$\begin{aligned}\mathbf{X}/\mathbf{I}_{k\times k} &= \mathbf{I}_{k\times k} - \mathbf{\Lambda}_1^t \mathbf{I}_{k\times k}^{-1} \mathbf{\Lambda}_1 \\ &= \mathbf{I}_{k\times k} - \mathbf{\Lambda}_1^2,\end{aligned}$$

is a diagonal matrix with only negative entries due to the fact that the singular values are larger than 1. Hence it only has negative eigenvalues and the matrix is not positive definite. Therefore, by Theorem 4.7, $\mathbf{\Sigma}_{ss}$ is not positive definite. In fact, if we applied the regularization $\mathbf{\Sigma}_{ss} + \epsilon\mathbf{I}$ as in Section 4.5, then by Theorem 4.7, we would need that $\epsilon \geq \sigma_1 - 1$, with σ_1 the largest singular value of $\mathbf{\Lambda}_1$. Such a condition would result in fixing an ϵ that can be large depending on the largest singular value of matrix $\mathbf{\Lambda}_1$. Alas, the results obtained with such a regularization method were not up to standard and inconsistent.

Unfortunately, it is still unknown why this matrix, which is supposedly positive semi-definite following Observation 4.5, and is of full rank, fails to be positive definite. Hence, we can only conclude that the matrix $\mathbf{\Sigma}_{ss}$ is not a covariance matrix since by definition a covariance matrix is positive semi-definite, unless the condition $\epsilon \geq \sigma_1 - 1$ is fulfilled, in that case $\mathbf{\Sigma}_{ss}$ becomes positive semi-definite. Instead of regularizing this matrix with a large ϵ, we decided to find an approximation that yielded satisfying results.

On the SWELL repository, Dhillon would compute $\mathbf{\Sigma}_{ss}^{-\frac{1}{2}}$, in this manner: first apply deterministic SVD on $\mathbf{\Sigma}_{ss}$. It yields,

$$\mathbf{\Sigma}_{ss} = \mathbf{U}_{ss}\mathbf{\Lambda}_{ss}\mathbf{V}_{ss}^t. \tag{4.5}$$

Then, he forms $\mathbf{\Lambda}_{ss}^{-\frac{1}{2}}$, the inverse square root of the diagonal matrix $\mathbf{\Lambda}_{ss}$. In order to do so, we just need to inverse and apply the square root on each element of $\mathbf{\Lambda}_{ss}$. Afterward, he would compute $\mathbf{\Sigma}_{ss}^{-\frac{1}{2}}$ as such,

$$\mathbf{\Sigma}_{ss}^{-\frac{1}{2}} \approx \mathbf{U}_{ss}\mathbf{\Lambda}_{ss}^{-\frac{1}{2}}\mathbf{V}_{ss}^t. \tag{4.6}$$

Alas, after multiple attempts at verifying the validity of this process and extensively searching in the literature, we could not find anything justifying this calculation. As we could not find any alternatives to compute an inverse square root of $\mathbf{\Sigma}_{ss}$ that would yield a real-valued matrix and satisfying results, we decided to settle for Dhillon's approximation from the SWELL repository in order to compute TSCCA.

Chapter 5

Competing Word Embedding Models

With regard to having other state-of-the-art models to compete against on word similarity task, we will briefly introduce a popular word embedding model, Word2vec and its different versions.

5.1 Word2vec

Word2vec (Mikolov et al., 2013a) is a shallow neural network, composed of two layers. It takes a large corpus as input and produce vectors corresponding to each word type. There are two varieties of Architectures within Word2vec, the Continuous Bag-of-Words model (CBOW) and the Skip-gram model.

One of its main strengths is its time efficiency and its simple implementation through Python. Since it is a shallow neural network it is fast, unlike deep learning models. Let us briefly explore Word2vec from the point of view of Mikolov et al. (2013a). One major difference to Eigenwords, is that Word2vec uses only local context information of the words in order to produce embeddings. However, Eigenwords algorithms incorporate global word occurrences from the **W** matrix in addition to the local context of words in the form of either the **L** and **R** or the **C** matrix.

5.1.1 CBOW

The goal of CBOW is to predict a target word from the other words in a sentence. For example, let the input be this sentence "Hello world is my program", if the

target word is "is", the context words would be
["Hello","world","my","program"], for a window size $h = 2$ (similar to OSCCA and TSCCA). The model would take this list of context as input and will try to predict the target word.

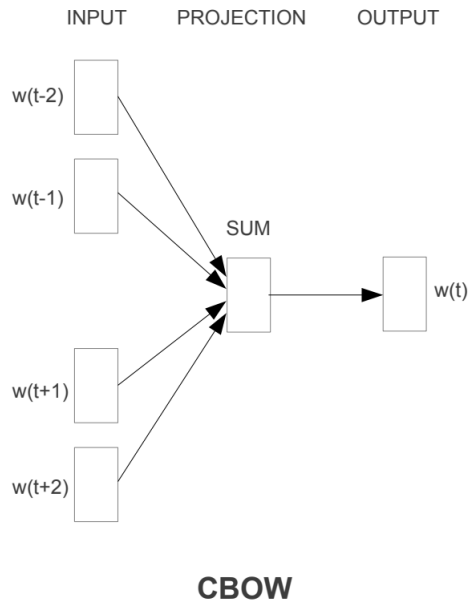


Figure 5.1: Image of the CBOW Architecture taken from (Mikolov et al., 2013a)

To be more specific, the model is composed of 3 layers, the input, the projection and the output layers. At the input layer, the $h = 2$ words in the context of the target word, in this case "is", are transformed into one-hot vectors. With each vector having the length v of the vocabulary. These vectors are filled with zeros except at the index corresponding to the represented context word, with a value of 1. For example, the word "world", would be represented by $(0, 1, 0, 0, 0)$. Afterward, the input layer, containing the $2h$ words of context, of dimension $2h \times v$ is projected into the projection layer of dimension $2h \times D$ using the shared projection matrix of dimension $v \times D$. It should be noted that the dimension D corresponds to the target rank k for Eigenwords algorithms. Additionally, it is evident from the architecture image that all words share the projection layer. All of the words would then be projected into the same position as a result. This means that the projection is unaffected by the context-specific order of words, and the model is subsequently referred to as Continuous Bag-of-Words.

The rows of the shared projection matrix of dimension $v \times D$, which are known as weights in neural networks, are actually the word embeddings. Indeed, the

output layer is a classifier function receiving vectors from the projection layer and it outputs probabilities for target words from the vocabulary (probability distribution). During the training, the goal of the architecture is to adjust those weights such that the predicted probabilities of the target words correspond to the actual words. In other words, it tries to minimize the loss function for this classification task. The CBOW model is a straightforward neural network with a single hidden layer. The model's objective is to obtain the weights, which are the word embeddings, after it has been trained to estimate the probability of a target word from a list of context words.

5.1.2 Skip-gram

Similarly to CBOW, the Skip-gram model is a simple neural network with one hidden layer. In essence, the Skip-gram model architecture contrasts with the CBOW model. Indeed, Skip-gram predicts surrounding context words from the target words, whereas CBOW predicts the target words from its contexts.

As an illustration, we shall use this same sentence as the example, "Hello world is my program". Now, let the input to our model be the word "is" in the middle, with a window size of $h = 2$, it will pair the target word with the word in its context in the rolling window size, ("is", "world"), ("is", "Hello"), and so on. Essentially, using these pairwise words as training data, the model will be trained to predict the probability of a word being the context of the given target word, "is". This difference in the architecture, would mean that each pair of target-context word is treated as an observation, in the case of a large corpus, statistically, Skip-gram performs better. The computational complexity is however also increased.

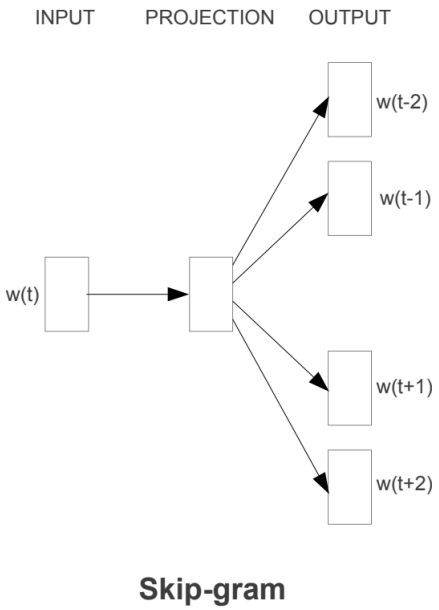


Figure 5.2: Image of the Skip-gram Architecture taken from (Mikolov et al., 2013a)

More specifically, much like CBOW, it has 3 layers, the input, the projection and the output. Here, the input layer receives only the target word in the form of a one-hot vector. This vector is of length v, the size of the vocabulary. Thereafter, the input vector of size $1 \times v$ is projected into the second layer of dimension $1 \times D$ by the projection matrix of dimension $v \times D$.

In the same fashion, the projection matrix, known as the weights, are the actual word embeddings. The output layer is just a classifier function that receives the vectors from the projection layer. It yields the h probabilities for the context words, before and after the given target word ($2h$). The training is done by minimizing the summed prediction error across all the $2h$ context words. Doing so, it will adjust the weights of the Skip-gram model.

Chapter 6

Web Scraping Daily Mail News Articles

In this thesis, we will work with real world text data, namely, webscraped news articles about the war between Russia and Ukraine. Throughout this chapter, we will display choices made. We will justify the reason for choosing this dataset, and subsequently explain the techniques employed to obtain it. Finally, we will describe the cleaning and preprocessing of the data before applying any word embedding algorithms.

6.1 Data

As a reminder, the goal of any word embedding model is to generate real valued vectors, that retain as well as possible the meaning of the words. In other terms, words that share semantic similarities should also be close to each other in the vector space. In order to harness those word vectors, we need to provide an adequate corpus of text data to the model. To do so, we decided to work with articles from Daily Mail, a highly-circulated British daily newspaper founded in 1896. There are several reasons for selecting this news outlet in particular. First, we decided to focus on creating embeddings for the English language. Thus, the Daily Mail, being a renowned English newspaper, becomes a natural choice. Secondly, the Daily Mail hosts a website[1] from which we can webscrape a large amount of articles.

Thus, the articles from the Daily Mail's website appear to be more than adequate to be the input data for the embedding models. Our choice of topic being the war in Ukraine, the scraped articles belong to the section about this war. Following multiple conflicts between Russia and Ukraine in the past decade, war broke out as

[1] https://www.dailymail.co.uk/home/index.html

the Russian military invaded Ukrainian grounds on the 24th of February 2022. This ongoing war has many socio-economical implications in the post-COVID world. One point of interest of this chapter would be to recover the embeddings built upon these articles. As the embedding models learn from the semantic meaning of a corpus, the choice of vocabulary and the standpoints of the authors will impact the word vectors.

6.1.1 Scraping

Web scraping is the process of retrieving data from websites. In order to give context, the content shown by a webpage when one inserts an URL into the web browser can be displayed using multiple technologies. The one we are interested in is HyperText Markup Language (HTML), the language used to create and organise the contents of a website with tags, each of these tags representing a different function within the website. Therefore, to scrape the relevant data from a website, one needs to inspect the HTML source code, and extract it. The challenge starts here, since extracting the isolated text data that is relevant to us is not a straightforward task, as most webpages are made of content besides plain text, mainly visual and structural elements, such as a navigation bar, advertisements, videos, headers, just to name a few. These structural elements are in general created with repetitive, mostly invariant boilerplate code. Thus, the process becomes incredibly complex when one tries to discern the relevant HTML tags containing text from the articles from the abundant structural boilerplate[2]. Not to mention that a lot of websites undergo periodic structure changes, as to create a more user-friendly experience, in the hopes of expanding their userbase (i.e. returning clients). Unfortunately, this process only makes it harder for the scraper to retrieve the desired data. Some webpages even contain Captcha verifications to differentiate humans and bots. We wondered if there was an easier way to obtain the text excerpts we want, without having to work through all this.

Fortunately for us, the Python package Newspaper[3] is the solution to our problem. Newspaper uses advanced algorithms to extract the relevant text information given an URL link as the input. It was built for the specific job of scraping online news outlet. As a consequence of this, we are not required to go through the hurdle of inspecting the HTML source code of the Daily Mail website ourselves.

As for the Daily Mail website, and more specifically within the section on the war between Russia and Ukraine, several available links to different articles related

[2]https://www.oreilly.com/library/view/mining-the-social/9781449368180/ch05.html
[3]https://newspaper.readthedocs.io/en/latest/

to this war can be found. Using Newspaper, one could extract the text information of each article by giving their URLs one at a time to the program. But such a process is too arduous.

A new question arises: how does one retrieve multiple article's URL links at once, so that the process of scraping would be reduced to looping through a list of those links? The answer to this problem is called web crawling, which consists of systematically browsing and retrieving the relevant information from a webpage. Here, the wanted information would be the 50 different article links on the Russo-Ukranian war section. To be brief, we used three libraries in order to build the web crawler: Requests[4], Urllib[5] and BeautifulSoup[6]. The web crawler fetches only the first 50 relevant links corresponding to articles about the war. By combining Newspaper package and the web crawler, we can scrape plain text from 50 articles at a time without encountering any errors. Any more than that, and the webpage considers the crawler a bot, which leads us to be blocked. The web scraped articles are concatenated and converted to a JSON file containing four variables before preprocessing:

- url: String, unique link to the article;
- date: Datetime, object representing the time of the publication;
- test: String, content of the article;
- author: String, the name of the author(s) of the article.

We chose to keep only these four variables for the purpose of removing duplicates and organizing the scraped data.

We managed to extract 2019 articles published between the 15-th February 2022 and the 16-th of April 2022. The concatenated text corpus, before any preprocessing, contains $4,194,348$ tokens and $55,514$ word types.

One thing worth mentioning is that, by limiting ourselves to articles about the war, we also limit the diversity of the used vocabulary. In fact, since we are scraping a dataset that is focused on a single topic rather than one that was created by scraping every article on the Daily Mail, regardless of topic, we may anticipate seeing fewer word types in the first case for the same number of total tokens scraped. The scraping of topic centered data can break the embedding

[4]https://requests.readthedocs.io/en/latest/
[5]https://docs.python.org/3/library/urllib.html
[6]https://beautiful-soup-4.readthedocs.io/en/latest/

model. More specifically, the model can over-fit on some words that are only used in very specific contexts. When using a topicless dataset as an input, one can expect the model to behave in the opposite fashion: with a limited amount of observations for each word, because of the richer diversity of the corpus and the fewer occurrences of certain words, the model may suffer from not being able to correctly learn the meaning of those words. However, if the models produce up to standard results during training on a topic-centered dataset, we can conclude that such models have a high sample complexity for rare words.

6.1.2 General Preprocessing

Text preprocessing is usually one of the first steps in NLP systems, and it is also one of utmost importance. If a corpus is not properly processed, one can easily obtain poor results. Despite its importance, text preprocessing for NLP related tasks and word embeddings has not received much attention in the literature. In a lot of instances it is overlooked (Dhillon et al., 2015) or barely touched upon (Yang et al., 2015).

Each downstream NLP task favors a different preprocessing approach. For cases such as sentiment analysis and emotion classification tests, Camacho-Collados and Pilehvar (2018) affirm in their conclusion that: "In general, a simple tokenization works equally or better than more complex preprocessing techniques such as lemmatization or multiword grouping". As for other tests, such as Part-of-Speech tagging or POS tagging, and Word similarity, a more complex preprocessing technique yields better results. Therefore, the choice of this step is crucial, as much of the performance gains of word embeddings are due to the transformed input data themselves. In a way, preprocessing can be considered as a parameter optimization step to our word embedding models. The existing literature does not specify a general rule to determine the best method for each situation. Most of the previously conducted research on this matter make use of different combinations of preprocessing steps, but very few discuss their impact on the results.

Therefore, we will perform two types of preprocessing, one where we will apply heavy preprocessing techniques and another one where the data will receive a minimum of preprocessing. Having two types of input data will help us find which preprocessing schemes are favored for different tests. Below, Table 6.1 shows the preprocessing steps applied to the training corpus for both schemes and the order in which they are applied.

Heavy preprocessing	Light preprocessing
1. Replace URLs by a placeholder	1. Replace URLs by a placeholder
2. Lower case the corpus	2. Lower case the corpus
3. Remove contractions of words	3. Remove contractions of words
4. Replace numbers by a placeholder	4. Replace numbers by a placeholder
5. Lemmatize the corpus	5. Remove non-alphanumeric chars.
6. Remove non-alphanumeric chars.	
7. Remove the stop words	
8. Remove words smaller than 3 chars.	
9. Remove words appearing less than 10 times	
10. Remove sentences less than 2 words	

Table 6.1: Table showing the sequential preprocessing steps for two different preprocessing schemes on the Daily Mail corpus.

All of the applied preprocessing steps mentioned in Table 6.1 reduce the vocabulary size, and are frequent routines applied in different fields of NLP (Bicalho et al., 2017; Pradana and Hayati, 2019).

The most basic step incorporated into both schemes is the lowercasing of words. If not done, the models would consider the same word with different letter cases as different instances. Such a situation is disadvantageous to the model, as the number of occurrences for a word would be divided among its different versions. Other shared steps between the two schemes include the replacement of the URL links and numbers by their respective generic place holders. We included these steps for the simple reason that our goal is not to learn URLs and numbers through the word embedding models. The last two common preprocessing steps are the removal of contractions of words and non-alphanumeric characters. In english, the apostrophe is a very important punctuation character, terms such as “aren’t" and "shouldn’t" are omnipresent in every text and can break an algorithm if not handled properly. Therefore, we will remove contractions of words, by replacing the above contractions with "are not" and "should not", just to name a few. Moreover, the majority of the noise polluting the corpus are the non-alphanumeric characters, and therefore the elimination of this noise will entail a positive impact on the results.

In the more complex scheme, we also apply lemmatization. Similar to lowercasing, the goal of lemmatization is to maximize the occurrences of words by reducing different words to their canonical form. For instance, "talks", "talking", and "talked" are transformed into the form "talk".

Afterwards, we remove all the stop words, words smaller than 3 characters, and

all the rare words that appear less than 10 times throughout the entire corpus. The last steps' objective is to get rid of noise and outliers. In essence, non-alphanumeric characters and stop words do not provide much information to infer content. Most of the time, they do not carry any meaning.

The heavy preprocessing's goal is to remove all noise in the corpus, such that only the key elements are left for our program to learn from. Such a drastic cleaning can be theoretically beneficial in learning word similarity, as most noise is removed. This process lets us reduce the overall size of the vocabulary, and increases the occurrences of the now reduced vocabulary. Doing so, the embedding model will receive more sample data for each word type.

On the other hand, applying only a simpler routine allows the corpus to retain a larger amount of words that appear less frequently. How the embedding models perform under such a corpus with a larger amount of noise and outliers can be interesting.

Finally, the preprocessed texts are tokenized using nltk's tokenizer [7]. The tokens are then used in Chapter 7.1 to build the different word matrices needed as input for the Eigenwords algorithms. As for Word2vec, it uses the tokens themselves as input to build the word embeddings.

The complex preprocessing leaves us with a corpus of $2,134,423$ tokens with $11,197$ vocabulary words and the light preprocessing yields a corpus of $3,821,794$ tokens with $32,354$ vocabulary words.

6.1.3 Description

On the topic of describing our data, we assembled Table 6.2 containing the 15 most used words and their frequency in the processed corpus.

As expected from the heavily preprocessed text, the most common words are related to the Russo-Ukrainian War and the number placeholder since every number, date, and statistic in the text about the war has been replaced by a common place holder. As for the lightly preprocessed text, words such as "the" and other stop words are the most frequent. This outcome was expected given that they are the most common words in the English language.

In order to illustrate the distribution of words within the corpus more accurately,

[7] https://www.nltk.org/api/nltk.tokenize.html

Heavy Preprocessing		Light Preprocessing	
Word	**Frequency (%)**	**Word**	**Frequency (%)**
numberPlaceHolder	2.72	the	5.66
ukraine	1.72	to	2.95
russian	1.65	of	2.74
said	1.36	in	2.32
russia	1.24	and	2.27
ukrainian	0.85	a	2.21
putin	0.80	numberPlaceHolder	1.52
war	0.54	on	1.15
force	0.54	s	1.03
city	0.49	ukraine	0.96
people	0.47	is	0.95
kyiv	0.46	that	0.95
country	0.44	russian	0.86
president	0.43	said	0.76
also	0.43	for	0.73

Table 6.2: Frequency table of the 15 most used words in the heavy and light preprocessed corpus.

we plot in Figure 6.1 the empirical frequencies of the words used in the corpus against Zipf's model of the words frequencies using Equation 2.1. On the x-axis the words' ranks are laid out, ordered from the most used to the least used, and on the y-axis we represent the counts of these words. Both axes are plotted on a logarithmic scale. We have side by side the heavy and light preprocessed text data.

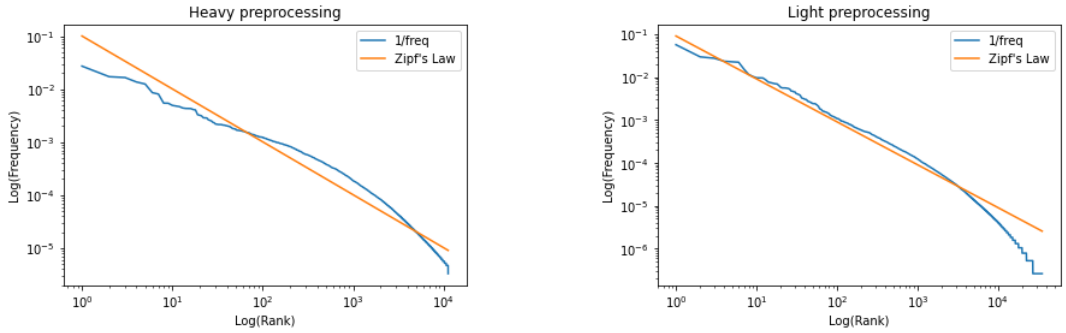


Figure 6.1: Logarithmic plots of the words' ranks against their frequencies from the heavy preprocessed corpus on the left and light preprocessed corpus on the right.

One interesting fact that can be deduced from the graphs is that both distributions follow the tendency of Zipf's law, although the most frequent and least frequent words tend to deviate from it. Most importantly, this is one of the conditions to easily solve the Eigenwords algorithms' constraints. Indeed, in Equation 4.1, one of the requirements to obtain the simplified form of the covariance matrices, is that the frequency of the words in a corpus needs to follow a Zipfian distribution.

To be more general, the frequency at which words appear and its relation with Zipf's law are an interesting topic on its own. It has been shown that most languages known to mankind follow Zipf's law (Manaris et al., 2006; Piantadosi, 2014). Furthermore, it has been shown that even if words are randomly drawn, it will not fail to follow Zipf's law either (Newman, 2005; Li, 1992). However, texts crafted by humans are far from being drawn randomly, they carry meaning, and in the case of this thesis revolve around a certain topic, and are thus not the result of a random draw. This subject, while it is interesting, is out of the scope of this thesis.

Chapter 7

Application on Daily Mail News Articles and Evaluation

7.1 Experimental Environment

Before taking any further steps into testing the models, we will summarize the previously detailed models and their parameters. We will also briefly talk about the practical development process of these models and some of the difficulties encountered. Next, we will delve deeper into different evaluation methods theoretically and their corresponding metrics. We will then fine-tune some parameters of the Eigenwords model using the most suited evaluation measure. Thereafter, we will evaluate Eigenwords against Word2vec algorithms. Lastly, the model that performs the best of these two is subjected to a qualitative evaluation.

7.1.1 Models and Parameters

As a quick reminder, both OSCCA and TSCCA entail the application of CCA on a pair of matrices that capture words' global count and local context information. This algorithm returns lower dimensional projections of both matrices, aiming to maximize the correlation between their canonical variables.
On the other hand, Word2vec's CBOW and SkipGram are shallow neural networks that rely solely on the local contexts of words as their single source of information.

Both of the highlighted approaches to word embedding share common parameters: the window size h and the hidden state size r. However, we must first fine-tune the parameters specific to Eigenwords, the oversampling and the power iteration parameters, respectively, p and q, before optimizing both programs on the shared parameters. As a reminder, these special parameters, used in OSCCA

and TSCCA, originate from the underlying implementation of RSVD.

7.1.2 Programming and Hardware Settings

The device hardware configurations affect how well a software operates, so we thought it was vital to mention that all the computations were done on a personal PC. The computer is running a 64-bit, 21H1 version of Windows 10 Home. The machine has 16 GB of RAM memory, and an AMD Ryzen 5 3600 6-Core Processor running at 3.60 GHz.

As part of this thesis, the code is based on "Eigenwords: Spectral Word Embeddings" (Dhillon et al., 2015) as the main reference. First, in order to recreate some results showed in the paper, we tried to install from Dhillon's github, the Spectral-Learning Toolkit, SWELL[1], the repository containing the Java code reproducing OSCCA, TSCCA and many more word embedding algorithms.

Before building SWELL, a set of dependencies such as jeigen[2] needed to be recursively built. However, when attempting this, errors related to version mismatch arose. First, as advised by Dhillon, we downgraded the Java version to the one both libraries were developed on, in an attempt to resolve the errors. Unfortunately this attempt was in vain, and we discovered that the problem resided in the incompatibility between SWELL and jeigen. In other words, SWELL was developed in 2014 to use a certain version of jeigen, but this library has received multiple updates after the maintenance of SWELL had been discontinued.

The main appeal to use jeigen to develop OSCCA and TSCCA on Java is that it offers matrix multiplication methods for dense-dense, sparse-dense, and sparse-sparse pairs of matrices. Indeed, the RSVD formulation of OSCCA and TSCCA requires to compute the multiplication of large dense and sparse matrices, namely the ones found in Equations (4.2), (4.3), and (4.4). Thus, in order to work with Eigenwords, we need those tools provided by jeigen to efficiently compute multiplications of different pairs of matrices.

Therefore, for the sake of productivity and to have a better understanding of the process of the word embedding techniques using CCA, we decided to recreate OSCCA and TSCCA on Python, instead of trying to repair SWELL and jeigen. Being flexible and having a simple syntax, Python is the most popular programming language. In our case, we used Anaconda's distribution of Python

[1] https://github.com/paramveerdhillon/swell
[2] https://github.com/hughperkins/jeigen

(version 3.8.8) through the Spyder IDE (version 5.1.5). Our choice was motivated by the fact that Anaconda's package dependency management system is robust and flexible. Furthermore, it contains multiple useful data management and processing modules such as SciPy's package for 2-D matrices[3], making the process of handling sparse-dense matrices much easier, the same way jeigen does for Java. To top it off, the vast library provided by Python is very appealing for a data scientist enthusiast. As a matter of fact, Python offers a dense variety of machine learning tools through many modules. Among those modules, we also used the Natural Language Toolkit (NLTK), in order to perform tokenization on the text corpus. Finally, in order to compute randomized SVD, the Scikit-Learn package[4] contains an implementation of the RSVD (RRF and RPI) based on the paper of Halko et al. (2011).

One thing worth discussing, is that for both OSCCA and TSCCA, Dhillon et al. (2015) briefly mention taking the square roots of the occurrence counts in the covariance matrices that carry word occurrence counts (i.e $\mathbf{\Sigma}_{ww}$, $\mathbf{\Sigma}_{wc}$, $\mathbf{\Sigma}_{cc}$, $\mathbf{\Sigma}_{ll}$, $\mathbf{\Sigma}_{lr}$, $\mathbf{\Sigma}_{rr}$ and so on). This transformation of the data, is not implemented in the source code of SWELL. However, OSCCA and TSCCA show very unstable and "bad" results when this transformation is not applied. In essence, since words follow a Zipfian distribution, by applying a square root on the counts of words, we can flatten the heavy tailed distribution of the words (Dhillon et al., 2015). Doing so, we also stabilize the variation of the entries in the matrices defined in Equation 4.2 and 4.3. As a result, the constructed embeddings are more stable (Stratos et al., 2015).

As for Word2vec, both architectures can be easily employed by writing a few lines of code using Gensim's module[5].

Finally, the implementation of the preprocessing steps described in Section 6.1.2 was challenging. For this we used Regular Expressions or regex. To put it simply, a regex is a group of characters that define a pattern using wildcards and other special characters such as "*" and "|". When applying it on a text, it can evaluate into multiple substrings (groups of characters) found within the text. Therefore, we used regex expressions to find patterns in the corpus to replace or remove entirely[6].

[3]https://docs.scipy.org/doc/scipy/reference/sparse.html
[4]https://scikit-learn.org/stable/index.html
[5]https://radimrehurek.com/gensim/models/word2vec.html
[6]https://regexr.com/

7.1.3 Evaluation Methods

First of all, the goal of this Chapter is to find the word embedding model that performs the best when applied to news articles. In order to do so, we test these models on multiple evaluation methods, which we will introduce in this section. Additionally, we fine-tune their parameters to bring them to their highest potential.

The methods to evaluate word vectors can be divided into two categories, intrinsic and extrinsic evaluations.

Intrinsic evaluations are methods that compare human judgments and word embeddings on the relation between words. The human judgments are manually created datasets based on human assessments of words, which can be collected from a small number of people evaluated in a controlled environment (Bakarov, 2018). The main appeal of intrinsic evaluation is that it is easy to compare models against each other and fast to compute (i.e. word similarity task).

On the other hand, extrinsic evaluation methods are based on the word vectors' ability to serve as features for supervised machine learning algorithms with the aim of completing a downstream NLP task. The performance of the supervised model on a particular dataset for the NLP task is used to assess the quality of the embeddings. As the evaluation is performed on a real world task, it is often more meaningful, as we can improve an existing supervised model on achieving, for example, sentiment classification. However, the problem with embeddings trained for a specific downstream task is that they perform poorly on other assignments. This is due to the fact that these tasks differ a lot and require different features from the embeddings. Thus, a global evaluation score for word embeddings does not exist in the case of extrinsic evaluation methods.

In our case, we chose two intrinsic evaluation approaches: word similarity and word analogy. In this section we will use these two schemes to test the models and analyze them. Later on, we will also explain the reason why we did not choose any extrinsic evaluation to test our models.

Word Similarity

The concept behind the word similarity method is that the distances between words in an embedding space should be comparable to human heuristic assessments of the real semantic distances between these words (Bakarov, 2018). For this test, one takes a dataset containing human judgments of pairs of words. These pairs

of words are scored by multiple humans on their similarities and relatedness. For example, on a scale from 0 to 10, coffee and mug could be scored as a 7.

The obtained set of distances and scores for all the pair of words present in both the word embedding space and the test set are then compared. Therefore, the task's objective is to assess how effectively the word vector representations capture human perceptions of word similarity.

There are two metrics involved in this task, the cosine similarity and Spearman's correlation (Faruqui et al., 2016). The cosine similarity is used to calculate the distance between two word vector representations. If a and b are two embeddings, then computing their cosine similarity will allow you to determine how similar they are in the vector space. After computing the cosine similarity between the word vectors, we acquire lists of word pairs that have been ordered based on human and vector space similarities. Finally, we compute the Spearman's correlation, that we denote as ρ_s in this chapter, between these sorted lists. The correlation should serve as an indicator of how effectively learned word embeddings represent people's natural perceptions of word similarity.

For this purpose, we will use three different datasets designed for evaluating word semantic similarity:

- **MEN**, 3000 pairs assessed by semantic similarity and relatedness using a scale from 0 to 50 (Bruni et al., 2014).
- **SimLex-999**, 999 pairs assessed by semantic similarity using a scale from 0 to 10 (Hill et al., 2015).
- **WordSim-353**, 353 pairs assessed by semantic similarity using a scale from 0 to 10 (Finkelstein et al., 2001).

A point of concern, would be the appropriateness of these test sets to evaluate the quality of the embedding models, which were trained on articles about the Russian-Ukrainian war. These three datasets are considered gold standard (best performing test data available) for word similarity evaluations, and are not biased towards a certain topic or theme, such as war. Since the compared words appearing in the test sets are most likely infrequent words of the corpus, obtaining high scores on these datasets would indicate that the model can capture the semantics of rare words. Hence, the models scoring high on these three sets would be more appropriate to be deployed for real world tasks. In a way, we can think of these gold standard test sets as a better representation of the general population's concept of

words' semantic similarity.

One thing to note is that the different test datasets use different notions of semantic similarity, and the subjects assessing the pair of words are also different from one set to another. In the **MEN** test set, it was assessed by treating similarity and relatedness as the same concept.

In fact, the notion of similarity and relatedness are subjective notions that are frequently erroneously interchanged. As an example, in **WordSim-353**, the concept of similarity and relatedness are mistaken by some test subjects. In this set, "cup" and "coffee" are rated more similar than "car" and "train". While "cup" and "coffee" are certainly related, as most people like to drink their coffee in a cup, they are not similar terms. Therefore, using the same embeddings on different sets can result in dramatically different outcomes (Bakarov, 2018), and models that account for the distinction between similarity and relatedness could perform worse than the ones who cannot.

Word Analogy

Let us introduce the second intrinsic test, word analogy. The purpose of this method is to evaluate if simple arithmetic operations with word vectors could imitate our understanding. The method is as follows: given three words a, b and c, this task's aim is to correctly identify a word d such that the relation $a : b$ is similar to $c : d$ (Bakarov, 2018). A simple illustration is more effective than a lengthy explanation in this case. If a = "russia", b = "moscow" and c = "france", then d = "paris". In this instance, the word embeddings should reflect the relation "country : its capital". In order to achieve this operation, Mikolov et al. (2013b) introduced the following formula to find the target word d. Let the word embeddings x_a, x_b and x_c, correspond to the words a, b and c. Then, one computes the vector $y = x_b - x_a + x_c$, which represents the ideal embedding for d. However, as this vector usually does not correspond to any existing word vector, the most logical solution is to find the closest instance w^* in the vector space, i.e. the next word that has the greatest cosine similarity with y:

$$w^* = \arg\max_{w \in V} \frac{x_w y}{\|x_w\| \|y\|}, \tag{7.1}$$

with V the set representing the vocabulary of the corpus, and w^* the solution to the optimization problem. This operation is also named 3CosAdd (Mikolov et al., 2013b). This task can only be used as a qualitative assessment of the embedding models as it does not have a global metric to evaluate their performance.

Extrinsic evaluations

In this thesis, we will not perform any extrinsic evaluation methods on the embeddings. We will explain our reasoning for this choice by using Part-of-Speech or POS tagging, an extrinsic evaluation method, as an example. Other extrinsic evaluation approaches are similar to POS tagging, and they consist of using the word embeddings as features in a supervised learner to achieve a downstream task.

In POS tagging, the goal of the supervised learner is to correctly classify words from a corpus into their distinct parts of speech. The POS classifiers are trained in a supervised manner using a special labeled corpus, called a treebank, made by linguists. These treebank datasets contain sentences where each word is assigned a POS label. This is our first point of concern. The webscraped texts from news articles do not include any POS labels. Manually assigning POS tags to over 4 million words is not efficient. The only solution would be to use a parser to assign these tags to the words. Unfortunately, such a method comes with its fair share of problems. The parser can incorrectly assign tags to some words and the supervised learner will learn these incorrect tags. Other extrinsic evaluation methods would require other types of specific label data for the supervised model to learn from. For example, in sentiment classification, each sentence would need a label indicating the sentiment. Ergo, extrinsic evaluation does not seem fit for our webscraped data.

Additionally, extrinsic evaluation methods can be divided into two steps. First, learn the word embeddings in an unsupervised manner, and then wield these embeddings as features for the supervised learner to train. Once again, this is a problem for our data. Usually, the unlabeled corpus used to learn the embeddings in the first step is different from the corpus in the second step, in order to avoid over-fitting. In our case, the corpus we have is consisting of just two versions of the same corpus, either heavily or lightly preprocessed.

In fact, we eliminated the grammatical structure of the sentences that make up the corpus by lemmatizing and removing stop words such as, "the", "a", "an" in the heavily preprocessed corpus. As a result, much of the data will lose its structural information, and hence seems poorly suited for POS tagging. The only solution left to us, is to use the light preprocessed corpus in both steps, but such a solution would lead to over-fitting, as we would be forced to use the same data on both the unsupervised and supervised models.

Therefore, for all the reasons mentioned above, we will not perform any extrinsic evaluation methods as they seem ill-fitted for the corpus data we are working with.

Critique of the Evaluation Methods

In the field of word embedding evaluations, Bakarov (2018) highlights some important problems. In the intrinsic tests, word embeddings are considered inadequate if they ignore the connections between words as established by our understanding of semantic relatedness or similarity, and considered good otherwise, when it accurately matches our perceptions of semantics. However, our own understanding of that concept is flawed. Considering the fact that there are so many different kinds of interactions between words, such as relatedness and similarity, whose definitions are also somewhat hazy as discussed above, it is unclear what kinds of relationships word embeddings ought to represent.

Furthermore, a good practice in machine learning is to split the data into a training, validation and testing sets. Thus, one can select the best parameters for the model while avoiding over-fitting. For intrinsic tasks, such as word similarity, it is not possible to split our data in such a fashion. The word embeddings are optimised only on the test sets which are usually small (3000 pairs of words in **MEN**) compared to the training set, which is the corpus containing millions of words.

Another cause for concern is that the tuning of the embeddings via intrinsic evaluation does not necessarily correspond to how well they would perform when faced with actual NLP problems or extrinsic tests (POS tagging, classification, etc) (Bakarov, 2018).

7.2 Tuning Eigenwords SVD parameters using Word Similarity

As mentioned above, the parameters of Eigenwords are the window size h, the target rank k, the oversampling p, and the power iteration q. Among them, the oversampling and power iteration are unique to the OSCCA and TSCCA models. More specifically, these parameters belong to the RSVD factorization in the process of Eigenwords. As these models have 4 parameters to tune, creating embedding vectors for each combination is not very efficient, especially since Eigenwords have a lengthy computation. Furthermore, in order to be comparable to Word2vec, we first need to adjust Eigenwords on p and q and set their value to their best iteration.

To do so, we fix $h = 2$ and $k = 200$, as suggested by Dhillon et al. (2015) for OSCCA and TSCCA. Then, we vary the oversampling and power iteration

parameters in the range of $\{50, 100, 150, 200, 250\}$ and $\{0, 1, 3, 5, 7\}$ respectively. The embeddings are trained on two corpus sets, the heavily and lightly preprocessed Daily Mail articles. For each combination of these parameters, we create five sets of word embeddings, using five different seeds. Then, we perform the word similarity task by computing the Spearman Correlation between the cosine similarity of our embeddings and the human judgments of similarity from the three test sets (**MEN**, **SimLex-999**, and **WordSim-353**).

However, before tuning Eigenwords SVD parameters, we must think about the nature of these parameters themselves. Going back to Section 3.2, we know that these parameters are the key to computing a more precise SVD decomposition in the sense of the 2-norm of matrices. By increasing the value of these parameters we trade off computation time of an already time consuming task in exchange for more accuracy. Therefore, we determined that in our case, selecting the optimal parameter values for tuning the model would entail balancing performance and accuracy across all test sets. In other words, we are looking for p and q such that increasing their value any further does not improve the correlation by a significant amount.

Moreover, in order to eliminate any random variability originating from RSVD, we averaged the results over five different seeds, and we will showcase the average Spearman correlations. As we cannot perform cross-validation, and split our data into training and testing sets, we decided to validate our results through this method.

Tables 7.1 and 7.2 showing the Spearman correlation may be found below. They were obtained by following the above scheme for OSCCA and TSCCA respectively using the heavily preprocessed corpus on the **MEN** dataset, highlighting the 10 largest values of each table.

OSCCA					
MEN	Heavy Preprocessing				
Power Iteration / Oversampling	$q = 0$	$q = 1$	$q = 3$	$q = 5$	$q = 7$
$p = 0$	26.2	40.9	48.6	50.8	51.6
$p = 50$	28.9	44.1	50.8	**52.1**	**52.5**
$p = 100$	30.5	43.7	51.1	**52.1**	**52.5**
$p = 150$	29.8	45.1	51.5	**52.5**	**52.8**
$p = 200$	32.8	45.9	51.5	**52.4**	**52.8**
$p = 250$	35.1	47.9	52.0	**52.9**	**53.2**

Table 7.1: Table showing the Spearman correlation ($\rho_p \times 100$) between the cosine similarity of the word embeddings and the human judgment scores from the **MEN** dataset by varying p and q. The correlations are averaged over five simulations of OSCCA using different seeds on the "heavy" corpus.

TSCCA					
MEN	Heavy Preprocessing				
Power Iteration / Oversampling	$q = 0$	$q = 1$	$q = 3$	$q = 5$	$q = 7$
$p = 0$	31.1	36.6	34.8	34.7	**35.2**
$p = 50$	23.8	31.7	34.6	34.3	34.5
$p = 100$	24.7	31.1	34.5	34.5	**35.0**
$p = 150$	26.7	32.6	34.5	**34.7**	**35.1**
$p = 200$	28.2	34.6	**35.2**	**35.6**	**35.6**
$p = 250$	28.2	34.4	**34.8**	**35.6**	**35.6**

Table 7.2: Table showing the Spearman correlation ($\rho_p \times 100$) between the cosine similarity of the word embeddings and the human judgment scores from the **MEN** dataset by varying p and q. The correlations are averaged over five simulations of TSCCA using different seeds on the "heavy" corpus.

From the results shown, it is clear that OSCCA captures the semantic similarity between words better than TSCCA for higher values of p and q. In other words, OSCCA outperforms TSCCA in terms of matching the human subjects' judgments of semantics when it comes to rating the word pairs in the MEN test set. Increasing both parameters yield higher, better correlations. However, around $p = 50$ and $q = 5$, the correlations are not enhanced as drastically anymore.

Furthermore, the most important factor in the high Spearman correlation, is the power iteration parameter q. Increasing q up to $q = 3$ seems to raise the correlation drastically across all values of p. Moreover, setting the parameter $q = 0$ for any parameter p, would yield either inconsistent or worse results overall for both models.

In contrast, let us take a look at Tables 7.3 and 7.4 showing the Spearman correlation for both OSCCA and TSCCA respectively, on the **MEN** dataset and, this time, using the lightly preprocessed corpus. Once again, we highlight the 10 largest values of each table.

OSCCA					
MEN	Light Preprocessing				
Power Iteration / Oversampling	$q = 0$	$q = 1$	$q = 3$	$q = 5$	$q = 7$
$p = 0$	12.0	19.4	24.1	**25.0**	**25.2**
$p = 50$	14.7	21.8	**24.7**	**24.4**	24.1
$p = 100$	15.5	22.2	**25.0**	**24.8**	**24.7**
$p = 150$	14.2	22.0	**24.6**	**24.4**	**24.4**
$p = 200$	15.3	22.2	23.6	24.1	24.3
$p = 250$	15.2	22.4	24.2	24.3	24.3

Table 7.3: Table showing the Spearman correlation ($\rho_p \times 100$) between the cosine similarity of the word embeddings and the human judgment scores from the **MEN** dataset by varying p and q. The correlations are averaged over five simulations of OSCCA using different seeds on the "light" corpus.

TSCCA					
MEN	Light Preprocessing				
Power Iteration / Oversampling	$q = 0$	$q = 1$	$q = 3$	$q = 5$	$q = 7$
$p = 0$	14.5	15.5	15.5	15.7	15.9
$p = 50$	14.6	15.6	16.6	**16.7**	**17.0**
$p = 100$	15.3	16.5	**16.9**	**16.8**	**17.1**
$p = 150$	14.1	14.9	16.0	**16.7**	**16.8**
$p = 200$	15.3	15.7	16.4	**16.7**	16.6
$p = 250$	14.4	15.9	**16.8**	**16.9**	**16.6**

Table 7.4: Table showing the Spearman correlation ($\rho_p \times 100$) between the cosine similarity of the word embeddings and the human judgment scores from the **MEN** dataset by varying p and q. The correlations are averaged over five simulations of TSCCA using different seeds on the "light" corpus.

The same conclusion can be reached for the light preprocessed corpus. The parameter q contributes the most around a value of 5 to obtain satisfying results. Meanwhile, the oversampling p does not need to be very high, 50 seems to work consistently for all the results. The major difference is that, testing the Eigenwords algorithms on the simple preprocessed corpus yields worse results for all combinations of parameters compared to the models trained on the heavy corpus. This is expected since the simple preprocessed corpus retains most of its stop words and has not been lemmatized. Therefore, it contains more words that do not contribute to the meaning of a sentence, and can be considered as noise to both models. Additionally, Halko et al. (2011) claim that $p \leq k$ is enough for satisfactory results. In our case $k = 200$ was set, and increasing p up to 200 or higher did not provide any noteworthy improvement compared to lower values of p. Likewise, the other tables showcasing the results obtained on the other two test sets demonstrate similar behaviors (see Appendix C).

To summarize, the Spearman correlation generally improves as the parameters become larger. TSCCA show more consistent correlations across all the tests, and OSCCA is more dependent on parameters p and q. In the case of higher values of p and q, OSCCA yields higher correlations than TSCCA for all test sets and all preprocessing types. Moreover, we can conclude that OSCCA and TSCCA are able to capture the semantic similarity between words. Indeed, p and q are only the parameters of the RSVD factorization. Increasing their values simply increases the precision of the factorization of the triple product of the covariance matrices. Thus, it means that the more precise factorization within the CCA algorithms

yields better correlation for all the test sets.

Finally, even if both Eigenwords algorithms perform better on average when the highest parameter settings are used, the computation time is significantly longer. Hence, instead of using the highest values, we decided to settle for $p = 50$ and $q = 5$ for the evaluation of word embedding models in the next sections. This combination of parameter values was selected since it provides satisfactory results across all tables and will reduce computing time.

7.3 Final Evaluation of the Unsupervised Models

In this section, we will use Word Similarity as an evaluation method to inspect the performance of OSCCA and TSCCA against the state-of-the-art Word2vec models. During this process, we will also analyze how well each model performs using the heavily and lightly preprocessed data. Finally, the best deemed model will undergo a qualitative word analogy test.

As mentioned in the last section, the oversampling and power iteration parameters will be set to $p = 50$ and $q = 5$ for OSCCA and TSCCA for the rest of the thesis. For every model, we will run five different simulations using 5 different randoms seeds. As a side note, we want to clarify that Word2vec uses random seeds to initialize the weights, which are the word vectors.

7.3.1 Word Similarity Test

In order to explore the different behaviors of Eigenwords and Word2vec models, we will set up the algorithms with multiple combination of parameters, h the window size specifying how large the context is, and k the numerical rank, which is also the size of the obtained word vectors. All the variations of the models are trained on either the heavy or lightly preprocessed datasets, and evaluated on the **MEN**, **SimLex-999**, and **WordSim-353** sets.

First, for all the models, we will set $h = 2$, as advised by Dhillon et al. (2015), and vary $k \in \{100, 200, 300\}$ to see the impact of k. Let us investigate Tables 7.5 and 7.6 representing these settings.

Model	k	MEN	SimLex	WordSim
OSCCA	100	**52.64**	21.28	56.05
	200	52.12	25.00	**57.36**
	300	49.67	**25.91**	53.38
TSCCA	100	39.63	11.21	49.38
	200	39.69	13.40	48.12
	300	40.21	16.28	46.15
CBOW	100	39.21	16.47	47.53
	200	38.68	16.52	46.75
	300	38.59	16.48	46.64
Skip-Gram	100	41.65	21.35	46.65
	200	39.65	21.16	43.80
	300	39.28	20.93	43.23

Table 7.5: Comparison of the Spearman correlation ($\rho_p \times 100$) for all evaluated models using the "heavy" corpus to train, with a window size set at $h = 2$. The best performance for each test set over all models is in bold. Every result is averaged over five iterations using five random seeds.

Model	k	MEN	SimLex	WordSim
OSCCA	100	23.60	11.72	37.38
	200	24.37	14.07	37.62
	300	**24.69**	16.34	**38.86**
TSCCA	100	14.91	11.36	38.24
	200	17.18	11.43	35.36
	300	16.23	12.91	34.68
CBOW	100	23.31	12.50	37.34
	200	23.50	13.25	37.51
	300	23.46	13.23	36.76
Skip-Gram	100	21.09	**16.86**	29.78
	200	20.19	16.12	29.67
	300	19.98	16.64	29.85

Table 7.6: Comparison of the Spearman correlation ($\rho_p \times 100$) for all evaluated models using the light corpus to train, with a window size set at $h = 2$. The best performance for each test set over all models is in bold. Every result is averaged over five iterations using five random seeds.

By examining Tables 7.5 and 7.6, we can notice that increasing k past 200 for $h = 2$, does not provide any dramatic changes in the correlations for any model,

across any test sets. However, it is clear that the OSCCA model performs better than its competitors in almost all situations, only the Skip-Gram model trained on the light corpus performs better, for the **SimLex-999** test set with $k = 100$.

Next we decided to fix $k = 300$ and vary $h = \{1, 2\}$. We did not increase h any further since the matrices generated within the Eigenwords algorithms become too large for our computer to handle. The following Tables 7.7 and 7.8 showcase the described settings and their results.

Model	**h**	**MEN**	**SimLex**	**WordSim**
OSCCA	1	43.83	25.62	**55.93**
	2	**49.67**	**25.91**	53.38
TSCCA	1	41.40	21.77	47.89
	2	40.21	16.28	46.15
CBOW	1	31.87	16.45	40.47
	2	38.59	16.48	46.66
Skip-Gram	1	32.87	18.82	40.37
	2	39.28	20.93	43.22

Table 7.7: Comparison of the Spearman correlation ($\rho_p \times 100$) for all evaluated models using the heavy corpus to train, with a target rank set at $k = 300$. The best performance for each test set over all models is in bold. Every result is averaged over five iterations using five random seeds.

Model	**h**	**MEN**	**SimLex**	**WordSim**
OSCCA	1	20.01	13.13	38.71
	2	**24.69**	16.34	**38.86**
TSCCA	1	17.22	13.66	38.02
	2	16.24	12.91	34.68
CBOW	1	19.92	15.26	37.89
	2	23.46	13.23	36.76
Skip-Gram	1	16.47	16.01	27.98
	2	19.98	**16.64**	29.84

Table 7.8: Comparison of the Spearman correlation ($\rho_p \times 100$) for all evaluated models using the light corpus to train, with a target rank set at $k = 300$. The best performance for each test set over all models is in bold. Every result is averaged over five iterations using five random seeds.

Once again, there is no doubt that the OSCCA model provides the highest correlations in almost every situation (test sets, combination of parameters). All

models show an increase in their correlations when the window size parameter is increased from $h = 1$ to $h = 2$. It seems that the models capture more information about the semantic similarity between words when this parameter is increased. This is expected, as we increase the context size, more information about the context of the words will be represented in the model, and more nuances can be captured. The only model that yields consistently worse results when increasing this parameter is the TSCCA algorithm.

Let us try to explain this behavior of TSCCA seen from Tables 7.7 and 7.8. We have a hypothesis to why TSCCA show worse results in those instances. As previously stated, we need to solve Equation (4.4) in order to run the TSCCA algorithm. However, the term $\mathbf{\Sigma}_{ss}^{-\frac{1}{2}}$ inside the equation has been shown to be problematic. In Section 4.5.2, we established that $\mathbf{\Sigma}_{ss}$, could not be positive definite and hence, we could not compute its inverse square root. In order to observe if the results would be different for a larger input corpus, we built the matrix $\mathbf{\Sigma}_{ss}$ by use of the "heavy" and "light" corpus, and proceeded to inspect both the obtained versions. To no surprise, the matrix had positive and negative eigenvalues, hence it was not positive definite. Since the inverse does not exist, no approximation, not even Equation (4.6) would solve the issue in the algorithm of TSCCA.

Therefore, increasing the window size, would in return, increase the amount of incorrectly approximated values in the estimated matrix $\mathbf{\Sigma}_{ss}$. This is our hypothesis on why TSCCA did not perform as well as according to Dhillon et al. (2015).

Moreover, it is clear that the results are particularly dependent on the type of preprocessing. The correlations obtained are much higher when the trained data are preprocessed with a more complex scheme. As mentioned in Sections 6.1.2 and 7.2, a corpus which has been laxly preprocessed carries more noise in its content than a heavily preprocessed corpus, and in consequence, any model learning from it displays worse correlations. Hence, the preprocessing step of the raw corpus can be considered a crucial optimization step of any word embedding model. From Table 7.5 compared to Table 7.6, for $h = 2$ and $k = 200$, it is evident that the performance of OSCCA drops from a correlation of 52.12 to 24.37. This performance drop applies to all the models tested here, without exceptions. Furthermore, across all tables, when comparing the results for all the models, we can observe that OSCCA handles both heavily and lightly preprocessed corpus better than the other models.

Finally, the correlations corresponding to either **MEN**, **SimLex-999**, and **WordSim-353**, have very different values from one test set to another. Correlations obtained with either **MEN** or **WordSim-353** are the highest. This is

expected, as the **MEN** and **WordSim-353** sets, as stated in Section 7.1.3, have very loose definitions for the notion of semantic similarity. Both datasets have a concept of relatedness and similarity that have been considered interchangeable.

Words are generally related to one other in their immediate contexts, and embedding models are good at detecting this pattern. However, it is more difficult for a word embedding model to determine whether those two terms are merely linked or truly semantically similar. As a result, terms that are related but semantically distinct from one another will receive a high cosine similarity, and datasets that do not make the distinction between both concepts will yield higher correlations. On the other hand, the **SimLex-999** has a more strict definition for the concept of similarity. Hence, we get lower correlation results.

Lastly, one point of worry we highlighted was how efficient and appropriate were the test sets at evaluating the pair of words, when our training set is centered around the topic of war. In order to illustrate the efficiency of the OSCCA model at catching semantic similarity for infrequent words, let the Table 7.9 show the most similar words that are diverse in themes.

Target word	Most similar words
color	{colour, yellow, blue, ribbon}
car	{truck, vehicle, jaguar, van}
love	{beautiful, thank, freedom, proud}
technology	{data, imagery, software, tech}

Table 7.9: Table listing the top 4 words that are the most similar to the target word using the cosine similarity. All the word vectors are computed using OSCCA with $h = 200$ and $k = 200$.

Hence, OSCCA is able to capture word similarity from the corpus, even for words that are rare, and the gold standard test sets are suitable as benchmarks to evaluate the models. As a side note, the most similar words for "color", include "blue" and "yellow", the colors of the Ukrainian flag.

7.3.2 Word Analogy

Lastly, in this final section we will work with the model that showed the best results, namely OSCCA. From our analysis, since the choice of preprocessing algorithm can be considered a parameter of the model, we decided to use the heavy preprocessing as it yields the highest correlations. Additionally, $h = 2$ will be used as it improves

the quality of the word vectors for the model, and we will set $k = 200$, as the increase to $k = 300$ generally did not enhance the outcomes.

In the word similarity task, the Spearman correlation was used as a global metric to evaluate the performance of the models. However, word analogy does not provide a metric to evaluate the overall accuracy of the predictions.

Nonetheless, let us tackle word analogy as the last qualitative test to investigate how the word vectors are perceived by the chosen model. Since it was trained on news articles about the war between Russia and Ukraine, the goal would be to check if the similarity between certain pair of words captured by OSCCA reflect our perception of those words in that context. On this account, we expect that this specific choice of topic would bear some interesting results.

Finally, using (7.1), we obtain the following results by selecting words and relations we deemed relevant in the appointed context.

$a : b$	$c :$	w^*
russia : france	moscow :	{denmark, italy, spain, paris}
russia : moscow	ukraine :	{hunting, barrett, controller, recognizes}
france : emmanuel	ukraine :	{volodymyr, volodmyr, vlodymyr, volodomyr}
russia : putin	ukraine :	{falter, dzhabarov, strive, recognizes}
putin : vladimir	zelensky :	{zelenskyy, zelenskiy, volodymyr, volodmyr}
civilian : refugee	people :	{migrant, concentration, resettlement, homes}
gas : rising	oil :	{soaring, surging, rise, hike}

Table 7.10: Table listing the top 4 words solution to (7.1), given the relation $a : b$, $c : w^*$.

Just from these few examples, it is clear that OSCCA manages to imitate the semantic meaning of the words to some extent. The relationship between the countries and their capitals or their presidents is almost captured by the model. The only instance where it fails, is when the words "russia" and "ukraine" are involved at the same time, contrarily to what we expect. This can be explained by the fact that the corpus we provided to the model was centered around the specific topic of the conflict between Russia and Ukraine. It makes sense that the connotations for those two nations were warped towards the meaning of the war. Thus, certain results deviate from our expectations, originating from a wider, more general context. Moreover, the socio-economical aspect of the war is somewhat captured: The terms gas and oil are related to some kind of inflation, and the terms civilian and people, are related to victims fleeing from the war.

Finally, it is a good practice to remind ourselves that the logical mathematical systems are learning from subjective inputs. Indeed, the model captures relationships between words written by the authors of the provided articles. Word embedding models, such as Eigenwords, do not have a critical mind of their own as they only learn by capturing patterns in the texts. Therefore, the bias and choice of words of these authors can directly shape the models' outcome.

Chapter 8

Conclusion

8.1 Results and Overview

This thesis served as an opportunity to investigate an alternative method to assemble word embeddings, a powerful complement for the ever-expanding field of NLP, and compare its performance to existing neural models.

In Chapter 2, we have demonstrated the incentives for the use of word embeddings and provided definitions of terms used over the course of this thesis. Chapter 3 provided an overview of CCA and the mathematical tools to calculate it. We explain that RSVD takes advantage of a randomized process to compute lower dimensional representations of the matrices more efficiently, while maximizing the precision of the factorizations. We motivate this approach by stating that the lower dimension k of the word vectors can be chosen arbitrarily, unlike other SVD factorization techniques, where the dimensions of the resulting projections are enforced by the number of singular values.

Chapter 4 defined the different word count matrices the Eigenwords algorithms are applied on. Moreover, two methods based on CCA were introduced, each one yielding word vectors. We provided the numerical results from Dhillon et al. (2015) showing the efficiency of their models, and discussed the mandatory step of regularizing by adding $\epsilon\mathbf{I}$ to the covariance matrices in order to compute the SVD formulation of CCA. Furthermore, we also highlighted the matrix $\mathbf{\Sigma}_{ss}$ from TSCCA and the reason it hinders the computation of the algorithm.

Chapter 5 briefly introduced Skip-Gram and CBOW architectures from the Word2vec model. They are fully developed shallow neural networks that are praised by the machine learning community for their effectiveness and simplicity. In Chap-

ter 6, the process of retrieving the data from the web was explained. It was divided into two steps, crawling and scraping. We also discussed our choice of using two different preprocessing schemes in order to shed light on the importance of this step for word embedding models. As we have shown in the Chapter 7, word embedding models that learn from a heavily preprocessed corpus obtain higher correlations on the word similarity task, as most noise inside the corpus has been filtered out.

Finally, in Chapter 7, we introduced two tasks, word similarity and word analogy. We tuned OSCCA and TSCCA on the word similarity task, and obtained the best oversampling and power iteration parameter values, which are respectively $p = 50$ and $q = 5$. In fact, Eigenwords algorithms perform better on average when the highest parameter settings are used, but the computation time is significantly longer, while the correlations obtained were comparable. Thus, $p = 50$ and $q = 5$ were deemed the most balanced in this regard. Then, we performed an evaluation of all the models using three test sets on the word similarity task by wielding all the best parameters for each model. We discovered that OSCCA was by far the model that performed the best in almost every situation. As for TSCCA, due to the matrix $\mathbf{\Sigma}_{ss}$ being non-positive definite, this model under-performed compared to the other displayed models, and compared to the expectations of Dhillon et al. (2015). However, it still presented satisfactory results despite the encountered technical problem. Finally, using a word analogy test on OSCCA, we showed that the model is capable of capturing semantic meaning of words from a topic centered corpus, in this case about the Russo-Ukrainian War. wd

8.2 Limitations and Avenues for Future Work

The process of Eigenword algorithms involves the approximation of multiple sparse covariance matrices (i.e $\mathbf{\Sigma}_{cc}$, $\mathbf{\Sigma}_{ll}$, $\mathbf{\Sigma}_{rr}$.) to diagonal matrices in order to calculate their inverse square roots. One avenue for improvement would be to compute an approximation of these large matrices that retains most of the values outside the diagonals, while keeping the sparse propriety of their inverse square roots. One such method could be the sparse CCA (SCCA) (Mai and Zhang, 2019). It transforms the CCA algorithm into an iterative penalized least squares problem. According to the authors, this method does not require any assumptions about the covariance matrices. Hence, their proposed SCCA should in theory be able to work without approximating the covariance matrices to their diagonal forms. They also affirm that SCCA can estimate the canonical pairs with a high precision.

As for deploying TSCCA, we displayed the problems associated with it, namely the matrix $\mathbf{\Sigma}_{ss}$ used in its algorithm. The issue being that the matrix is not

positive definite and its inverse does not exist, one cannot compute TSCCA without replacing the non-existent square root inverse of $\mathbf{\Sigma}_{ss}$ by an approximation. Hence, a future area of improvement would be to find another regularization method or an approximation that is more appropriate, and could potentially repair TSCCA.

Another point of interest is that in the Eigenwords models, the CCA approach aims to learn linear transformations of the original word matrices such that their correlation is maximized. However, there may be nonlinear transformations producing higher correlated matrices. This is where Deep CCA (Andrew et al., 2013) could be an alternative that is worth exploring. It takes advantage of deep learning schemes to find nonlinear mappings, and therefore could be a potential avenue for improving Eigenwords algorithms.

On another note, Dhillon et al. (2015) claim that Eigenwords algorithms can detect polysemy and generate context-specific embeddings for words. Extrinsic evaluations are the only way to test such a statement. Unfortunately, our work has not explored extrinsic methods for word embeddings due to the choice of the data used for this paper. Finally, incorporating downstream NLP tasks to evaluate the models on real world situations would be a great improvement for the models in general, and would help in avoiding word vectors over-fitting on word similarity test sets.

Appendix A

Singular Value Decomposition. Proof of equation (3.9):

The following proof is from Lay et al. (2016):

Proof. Suppose $\{v_1, ..., v_n\}$ is an orthonormal basis of $\mathbb{R}^n$ consisting of eigenvectors of $\mathbf{A}^t\mathbf{A}$, arranged so that the corresponding eigenvalues of $\mathbf{A}^t\mathbf{A}$ satisfy $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$, and suppose $\mathbf{A}$ has r nonzero singular values σ_i. Then $\{\mathbf{A}v_1, ..., \mathbf{A}v_n\}$ is an orthogonal basis for Col $\mathbf{A}$, and rank$(\mathbf{A}) = r$. Normalize each $\mathbf{A}v_i$ to obtain an orthogonal basis $\{u_1, ..., u_r\}$, where

$$u_i = \frac{\mathbf{A}v_i}{\|\mathbf{A}v_i\|} = \frac{\mathbf{A}v_i}{\sigma_i}$$

and

$$\mathbf{A}v_i = \sigma_i u_i \qquad (1 \leq i \leq r) \tag{A.1}$$

Now extend the vectors $\{u_1, ..., u_r\}$ to an orthonormal basis $\{u_1, ..., u_m\}$ of $\mathbb{R}^m$, and let

$$\mathbf{U} = \begin{bmatrix} u_1 & u_2 & ... & u_m \end{bmatrix} \qquad \text{and} \qquad \mathbf{V} = \begin{bmatrix} v_1 & v_2 & ... & v_n \end{bmatrix}$$

By construction, $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices. Also, from (A.1),

$$\mathbf{AV} = \begin{bmatrix} \mathbf{A}v_1 & ... & \mathbf{A}v_r & 0 & ... & 0 \end{bmatrix} = \begin{bmatrix} \sigma_1 u_1 & ... & \sigma_r u_r & 0 & ... & 0 \end{bmatrix}$$

Let $\mathbf{D}$ be the diagonal matrix with diagonal entries $\sigma_1, ..., \sigma_r$, and let $\mathbf{\Lambda}$ be as in

equation (3.8). Then

$$\begin{aligned}
\mathbf{U\Lambda} &= \begin{bmatrix} u_1 & u_2 & \dots & u_m \end{bmatrix} \left[\begin{array}{cccc|c} \sigma_1 & & & 0 & \\ & \sigma_2 & & & \\ & & \ddots & & 0 \\ 0 & & & \sigma_r & \\ \hline & & 0 & & 0 \end{array}\right] \\
&= \begin{bmatrix} \sigma_1 u_1 & \dots & \sigma_r u_r & 0 & \dots & 0 \end{bmatrix} \\
&= \mathbf{AV}
\end{aligned}$$

Since $\mathbf{V}$ is an orthogonal matrix, it follows that

$$\mathbf{U\Lambda V}^t = \mathbf{AVV}^t = \mathbf{A}$$

□

Appendix B

Proof of the SVD decomposition of CCA

Proof. Let us define $\phi_a = \mathbf{\Sigma}_{aa}^{-\frac{1}{2}} u$ and $\phi_b = \mathbf{\Sigma}_{bb}^{-\frac{1}{2}} v$, with $u \in \mathbb{R}^{m_1}$ and $v \in \mathbb{R}^{m_2}$, due to the Constraints (3.5), we have that

$$\begin{aligned}
\phi_a^t \mathbf{\Sigma}_{aa} \phi_a = 1 &\iff u^t \mathbf{\Sigma}_{aa}^{-\frac{1}{2}} \mathbf{\Sigma}_{aa} \mathbf{\Sigma}_{aa}^{-\frac{1}{2}} u \implies u^t u = 1 \\
\phi_b^t \mathbf{\Sigma}_{bb} \phi_b = 1 &\iff v^t \mathbf{\Sigma}_{bb}^{-\frac{1}{2}} \mathbf{\Sigma}_{bb} \mathbf{\Sigma}_{bb}^{-\frac{1}{2}} v \implies v^t v = 1 \\
\phi_a^t \mathbf{\Sigma}_{ab} \phi_b &= u^t \mathbf{\Sigma}_{aa}^{-\frac{1}{2}} \mathbf{\Sigma}_{ab} \mathbf{\Sigma}_{bb}^{-\frac{1}{2}} v.
\end{aligned}$$

Hence, the optimization Equation (3.1) becomes,

$$\arg\max_{\phi_a, \phi_b} (u^t \mathbf{\Sigma}_{aa}^{-\frac{1}{2}} \mathbf{\Sigma}_{ab} \mathbf{\Sigma}_{bb}^{-\frac{1}{2}} v),$$

under the constraints $u^t u = 1$ and $v^t v = 1$. Now let us apply SVD,

$$\mathbf{\Sigma}_{aa}^{-\frac{1}{2}} \mathbf{\Sigma}_{ab} \mathbf{\Sigma}_{bb}^{-\frac{1}{2}} = \mathbf{U} \mathbf{\Lambda} \mathbf{V}^t,$$

with $\mathbf{U}$ and $\mathbf{V}$ the right and left singular vectors corresponding to its singular value matrix $\mathbf{\Lambda}$. Since $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrices and their columns are an orthonormal basis. Accordingly, we have that $u^t \mathbf{U}$ and $\mathbf{V} v$ are equal to either 0 or 1. Therefore,

$$u^t \mathbf{\Sigma}_{aa}^{-\frac{1}{2}} \mathbf{\Sigma}_{ab} \mathbf{\Sigma}_{bb}^{-\frac{1}{2}} v = u^t \mathbf{U} \mathbf{\Lambda} \mathbf{V}^t v = \sigma_{uv}.$$

From this, it is easy to see that maximizing Equation (3.1) is equivalent to finding the biggest singular value σ_{uv} corresponding to a certain group of left and right singular vectors. This largest singular value is in fact the largest correlation coefficient between $\mathbf{X}_a$ and $\mathbf{X}_b$. □

Appendix C

Extra tables

Here are multiple tables showing the Spearman correlation ($\rho_p \times 100$) between the cosine similarity of the word embeddings and the human judgment scores by varying p and q. The correlations are averaged over 5 simulations using different seeds. The simulations were run with either OSCCA or the TSCCA algorithms using either the heavy or the light preprocessed data. The test sets used to evaluate the correlations were the **MEN**, **SimLex-999**, and **WordSim-353** sets.

The tables are ordered by their type of preprocessing, then by their test sets, and finally by the type of Eigenwords algorithms deployed.

OSCCA					
SimLex-999	Heavy Preprocessing				
Power Iteration / Oversampling	$q = 0$	$q = 1$	$q = 3$	$q = 5$	$q = 7$
$p = 0$	15.6	21.2	24.3	25.0	25.5
$p = 50$	13.3	20.5	24.7	25.1	**25.6**
$p = 100$	17.5	22.6	24.8	**25.8**	**25.7**
$p = 150$	15.3	22.9	**25.8**	**26.1**	**26.1**
$p = 200$	17.0	22.5	25.0	**26.1**	**26.2**
$p = 250$	18.3	22.6	24.9	**25.5**	**25.7**

Table C.1: Table showing the Spearman correlation ($\rho_p \times 100$) between the cosine similarity of the word embeddings and the human judgment scores from the **SimLex-999** dataset by varying p and q. The correlations are averaged over 5 simulations of OSCCA using different seeds on the heavy corpus.

TSCCA					
SimLex-999	Heavy Preprocessing				
Power Iteration / Oversampling	$q = 0$	$q = 1$	$q = 3$	$q = 5$	$q = 7$
$p = 0$	12.7	**15.00**	**14.8**	**14.5**	**14.4**
$p = 50$	10.4	13.1	13.0	13.4	13.7
$p = 100$	13.8	**15.8**	**14.7**	14.2	**14.5**
$p = 150$	13.4	13.2	14.2	**14.6**	**15.1**
$p = 200$	13.9	13.8	13.9	13.8	14.3
$p = 250$	13.9	**14.8**	14.0	14.1	**14.6**

Table C.2: Table showing the Spearman correlation ($\rho_p \times 100$) between the cosine similarity of the word embeddings and the human judgment scores from the **SimLex-999** dataset by varying p and q. The correlations are averaged over 5 simulations of TSCCA using different seeds on the heavy corpus.

OSCCA					
WordSim-353	Heavy Preprocessing				
Power Iteration / Oversampling	$q = 0$	$q = 1$	$q = 3$	$q = 5$	$q = 7$
$p = 0$	31.6	46.4	53.0	42.2	55.1
$p = 50$	35.3	49.1	55.76	**57.4**	**59.0**
$p = 100$	34.2	49.2	54.9	56.6	57.2
$p = 150$	36.3	52.4	57.9	**58.1**	**57.9**
$p = 200$	42.2	53.6	**58.4**	**59.1**	**59.1**
$p = 250$	40.0	53.3	**57.6**	**58.8**	**58.9**

Table C.3: Table showing the Spearman correlation ($\rho_p \times 100$) between the cosine similarity of the word embeddings and the human judgment scores from the **WordSim-353** dataset by varying p and q. The correlations are averaged over 5 simulations of OSCCA using different seeds on the heavy corpus.

TSCCA					
WordSim-353	Heavy Preprocessing				
Power Iteration / Oversampling	$q = 0$	$q = 1$	$q = 3$	$q = 5$	$q = 7$
$p = 0$	41.8	45.6	46.7	47.7	48.5
$p = 50$	40.7	44.9	48.2	48.2	49.2
$p = 100$	48.4	40.1	49.7	**51.8**	**51.7**
$p = 150$	45.1	49.7	53.5	**53.3**	**53.0**
$p = 200$	41.5	48.2	**50.5**	**52.6**	**53.0**
$p = 250$	43.6	50.6	**52.3**	**52.8**	**52.6**

Table C.4: Table showing the Spearman correlation ($\rho_p \times 100$) between the cosine similarity of the word embeddings and the human judgment scores from the **WordSim-353** dataset by varying p and q. The correlations are averaged over 5 simulations of TSCCA using different seeds on the heavy corpus.

OSCCA					
SimLex-999	Light Preprocessing				
Power Iteration / Oversampling	$q = 0$	$q = 1$	$q = 3$	$q = 5$	$q = 7$
$p = 0$	11.3	13.9	13.9	13.6	13.6
$p = 50$	11.3	**14.3**	14.2	14.1	**14.3**
$p = 100$	9.4	14.0	**14.3**	14.2	14.2
$p = 150$	8.7	13.9	**14.3**	**14.3**	**14.4**
$p = 200$	12.7	**14.3**	**14.4**	**14.5**	**14.4**
$p = 250$	11.3	**14.7**	**14.7**	**14.6**	**14.4**

Table C.5: Table showing the Spearman correlation ($\rho_p \times 100$) between the cosine similarity of the word embeddings and the human judgment scores from the **SimLex-999** dataset by varying p and q. The correlations are averaged over 5 simulations of OSCCA using different seeds on the light corpus.

TSCCA					
SimLex-999	Light Preprocessing				
Power Iteration / Oversampling	$q = 0$	$q = 1$	$q = 3$	$q = 5$	$q = 7$
$p = 0$	8.6	10.1	11.9	11.6	11.5
$p = 50$	10.2	12.0	12.1	11.7	12.3
$p = 100$	12.2	**12.6**	12.2	**12.4**	**12.6**
$p = 150$	11.3	12.3	**12.5**	**12.5**	**12.5**
$p = 200$	9.4	12.0	12.2	11.9	**12.6**
$p = 250$	12.0	**12.6**	12.1	**12.8**	**12.6**

Table C.6: Table showing the Spearman correlation ($\rho_p \times 100$) between the cosine similarity of the word embeddings and the human judgment scores from the **SimLex-999** dataset by varying p and q. The correlations are averaged over 5 simulations of TSCCA using different seeds on the light corpus.

OSCCA					
WordSim-353	Light Preprocessing				
Power Iteration / Oversampling	$q = 0$	$q = 1$	$q = 3$	$q = 5$	$q = 7$
$p = 0$	21.6	30.6	36.7	**38.0**	**38.4**
$p = 50$	26.7	34.3	**37.8**	**37.6**	**37.5**
$p = 100$	20.7	32.6	**38.4**	**38.0**	**37.7**
$p = 150$	24.8	35.3	**37.5**	37.1	37.0
$p = 200$	22.6	36.0	36.5	37.1	37.2
$p = 250$	29.1	**37.9**	37.1	36.9	37.1

Table C.7: Table showing the Spearman correlation ($\rho_p \times 100$) between the cosine similarity of the word embeddings and the human judgment scores from the **WordSim-353** dataset by varying p and q. The correlations are averaged over 5 simulations of OSCCA using different seeds on the light corpus.

TSCCA					
WordSim-353	Light Preprocessing				
Power Iteration / Oversampling	$q = 0$	$q = 1$	$q = 3$	$q = 5$	$q = 7$
$p = 0$	28.0	34.0	34.0	**34.8**	34.2
$p = 50$	27.1	32.14	33.2	**34.7**	34.0
$p = 100$	**35.7**	**34.9**	33.7	**34.6**	**34.7**
$p = 150$	31.2	33.0	**35.3**	33.2	**35.6**
$p = 200$	33.4	33.5	**34.6**	34.2	**34.6**
$p = 250$	33.4	**35.7**	34.2	33.5	34.1

Table C.8: Table showing the Spearman correlation ($\rho_p \times 100$) between the cosine similarity of the word embeddings and the human judgment scores from the **WordSim-353** dataset by varying p and q. The correlations are averaged over 5 simulations of TSCCA using different seeds on the light corpus.

Bibliography

Andrew, G., Arora, R., Bilmes, J., and Livescu, K. (2013). Deep canonical correlation analysis. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1247–1255, Atlanta, Georgia, USA. PMLR.

Bach, F. and Jordan, M. (2005). A probabilistic interpretation of canonical correlation analysis. Technical Report 688, Department of Statistics, University of California, Berkeley.

Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., and van der Vorst, H. (2000). *Templates for the solution of algebraic eigenvalue problems: a practical guide*. Society for Industrial and Applied Mathematics.

Bakarov, A. (2018). A Survey of Word Embeddings Evaluation Methods. *Computing Research Repository*.

Bicalho, P., Pita, M., Pedrosa, G., Lacerda, A., and Pappa, G. L. (2017). A general framework to expand short text for topic modeling. *Information Sciences*, 393:66–81.

Bruni, E., Tran, N. K., and Baroni, M. (2014). Multimodal Distributional Semantics. *Journal of Artificial Intelligence Research*, 49:1–47.

Camacho-Collados, J. and Pilehvar, M. T. (2018). On the Role of Text Preprocessing in Neural Network Architectures: An Evaluation Study on Text Categorization and Sentiment Analysis. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP), Brussels, Belgium*, pages 40–46. Association for Computational Linguistics.

Dhillon, P. S., Foster, D. P., and Ungar, L. H. (2015). Eigenwords: spectral word embeddings. *The Journal of Machine Learning Research*, 16(1):3035–3078.

Dudoit, S., Fridlyand, J., and Speed, T. P. (2002). Comparison of Discrimination Methods for the Classification of Tumors Using Gene Expression Data. *Journal of the American Statistical Association*, 97(457):77–87.

Ewerbring, L. M. and Luk, F. T. (1989). Canonical correlations and generalized SVD: Applications and new algorithms. *Journal of Computational and Applied Mathematics*, 27(1):37–52.

Faruqui, M., Tsvetkov, Y., Rastogi, P., and Dyer, C. (2016). Problems With Evaluation of Word Embeddings Using Word Similarity Tasks. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP, Berlin, Germany, August 2016*, pages 30–35. Association for Computational Linguistics.

Finkelstein, L., Gabrilovich, E., Matias, Y., Rivlin, E., Solan, Z., Wolfman, G., and Ruppin, E. (2001). Placing search in context: The concept revisited. *ACM Transactions on Information Systems - TOIS*, 20:406–414.

Ford, W. (2014). *Numerical Linear Algebra with Applications using MATLAB*. Elsevier, Academic Press.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Gubner, J. A. (2006). *Probability and Random Processes for Electrical and Computer Engineers*. Cambridge University Press, Cambridge.

Halko, N., Martinsson, P. G., and Tropp, J. A. (2011). Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Review*, 53(2):217–288.

Hansen, P. C. (1987). The truncated SVD as a method for regularization. *BIT Numerical Mathematics*, 27:534–553.

Hill, F., Reichart, R., and Korhonen, A. (2015). SimLex-999: Evaluating Semantic Models With (Genuine) Similarity Estimation. *Computational Linguistics*, 41(4):665–695.

Horn, R. A. and Johnson, C. R. (2013). *Matrix Analysis*. Cambridge University Press.

Horn, R. A. and Zhang, F. (2005). Basic Properties of the Schur Complement. In Zhang, F., editor, *The Schur Complement and Its Applications*, Numerical Methods and Algorithms, pages 17–46. Springer US.

Hotelling, H. (1936). Relations Between Two Sets Of Variates. *Biometrika*, 28(3-4):321–377.

Hoy, M. B. (2018). Alexa, siri, cortana, and more: An introduction to voice assistants. *Medical Reference Services Quarterly*, 37(1):81–88.

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

Lay, D. C., Lay, S. R., and McDonald, J. (2016). *Linear algebra and its applications*. Pearson, Boston, 5th edition.

Li, W. (1992). Random texts exhibit Zipf's-law-like word frequency distribution. *IEEE Transactions on Information Theory*, 38(6):1842–1845.

Mai, Q. and Zhang, X. (2019). An iterative penalized least squares approach to sparse canonical correlation analysis. *Biometrics*, 75(3):734–744.

Manaris, B., Pellicoro, L., Pothering, G., and Hodges, H. (2006). Investigating Esperanto's Statistical Proportions Relative to other Languages using Neural Networks and Zipf's Law. In *IASTED International Conference on Artificial Intelligence and Applications, part of the 24th Multi-Conference on Applied Informatics, Innsbruck, Austria, February 13-16, 2006*, pages 102–108. IASTED/ACTA Press.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Meyer, C. D. (2000). *Matrix Analysis and Applied Linear Algebra*. Society for Industrial and Applied Mathematics.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*.

Mikolov, T., Yih, W.-t., and Zweig, G. (2013b). Linguistic Regularities in Continuous Space Word Representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia. Association for Computational Linguistics.

Newman, M. E. J. (2005). Power laws, Pareto distributions and Zipf's law. *Contemporary Physics*, 46(5):323–351.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

Penrose, R. (1955). A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51(3):406–413.

Piantadosi, S. T. (2014). Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic Bulletin & Review*, 21(5):1112–1130.

Plisson, J., Lavrac, N., and Mladenic, D. (2004). A Rule based Approach to Word Lemmatization. In *Proceedings of IS04*. Department of Knowledge Technologies.

Pradana, A. and Hayati, M. (2019). The Effect of Stemming and Removal of Stopwords on the Accuracy of Sentiment Analysis on Indonesian-language Texts. *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*, 4(4):375–380.

Rose, T., Stevenson, M., and Whitehead, M. (2002). The Reuters Corpus Volume 1 -from Yesterday's News to Tomorrow's Language Resources. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02), Las Palmas, Canary Islands - Spain*, pages 29–31. European Language Resources Association (ELRA).

Stratos, K., Collins, M., and Hsu, D. (2015). Model-based Word Embeddings from Decompositions of Count Matrices. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Beijing, China, July 26-31, 2015*, pages 1282–1291. Association for Computational Linguistics.

Tibshirani, R., Hastie, T., Narasimhan, B., and Chu, G. (2003). Class Prediction by Nearest Shrunken Centroids, with Applications to DNA Microarrays. *Statistical Science*, 18(1):104–117.

Uurtio, V., Monteiro, J. M., Kandola, J., Shawe-Taylor, J., Fernandez-Reyes, D., and Rousu, J. (2017). A Tutorial on Canonical Correlation Methods. *ACM Computing Surveys*, 50(6):95:1–95:33.

Vitale, E. (2017). *Syllabus Algèbre Linéaire*. Ecole de Mathématique - Faculté des Sciences. Université Catholique de Louvain.

Yang, C. (2013). Who's afraid of George Kingsley Zipf? Or: Do children and chimps have language? *Significance*, 10(6):29–34.

Yang, S., Lu, W., Yang, D., Yao, L., and Wei, B. (2015). Short Text Understanding by Leveraging Knowledge into Topic Model. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado*, pages 1232–1237. Association for Computational Linguistics.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
Faculté des sciences
Place des Sciences, 2 bte L6.06.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/sc