University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Data Science

Optimization of embedding alignment for bias analysis

Supervisor
Prof. FRANCESCO RINALDI
University of Padova

Co-supervisor
Prof. GIOVANNI DA SAN MARTINO
University of Padova

Master Candidate
WU XIANLONG

Student ID
2038500

Academic Year
2022-2023

Abstract

Natural Language Processing(NLP) is a field of Artificial Intelligence which also related to linguistics. NLP tasks usually rely on **word embedding**, the representation of words. However, bias in the words can also propagate to its word embedding, thus resulting in a bias representation. The study of the bias in word embedding is of a great interests nowadays. In this thesis, we will focus on methods of finding the bias hidden in the word embedding with **alignment**.

This thesis consists two parts, the first part, we will focus on the general introduction to the word embedding, biases and the alignment technique. more specifically, with the first chapter, we will introduce the word embedding and the semantic and syntactic property. In the chapter part, we will introduce the theoretical background of the optimization and linear programming. In the third chapter, we will introduce to the biases and the diachronic evolution of the meaning of words in different years. While in the second part of this thesis, we will focus on different alignments found by different techniques and we compare their performances of finding the diachronic change between the word embedding of English words in year 1890 and 1990. Later we will focus on the zero norm formulation and try to implement different technique to reduce the computational time. In chapter four, we will construct the alignments and similarity measure. In the last two chapters we will perform the experiment and we compare different algorithms for computation reduction.

Contents

Listing of figures

Listing of tables

Listing of acronyms

FTC	Fundamental Theorem of Calculus

1 Introduction

Before diving into the thesis, the first thing to be done is to clarify the term **word embedding**. The word embedding is essentially a vectorial representation of a word which is widely used in Computer Science and also NLP. Mathematically speaking, it is a mapping e from a vocabulary of N words $D = \{w_i\}_{i=1,\ldots,N}$ to a d-dimensional space $V = \{\vec{w_i}\}_{i=1,\ldots N}$ [5] and this mapping can be represented as:

$$e : \quad D \longrightarrow V$$

With the above formulation, we can map a word into a vector representation. Notice that $V \cong R^d$, the selection of dimensionality for word embedding is a well-known open problem. In most NLP applications, the dimensionality d is either selected ad hoc or by grid search, however, in this thesis, we will use $d = 300$ as it is not only the most common choice of d [6], but we also want to stay aligned with groundbreaking papers such as [7], [8] and [9].

1.1 Distributional hypothesis and vector semantics

In the 1950s, a hypothesis called the **distributional hypothesis** was first formulated by linguists such as Joos, M. (1950)[10], Harris (1954) [11] and Firth (1957) [12]. Within this hypothesis, it stated that words in similar contexts tend to have similar meanings, the link between similar-

ity in how words are distributed and similarity in what they mean is called the distributional hypothesis. Based on this hypothesis, we can introduce the so-called **distributional semantics** which adopts the idea that it is possible to study the linguistic elements starting from the distributional hypothesis. In a 1957 paper, *The measurement of Meaning* [**?**], they notice that it is possible to retrieve a 3-dimensional vector from a word.
Harris (1954) in his work *Distributed Structure* made another important contribution, he wrote:

> Here we will discuss how each language can be described in terms of a distributional structure, i.e. in terms of the occurrence of parts (ultimately sounds) relative to other parts, and how this description is complete without intrusion of other features such as history or meaning.

Furthermore, Joos (1950) and Firth (1957), they found out that the words that are being used in the same context are more likely to have a close semantic similarity. Now, we can provide with some examples.

1. Can I use your ......?
2. Sure, the ......is in the living room.
3. Shhh, he's on the ....
4. Pick up the ......and call me.

For the examples above, there is a lot of options of words that can be used in some of these contexts. However, in order to find words that can appear in all of them are more likely to be synonymous. Let us give some examples, for words such as *sofa* will be perfectly fine option for sentences 1, 2 and 3. The word *TV* appears to be more suitable for sentences 1 and 2. And the word *bathroom* can appear in only the first sentence. Furthermore, word such as *volcano* cannot appear in any of these sentences above while words such as *telephone* and *dog and bone* have exactly the same meaning and thus they can be filled into all of these sentences.

Another term that is useful for this thesis is the so-called **vector semantics**, it instantiates the distributional hypothesis by learning representations of the words which is also called the

embedding, directly from the distributions in texts. In order to obtain the embedding representation for the words, linear algebra and machine learning tools are necessary. The word embedding can be retrieve using different algorithms, and the algorithm tries to match each word in the vocabulary to its unique vectorial representation or namely their word embedding. This will be discussed in more detail in the following sections.

1.2 Semantic and syntactic property of embedding

In the previous section, we have discussed in general about word embedding, distributional hypothesis which is related to the semantic meaning of the words. In this section, we will move to the semantic and syntactic properties of embedding, but before starting it, we will first discuss the metric to determine the similarity between embedded words.

1.2.1 Cosine Similarity

When determining the similarity of two vectorial representations, there are various ways of doing so. The most common one can be using the Minkowski distances which is defined as:

Definition 1 *The **Minkowski distance** of order p (p an integer) between two points*

$$X = (x_1, x_2, \dots, x_n) \quad and \quad Y = (y_1, y_2, \dots, y_n)$$

is defined as:

$$L^p(X, Y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}}$$

For the same points given, the Minkowski distance might change according to the order p, an popular choice of p is when $p = 2$, in this specific case, the Minkowski distance is the Euclidean distance.

Even though, the Minkowski distance is a really easy choice for the similarity identification, however, it still faces some difficulties such as the proper choice of the order p. Thus, here we will use another metric which instead of determining similarity of representations based on the distance, we use the angle between vectors to perform the task, and this new method is called the **cosine similarity**. Notice that, we have successfully avoid choosing the order p as the cosine similarity does not measure the distance in the first place, it considers only the cosine of the angle between two vectors, which means that two words that have a smaller angle are more similar in space. Now, we will give the definition of the cosine similarity.

Definition 2 *Let v, $w \in D$ be two words and let $\vec{v}, \vec{w} \in V$ be the corresponding vectors. The* ***cosine similarity*** *between $\vec{v}$ and $\vec{w}$ can be computed as:*

$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{||\vec{v}||_2 ||\vec{w}||_2}$$

Due to the property of the cosine function, we know that the cosine similarity $\cos(\vec{v}, \vec{w}) \in [-1, 1]$. With this technique, the denominator of the function performs a normalization over the dot product, which can provide us a more robust identification of two embedding. In this thesis, since the embedding of each word is normalized to 1 and the denominator of equation (2) takes value 1 and it can be then be rewritten in a simplified form:

$$\cos(\vec{v}, \vec{w}) = \vec{v} \cdot \vec{w}$$

Here, we will give some examples of the cosine similarity measure taking from [13] by Atienza (2018) which considers a 50-dimensional embedding vectors.

$$\cos(\overrightarrow{father}, \overrightarrow{mother}) = 0.890903844289$$
$$\cos(\overrightarrow{ball}, \overrightarrow{crocodile}) = 0.890903844289$$

Let us do a quick check of the example above. For words *father* and *mother*, they are clearly similar words as they can be labeled within the same semantic class, for example, they are the parent of a person. Indeed, the cosine similarity give us a very high result that is close to 1. And the words *ball* and *crocodile* belong to different semantic class, thus should return a low similarity value, indeed, we get 0.274.

1.2.2 Analogy of word embedding

One important semantic property of word embedding is the so-called **analogy**. It can capture the relational meanings between the word embedding [14] through vector arithmetic. A word analogy is essential a statement of the form *'x is to y as a is to b '*. The word *a* and *x* can be transformed to get *b* and *y* in the same way, and vice-versa. Since this transformation is invertible, thus, we can state:

Definition 3 *A word analogy f is an invertible transformation that holds over a set of ordered pairs S if and only if*

$$\forall x, y \in S, \quad f(x) = y \wedge f^{-1}(y) = x.$$

when f is of the form $\vec{x} \rightarrow \vec{x} + \vec{r}$*, it is a* ***linear word analogy****.*

Let us give another definition.

Definition 4 *If we consider three words, a, b, x. We can find the solution y of the analogy a : x = b : y(a is to x as b is to y) by computing* $\operatorname{argmin}_x L^2(y, \vec{b} + \vec{x} - \vec{a})$ *in which the term* L^2 *represents the Euclidean distance.*

Now, let us report some examples from [15], within this pre-trained 50-dimensional embedding space, we have the following analogy:

$$\vec{woman} - \vec{man} + \vec{son} \approx \vec{daughter}$$

This analogy does make sense as the relational meaning between the pair of words (*woman*, *man*) and (*daughter*, *son*) are captured. More specifically, for both pairs, they have the form of (*female*, *male*). Another example can be:

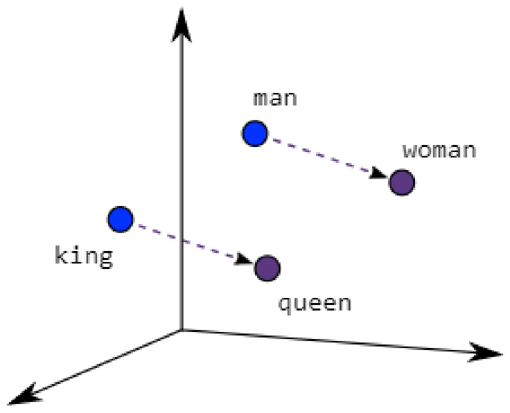
$$\vec{China} - \vec{Beijing} + \vec{Tokyo} \approx \vec{Japan}$$

With this analogy, again it captures the semantic relation of the pairs (*China*, *Beijing*) and (*Japan*, *Tokyo*). They are both have the form (country, capital). One last example we will give is as follow:

$$\vec{worst} - \vec{bad} + \vec{big} \approx \vec{biggest}$$

Again, within this analogy, the semantic relation of the pairs (*worst*, *bad*) and (*biggest*, *big*) as they both have the form (superlative, basic).

In order to show better the concept, here, we will cite [16] another example with figure (**??**).



With the figure above, we can see how the words *king* and *man* shift to words *queen* and *woman* in the same manner.

1.3 Embedding models

Until now, we have discussed briefly about the basics of the word embedding and its semantic properties, however, we haven't discussed how to obtain the word embedding. Within this section, we will dive deeper into the relevant models for retrieving word embedding.

The easiest on is the so-called **one hot encoding** which takes each word into its own class represented by a d-dimensional unit vector. With this method, in order to guarantee that each pair of word and embedding vector is unique, the dimension d should be larger or equal to the number of words in the vocabulary. This setup results in a very sparse representation of words, and d is much larger than 300.

Another important and successful model is the so-called **Word2Vec** model which is first introduced by Mikolov et al. [17] in 2013. Within this paper, they proposed two novel model architectures for computing continuous vector embedding of words, namely **CBOW model** and **Skip-Gram model**. The CBOW model output a word embedding considering the words in a sliding window, moving one word at time even though the exact output method differs and we will discuss more about them in the following section.

And finally, the last model to be introduced in this thesis is the **GloVe** model [18] developed by Stanford's NLP group in 2014 which uses an unsupervised learning algorithm for obtaining vector representations for words. Training is performed only on the nonzero elements in a word-word co-occurrence matrix.

Notice that, unlike one hot encoding which give sparse vectors and large dimensionality d. Both Word2Vec and GloVe model returns vectors that are typically dense and have much smaller dimensionality d compared to the size of the vocabulary N. The embedding obtain with the models explained above can be labelled as **static embedding** as they are trained over a fixed corpus and they do not change depending on the context of the word. There is also its counterpart, the **dynamic embedding** which learns the dynamic contextual embedding, one of such model is *BERT* [5]. We will not discuss more about BERT, as in this thesis, our embedding belongs to the type of static embedding.

1.3.1 One Hot Encoding

Previously, we have introduced briefly various models for word embedding, now, we will discuss them in more detail. First, we start with the one hot encoding model. As explained before, the one hot encoding represents the words with unit vectors, hence, each word is mapped to a unit vector whose value are all zero except for the index of the word itself in the vocabulary. Since it's a unit vector, thus, the non-zero entry of the embedding will take value 1 and consequently, the norm of such word embedding is always equal to 1. Other than the previous stated problem with high dimensionality d. There is another potential issue for this method: since all the unit vectors are orthogonal to each other and thus they are dissimilar to each other, in case of classification, this might be beneficial as we want the different class to be as separable as possible. However, it can be an potential limitation if we want to preserve some similarity.

Now, let us give an example of one hot encoding so that it can be understood better. Let us consider the following sentence and let us call it D:

I left home last summer and I went to Paris and Rome.

Then length or the cardinality of sentence D is: $|D| = 10$, which means that, in order to preserve the one to one correspondence between word and word embedding vector, we must represent these word in $R^1 0$ as there are 10 unique words in this sentence. Let us represent these word with one hot encoding and we consider the order of the appearance of the word in the sentence as the position of the entry of the unit vector, and thus we have:

I	[1 0 0 0 0 0 0 0 0 0]
left	[0 1 0 0 0 0 0 0 0 0]
home	[0 0 1 0 0 0 0 0 0 0]
last	[0 0 0 1 0 0 0 0 0 0]
summer	[0 0 0 0 1 0 0 0 0 0]
and	[0 0 0 0 0 1 0 0 0 0]
went	[0 0 0 0 0 0 1 0 0 0]
to	[0 0 0 0 0 0 0 1 0 0]
Paris	[0 0 0 0 0 0 0 0 1 0]
Rome	[0 0 0 0 0 0 0 0 0 1]

And finally, let us summarize the pros and cons of one hot encoding. The advantage of this technique is that it is highly intuitive and very easy to compute. Recall that, it might requires more memory because as the size of the vocabulary increases, the dimensionality increases accordingly and will result in a larger memory consumption. Another disadvantage is that it does not preserve the inner relation between the words and by its construction, it does not exploit the context of the words as it can be seen as a random assignment of unit vector to the vocabulary. As explained before, due the orthogonality between the unit vectors, all the embedding are equally irrelevant to each other, which makes the embedding fail at providing meaning measure. For example, *Beijing* and *Tokyo* are both capitals and for sure we expect them to be more similar than *Beijing* and *apple*. For all the reasons stated above, within this paper, we will not use one hot encoding for word embedding.

1.3.2 Word2Vec

The Word2Vec mainly contains two models. The first one is the CBOW model which is short for Continuous Bag-of-Words model. With this method, it outputs the target word according to its surrounding words which is called **context words** [19]. Let us give an example of the context words. Let us consider a text corpus with a sliding window of size h, it can slide one word forward each time. At each time instance, we focus on the center word of the sliding window and we output based on all the other $h - 1$ surrounding words and these words are the context words. Let's consider a sliding window of the size $h = 5$ and let us assume we have the following sentence:

I love giraffes because I love their long necks

Let us consider the first 5 words as the size of the sliding window is 5, then we have the center word of this time instance is then the word *giraffes*. Notice that our corpus has 7 unique words, namely, *I, love, giraffes, because, their, long, necks*. Representing them with one hot encoding requires a 7-dimensional representation. Let us follow the order of these 7 unique words and with one hot encoding, we have:

I	[1 0 0 0 0 0 0]
love	[0 1 0 0 0 0 0]
giraffes	[0 0 1 0 0 0 0]
because	[0 0 0 1 0 0 0]
their	[0 0 0 0 1 0 0]
long	[0 0 0 0 0 1 0]
necks	[0 0 0 0 0 0 1]

In the first time instance, our context words are *I, love, because, I*. For the current center word *giraffes*, its output is thus calculated as the average of the embedding of the context words, and we have the output of *giraffes*.

Notice that, in the calculation above, we simply perform the sum of the vectors, thus in this case, the order of the vectors doesn't change the result.

$$\left(\begin{array}{ccccccc} I & & love & & because & & I \\ \begin{bmatrix}1\\0\\0\\0\\0\\0\\0\end{bmatrix} & + & \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\end{bmatrix} & + & \begin{bmatrix}0\\0\\0\\1\\0\\0\\0\end{bmatrix} & + & \begin{bmatrix}1\\0\\0\\0\\0\\0\\0\end{bmatrix}\end{array}\right)/4 = \begin{bmatrix}0.5\\0.25\\0\\0.25\\0\\0\\0\end{bmatrix}$$

Let us dive into the model that trains for the embedding. Here, a neural network is used for this task which is fully connected. It has three different layers, namely, the **Input layer**, **Hidden layer** and **Output layer**. The Schematic representation of the CBOW architecture is showed in figure (**??**). We can see that the input context word vector $\vec{x}$ is fed to the input layer and then it passes through the fully connected layers with ReLu activation function to Hidden layer, produces a hidden representation of the input, later it further passes to the output layer with again fully connected layers activated with the softmax function, and finally at the output, we have the predicted word vector of the center word $\vec{y}$. The relevant hyperparameters of this network are the dimensionality of the embedding d and the cardinality of the vocabulary $|D = N|$. From now on, we will fix the letter d for the dimensionality of the embedding and the letter N as the number of words of the vocabulary. Moreover, we have also bias terms a, b and the weight matrices **M** $\in R^{d \times N}$, **N** $\in R^{N \times d}$. CBOW uses a **cross entropy loss** as its loss function and it is defined as:

$$J = -\sum_{k=1}^{N} \vec{y_k} \log(\vec{y_k})$$

By the end of the training process, we learn the optimal weight matrices **M** and **N**, the final word embedding is then given by the average of **M** and **N** transposed.

Another architecture is the **Skip-Gram model** which acts as the reverse of CBOW. The skip-gram model seeks to classify whether the center word belongs to the context or not based on another word in the sentence. In order to demonstrae better the concept, let us give the following example. Consider the following sentence:

. . . lemon, a tablespoon of apricot jam, a pinch . . .

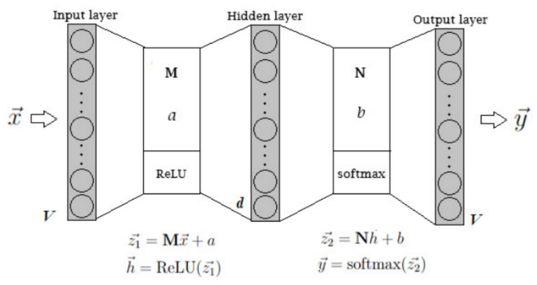

Input layer
Hidden layer
Output layer
$\vec{x}$
$\vec{y}$
M
a
ReLU
N
b
softmax
V
d
V
$\vec{z}_1 = \mathbf{M}\vec{x} + a$
$\vec{h} = \mathrm{ReLU}(\vec{z}_1)$
$\vec{z}_2 = \mathbf{N}\vec{h} + b$
$\vec{y} = \mathrm{softmax}(\vec{z}_2)$

Figure 1.1: CBOW architecture.

Let us now focus on the central part of this sentence and consider a sliding window of size $h = 5$ that is the words between *tablespoon* and *a*. The center and the target word for this sliding window is thus the word *apricot*.
Considering a general pair of words (w, c) of the corpus, in which c is the possible context word. Our classifier tries to classify if word c is a real context word based on the probability $P(+|w, c)$. With the example above, we should expect a positive classification if c $=$ *jam* and return negative if c $=$ *air*. The intuition behind this is that a word is likely to be close to the target if its embedding vector is also similar to the target ones. With the cosine similarity measure introduced in the previous section, now we can quantity the similarity between word w and c and we can finally get the probability $P(+|w, c)$:

$$P(+|w, c) = \sigma(\vec{w} \cdot \vec{c}) = \frac{1}{1 + e^{-\vec{w} \cdot \vec{c}}}$$

where σ is the sigmoid function. Notice that the cosine similarity inside is generally higher when words are close to each other, thus, correspond to a higher probability value. Assumption that all context words are independent is make for the skip-gram model, thus, the probability of a set of h words $c_1, \dots, c_h$ are words in a real context window for w is:

$$P(+|w, c_1, \dots, c_h) = \prod_{i=1}^{h} \sigma(c_i \cdot w)$$

The model considers two embedding for target and contest word. The two embedding are then gathered in matrices $W \in R^{d \times N}$ and $C \in R^{d \times N}$. Later they will be concatenated and return the matrix θ.

The skip-gram model starts by assigning a random embedding vector for each of the N word in the vocabulary. Then, it modifies the embedding of each word iteratively by trying to make nearby words to have similar embedding, in another word, given a word w and other words close in the text, the embedding of w shifts to be more like the embedding of the nearby words and less like the embedding of the word that occur far away from w. Here, we have a binary classifier, in order to train this binary classifier, we will use the **Skip-Gram with Negative Sampling(SGNS)**. For training, we will need two set of samples, a positive set and a negative set of the form (w, c). Let us denote them as:

$$S_+ := \{(w, c_+)\}\textbf{and} \quad S_- := \{(w, c_-)\}$$

Going back to the previous example, for the target word *apricot*, word pairs such as (*apricot*, *jam*) and (*apricot*, *tablespoon*) are in the positive set S_+. While word pairs such as (*apricot*, *space*) and (*apricot*, *air*) should be in the negative set S_-. In the training, we usually have more negative class words than positive class words as the corpus is usually really large, and for this reason, we put the term Negative Sampling in the name of this method. This algorithm has one hyper-parameter which is the ratio of the number of positive sample pairs and the number of negative sample pairs, and we denote it as k. Thus, for each positive sample, k negative sample are generated according to the weighted unigram frequency of w. The definition of the loss function takes into consideration both the idea of maximizing similarity between positive pairs and at the same time minimizing similarity between negative paris for each word w in the corpus, and the loss can be written as:

$$\begin{aligned} L(\theta) &= -[\log(P(S_+|w, c)) + \sum_{i=1}^{k} \log(P(S_-|w, c))] \\ &= -[\log(\sigma(\vec{w} \cdot \vec{c})) + \sum_{i=1}^{k} \log(\sigma(-\vec{w} \cdot \vec{c}))] \end{aligned} \tag{1.1}$$

For minimization of this loss function, one can use the stochastic gradient descent. And another hyper-parameter should be mentioned is the size of the sliding window h. A smaller sliding window forces a similar word embedding with the same semantic role, for example *Paris* and *Rome* While a large gives the topical relatedness, such as *pope* and *Rome*. Hence, usually the size of the sliding window is a moderate value which is less or equal to 10.

1.4 The alignment

Mathematically speaking, the **word alignment** is essentially a mapping function A between two vector spaces of two different set of word embedding. Considering two sets of words and their corresponding set of word embedding, the word embedding of word in the set can be defined as:

$$e_1 : D_1 \rightarrow V_i \quad e_2 : D_2 \rightarrow V_2$$

in which D represents the vocabulary sets and we have $V_1 \cong R^{d_1}$ and $V_2 \cong R^{d_2}$. In this thesis, we will consider the dimensionality of the two embedding to be the same. Given two word $x \in D_1$ and $y \in D_2$, the alignment between the embedding space will try to look for a linear map A of the form [20]:

$$A : V_1 \rightarrow V_2 \quad \textbf{s.t.} \quad A(e_1(x)) \approx e_x(y)$$

This method is widely used in the **cross-lingual embedding models** as we can set the two vocabularies D_1 and D_2 to be the vocabularies of two different languages, however, its application can also expand to the embedding of the same language [21]. We will further discuss it in the following section.

The joint embedding space of the cross-lingual embedding is the main application of the cross-lingual embedding model. The **cross-lingual transfer** is really useful for some NLP task. It is composed by modelling on data of one language and then applying to another based on shared cross-lingual features.

Let us take **sentiment analysis** for an example of the cross-lingual task. It has to determine the polarity of the sentiment such as positive and negative of text in different languages [22]. Mogadala and Rettinger [23] evaluate their embedding with the multilingual Amazon product review dataset of Prettenhofer and Stein [24]. Another widely used application is the **machine translation** task, which seeks to translate the entire text of corpus into another language. One example is done by Zou et al. [25] who uses a phrase-based machine translation to evaluate the embedding.

Before diving into the word alignment models, we will first classify the alignment. They can be classified based on the input and output. For example, we can have the **Word to Word** alignment method, which takes an embedding in the first embedding space e_1 and output the embedding of semantically similar token in the second embedding $e_2(x^*)$. When we want to input and output sentences, we then have the so-called **Sentence to sentence** alignment, we can further expand it to the document class and we have the **document alignment model**. Since for thesis, we use only the word to word alignment model, thus, we will not discuss further about the sentence alignment and the document alignment.

1.4.1 Word alignment

For word embedding trained on different text corpora, they exhibit similar geometric patterns and behaviors. This phenomenon is first observed by Mikolov [26] in 2013. With this idea, a hypothesis of the transformation of the embedding space can be an linear operation. Now, we will introduce various mapped based methods for finding the word alignment. We will consider two different sets of embedding e_1 and e_2 training on two different corpora D_1 and D_2. We try to search for a linear map A by finding the corresponding transformation matrix of the two embedding spaces W:

$$A : V_1 \longrightarrow V_2$$

and we have that $\vec{x} \rightarrow W\vec{x} \approx \vec{y}$ in which $\vec{x} = e_1(X) \in D_1$ and $\vec{y} = e_2(y) \in D_2$ are the corresponding word embedding of words x and y and $W \in R^{d \times d}$. It means that for a cross-lingual alignment, it gives us the translation of word x as y. As for the embedding space of the same language, words x and y are simply the same word with different representations. Furthermore, since the linear map is a bijection, thus, we can make the following assumption:

- Two vocabularies have the same cardinality, $D_1| = |D_2|$ and

- For all the words in the first vocabulary, there exists a corresponding y in the second vocabulary that is $\forall x \in D_1, \quad \exists y \in D_2$.

The first method we will introduce here is the **regression method**, the alignment is obtained by maximizing the similarity between embedding. In the work done be Mikolov et al. [26] in 2013, he showed that using the n most frequent words in two vocabularies, the linear transformation matrix W can be obtained by minimizing the mean squared error between word embedding $\vec{x}$ and $\vec{y}$ using stochastic gradient descent. More specifically, the function is:

$$\Omega_{REG}(R) = \sum_{i=1}^{n} ||R\vec{x_i} - \vec{y_i}||^2 \quad \textbf{with} \quad R \in R^{d\times d}$$

It can be further written into the matrix form:

$$\Omega_{REG}(R) = ||RX - Y||_F^2 \quad \textbf{with} \quad R \in R^{d\times d}$$

in which matrices $X, Y \in R^{d\times N}$ are the embedding matrices of the two vocabularies. For each column of the matrices X and Y, the column corresponding to the same index in both embedding are the corresponding embedding $\vec{x_i}$ and $\vec{y_i}$. The $||\,||_F$ is the Frobenius norm. And finally, we will minimize this objective function and then obtain the alignment matrix W:

$$W = \underset{R\in R^{d\times d}}{\text{argmin}}\, \Omega_{REG}(R)$$

Further regularization such as l_2 regularization has been added by Ruder et al. [22] in 2019.

Another method is the **orthogonal methods**. This idea was proposed by Ruder [22] by adding the orthogonal constraints to the regression model. And our problem now becomes a classic problem in Data Science which is the **orthogonal Procrustes problem**.

$$W = \underset{R\in R^{d\times d}}{\text{argmin}} ||RX - Y||_F \quad \textbf{subjected to} \quad R^T R = I_d$$

The closed form solution to the procrustes problem is found by Schöneman [27] in 1966. He solved it with the singular value decomposition and it can be solved efficiently, and the solution is:

$$W = VU^T \quad \textbf{where} \quad Y^T X = U\Sigma V^T$$

We will also report the proof here.

$$\begin{aligned}
R &= \underset{\Omega}{\operatorname{argmin}} \|\Omega A - B\|_F^2 \\
&= \underset{\Omega}{\operatorname{argmin}} \|A\|_F^2 + \|B\|_F^2 - 2 < \omega A, B >_F \\
&= \underset{\Omega}{\operatorname{argmax}} < \Omega A, B >_F \\
&= \underset{\Omega}{\operatorname{argmax}} < \Omega, BA^T >_F \\
&= \underset{\Omega}{\operatorname{argmax}} < \Omega, U\Sigma V^T \\
&= \underset{\Omega}{\operatorname{argmax}} < U^T \Omega V, \Sigma >_F \\
&= \underset{\Omega}{\operatorname{argmax}} < S, \Sigma >_F \quad \textbf{where} \quad S = U^T \Omega V
\end{aligned}$$

Since matrix S is an a product of orthogonal matrices, thus in order to maximize it, the matrix S must be an identity matrix, Thus,

$$\begin{aligned}
I &= U^T R V \\
R &= UV^T
\end{aligned}$$

and the matrix R is our final output alignment matrix W.

The orthogonal constraints has been used by various researchers. For example, Xing et al [28] in his work *Normalized word embedding and orthogonal transform for bilingual word translation* states that the orthogonal constraints leads to the preservation of length normalization. Other researcher such as Artetxe et al. [29] motivates it as a means to ensure monolingual invariance.

2 Linear Programming and Frank Wolfe method

2.1 Introduction to Linear Programming

Within this section, the basic idea of Linear Programming will be introduced as it will be required by the subsequent tasks. A **Linear Programming problem** is a minimization/maximization problem subjected to linear constraints formulated by linear equations and/or linear inequalities, which takes the general form of:

$$\begin{aligned} \max \quad & c_1x_1 + \cdots + c_nx_n \\ \text{s.t.} \quad & a_{11}x_1 + \cdots + a_{1n}x_n \sim b_1 \\ & \quad \vdots \qquad \vdots \\ & a_{m1}x_1 + \cdots + a_{mn}x_n \sim b_m \end{aligned} \tag{2.1}$$

in which the symbol $\sim$ is used to represent operators such as $\geq$, $\leq$ and $=$ and the terms a_{ij}, $b_i, c_j \in \mathbb{R}$ (i = 1, ..., m, j = 1, ..., n) are the corresponding parameters. Here, we only use the maximization problem, however, a minimization problem is equivalent to a maximization problem which means that we can obtain the result of a minimization problem denoted with $\min f(x)$ by solving a maximization problem $\max -f(x)$. Notice that the constraints with $\sim$ forms a region where the possible solution locates, and we call it the **feasible set** or **feasible region**. A vector x $\in \mathbb{R}^n$ in the feasible set is called a **feasible solution**. A feasible solution is a solution to the linear programming problem but not necessarily the best solution. Let us denote the vector $\tilde{x} \in \mathbb{R}^n$ an **optimal solution** of the linear program problem if vector $\tilde{x}$ is a feasible solution and if given a random feasible solution x in the feasible set, the inequality $c^T\tilde{x} \geq c^T x \forall$ holds true. The corresponding value $c^T\tilde{x}$ is called the **optimal value** of this linear program.

Furthermore, there is more compact form of the linear program which we call it a **standard form**, which is:

$$\begin{aligned} \max \quad & c^T x \\ \textbf{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{2.2}$$

in which A $\in \mathbb{R}^{m \cdot n}$, b $\in \mathbb{R}^m$, c $\in \mathbb{R}^n$ and x $\in \mathbb{R}^n$ a vector that contains the variables of the problem.

Before reporting some important theorems for the linear programming which are called **Fundamental Theorem of Linear Programming**, we first report some basic definitions.

Definition 5 *A **closed half-space** in R^n is a set having the form $\{x \in R^n : a^T x \leq \beta\}$ where $a \in R^n \backslash \{0\}$ and $\beta \in R$. If we it explicitly, we have $a_1 x_1 + \cdots + a_n x_n \leq \beta$*

Definition 6 *A **hyperplane** in R^n is a set having the form $\{x \in R^n : a^T x = \beta\}$ where $a \in R^n \backslash \{0\}$ and $\beta \in R$.*

Definition 7 *A **polyhedron** is the intersection of a finite number of closed half-spaces. Equivalently a polyhedron is a set Ω that can be written in the following form:*

$$\Omega\{x \in R^n : Ax \leq b\}$$

with A$\in R^{m \times n}$andb $\in R^m$.

Furthermore, an extreme point is called **vertex** when dealing with a polyhedron. Now, we further introduce a proposition for the connection between polyhedron and convex set.

Proposition 1 *The following properties hold:*

- *every closed half-space is convex*
- *intersection of convex sets is a convex set*
- *every polyhedron is convex*

With proposition above, we know that a linear programming problem is also a convex problem. And finally we can introduce the fundamental theorem of linear programming which is:

Theorem 1 *Given a linear program, one and only one of the following alternatives holds:*
(a) the problem has an optimal solution (which is a vertex of the polyhedral feasible set) at least;
(b) the problem is unfeasible, i.e., it has no feasible solutions;
(c) the problem is unbounded, i.e., for every $K \in \mathbb{R}$, there exists a feasible solution x such that $c^T x \geq K$ ($c^T x < K$ for minimization problems).

With the theorem above, we can now solve a linear programming problem with an efficient way other than the brute force search which is by check only the vertex of the polyhedral feasible set as if the optimal solution exists, it is for sure one of the vertex of the polyhedron.

2.2 Nonlinear Programming

A nonlinear programming problem is a problem such that minimizes/maximizes a nonlinear objective function subjected to constraints formulated by inequalities or equalities. For the sake of simplicity, we will fix the notation first before going further. Let us denote the feasible set as Ω and the objective function as $f\colon \Omega \longrightarrow \mathbb{R}$. Then the problem can be represented as:

$$\min f(x) \quad \textbf{s.t.} \quad x \in \Omega \tag{2.3}$$

and

$$\max f(x) \quad \textbf{s.t.} \quad x \in \Omega \tag{2.4}$$

Here we will report some definitions hold for the problem of the form 2.3. As explained in the chapter before, since these minimization and maximization problem can be transformed, thus they also hold for the maximization problem of the form (2.4).

Definition 8 *An optimization problem of the type* (2.3) *is said to be infeasible if* $\Omega = \Phi$*, that is, if there are no feasible solutions.*

Definition 9 *An optimization problem of the type* (2.3) *is said to be unbounded(below) if however, if you choose a value* $M \geq 0$*, there is a point* $x_M \in \Omega$ *such that* $f(x_M) \leq -M$*.*

Definition 10 *An optimization problem of the type* (2.3) *is said to have an optimal solution if there exists a* $x^* \in \Omega$ *such that it results* $f(x^*) \leq f(x) \forall x \in \Omega$*. The corresponding value* $f(x^*)$ *is the optimal value.*

We can classify the optimization problem into three different types according to the structure of the feasible set Ω.

- **Continuous Optimization problem** where we have real value variable $x \in R^n$, thus, we have the feasible set $\Omega \subseteq R^n$.
- **Integer Optimization problem** whose variables takes integer values $x \in Z$, thus, we have that the feasible set $\Omega \subseteq Z^n$.
- **Mixed Integer problem** with a subset of variables are constrained to be integer.

In this thesis, the test dataset requires variable to take real values, thus, only the continuous optimization will be discussed.

2.2.1 Continuous Optimization

Here, we represent some relevant definitions.

Definition 11 *A point $x^* \in \Omega$ is a* ***global minimum*** *for f in Ω, if $f(x^*) \le f(x) \quad \forall x \in \Omega$.*

Definition 12 *A point $x^* \in \Omega$ is a* ***strict global minimum*** *for f in Ω, if $f(x^*) \le f(x) \quad \forall x \in \Omega, x \neq x^*$.*

Definition 13 *A point $x^* \in \Omega$ is a* ***local minimum*** *for f in Ω, if there exists a neighborhood $B(x^*;\rho)$, with $\rho > 0$ such that $f(x^*) < f(x)\, \forall x \in \Omega \cap B(x^*;\rho)$.*

Now, let us consider a continuous optimization problem with the following form:

$$\begin{array}{c} \min \quad f(x) \\ x \in \omega \end{array} \tag{2.5}$$

where $f\colon \Omega \to R$ is a continuous function in R^n. When minimizing this objective function, for the consideration of the computation, it is crucial that we find a proper set of direction to go for our algorithm, and these directions are the so-called the **descent directions** and the necessary condition for the optimality which in this thesis, we will use the **first order optimality condition**. The descent directions help us to determine a proper direction for our algorithm to proceed and the optimality condition can help us to identify such directions. In order to understand better, let us consider the following definitions and proposition.

Definition 14 *We define the set of descent directions for f in $\bar{x}$ as follows:*

$$D(\bar{x}) = \{d \in R^n : \exists \delta > 0 \quad \textbf{\textit{s.t.}} \quad f(\bar{x} + \alpha d) < f(\bar{x}), \forall \alpha \in (0, \delta)\}$$

Proposition 2 *Let f be a continuous differentiable function. If $x^8 \in \Omega$ is a local(global) minimum for our problem, then*

$$D(x^8) = \Phi$$

Considering both definition (14) and proposition (2), we can get the following results.

Proposition 3 *Assume that f: $\Omega \to R$ is a continuous differentiable and let $d \in \Omega$ be a nonzero vector. If we have:*

$$\nabla f(x)^T d < 0$$

then d is a descent direction for f in x. If we further have that f is a convex function and d is a descent direction for f in x, then the condition above is satisfied.

The proposition above gives us a characteristic of the first order information of the descent directions. The second result is:

Theorem 2 *Let f be a continuous differentiable function. If $x^* \in \Omega$ is a local(global) minimum of problem* (2.5)*, then*

$$\nabla f(x^*) = 0$$

The theorem (2) is really useful for proving the convergence of the Frank-Wolfe algorithm which will be explained in more detail in the following chapter.

2.2.2 Existence condition

For the completeness and well-posedness of the problem, we will now discuss about the existence conditions of the continuous optimization problem. In fact, we are not guaranteed to have the minimum point to be in the feasible set Ω, we have:

- $\Omega = \Phi$
- $\Omega \neq \Phi$, but function f is unbounded from below on Ω, i.e. $inf_{x \in \Omega} f(x) = -\infty$
- $\Omega \neq \Phi$, f is bounded from below on Ω, but there exists no global minimum point for f in set Ω.

Hence, now, we can establish the sufficient conditions for the existence of a global minimum of the the continuous optimization problem. First, we will introduce a new proposition related to the existence of solution given a continuous optimization problem.

Proposition 4 *Let $\Omega \subset R^n$ be a non-empty and compact set. Let f be a continuous function defined on Ω. Then there exists a global minimum point for f in Ω.*

Notice that this proposition holds true only to problem with compact feasible set. Furthermore, for the constrained problem, since we have that $\Omega \subset R^n$, thus if we have closed and limited set Ω, then it is also compact. In case of a non-constrained problem or if set Ω is closed but not limited, in order to make the solution exists, the identification of some subset of set Ω that contain the optimal solution of the continuous problem is required.

Let us report more useful definitions and propositions. The first definition we will give is the **level set**

Definition 15 *Let $\Omega \subseteq R^n$ and f: $\Omega \to R$. A **level set** for f on Ω is a non-empty set of the form:*

$$L(\alpha) := \{x \in \Omega : f(x) \leq \alpha\}$$

in which $\alpha \in R$. Essentially, the level set represents the set of input variables corresponding to the value of the objection function lower than a threshold value. For example, give a random point x $\in \Omega$. The corresponding level set L(f (x)) corresponds to the set of points x_i which has an corresponding objective function value lower than $\alpha = f(x)$.

Now, we can introduce more propositions.

Proposition 5 *Let $\Omega \subseteq R^n$ and f be a continuous function defined on Ω. Suppose there exists a level set of f on Ω that is not empty and compact. Then there exists a global minimum point for f in Ω.*

Proposition 6 *Let $\Omega \subseteq R^n$ and f be a continuous function defined on Ω. Then all the level sets $L(\alpha) := \{x \in \Omega : f(x) \leq \alpha\}$ of f in Ω are compact if and only if the following condition is satisfied:*

- *Let $\{x_k\}$ be a sequence of $x_k \in \Omega$ such that* $\lim_{k\to\infty}\|x_k\| = \infty$*, then it follows that*

$$\lim_{k\to\infty} = \infty$$

Combining the two above mentioned propositions, now we have:

Proposition 7 *Let $\Omega \subseteq R^n$ be a closed set and let f be a continuous function on Ω and assume that f is coercive on Ω, that is*

$$\lim_{k\to\infty} f(x_k) = \infty$$

for each sequence $\{x_k\}$, with $x_k \in \Omega$, such that $\lim_{k\to\infty\|x_k\|} = \infty$*. Then there exists a global minimum of f on Ω.*

With the last proposition, now we can conclude the part of the sufficient condition for the existence of the continuous optimization problem with a closed feasible set Ω.

2.2.3 Convex and concave programming

Convex programming problems deal with convex optimization. For the introduction of the discussion, here, we will report more relevant definitions.

Definition 16 *Let us consider a set $C \subseteq R^n$. C is a **convex set** if, $\forall\, x, y \in C$ and $\forall \alpha \in [0, 1]$, it holds:*

$$\alpha x + (1-\alpha)y \in C \tag{2.6}$$

Definition 17 *Let set $C \subseteq R^n$ be a convex set and let f: $C \longrightarrow R$. f is convex on **C** if, given any x, $y \in C$ and $\alpha \in [0, 1]$, we have that*

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y) \tag{2.7}$$

*Then, f is **strictly convex** on C if, given any $x, y \in C$, with $x \neq y$ and $\alpha \in (0, 1)$ have that*

$$f(\alpha x + (1-\alpha)y) < \alpha f(x) + (1-\alpha)f(y) \tag{2.8}$$

Definition 18 *A point $x \in R^n$ is a* ***convex combination*** *of points $v_i, \ldots, v_m \in R^n$ if we have:*

$$x = \sum_{i=1}^{m} \alpha_i v_i, \quad \sum_{i=1}^{m} \alpha_i = 1, \alpha_i \geq 0, i = 1, \ldots, m$$

Definition 19 *A point $x \in R^n$ is a* ***proper convex combination*** *of points $v_1, \ldots, v_m \in R^n$ if we further have $\alpha_i \in (0, 1)$ for all $i \in \{1, \ldots, p\}$.*

Now, let us further report some operations preserving convexity that are useful for this thesis.

Definition 20 *Given a convex set $\Omega \subseteq R^n$ a point $\bar{x} \in \Omega$ is an* ***extreme point*** *of Ω if $\bar{x}$ cannot be given as a proper convex combination of two points in Ω. That is, if we cannot find two points $y, z \in \Omega$ such that $\bar{x} = \alpha x + (1 - \alpha)z$, with $\alpha \in (0, 1)$.*

Proposition 8 *Let $\Omega \in R^n$, be a convex set and $f_i : \Omega \to R$, $i = 1, \ldots, m$ be convex functions defined in Ω. We have that the function*

$$f(x) = \sum_{i=1}^{m} \alpha_i f_i(x)$$

with $\alpha_i \geq 0, i = 1, \ldots, m$ is a convex function over Ω. Furthermore, if there also exists an index i such that $\alpha_i > 0$ and f_i strictly convex over Ω, then f is strictly convex over Ω.

As for the concave programming problem, we have the following definitions.

Definition 21 *Let set $C \subseteq R^n$ be a convex set and let f: $C \longrightarrow R$. f is* ***concave*** *on C if, given any $x, y \in C$ and $\alpha \in [0, 1]$, we have that*

$$f(\alpha x + (1 - \alpha)y) \geq \alpha f(x) + (1 - \alpha)f(y) \tag{2.9}$$

Then, f is ***strictly concave*** *on C if, given any $x, y \in C$, with $x \neq y$ and $\alpha \in (0, 1)$ have that*

$$f(\alpha x + (1 - \alpha)y) > \alpha f(x) + (1 - \alpha)f(y) \tag{2.10}$$

Definition 22 *A **convex programming problem** is a minimization problem of the form:*

$$\min f(x) \quad s.t. \quad x \in \Omega \tag{2.11}$$

where set Ω is a convex set and f is concave on Ω.

Proposition 9 *Let $C \subseteq R^n$ be a convex set and f a convex (strictly convex) function on C. Then any(one) minimum point of f on C is also (the only) global minimum.*

Besides the convex programming problem, the **concave programming problem** is another type of minimization problem. Now we will provide more definitions about concave programming problem.

Definition 23 *A **concave programming problem** is a minimization problem of the form:*

$$\min f(x) \quad \boldsymbol{s.t.} \quad x \in \Omega \tag{2.12}$$

where Ω is a convex set and f is concave on Ω, or, equivalently:

$$\max f(x) \quad \boldsymbol{s.t.} \quad x \in \Omega \tag{2.13}$$

where Ω is a convex set and f is convex on Ω.

For both types of programming problem, when dealing with the non-strictly case, we might end up stuck in a local minimum, thus, solving such type of problem becomes harder when it a larger number of local minimum points. For the concave programming problem, it usually consists a larger number of such points, thus making it more challenging to tackle compared to the convex programming problem. Now, let us give two theorems related to the global minima of the concave programming problem.

Theorem 3 *Let $\Omega \subseteq R^n$ be a closed convex set, and let f be concave (and non-constant) on Ω. Hence, if there is a global minimum of f on Ω, this lies on the boundary of Ω.*

Theorem 4 *Let $\Omega \subseteq R^n$ be a polyhedron with at least one vertex and let f be a concave function with global minima in Ω. Then, there exists a global minimum of f in Ω that coincides with a vertex of the polyhedron Ω.*

The theorem above give us a nice way to deal with the concave programming problem with a polyhedral feasible set as one can focus on the set of vertices of the feasible set to get the global minimum of the problem.

2.2.4 Linearization of nonlinear problem

Nonlinear problem is generally more complicated and hard to solve, however, for some cases a nonlinear problem can be transformed into a linear one (2.1). For example, here in this thesis, we will report one called **linearization of absolute value problem** which will be useful in the following chapters.

The **Absolute value problem** is a type of problem of the following form:

$$\min_{x,y} \sum_j c_j|x_j| + \sum_k d_k y_k \quad \textbf{s.t.} \quad (x,y) \in \Omega \tag{2.14}$$

where set Ω is the feasible region and it is a polyhedron, and furthermore, we have that $c_j > 0$. In order to tackle this problem, first we perform the following transformation of the variables which is representing a real number with its positive and negative part:

$$x = x_j^+ - x_j^-, x_j^+ \geq 0, x_j^- \geq 0$$

The transformation above holds true for all real numbers, furthermore, we can differentiate two different cases:

- $x_j \geq 0 \Rightarrow x_j^+ = x_j + \partial = |x_j| + \partial$ and $x_j^- = \partial$
- $x_j < 0 \Rightarrow x_j^+ = \partial$ and $x_j^- = -x_j + \partial = |x_j| + \partial$

with $\partial \geq 0$. For both cases above, when $\partial = 0$, we have one of the negative or positive part will be 0 and the other one will be $|x_j|$. For the first case, when $x_j \geq 0$, we have that the negative part $x_j^- = \partial = 0$, the absolute value of $|x_j|$ equals to x_j and for the second case, when x_j is a negative number, we have that $x_j^+ = \partial = 0$ and we have the absolute value $|x_j|$ equals to -x_j. Further more, we can sum the positive part and the negative part and we have:

$$x_j^+ + x_j^- = |x_j| + 2\partial$$

We can then replace the relation above to the absolute value problem (2.14) with $\delta = 0$. Now the problem can be written as:

$$\begin{aligned} \min_{x,y} \quad & \sum_j c_j(x_j^+ + x_j^-) + \sum_k d_k y_k \\ \textbf{s.t.} \quad & (c_j^+ - x_j^-, y) \in \Omega \quad x_j^+ \geq 0, x_j^- \geq 0, \forall j = 1, \dots, n \end{aligned}$$

in which set Ω the feasible set.

Another type of transformation can be made is by represent the absolute value of x_j as the maximum between x_j and $-x_j$:

$$|x_j| = \max\{x_j, -x_j\}$$

We can also replace this formulation to the problem (2.14) and we have the following problem:

$$\min_{x,y} \sum_j c_j \max x_j, -x_j + \sum_k d_k y_k \quad \textbf{s.t.} \quad (x, y) \in \Omega$$

We can further transform the problem above by introducing a new variable z_j to our formulation. We can now replace the max with our new variable z_j, and then the original problem (2.14) becomes:

$$\min_{x,y,z} \sum_j c_j z_j + \sum_k d_k y_k \quad \textbf{s.t.} \quad z_j = \max\{x_j, -x_j\}, (x, y) \in \Omega \tag{2.15}$$

The constraint about can be further transformed as setting the variable z_j to be the maximum between between x_j and $-x_j$ is equivalent to constraining x_j to be ranging between $-z_j$ and z_j, and thus, our problem now can be rewritten as:

$$\min_{x,y,z} \sum_j c_j z_j + \sum_k d_k y_k \quad \textbf{s.t.} \quad -z_j \le x_j \le z_j, j = 1, \dots, n, (x,y) \in \Omega \tag{2.16}$$

Now let us demonstrate the linearization technique in a Data Science scenario. First suppose we want to build up a **linear regression model** of the following form:

$$y = a^T x + b$$

We have that $x \in R^n$ a real valued input vector of our model, and $y \in R$ is a real valued output of the model. Furthermore, $a \in R^n$ and $b \in R$ are the relevant parameters. In Data Science applications, we have a finite set T of the input and output sample pairs, the set T is often called the **training set**, it has the form:

$$T := (x^1, y^1), \dots, (x^m, y^m)$$

in which the superscript represents the sample of the training set T. Now we want to obtain the variables a and b giving the training set T and we hope those found variables can help us to identify the label y^j giving a new sample x^j. In order to achieve this goal, we can minimize the error over the training set T, and the error E_i for a general input-output pair in the training set T can be written as:

$$E_i = y^i - (a^T x^i + b)$$

When minimizing this error, we have various formulations to do so, a popular one is by considering the square loss over the training set and we obtain the desired parameters by solving the well-known least-square problem which has the following form:

$$\min_{a,b} \quad \sum_{i=1}^{m} (y^i - a^T x^i - b)^2$$

Another option can be done with the absolute value formulation, which is, instead of considering the square error, we seek to optimize over the absolute value of the error, and thus the formulation can be written as:

$$\min_{a,b} \quad \sum_{i=1}^{m} |y^i - a^T x^i - b|$$

Now, we can apply the linearization techniques (2.15) and (2.15) to the absolute value formulation and it can be rewritten as:

$$\min_{a,b,z} \sum_{i=1}^{m} z_i \quad \textbf{s.t.} \quad |y^i - a^T x^i - b| \leq z_i, \forall i = 1, \ldots, m$$

and

$$\min_{a,b,z} \sum_{i=1}^{m} z_i \quad \textbf{s.t.} \quad -z_i \geq y^i - a^T x^i - b \leq z_i, \forall i = 1, \ldots, m$$

This formulation is called the **Least Absolute Deviation** (LDA). Thanks to the linear constraints provided with this linearization, now, this problem can be solved with the linear programming techniques which makes it very easy to solve.

2.3 The Frank-Wolfe method

There are various techniques for solving a constrained optimization problem, here, we will report the so-called Frank-Wolfe method which is also called conditional gradient method. It is an iterative first-order optimization originally proposed by Marguerite Frank and Phili Wolfe in 1956 to solve quadratic programming problem with linear constraints. It is a very popular method for solving constrained optimization problem as it is simple to solve and unlike the projected methods, it does not require the projection back to the feasible set which reduced the computational cost. Furthermore, as stated before, thanks to the fundamental of linear programming, we only need to search for the vertices of the polyhedron set, thus, combing all the reasons discussed before, in this thesis, we will focus on this particular method and its variants for solving efficiently our constrained optimization problem.

2.3.1 Formulation of Frank-Wolfe method

The Frank-Wolfe algorithm proceeds by solving iteratively the so-called Frank-Wolfe problem with the form:

$$\begin{aligned} \min \quad & f(x) \\ x \in & \Omega \end{aligned} \tag{2.17}$$

in which the objective function f(x) is a continuous differentiable function and set $\Omega \subseteq R^n$ a convex compact set.

Now, we will dive deeper into the Frank-Wolfe algorithm. We first start with a random feasible solution in the feasible set and then at each iteration x_k, we search for the new descent direction by solving the following Frank-Wolfe problem:

$$\min_{x \in \Omega} \quad \nabla f(x_k)^T(x - x_k)$$

Notice that, the problem provided above is in fact a simplified version of the linear approximation of function f in point x_k, the original linear approximation is:

$$\min_{x \in \Omega} \quad f(x_k) + \nabla f(x_k)^T(x - x_k)$$

And in figure (3.1), we can see the full Frank Wolfe iteration. The solution gives us a feasible search corner or vertex of the polyhedron, and the next iterate is given by a convex combination of current vertex and the previous iterate adjusted by the step size. Furthermore, since at each iteration x_k, the value of the objective function $f(x_k)$ is a constant, thus can be ignored in the optimization process. Furthermore, from the compactness of the set Ω, we have that there exits a solution $\hat{x}_k \in \Omega$ for the linearized problem. Now, we can introduce the first order optimality condition for the Frank-Wolfe problem. Recall that we have already introduced the descent directions and the first order optimality condition for a continuous objective function. The same argument holds true for the Frank-Wolfe problem. In fact, we have the following result.

Proposition 10 *Let $x^* \in \Omega$ be local minimum for problem*

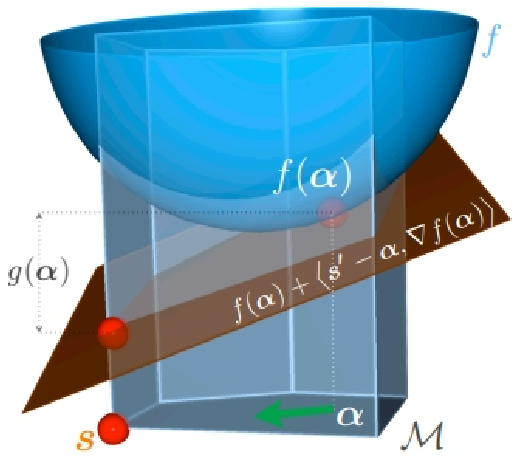


Figure 2.1: Full Frank Wolfe [1].

$$\min \quad f(x) \quad \textbf{\textit{s.t.}} \quad x \in \Omega$$

with $\Omega \subseteq R^n$ *convex and* $f \in C^1(R^n)$. *Then we have*

$$\nabla f(x^*)^T(x - x^*) \geq 0, \quad \forall x \in \Omega$$

Proposition 11 *Let* $\Omega \subseteq R^n$ *be a convex set and* $f \in C^1(R^n)$ *be a convex function.* $x^* \in \Omega$ *is a global solution of the following problem:*

$$\min \quad f(x) \quad \textbf{\textit{s.t.}} \quad x \in \Omega$$

if and only if

$$\nabla f(x^*)^T(x - x^*) \geq 0 \quad \forall x \in \Omega$$

Going back to the original Frank-Wolfe problem (2.3.1). We might have the following two different cases. The first one is when the optimality condition is reached, that is:

$$0 = \nabla f(x_k)^T(\hat{x}_k - x_k) \leq \nabla f(x_k)^T(x_k - x_k) \quad \forall x \in \Omega$$

Another possible case is when $\nabla f(x_k)^T(\hat{x}_k - x_k) < 0$ which means that there exists a direction that can further decrease the value of the objective function $f(\mathrm{x})$ and this new descent direction in x_k can then be written as $d_k = \hat{x}_k - x_k$. And subsequently, the new point is given by $x_{k+1} = x_k - \alpha_k d_k$, in which $\alpha \in (0, 1]$ is a properly chosen step size. There are various ways for choosing a proper step size *alpha* including line search. We will leave the relevant discussion in the following section. Before jumping into the next section, a pseudo-code of the general Frank-Wolfe algorithm will be reported.

Algorithm 2.1 The general Frank-Wolfe algorithm

```
1: Choose a random starting point $x_1 \in \Omega$
2: for $k = 1, \dots$
3:     Set $\hat{x_k} = \operatorname{argmin}_{x \in \Omega} \nabla f(x_k)^T (x - x_k)$
4:     If $\hat{x_k}$ satisfies some specific condition, then STOP
5:     Set $x_{k+1} = x_k + \alpha(\hat{x_k} - x_k)$, with $\alpha_k \in (0, 1]$
6: end for
```

Notice that, in line (5) of the Frank-Wolfe algorithm (2.1). We restricted the step size to admit a maximum possible value equals to 1, this is essential for the Frank-Wolfe algorithm as it can guarantee that at each Frank-Wolfe iteration, the newly found point x_{k+1} must be a feasible point within the feasible set Ω.

2.3.2 Step size

When solving an optimization problem, it is also very to choose a proper step size α_k at each iteration of the algorithm. However, choosing a proper step size α_k is not a simple task. The first one to be discussed is the so-called **line search** which is an iterative approach to find a local minimum of a nonlinear function using its gradients. It is composed by two phases, the first phase a search direction will be computed and then in the second phase, we will choose an acceptable step length that satisfies certain standard condition. The line search methods can be further divided into two categories, namely the exact search and inexact search. The exact search method seeks to find the **exact** minimizer at each iteration and the **inexact** search computes the step length to satisfy certain conditions. We will not give too much details as the line search method will not be used in this thesis, thus, we will not mention too much about it. However, we will still report the pseudo-code for the basic line search algorithm.

Another type of step size is the so-called **fixed step size** which means that a step size remains unchanged throughout the entire iterations.

Algorithm 2.2 The line search algorithm

```
1: Pick the starting point $x_0$
2: while $f_x := f(x_k)$ does not converge to a local minimum
3:     Choose a descent direction $d_k$ starting at $x_k$ such that $\nabla f(x_k)^T d_k < 0 \quad for \quad \nabla f(x_k) \neq 0$
4:     Find a step length $\alpha_k$
5: end while
```

$$\alpha_k = p, \quad \text{with} \quad p > 0, \quad k = 0, 1 \ldots$$

In this thesis, we will use a unit step size which has $\alpha_k = p = 1$ for all the $k = 0, 1 \ldots$. We will further provide a proposition for the convergence of the unit step size for the Frank-Wolfe algorithm.

Proposition 12 *Let us consider the problem*

$$\min \quad f(x) \quad \boldsymbol{s.t.} \quad x \in \Omega$$

with $f \in C^1(R^n)$ concave function lower bounded on Ω, and $\Omega \subseteq R^n$ polyhedron. The Frank-Wolfe Algorithm (2.1) *with unit step size converges to a stationary point in a finite number of steps.*

The proposition above gives us the guarantee of convergence to a stationary point in finite number of steps. In order words, we might not converge to global minimum, but we will at least converge to a stationary point.

The last rule for the selection of a proper step size involved in this thesis is using a diminishing step size. It is used to guarantee the converges of the stochastic methods such as the **stochastic gradient descent method**. Without a diminishing step size, the stochastic gradient descent method might not converge as it replaces the actual gradient by an estimate of it or in another word, a randomly selected subset of the data.
In this thesis, we will use the diminishing step size for the **Block Coordinate Frank Wolfe algorithm** which optimizes over only a block of the variable. The choice of a diminishing step size is different from the above mentioned case. When optimization over a random block of

the variable, we do not have the inexact information of the current iterate which might lead to an irregularity to the sequence of the values found by the algorithm, thus, we will use a diminishing step size for this algorithm. Further details will be provided in the following chapter.

2.3.3 Block-coordinate Frank-Wolfe

The **block-coordinate methods** represents a class of methods that solve the optimization problem by performing gradient steps along alternating sub-block of the coordinates. The block-coordinate methods become useful when we want to minimize over a variable in R^d and the dimensionality n large. It first split the optimization process into a sequence of simpler optimizations. In this case, computation for the entire variable can be very expensive, thus considering only a sub block of the variable makes the computation much cheaper. However, we have to keep in mind that even though the computation is faster, but since we do not use all the information of the current iteration and the new descent direction is determined by optimizing over only a sub-block of the variable, thus, we might get the optimal value for at each iteration.

Before going further, let us first report the general shceme of the block-coordinate methods.

Algorithm 2.3 Block-coordinate method

```
1: Choose a random starting point $x_1 \in \Omega$
2: for $k = 1, \dots$
3:     If $\hat{x_k}$ satisfies some specific condition, then STOP
4:     Pick coordinate i from 1 to n and set
           $$s_k^{(i)} = \underset{x^{(i)} \in R}{\operatorname{argmin}} f(x^{(i)}, x_{-i})$$
5:     Use $s_k^{(i)}$, with $i = 1, \dots, n$ to build $x_{k+1}$
6: end for
```

in which the term x_{-i} represents the set of all variables but the selected block of variable $x^{(i)}$. This is a very general scheme for the block-coordinate method. The selection of block can be done using different technique, furthermore, at step (4) of the algorithm, it is possible to introduce two different schemes. One is the **Gauss-Seidel Scheme** and the other one is the **Jacobi Scheme**. With the Gausee-Seidel scheme, at step (4), we set the coordinate x_{-i} to be the most up-to-date value, that is:

$$x_{-i} = (s_k^{(1)}, \dots, s_k^{(i-1)}, x_k^{(i+1)}, \dots, x_k^{(n)})$$

and at step (5), we simply set:

$$x_{k+1}^{(i)} = s_k^{(i)}$$

This scheme has a faster convergence, however, it suffers from the fact that it can be done only in the sequential order, thus, we will introduce the Jacobi scheme will enable the minimization in parallel. At step (4) with the Jacobi scheme, we have:

$$x_{-i} = (x_k^{(1)}, \dots, x_k^{(i-1)}, x_k^{(i+1)}, \dots, x_k^{(n)})$$

and at step (5), we gather the information obtained from the different minimization steps to get a new iterate to guarantee the decrease of the objective function. The most important advantage of the Jacobi scheme is that it allows the computation to be done in a parallel manner which gives a fast computation, however, it also has one major drawback, that is: The update of the Jacobi scheme does not take into account the intermediary iterates until we finish all the coordinates.

Another interesting thing about the block-coordinate method is how to select the block. Here we will report two classes of block selection strategies. The first one is the classic strategies and the second one is the randomized block selection strategies in which a random procedure is considered.
Now, let us discuss first the classic block coordinate selection strategies, considering b blocks, we have:

- **Cyclic order**: Run all blocks in cyclic order starting from the very first block to the very last block.

- **Almost cyclic order**: Each block i $\in \{1, \dots, b\}$ picked at least every $\beta < \infty$ successive iterations.

- **Gauss-Southwell**: At each iteration, pick block i so that:

$$i = \underset{j \in \{1,\ldots,b\}}{\text{argmin}} \|\nabla_j f(x_k)\|$$

The cyclic order selection proceeds by moving to the next block in a cyclic order, the almost cyclic order one requires that all the b blocks $i \in \{1, \ldots, b\}$ will be picked at least once every finite β iterations. And finally, the Gauss-Southwell method that is a greedy method and it seeks to pick the block that has the largest gradient norm, thus, providing a faster descent direction.

As for the randomized selection strategies, as one can tell from the name, this class of strategies requires randomized procedure. The first one is the seeks to create randomness by random permutation of the set and the other one performs a direct random selection of the block. More specifically, they are:

- **Random Permutation**: Run cyclic order on a set of permuted indices

- **Random Sampling**: Randomly select a block i for update

The cyclic order of the random permutation means that we fix the position and then run cyclic order, then at each iteration, we make a random permutation of the set of indices, thus, result in a random selection. As for the random sampling approach, at each iteration, we randomly select one block to optimization.

Here, we will give some examples of the these methods. We assume that we have only 3 blocks and each with a dimensionality of 1, and thus, we have:

$$\text{Cyclic}: \quad (1 \rightarrow 2 \rightarrow 3) \rightarrow (1 \rightarrow 2 \rightarrow 3) \rightarrow (1 \rightarrow 2 \rightarrow 3) \ldots$$

$$\text{Random Permutation}: \quad (2 \rightarrow 1 \rightarrow 3) \rightarrow (3 \rightarrow 1 \rightarrow 2) \rightarrow (1 \rightarrow 2 \rightarrow 3) \ldots$$

$$\text{Random Sampling}: \quad 3 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 1 \rightarrow 3 \ldots$$

With the introduction of block-coordinate method above, now, we can discuss the **Block-coordinate Frank-Wolfe** method. It is essentially the standard Frank-Wolfe algorithm but considers a random sampling of the block at each iteration and its result comes from the optimization of the objective function over the selected block. Comparing with the original Frank-Wolfe method, it is clear that the block-coordinate methods help the algorithm to be solved faster at each iteration as the problem becomes easier by considering one block each time.

This method was first proposed by Lacoste-Julien et al. [1] in their groundbreaking work *Block-Coordinate Frank-Wolfe Optimization for Structural SVMs* in 2013. In their experiments, they applied the block coordinate method to the structural SVMs problem and they showed that despite the lower iteration cost of the block-coordinate Frank Wolfe method, a similar rate in duality gap to the full Frank Wolfe method is achieved. Here, we will report the pseudocode of the Block-coordinate Frank Wolfe algorithm.

Algorithm 2.4 Block-coordinate Frank Wolfe algorithm

```
1: Let $x^{(1)}$ a starting point and $x_1 \in \Omega = \Omega^{(1)} \times \Omega^{(2)} \times \ldots \times \Omega^{(n)}$
2: for $k = 1, \ldots, K$
3:     Pick i at random in { 1, . . ., n}
4:     Find $s_{(i)} := \text{argmin}_{s^*_{(i)} \in M^{(i)}} < s^*_{(i)}, \nabla_{(i)} f(x^{(k)}) >$
5:     Let $\alpha$ be a properly chosen step size
6:     Update $x^{k+1}_{(i)} := x^{(k)}_{(i)} + \alpha(s_{(i)} - x^{(k)}_{(i)})$
7: end for
```

Notice that this is again just a general scheme of the block-coordinate Frank Wolfe method. One can modify according step size or block selection method. Further, it is also possible to consider more blocks at each iteration so that at each iteration, more information will be provided and thus a better result, however, it is still with the cost of more computation.

3
Cultural Semantic Conditioning and Biases

Nowadays, there are a lot of online source for the pre-trained embedding obtained with the above mentioned methods, one example can be the Word2Vec. However, these results are not always that trust-worthy or fair as Petreski and Hashim found out that the bias or cultural peculiarities of the training corpora can be also passed and hid inside the word embedding. In another word, our trained embedding also contains bias or cultural peculiarities related to the training corpora. With this in mind, several studies have proved that the ability of the word embedding to reflect different types of particular prejudice, such as: gender bias, racial bias, homophobia and discrimination.

3.1 Biases

In the *Oxford English Dictionary*, the term *bias* in general means that: *Tendency to favour or dislike a person or thing, especially as a result of a pre- conceived opinion; partiality, prejudice* [30].

Now we will introduce different types of bias of the word embedding. The first one is the **gender bias**. However, in the field of AI fairness, it is often described as: *skew that result in undesirable impacts [31] or more precisely computer systems that systematically and unfairly discriminate against certain individuals or groups of individuals in favor of others [32].* Blodgett et al. [33] in his work *Language (Technology) is Power: A Critical Survey of" Bias" in NLP* stated that:

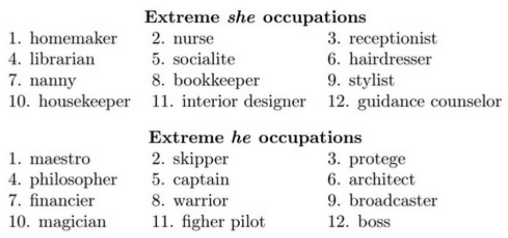
> Large body of work analyzing “bias” in natural language processing (NLP) systems has emerged in recent years, including work on “bias” in embedding spaces (...). Although these papers have laid vital groundwork by illustrating some of the ways that NLP systems can be harmful, the majority of them fail to engage critically with what constitutes “bias” in the first place.

Friedman and Nissenbau proposed a new interesting framework for the bias analysis by splitting them into **preexisting, technical** and **emergent**. By dividing the problem into this scheme, we have that the preexisting bias is related to the input data with which the algorithm is trained. While the technical bias refers to the bias caused by technical constraints. And finally, the emergent bias refers to bias related to the usage with a real user. Papakyriakopoulos et al. in their 2020's work *Bias in Word Embeddings* demonstrated that in reality, the word embedding suffers from all three types of bias mentioned before. To the interest of this thesis, we will focus only on the preexisting bias. This type of bias is of our interests as it is strongly related to social discrimination, more precisely, it represents the discrimination of one social group from the member of another social group. Now, we will focus on introducing different types of societal biases.

3.1.1 Societal biases

The very first societal bias to be introduced is the so-called **gender bias**. In NLP, this type of bias has a huge impact on our embedding, and it originates from the implicit sexism permeating the society and can be seen in the input corpora for the embedding training. Thus, our embedding will learn the bias pattern and will provide a biased output. One example can be the relations of the gender bias associated with the (male, female) pair and a semantic domain such as (work, family) pair. One very important paper about this topic is the *Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings* by Bolukbasi et al. Bolukbasi focus on the *w2vNEWS* embedding and he successfully detected the biases association such as indicated in the title, a man is biased towards programmer and a woman is biased toward a Homemaker. Within his work, he first highlight the occupational biases by measuring the distance between occupations and the (he, she) pair (2.4). They further studies about the analogies exhibiting stereotypes by generating pairs of word (x, y) such that they have the following relation with the pair (he, she):

$$he : she = x : y \quad \textbf{or} \quad \vec{she} - \vec{he} \approx \vec{y} - \vec{x}$$



Extreme *she* occupations

1. homemaker	2. nurse	3. receptionist
4. librarian	5. socialite	6. hairdresser
7. nanny	8. bookkeeper	9. stylist
10. housekeeper	11. interior designer	12. guidance counselor

Extreme *he* occupations

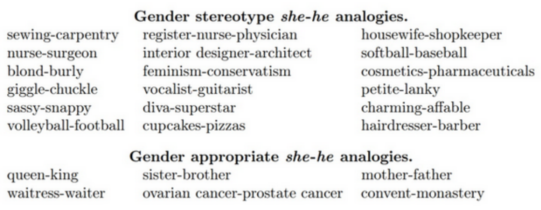
1. maestro	2. skipper	3. protege
4. philosopher	5. captain	6. architect
7. financier	8. warrior	9. broadcaster
10. magician	11. figher pilot	12. boss

Figure 3.1: Figure of automatically generated analogies for the she-he pair by Bolukbasi et al. [2] (2016).

Considering all the analogies of the corpora of the form $(he, she) = (a, b)$, we can define a score for all the possible pairs (x, y):

$$S_{A,B}(X, Y) = \begin{cases} \cos\left(\vec{a} - \vec{b}, \vec{x} - \vec{y}\right) & \textbf{if} \quad \|\vec{x} - \vec{y}\| \leq \delta \\ 0 & \textbf{otherwise} \end{cases}$$

in which the Greek letter δ is the threshold value for the evaluation of the score, that is, a valued non zero score will be generated only if the vector pair of x and y are close enough in term of the norm of their difference. And in the following figure (3.2), within the figure, we can see that in the upper part, a list of the gender biases following the she-he analogies, and at the bottom, we have the list of appropriate gender she-he analogies.



Gender stereotype *she-he* analogies.

sewing-carpentry	register-nurse-physician	housewife-shopkeeper
nurse-surgeon	interior designer-architect	softball-baseball
blond-burly	feminism-conservatism	cosmetics-pharmaceuticals
giggle-chuckle	vocalist-guitarist	petite-lanky
sassy-snappy	diva-superstar	charming-affable
volleyball-football	cupcakes-pizzas	hairdresser-barber

Gender appropriate *she-he* analogies.

queen-king	sister-brother	mother-father
waitress-waiter	ovarian cancer-prostate cancer	convent-monastery

Figure 3.2: Figure of selected list of occupations that are closest to *he* or *she* in the w2vNEWS embedding.

Bolukbasi et al. later proposed a debiasing algorithm that tries to correct the bias of the words by finding a gender subspace by two binary extremes on pairs of gender specific words. And finally, in 2022, Caliskan et al. [34] studies how gender bias permeate by studying the frequency, the part-of-speech, the conceptual clusters and the emotional characterization associated with each gender. And finally, we have to keep in mind that even gender is more complicated than a simple polarity, we will still rely on the polar classification due to the limitation inherent in data.

Another type of bias is the so-called **racial bias** which means the discrimination from one group towards a specific ethnic group. One example can be *Semantics derived automatically from language corpora contain human-like biases* written by Cliskan et al. [35] in 2017. They used analogy for the racial bias association and they showed that, the word embedding of the African American names appear to be closer to negative words rather than the positive ones, the same pattern doesn't appear for the European American names. And with this example, a clearly reflection of the racial bias towards the African Americans can be observed.

Garg et al. [36] in his paper *Word embeddings quantify 100 years of gender and ethnic stereotypes* found the occupational bias related to the proper names depending on the ethnicity. They achieved it by measuring the distance between elements of a set of proper names and a set of occupations. Furthermore, some other paper also suggest that the presence of racial bias hidden in the NLP system [37] [38].

Another type of bias is **political bias**. Bias does not only emerge in racial context, but also exist in a political context. In 2020, Gordon et al. [39] demonstrated that the technique developed by Bolukbasi et al. [2] can be expanded for the study of the political bias. They considered the American politics and thus a binary choice of party between the Democratic pole and Republican pole. The author analysis the most frequent word of the tweet by the two political parties, and then the binary pairs are obtained. And finally they arrived to the conclusion that in order to make a comprehensive analysis, it is necessary to model the bias along multiple axes as the politics is way more complicated than a simple binary choice.

3.1.2 Bias propagation and amplification

The word embedding has the ability to both propagate and amplify the intrinsic bias hidden in the language itself and it is mainly due to two reasons. The first one is the large usage of the word embedding and the second one is that the NLP algorithms are built so that they can influence people's behaviors, and let us now see two examples.

The first one is the **search engine**. In the work by Bolukbasi et al. [2], they noticed that, given a search query *cmu computer science phd student* for searching the computer science phd student at the Carnegie Mellon University. For two pages with the same content but only differs by the name: Mary and John. Despite this, the bias hidden in the embedding results in picking John before Mary. In this case, the word *programmer* is biased towards or closer to *male* or the male name *John* rather than *female* or the female name *Mary*. This poses an issue and in the field of computer science as given in the example above, woman might face a large gender gap due to the bias hidden inside of the embedding and subsequently, inside the search engine algorithm.

The second one we would like to discuss is the **NLG tasks**. NLG is short for Natural Language Generation, it is a class of methods used to generate the so-called **human-readable** language. The NLG tasks have a wide range of applications, some of the examples can be: Chat bots, automatic translation, virtual assistants and etc. In this work done by Sheng et al. [40]in 2021. They stated the importance of understanding how societal biases in NLG applications which can interact with its users directly, resulting in problem. Furthermore, these biases originate from the training of the NLG algorithm and also originate from the word embedding.

In conclusion, bias propagation and amplification can appear in various environments, they can pose different trouble and hinder the performance of our AI system, and these bias can come directly from the word embedding itself and thus they should be handled with care.

3.2 Diachronic change of the embedding

Before, we have discussed about the biases more precisely societal biases and their relevant presence in AI or NLP applications. Within this section, we will focus on the diachronic change of the word embedding which considers the embedding of the same word trained with the copora of the same language at different time. First, we will introduce the concept of **semantic change**. Bloomfield [41] in his 1933 work *Language* defined the semantic change as:

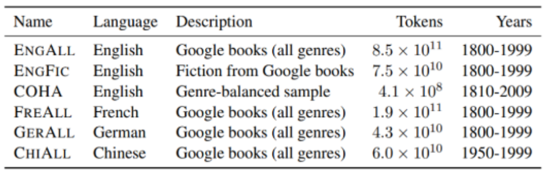
> Innovations which change the lexical meaning rather than the grammatical function of a form, are classed as change of meaning or semantic change.

According to the dictionary, the term *diachronic* [42] means, *relating to, or studying the development of a phenomenon through time*. Thus studying the diachronic change of the embedding basically means to find the evolution of the embedding between two labelled time. Now we will report two relevant studies about the diachronic semantic change.

3.2.1 Histwords project

The first project to be discussed is the famous *Histwords* project, which was done by a group of researchers at Stanford University. In their research paper *Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change* [3] published in 2016, they demonstrated the possibility of studying the diachronic change of words by analysing the relevant embedding space. Moreover, the embedding spaces are trained with the same technique and with the text at different time period, spanning for almost 2 centuries with texts written between 1800 and 1990. In this corpus, 4 languages are available: they are English, German, French and Chinese. A table of the characteristics of the datasets built for the Histwords project can be seen in the figure (3.3). One important reason for including this project work in the thesis is that in the experiment, we used the open source embedding of the Histword, in fact, we selected the embedding of English words of the year 1890 and of the year 1990 respectively, more details will be provided in the following chapter.

Now, let us talk about more the dataset of the histwords project. The project is based on two corpus, one is the Google n-gram corpus [43], [44], the second one used is the corpus of historical American English [45]. The Google n-gram corpus is very large and it contains almost 6% of all the books ever published, however, this rich collection of information also introduces a significant amount of corpus artifacts and noise. The Corpus of Historical American English on the other hand is much smaller and cleaner than the Google n-gram, it contains 457 million words coming from the 1820s-2010s and it is balanced by genre decade by decade.



Name	Language	Description	Tokens	Years
ENGALL	English	Google books (all genres)	8.5×10^{11}	1800-1999
ENGFIC	English	Fiction from Google books	7.5×10^{10}	1800-1999
COHA	English	Genre-balanced sample	4.1×10^{8}	1810-2009
FREALL	French	Google books (all genres)	1.9×10^{11}	1800-1999
GERALL	German	Google books (all genres)	4.3×10^{10}	1800-1999
CHIALL	Chinese	Google books (all genres)	6.0×10^{10}	1950-1999

Figure 3.3: Characteristics of the datasets built for the project Histwords[3].

Different state-of-the-art models were used for the training of the histwords embedding such as PPMI, SVD and SGNS using the framework provided by Levy et al. [46]. But the latter one seems to be a better one in terms of the performance.

The next step is to compute the alignment matrix, in order to do so, the Orthogonal Procrustes construction is used between each pair of diachronic embedding. Considering a matrix $X^{(t)} \in R^d \times N$ which is the embedding matrix of the vocabulary with N words at the time t, and the dimensionality of each word embedding is d. Thus, this diachronic change matrix can thus be computed by solving the following problem:

$$W^{(t)} = \underset{Q^TQ=I}{\text{argmin}} \|QX^T - X^{(t+1)}\|_F$$

The final solution $W^{(t)} \in R^{d \times d}$, and it can be solved with the method discussed in the previous chapter. And the last step of this project is to quantify the diachronic change of the word embedding over time, they proposed two approaches. The first one is measuring the similarity of the word pairs, the second one is by quantifying the overall shift of an individual word with the Spearman correlation.

Let us give some examples of the final result. Given three different words of different year: *gay* (1900s), *broadcast* (1850s) and *awful* (1850s), as a result of the approaches reported before, these diachronic change of these words can be see in the figure (3.4). From this figure we can notice that, the word *gay* has changed significantly its meaning, as its original meaning is closer to *daft* and shifts its meaning slowly to *pleasant*, *witty* and it appears to have the modern day meaning related to the secual orientation after the 1990s. For the word *broadcast*, the similar trend can be found, as it shifted significantly its meaning from the original *sow*, *seed* to *newspaper* and then gradually takes the modern day meaning such as close to *radio* etc which is connected to contemporary technologies. As for the word *awful*, we can see that it shifted its meaning from *majestic*, *solemn* in the 1850s to *appalling*, *terrible* in the 1900s and finally arrived to the modern meaning in the 1990s.

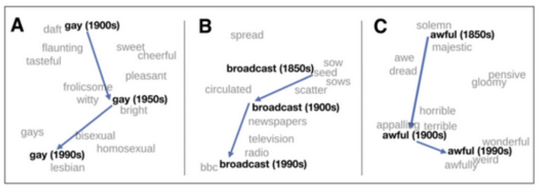


Figure 3.4: Diachronic change of different words in the project Histwords[3].

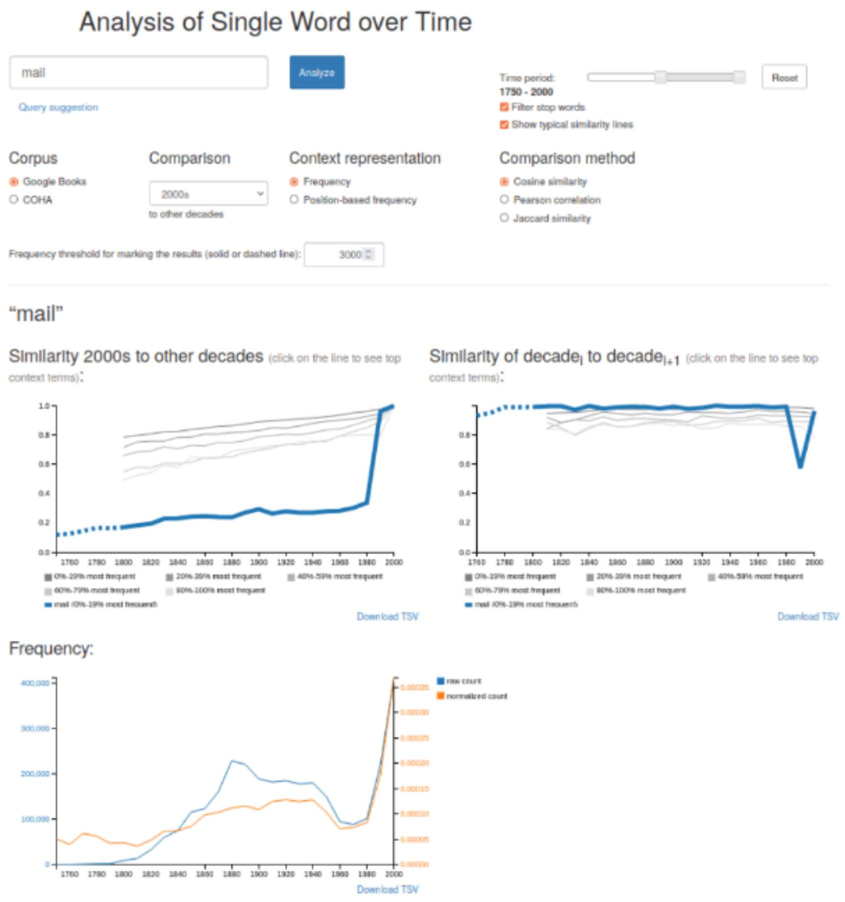
3.2.2 Exploring Word Evolution

In the paper *Every Word has its History: Interactive Exploration and Visualization of Word Sense Evolution* by Jatowt et al [4], an interactive framework is introduced and it is published on the web page. This is really powerful as it allows the users to study the diachronic change of the meaning of the words.This online platform performs various investigation at different lever once given the query word, these analysis are: *word analysis, contrastive word pair analysis, multi-word analysis and temporal context analysis*. Since in the experiment, we used the word analysis, thus later, we will give a description of the system basing on the word analysis. And finally this project uses the same dataset as the Histword project, namely the Google n-gram corpus and the COHA.

Now, given a query word w at a decade time d, the framework automatically find the word embedding $\vec{w_d}$. Then a word analysis is perform as it evaluates the degree of changes a query word experience across time. The word embedding at time d is compared to any other decade of a fixed time range. The web page provided several possible methods to make the comparison, they are: **Cosine Similarity, Pearson correlation, and Jaccard similarity**. Let us focus on the cosine similarity measure. The similarity measure considered by the Exploring Word Evolution is:

$$S_E(\vec{w_{d1}}, \vec{w_{d2}}) := |\cos(\vec{w_{d1}}, \vec{w_{d2}})|$$

Furthermore, we provided a screen shot (3.5) of the web page showing the output for the word *mail*. The plot on the top right of the figure is the plot of the cosine similarity between the query at decade d and the one in the past decade. The plot at the bottom left is the plot of the raw word count and its normalized frequency through out the time. And finally, the plot on the right is the plot similarity plot of the word embedding of two consecutive decades which can provide a better vision of the evolution of the meaning of the query word. And we can see that the meaning of the word *mail* has changed dramatically after 1980s.


Analysis of Single Word over Time

mail

Time period: 1750 - 2000

[✓ Filter stop words]

[✓ Show typical similarity lines]

Corpus: [✓ Google Books] COHA

Comparison: 2000s to other decades

Context representation: [✓ Frequency] Position-based frequency

Comparison method: [✓ Cosine similarity] Pearson correlation Jaccard similarity

Frequency threshold for marking the results (solid or dashed line): 3000

"mail"

Similarity 2000s to other decades (click on the line to see top context terms):

Similarity of decade$_i$ to decade$_{i+1}$ (click on the line to see top context terms):

Frequency:

Figure 3.5: Result of the word *mail* in the Exploring Word Evolution[4].

3.3 The cultural contamination by alignment

From the previous discussion, we know that bias or cultural peculiarities can be hidden insider the word embedding, it can be also disseminated by the alignment between different embedding space, the alignment does not only align the embedding but it also carries the bias. One of the most popular example can be the alignment between the cross-lingual word embedding. Let us take the example proposed by Zhang et al. [47] in the paper *Are Girls Neko or Shojo? Cross-Lingual Alignment of Non-Isomorphic Embeddings with Iterative Normalization*. With their work, they found out that it can be hard to build a cross-lingual alignment between two languages using the orthogonal method which is related to the cultural differences related to the languages. They further proposed a solution to deal with this problem which is by using an **iterative normalization** before applying the alignment so that one can try to correct the monolingual embedding.

One example is provided by Zhang which focus on the translation between English and Japanese. They noticed that the orthogonal alignment failed to translate the Japanese word *shojo* which has the meaning of *girl*. This phenomenon is due to the fact that in Japanese embedding, the word *shojo* and *girl* have different most similar words which means in the context of Japanese, they are not semantically similar. Furthermore, the most similar word of *shojo* in the Japanese embedding is the word *neko* which means *cat* as in Japanese culture, cats are considered to be a feminine animal which is totally different from the English as in English, words *girl* and *cat* are not similar.

Some other work that worth mentioning is done by Søgaard et al.[48] in which they found the limitations of unsupervised machine translation. Their critique is based on that:

> Unsupervised approaches to learning crosslingual word embeddings are based on the assumption that monolingual word embedding graphs are approximately isomorphic, that is, after removing a small set of vertices (words).

They used various unsupervised models rely on the orthogonal method that are developed by Conneau et al. [49]. They reached to the conclusion that monolingual word embedding are far from being isomorphic by studying the nearest neighbor graphs of the embedding space. This result still holds for two closely related languages such as English and German.

4 Models

In previous chapters, we have introduction the general description of word embedding, now let us introduce the exact framework to be used in this thesis. From now on, we will follow the following notations. Since we will use two word embedding spaces of the same language, thus we will define e_1 and e_2 as:

$$e_1 : D \to V_1 \quad e_2 : D \to V_2$$

with D_1, D_2 the vocabularies and V_1, $V - 2$ the corresponding embedding spaces. Considering both vocabulary to be the same and we will further have the number of words are $|D| = N$. We will also denote X, Y $\in$ $R^{d\times N}$ two matrices of the word embedding, each column is the embedding representation of the word. Thus, we have for any word $w_i \in D$, its representation are the columns of X and Y as $e_1(w_i) = \vec{x_i}$ and $e_2(w_i) = \vec{x_i}$. Here, we want to find a technique that can automatically highlight the bias based on the alignment. And it is showed in the figure (4.1). We start with a word w from the vocabulary D. Later we have its embedding in different spaces V_1 and V_2 are $\vec{x}$ and $\vec{y}$ respectively. And our alignment A seeks to find an alignment matrix $W \in R^{d\times d}$ to minimize the mapping error between the two word embedding $\vec{x}$ and $\vec{y}$.

Our experiment considers three different base alignments according to different selection of norm. All three alignments are the map of the form:

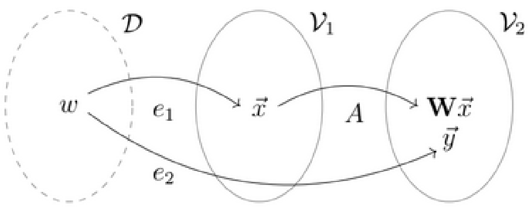


Figure 4.1: Representations of w and the alignment map A.

$$A_i : \quad V_1 \longrightarrow V_2$$

which transform the embedding $\vec{x}$ in the first embedding space to be as close to its representation $\vec{y}$ in the second space via the corresponding transformation matrix W, which is $\vec{X} \to W_i\vec{x} \approx \vec{y}$. These mapping can be seen in the figure (4.2).

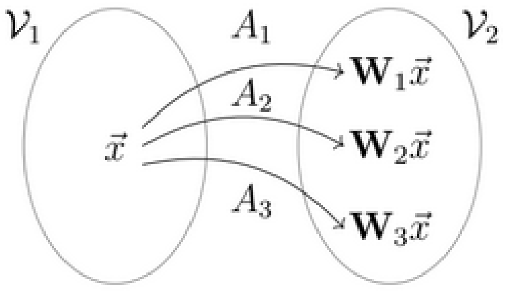


Figure 4.2: Representations of all three alignments.

Now we will provide more details about the construction of these alignments. Our goal is to find a matrix that can minimize the error or difference between the mapped representation $W\vec{x}$ and its target representation $\vec{y}$ in the other embedding space, these difference can be written in matrix form that is $WX - Y$. Now we want to minimize this difference by considering the norm of the matrix $WX - Y$ and we consider three types of norm, the l2, l1 and zero norm that in the end will return different alignment matrix W_i.

4.1 W_1: Orthogonal Procrustes

The first norm we will consider is the l2 norm which is involved with the square of the error of each dimension of the difference of the embedding, when applied to the matrices, it is also called the Frobenius norm, thus our alignment problem becomes the **orthogonal Procrustes problem** and our first alignment matrix W_1 is obtained by solving:

$$W_1 = \underset{W \in R^{d \times d}}{\operatorname{argmin}} \|WX - Y\|_F \quad \textbf{s.t.} \quad WW^T = I$$

and here with this alignment we seeks to minimize the global squared error for each entry in this error matrix $A = WX - Y$. Recall from the previous chapter, this problem has a close from solution, and it is:

$$W_1 = UV^T \quad \textbf{with} \quad U\Sigma V^T = SVD(YX^T)$$

It is very efficient to compute, as notice that it requires the SVD which is essentially matrix multiplication and with Python, there are already several efficient libraries that can handle the matrix multiplication efficiently, such as: Numpy, JAX.

4.2 W_2: L1 method

Another norm we can use for this problem is the **l1 norm**, unlike the l2 norm, instead of considering the squared error, it considers the pure difference between the word. And our alignment matrix W_2 is found by solving:

$$W_2 = \underset{W \in R^{d \times d}}{\operatorname{argmin}} \|WX - Y\|_1$$

Again, we consider the global error of the error matrix A, we can further reconstruct the problem by considering the global error as the sum of the error of each individual word embedding, and that is:

$$
\begin{aligned}
W_2 &= \underset{W \in R^{d \times d}}{\operatorname{argmin}} \sum_{i=1}^{N} \| W\vec{X_i} - \vec{Y_i} \|_1 \\
&= \underset{W \in R^{d \times d}}{\operatorname{argmin}} \sum_{i=1}^{N} \| W\vec{x_i} - \vec{y_i} \|_1
\end{aligned}
$$

Notice that the letter X and Y with subscript i represents the i-th column of the corresponding matrix, for the sake of simplicity, we denote x as a general word embedding in X and y as a word embedding in Y for the i-th word, it can be denoted as x_i and y_i. The superscript in this thesis will denote the row of a matrix. And finally with these notations, considering also $d = 300$, we will represent the j-th row and i-th column of the matrix Y as Y_i^j and our problem can be rewritten as:

$$
\begin{aligned}
W_2 &= \underset{W \in R^{d \times d}}{\operatorname{argmin}} \sum_{i=1}^{N} \sum_{j=1}^{300} | W^j x_i - y_i^j | \\
&= \underset{W \in R^{d \times d}}{\operatorname{argmin}} \sum_{j=1}^{300} \sum_{i=1}^{N} | W^j x_i - y_i^j |
\end{aligned}
$$

Furthermore, solving the problem above is equivalent to solving 300 sub-problems of the form:

$$
\min_{w^j \in R^d} \sum_{i=1}^{N} | W^j x_i - y_i^j | \quad \forall j = 1, \ldots, 300
$$

As explained in the previous chapter, we can perform the linearization technique to the problem and we have the following linear problem:

$$
\min_{w^j \in R^d, z \in R^N} \sum_{i=1}^{N} z_i \quad \textbf{s.t.} \quad -z_i \leq w^j x_i - y_i^j \leq z_i \quad \forall j = 1, \ldots, 300
$$

where $z \in R^N$ the new variable to optimize. Notice that such decomposition of the problem is required, as with the very original construction, solving the problem requires the consideration of almost 5 million constraints at once, thus makes the problem way more complicated and hard to solve.

4.3 W_3: Zero norm

The last norm we consider is the **zero norm**, unlike previous two norms that consider the scale of the error, the zero norm forces the sparsity of the error vector $z \in R^N$, in our previous constructions, we refer the entry $z_i \in R^d$ as the sum of error over all the dimensions of the embedding of the i-th word. Let us first define the zero norm problem. Given a polyhedral set, the problem of finding the minimum number of nonzero components of a vector in the set is the zero norm problem. And we can write it formally as:

$$\min_{min} \quad ||z||_0, \quad z \in \Omega \tag{4.1}$$

where we denote the zero norm of z as $||z||_0 = \textbf{card}\{z_i : z_i \neq 0\}$. This problem is NP-hard [50], thus in order to make it tractable, we can further write it as:

$$||z||_0 = \sum_{i=1}^{N} s(|z_i|)$$

in which $s : R \rightarrow R^+$ is the **step function** such that:

$$s(t) = \begin{cases} 1 & t > 0 \\ 0 & t \leq 0 \end{cases}$$

Replacing the step function in the previous form with a continuously differentiable concave function $v(t) = 1 - \exp^{-\alpha t}$, with $\alpha > 0$, the objective function of the zero norm become:

$$\min_{z,y \in R^N} \quad \sum_{i=1}^{N} (1 - \exp^{-\alpha y_i}) \quad \textbf{s.t.} \quad z \in \Omega \quad -y_i \leq z_i \leq y_i, \quad i = 1, \ldots, N$$

With this replacement, the convergence of the Frank-Wolfe method to a vertex stationary point with uni step size is guaranteed, in fact, this convergence result was proved to be value for a general class of concave programming problem. And now, we can apply this framework to our problem, and we can find the third alignment matrix W_3 by solving:

$$W_3 = \underset{W,A,z}{\text{argmin}} ||z||_0 \quad \textbf{s.t.} \quad -a_i \leq Wx_i - y_i \leq a_i, \quad e^T a_i = z_i$$

in which $z \in R^N$ is the error vector with entry i being the sum of the i-th column a_i of the error matrix A and e^T is a vector of all its entries equal to 1. We will focus on this problem and use the Frank-Wolfe algorithm and its variant to solve it. For this problem, we have three variables that are concatenated in variable x, since there is no component of w_j and d_i^j in the objective function, thus, we have:

$$\frac{\partial f}{\partial w_j^l} = \frac{\partial f}{\partial d_i^j} 0 \quad \textbf{with} \quad \forall j = 1, \ldots, d, l = 1, \ldots, d, i = 1, \ldots, N$$

Thus, we further have:

$$\nabla f(x_k) \cdot x = \nabla f(z_k) \cdot z$$

and finally, our minimization problem becomes:

$$\min_{A,W,z} \nabla f(z_k) \cdot z \quad \textbf{s.t.} \quad -a_i \leq Wx_i - y_i \leq a_i, \quad e^T a_i = z_i$$

Here, we will report the pseudocode of the Frank-Wolfe algorithm for the zero norm problem, we will set $g_k := \nabla f(z_k) \in R^N$, and the cut-off condition ε.

Algorithm 4.1 The general Frank-Wolfe algorithm for the zero-norm problem

```
 1: k ← 0
 2: $(g_k)_i \leftarrow (g_0)_i, \quad \forall i = 1, \dots, N$
 3: A, W, z ← $\text{argmin}_{A,W,z} \nabla f(z_k) \cdot z \quad$ **s.t.** $\quad A, W, z \in \Omega$
 4: k ← 1
 5: **while** $\sum_{i=1}^{N} (g_k)_i (z_i - (z_k)_i) \geq -\varepsilon$
 6:     $(z_k)_i \leftarrow z_i \quad \forall i = 1, \dots, N$
 7:     $(g_k)_i \leftarrow \alpha \exp -\alpha (z_k)_i \quad \forall i = 1, \dots, N$
 8:     A, W, z ← $\text{argmin}_{A,W,z} \nabla f(z_k) \cdot z \quad$ **s.t.** $\quad A, W, z \in \Omega$
 9:     k ← $k + 1$
10: **end while**
```

Similar to the l1 problem, this problem again requires almost 5 million constraints, in order to obtain a lighter version of this problem, we will perform the following transformation.

For each entry z_i of the error vector, the error of the i-th word is:

$$e^T a_i = \sum_{i=1}^{300} a_i^j = z_i$$

apply it to the objective function, we then have:

$$\begin{aligned} \nabla f(z_k) \cdot z &= \sum_{i=1}^{N} (g_k)_i z_i = \sum_{i=1}^{N} (g_k)_i \sum_{j=1}^{300} a_i^j \\ &= \sum_{i=1}^{N} \sum_{j=1}^{300} (g_k)_i a_i^j \\ &= \sum_{j=1}^{300} \sum_{i=1}^{N} (g_k)_i a_i^j \end{aligned}$$

and our minimization problem now becomes:

$$\min_{A,W} \sum_{j=1}^{300} \sum_{i=1}^{N} (g_k)_i a_i^j \quad \textbf{s.t.} \quad -a_i \leq W x_i - y_i \leq a_i$$

Thanks to the concave nature of our approximated objective function, we have that the gradient is always greater or equal to 0, thus the problem above can be again divided into 300 sub-problem, and they have the form:

$$\min_{a^j,w^j} \sum_{i=1}^{N} \quad \textbf{s.t.} \quad -a_i^j \leq Wx_i - y_i \leq a_i^j \quad \forall j = 1, \ldots, 300$$

Denote Ω_j as the feasible set for the j-th sub-problem, now the pseudocode becomes:

Algorithm 4.2 The general Frank-Wolfe algorithm for the zero-norm problem

```
 1: k ← 0
 2: $(g_k)_i \leftarrow (g_0)_i, \quad \forall i = 1, \ldots, N$
 3: $a^j, w^j \leftarrow \text{argmin}_{a^j,w^j} \sum_{i=1}^{N} (g_k)_i a_i^j$   s.t.   $a^j, w^j \in \Omega_j, \quad \forall j = 1, \ldots, 300$
 4: k ← 1
 5: while $\sum_{i=1}^{N} (g_k)_i (\sum_{j=1}^{300} a_i^j - (z_k)_i) \geq -\varepsilon$
 6:     $(z_k)_i \leftarrow \sum_{j=1}^{300} a_i^j \quad \forall i = 1, \ldots, N$
 7:     $(g_k)_i \leftarrow \alpha \exp -\alpha (z_k)_i \quad \forall i = 1, \ldots, N$
 8:     $a^j, w^j \leftarrow \text{argmin}_{a^j,w^j} \sum_{i=1}^{N} (g_k)_i a_i^j$   s.t.   $a^j, w^j \in \Omega_j \quad \forall j = 1, \ldots, 300$
 9:     k ← $k + 1$
10: end while
```

Thanks to the proposition (12), we are sure that this algorithm converges in a finite number of steps. Another modification can be applied is the reduction of dimension method which was proposed by Rinaldi el al. [51]. Within this work, a new technique based on the idea that if a word is correctly aligned, or equivalently the entry corresponds to that word is 0 and will remain unchanged for the rest of the iterations. Hence, computations related to those words by be skipped and thus a significant save of computation time by performing the following transformation.

$$\begin{aligned}
(z_k)_i = 0 &\Rightarrow \sum_{i=1}^{300} a_i^j = 0 \quad \forall j = 1, \ldots, 300 \\
&\Rightarrow 0 \leq w^j x_i - y_i \leq 0 \\
&\Rightarrow w^j x_i - y_i = 0
\end{aligned}$$

Considering a set of indices of non zero entry of z_k, which we denote with $I = \{i = 1, \dots, N, \quad \textbf{s.t.} \quad (z_k)_i \neq 0\}$, thus the j-th sub-problem becomes:

$$\begin{aligned} \min_{a^j, w^j} \quad & \sum_{i \in I} (g_k)_i d_i^j \\ \textbf{s.t.} \quad & w^j x_i - y_i = 0 \quad \forall \notin I \\ & - d_i^j \leq w^j x_i - y_i \leq d_i^j \quad \forall i \in I \end{aligned}$$

As discussed before, we can also apply the block-coordinate method for this problem which considers one block at a time. Here, we will use a random sampling of the block to be optimized, thus, we only need to add a random selection of the block between line 4 and 5 in the pseudocode.

Notice that we can also define an easier objective function, which is instead of considering the zero norm of the error vector z_k, we will try to minimize the zero norm of each word, or equivalently, we try to make the error matrix to contain as many zeros as possible. Comparing with the original formulation, our new problem becomes easier, as forcing one entry of z_k to be zero means that for that column of the error matrix to be all zero. Hence, minimize the zero norm of the error vector z_k means trying to make all columns to be 0 which is a way harder problem. The new gradient is still a vector, for the sake of simplicity, we will denote $(g_k)_i^j$ as the gradient related to the variable d_i^j and now we can write the pseudocode as:

Algorithm 4.3 The general Frank-Wolfe algorithm for the zero-norm problem (Version 2)

```
1: k ← 0
2: $(g_k)_i^j \leftarrow (g_0)_i^j, \quad \forall i = 1, \dots, N \quad \forall j = 1, \dots, 300$
3: $a^j, w^j \leftarrow \text{argmin}_{a^j, w^j} \sum_{i=1}^N (g_k)_i^j d_i^j \quad$ s.t. $a^j, w^j \in \Omega_j, \quad \forall j = 1, \dots, 300$
4: k ← 1
5: while $\sum_{i=1}^N (g_k)_i^j (d_i^j - (a_k)_i^j) \geq -\varepsilon$
6:     $(a_k)_i^j = d_i^j$
7:     $(g_k)_i^j \leftarrow \alpha \exp -\alpha d_i^j \quad \forall i = 1, \dots, N \quad \forall j = 1, \dots, 300$
8:     $a^j, w^j \leftarrow \text{argmin}_{a^j, w^j} \sum_{i=1}^N (g_k)_i^j d_i^j \quad$ s.t. $a^j, w^j \in \Omega_j \quad \forall j = 1, \dots, 300$
9:     k ← $k + 1$
10: end while
```

4.3.1 Similarity measure

In this very last section of this chapter, we will include the important similarity measures for our experiment. The first one is the **absolute cosine similarity** and we denote it as s_i. For the i-th aligned word, its absolute cosine similarity is thus defined as:

$$s_i(w) = |\cos(\vec{y}, W_i\vec{x})|$$

in which the subscript of the alignment matrix refers to different matrix obtained with various techniques and vectors $\vec{x}, \vec{y}$ are the usually word embedding representation of w in two embedding spaces. The second useful measure is the **theta measure**, which considers the common number of N closest neighbors before and after the alignment. The intuition is that if two words have similar semantic meaning, that means that they are likely to have more words in common. With this intuition, the theta measure can be defined as:

$$\theta_k(\vec{x}, \vec{y}) = \frac{|\{x_1, \ldots, x_k\}| \bigcap \{y_1, \ldots, y_k\}}{k}$$

where the sets $\{x_1, \ldots, x_k\}$ and $\{y_1, \ldots, y_k\}$ are the set of k nearest words of $\vec{x}$ and $\vec{y}$ in the corresponding embedding space. And below is attached one diagram of the role of the similarity measure.

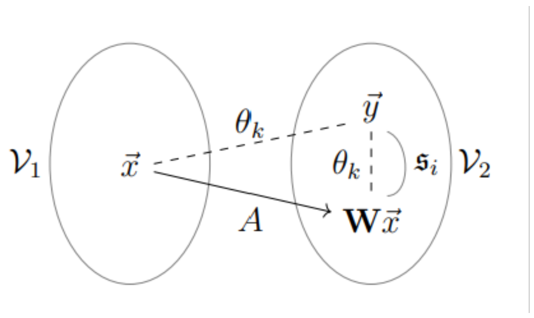


Figure 4.3: Diagram of similarity measure.

These two measure are essentially similar, as saying that two words have more common neighbors means that they are close to each other and as a consequence, the angle between them should be small and thus a lower absolute cosine similarity. One final note to the theta measure is the choice of the number of closest neighbors. Of course more neighbors considered means a higher computation load, on the other hand, a too small number of neighbors might not give us enough information to make meaningful statement, hence, in our experiment, we consider 20 closest neighbors as a trade-off between those two factors.

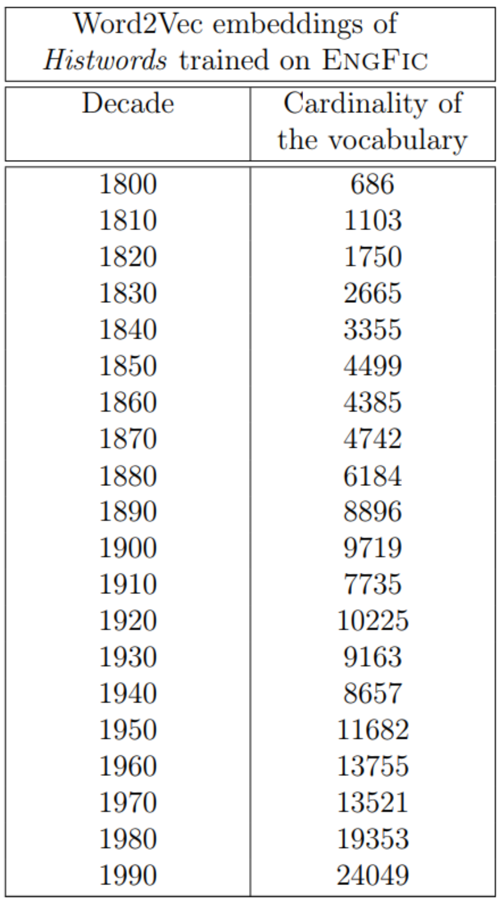
5
Dataset and the experimental setup

As explained in the previous chapter, the dataset used in this experiment is a pre-trained 300-dimensional of the **Histwords** project. More specifically, we use the embedding trained on English corpora ENGFIC which is made up by English fiction books with 7.5×10^10 tokens. There are 20 different sets of embedding for different decade between the year 1890 and 1990 and of course with various number of words. From the figure (5.1) below, the number of word embedding of each decade is reported.

We selected the year 1890 and 1900, thus we have the two embedding are:

$$e_{1890} : D_{1890} \to V_{1890} \quad \textbf{and} \quad e_{1990} : D_{1990} \to V_{1990}$$

in which again the letter D_i the vocabularies of related to the embedding. From the figure above we can see that there are 8896 words in the vocabulary of the year 1890 and 24049 words in the year 1990. Our hypothesis assumes the two embedding to have the same dimensionality which means both the dimensionality of the embedding vector d and the number of words in the vocabulary N must be the same for the two dataset, thus, we will use the intersection between these two vocabularies as the common vocabulary for both embedding spaces. There are 8810 words in the common vocabulary. Thus, for our experiment, we will use X as the embedding representation matrix in the space of 1890 and Y in the space of 1990.



Word2Vec embeddings of *Histwords* trained on EngFic	
Decade	Cardinality of the vocabulary
1800	686
1810	1103
1820	1750
1830	2665
1840	3355
1850	4499
1860	4385
1870	4742
1880	6184
1890	8896
1900	9719
1910	7735
1920	10225
1930	9163
1940	8657
1950	11682
1960	13755
1970	13521
1980	19353
1990	24049

Figure 5.1: Cardinality of the vocabulary.

Within our experiment, we will use two different test set, the first one is composed by the top and bottom 150 words based on their theta measure and the second test set will be a test sets selected based on literature, personal annotation. We will discuss both test sets in more details.

Let us focus on the first dataset of the top and bottom words. In order to extract this dataset, we first consider the theta measure θ_k between all the words in two embedding matrices and we use subscript k to represent the number of closest neighbors to be considered. We compute the theta measure for all the words in the vocabulary and then sort them based on their theta measure value in ascending order. The result can be found in the figure (5.2). From the figure, we can see that the theta measure took all its possible values in a discrete way. Furthermore, we can also see that some words tends to be stable as their common closest neighbors don't really change, and some words tends to be unstable as none of their closest neighbors in the 1890 remain in the closest neighbors of the same word in 1990. Which confirmed the fact that the meaning of some words has a more rapid change.

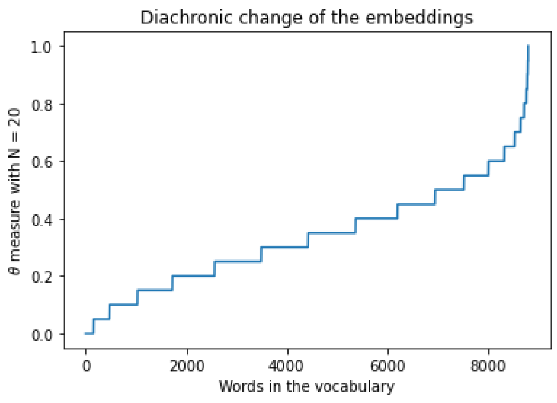


Figure 5.2: Sorted θ_{20} for words in the common vocabulary.

Now, we will take the top and bottom 150 words of the vocabulary based on the sorted list of theta measure as those two groups of words represent the word with the most and least rapid change between the year 1890 and 1990. Now, we will report the two set of words below.

Bottom 150 words
'tut', 'ta', 'roland', 'guy', 'palmer', 'warden', 'jerry', 'veteran', 'unheard', 'check-

ing', 'exercises', 'ex', 'priscilla', 'wells', 'continues', 'corrected', 'luckily', 'maps', 'presided', 'eventually', 'jem', 'eminently', 'compound', 'delivery', 'distributed', 'consigned', 'imagining', 'judy', 'accomplishment', 'ultimately', 'enlightened', 'excluded', 'swamp', 'breasts', 'lodger', 'creditors', 'damn', 'brussels', 'minstrel', 'carrier', 'noah', 'interpreted', 'missionary', 'bearings', 'mac', 'secondly', 'flourished', 'sate', 'available', 'twelvemonth', 'production', 'spit', 'jason', 'crook', 'transport', 'contemplate', 'aforesaid', 'meg', 'assailed', 'barker', 'divinity', 'assisting', 'toilet', 'unequal', 'exempt', 'random', 'practise', 'representation', 'exaggerated', 'foreigner', 'reminding', 'brand', 'whatsoever', 'plight', 'overtake', 'georgie', 'availed', 'keawe', 'dad', 'proclaimed', 'literally', 'uses', 'hereditary', 'choosing', 'compromise', 'dismiss', 'correspondent', 'artificial', 'hermiston', 'partial', 'obviously', 'snatch', 'offensive', 'dorrit', 'deborah', 'nell', 'significant', 'speculation', 'rosa', 'nick', 'banished', 'illustration', 'foster', 'viewed', 'favoured', 'kindred', 'esther', 'urge', 'jessie', 'joyous', 'flora', 'tony', 'jove', 'dow', 'detained', 'monarch', 'judging', 'briefly', 'extensive', 'charlie', 'ideal', 'positively', 'convenience', 'daniel', 'division', 'scandal', 'ward', 'travellers', 'archie', 'assigned', 'including', 'reserved', 'type', 'gypsy', 'mess', 'communicate', 'acknowledged', 'chanced', 'waverley', 'cranford', 'headed', 'egyptian', 'gay', 'rob', 'rude', 'nanny', 'gavin', 'of', 'and', 'vis'

Top 150 words
'bout', 'wist', 'fifteenth', 'grate', '3', 'beale', 'rum', 'ter', 'lamorak', 'kay', 'ane', 'perspiration', 'precipice', 'sung', 'sixth', 'blaze', 'slew', 'bors', 'trousers', 'fourteen', 'youngest', 'sweat', 'virgin', '2', 'haired', 'cliff', 'eighteen', 'sang', 'pistol', 'finest', 'sheep', 'fired', 'secretary', 'shield', 'francs', 'dawn', 'gown', 'accepted', 'daughters', 'burning', 'sisters', 'growing', 'fifteen', 'sing', 'weight', 'sunday', 'playing', 'bottle', 'mountain', 'played', 'forty', 'launcelot', 'heads', 'france', 'dozen', 'm', 'weeks', '1', 'thirty', 'third', 'sword', 'months', 'knight', 'hair', 'children', 'days', 'mother', 'few', 'years', 'father', 'two', 'gude', '5', 'marhaus', 'thirteenth', 'ninth', 'fourteenth', 'ony', 'ector', '4', 'sixteenth', 'nineteen', 'percivale', 'nae', 'ower', 'gareth', 'dinadan', 'eighty', 'centuries', 'burned', 'lantern', 'seventy', 'en', 'eleven', 'sofa', 'coach', 'knife', 'tristram', 'sigh', 'twelve', 'breakfast', 'wine', 'eight', 'fifty', 'seven', 'carriage', 'trees', 'glass', 'thousand', 'twenty', 'understand', 'morning', 'three', 'ole', '9', '14', 'gaheris', '8', 'muckle', 'thirteen', 'mair', 'ninety', 'sixteen', 'gawaine', 'fourth', 'song', 'nine', 'six', 'ten', 'five', 'hundred', 'fire', '27', '30', '19', '15', '16', '25', '7', 'mordred', 'hae', 'four', '23', '21', '24', '13', '17', '20', '12', '18'

The second dataset is a combination of three different dataset selected by Elena [52] which are reported below.

H
fun, fond, terrific, tremendous, awe, grin, smart, egregious, sad, smug, facetious, bully, gay, fatal, awful, nice, broadcast, monitor, record, guy, call, awesome, terrible, terrific, naive, demagogue, guy, mouse, queer, nigger, jaw, kill, astound, knave, knight, recording, bitch, tape, checking, diet, sex, plastic, transmitted, peck, honey, hug, windows, bush, apple, sink, click, handle, instant, twilight, rays, streaming, ray, delivery, combo, candy, rally, snap, mystery, stats, sandy, shades, god, propaganda, atomic, toilet, halloween, king
B
chink, colored, indian, african, foreign, lesbian, gypsy, elderly, handicapped, homos, homosexual, alien, master, slave, retarded, tranny, tribe, girl, boy, man, woman, housekeeper
S
house, tree, table, lamp, book, shoe, mirror, box, fork, chair, telephone, bottle, stove, engine, wallet, boat, pencil, box, cup, plate, paper, stereo, leaf, stick, cloud, shampoo, hat, painting, clothes, watch, window, key, pillow, water, fire, book, door, street, path, bird, horse, cat, dog, fox, fish, school, paper, fountain, cage, ink, pen, bone, forniture, dictionary, umbrella, scissor, hammer, rubbish

For the set H, it has the words subjected to historical semantic change according to the literature. Set B contains words that are subjected to certain type of cultural bias or due to racial bias, etc. and they are personally annotated or found online. The set S are words that are more likely to be stable with all the words belong to the **inanimate objects** and **common animals and natural elements** classes as usually these classes are influenced less by the cultural change over time. Thus, if we plot the theta measure of these three datasets, we will expect set H to have the lowest theta measure as these are words that are subjected to semantic changes according to the literature which is possibly the most reliable source for such classification. Furthermore, we expect the set S to have the highest theta measure and the set B to be the intermediate one.

Notice that not all the words in this test set are present in our common vocabulary, in fact, 25 words of set H are not in the common vocabulary, the number of missing word are 9 and 6 for the set B and S respectively. Thus, as explained before, in order to have the common vocabulary, we will not consider all the missing words in the common vocabulary.

One thing to be noticed is that in the list of the top words, a lot of them are just numerical numbers, it is not surprising that the numbers tend to be stable as they are more objective compared to other classes of words. Furthermore, we will also report in the table below the mean value and variance of the theta measure θ_k for all the words in the top and bottoms lists. We will use these values to compare the effectiveness of our alignments. As for the second dataset, we will apply the theta measures for all three sets H, B and S and again we will report these values in the same table.

Set	**bottom**	**top**	**H**	**B**	**S**
$\theta(\vec{x},\vec{y}) \pm \sigma_{\theta(\vec{x},\vec{y})}$	0.0003 ± 0	0.7977 ± 0.0035	0.3326 ± 0.0461	0.35 ± 0.0373	0.4806 ± 0.027

From the table above, we can see that for the theta measure for the bottoms words are really close to zero and the theta measure of the top words are really high, with a mean value of almost 0.8, but this phenomenon is not surprising to us as we have purposely selected those words. As for the second test set, we can see that even though the difference between the mean value of set H and B are close, but recall that all the words of set B are personally annotated or found online, thus the conclusion of semantic change of these words are not as reliable as the other two sets. However they still follows our expectation as set S has the highest mean theta measure and then followed by set B and then H. With these results, we can confirm that the sets H, B, S are properly selected.

5.1 Comparison of the alignments

Within this subsection, we will focus on comparing the alignment obtained with different methods. We will classify them into three general groups by the formulation of the objective functions as discussed in the previous chapter. Recall that these methods are:

- W1 obtained by solving the orthogonal procrustes problem.
- W2 obtained by minimizing the L1 norm with linear decomposition.
- W3 obtained by minimizing the zero norm of the error matrix .

Notice that we have already discussed in detail about the Block-coordinate methods and we will further add the block-coordinate Frank-Wolfe algorithm and we will denote it with W_4. Furthermore, we will propose two different initialization point for the Frank-Wolfe algorithm. For the initialization of the Frank-Wolfe algorithm, as explained before, we usually just pick a random starting point and then solving for a vertex of the polyhedron formed by the linear constraints. However, we are also free to choose our initialization point such as using the one obtained by the Procrustes and L1 norm problem. The intuition behind this is that, even though we have constructed different objective function, but since the minimization of the error in terms of the L2 norm, L1 norm and zero norm are essentially performing similar tasks, thus we can assume that the solution provided by the orthogonal procrustes method and L1 methods are close to the solution of the zero norm problem, or at least close to one of its local minimum. Hence, by applying those new starting point, we expect that our algorithm will converge faster than the random initialization. Thus, we will denote W_5 for the Frank-Wolfe method initialized with the solution of the L1 norm method and W_6 as for the Frank-Wolfe method initialized with the solution to the orthogonal procrustes problem. However, the feasibility of the initialization point provided by different base line models should be guaranteed first. More precisely, notice that in our formulation, for each entry of the matrices X and Y and also for each row j of the matrix W, we have the following constraint:

$$-d_i^j \leq Wx_i - y_i \leq d_i^j \quad \forall j = 1, \ldots, 300$$

One hidden constraint on the entry d_i^j is that it takes only non negative values. Indeed, from the constraint above, we have that the only possibility that the inequality $-d_i^j \leq d_i^j$ hold is by letting variable d_i^j takes non negative values. This is really important as these constraints form the feasible set of our problem and for the Frank-Wolfe algorithm, one important thing is that it is a projection free algorithm as at each Frank-Wolfe step, we move towards one of the vertex of the polyhedron and we also set the step size to be smaller or equal to 1, thus, the new point generated by the Frank-Wolfe algorithm for sure is a feasible solution by our construction. On the other hand, if we start with a point outside of the feasible region, then the Frank-Wolfe method will not work and it will give us a wrong result. We can perform the analysis first with the error matrix obtained by the L1 norm method. For the result provided by L1 method, this non-negativity constraint is automatically satisfied as it is also the constraint for the L1 norm method. However, if we take a look at the error matrix provided by the procrustes method, we can see that this constraint is not satisfied as some of these entries are negative values. Thus, in order to make the solution of orthogonal procrustes method a feasible solution, we will first need to perform the projection of the result back to the feasible set. As discussed before, one of the constraint is that all a_t^j must take non negative value, thus projecting the non negative value back to the feasible set means to take the positive part of the negative entries. With this transformation, we can guarantee that our initialization is feasible.

And finally, we will introduce one more way to find the transformation matrix W but will not be considered in current work, which is by considering the top words according to the theta measure and then solve this new dataset with procrustes method as it is really efficient, later we choose the next word that has the highest theta measure and we add it to our dataset and again we apply the procrustes method. Later, the algorithm keeps going until we find a new word such that by adding that new word to our dataset, the mean value of the error increases by δ with δ being the threshold value. Since, it is based on a different technique, thus we will denote it with simply W without any subscripts. The intuition behind this final formulation is that by focusing only on the top words, we will find a matrix that can align well the stable words.

Before going further, we will first set up all the necessary parameters for the algorithms. More specifically, all the parameters required by the Frank Wolfe method such as cut-off condition ε and hyper-parameter α that is used for the approximation of the zero norm. For the ε, we decided to choose 10^{-12} and as for the hyper parameter α, we will set it to $\alpha = 5$. As for the step-size of the Frank Wolfe algorithm, we will set it to be a unit step size for the full Frank Wolfe algorithm and with the block-coordinate method, we will try to use a diminishing step sizes: $\frac{1}{k+1}$, $\frac{1}{k^2+1}$ and $\frac{1}{\sqrt{k}+1}$. Where the letter k refers to the number of iteration, notice that all these three step sizes are all lower or equal to 1 which can guarantee that the new iterate is still feasible. The difference between these three step sizes are the speed of converging to 0 when the iteration k keeps growing. In fact, out of all 3 step sizes, at the same iteration, $\frac{1}{\sqrt{k}+1}$ will have the highest value as it has the lowest speed of converging to 0 and on the other hand, $\frac{1}{k^2+1}$ will give the smallest step size value as it converges faster comparing to the other two step sizes. In conclusion, we can use these different step size to control the number of iteration needed for the convergence of the Block-Coordinate Frank Wolfe method.

5.2 Evaluation methodology

The experiment is composed by two different parts, the first part is about the general evaluation, we take both test sets explained in the previous subsection and given a word w in those sets, we will compute the mean value of the theta measure $\theta(W_i\vec{x}, \vec{y})$ and the mean of the absolute cosine similarity $s_i(w)$ after applying the alignments for all alignments discussed before. The second part of the experiment performs the analysis based on the top and bottom words based on s_i in the ascending order and we denote the top lists as H_i and the bottom list as L_i with the index i representing different alignments. We expect the words in the bottom list are more likely to be subjected to the semantic bias through the alignment W_i and words in the top lists H_I are less likely to be subjected to the semantic bias through the alignments. We later computer also the theta measure for the top and bottom list H_i and T_i for all the alignments.

Notice that the first three alignments W_1, W_2, W_3 are associated with different formulation of the objective functions, thus we will use them as the baseline models for our experiment. Later we will focus on the zero norm method and and we will try to implement some variants of the Frank- Wolfe algorithm to speed up the computation such as Block-coordinate Frank-Wolfe method and Frank-Wolfe method initialized with the solution of the Procrustes problem and L1 norm problem.

6 Experiment and conclusion

Within this chapter, we will report the result of our experiments and their respective analysis. Before we start, we will first perform a general evaluation on the test sets, more specifically, the sets H, B and S. Recall that these sets come from literature, personal annotation,etc which means that they are not supported by our evaluation measure. On the contrary, the top and bottom lists comes from the sorted theta measure array, thus they are automatically verified. As for the top and bottom lists, the same procedures are repeated as they are already verified by the theta measure, and with this test, we can check the effectiveness of the alignments. Considering a vocabulary D and for all word $w \in D$ in the set top/bottom lists, H, B and S, we compute their corresponding absolute cosine similarity values after applying three different based line alignments models and we later obtain the absolute cosine similarity $s_1(w), s_2(w), s_1(w)$ after the alignment i, where $\vec{x} \in V_{1890}$ and $\vec{y} \in V_{1990}$. The mean and variance of the absolute cosine similarity of our datasets without applying the alignments are (6.1):

Set	**bottom**	**top**	**H**	**B**	**S**
$s(w) \pm \sigma_{s(w)}$	0.266 ± 0.010	0.472 ± 0.010	0.378 ± 0.014	0.414 ± 0.018	0.458 ± 0.012

Table 6.1: Original theta measure without alignments.

Notice that here we have that the order based on the absolute cosine similarity are: bottom < H < B < S < top which are in line with our choice of these sets. Furthermore, the absolute cosine similarity after applying three alignments are in the following table (6.2):

Set	$s_1 \pm \sigma_{s_1}$	$s_2 \pm \sigma_{s_2}$	$s_3 \pm \sigma_{s_3}$
Top	0.614 ± 0.010	0.678 ± 0.009	0.677 ± 0.009
Bottom	0.362 ± 0.014	0.412 ± 0.015	0.411 ± 0.156
H	0.512 ± 0.022	0.569 ± 0.026	0.568 ± 0.026
B	0.557 ± 0.018	0.622 ± 0.018	0.621 ± 0.018
S	0.611 ± 0.012	0.668 ± 0.009	0.667 ± 0.009

Table 6.2: Table of result of absolute cosine similarity for all three base line models.

We will further provide a plot (6.1) of the absolute cosine similarity measures for both with and without alignments and for all the three base line models. From the table, we can see that for each alignment, they can really align the two dataset as the absolute cosine similarity has increased after the alignment which confirms the effectiveness of the alignment techniques. And more importantly, we can see that the order H < B < S based on the absolute cosine similarity is also respected and there are clearly some margins between set H, B and S that can guarantee the separability between these sets. Furthermore, from the table we can also see that the procrustes method gives the poorest result as it has an absolute cosine similarity value that is on average 0.5 lower than the other two for all the sets. We can also see that the second and third alignment have similar performance as they give almost equivalent results for all the set considered in our experiment.

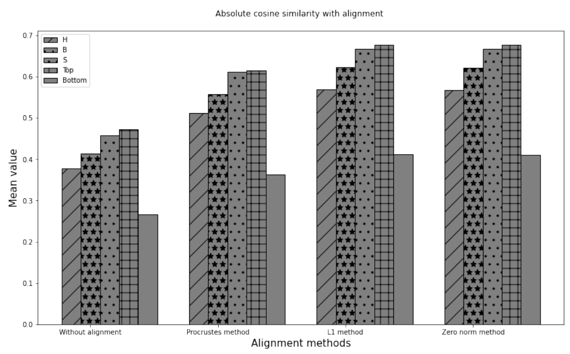


Figure 6.1: Result of absolute cosine similarity for all three base line models.

Similarly, we can also calculate the theta measure before and after the corresponding alignment and a bar plot with is presented in (6.2) and the result are also reported in the following table (6.3):

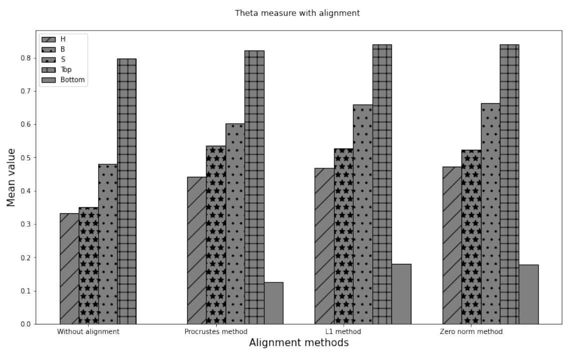


Figure 6.2: Result of theta measure ($K = 20$) for all three base line models.

Set	$\theta(W_1\vec{x},\vec{y}) \pm \sigma_{\theta(W_1\vec{x},\vec{y})}$	$\theta(W_2\vec{x},\vec{y}) \pm \sigma_{\theta(W_2\vec{x},\vec{y})}$	$\theta(W_3\vec{x},\vec{y}) \pm \sigma_{\theta(W_3\vec{x},\vec{y})}$
Top	0.821 ± 0.0	0.839 ± 0.0.	0.896 ± 0.0
Bottom	0.125 ± 0.0	0.18 ± 0.0	0.178 ± 0.0
H	0.441 ± 0.0	0.468 ± 0.0	0.472 ± 0.0
B	0.535 ± 0.0	0.527 ± 0.0	0.53 ± 0.0
S	0.602 ± 0.0	0.658 ± 0.0	0.663 ± 0.0

Table 6.3: Theta measure for all the test sets after applying all three alignments.

Comparing with the theta measure without alignment, we can notice that with all the alignments, the theta measure for all the sets increases which indicates that the alignments are correctly applied and we can also see that not only all of their theta measures increase with the alignment but also their corresponding order is respected. More importantly, we can notice that the theta measure for the bottom list has increased significantly with the alignments. Recall that these words are selected based on the lowest theta measures and the original mean value of the theta measure for the low list is essentially zero which means that they are the ones with the most rapid changes and their meanings have changes completely. However, with the alignment, it seems like we can mitigate this change as they provide a descent improvement to the mean values, more specifically, the mean value increases from 0 to almost 0.2 for some alignments. Furthermore, if we focus on the differences between the sets, we can see that the difference between set H and B are more evident after applying the alignments as their gap has increased a lot. Furthermore, from the plot we can see that the alignments W_2 and W_3 seem like to have a slightly better performance than the procrustes method for all the sets.

The second experiment we focus on the second test set namely, H, B, S and we later perform the evaluation of the top and bottom of the sorted ranking according to the absolute cosine similarity value s_i. We first consider only the test set H, B, S as we have already verified their level of bias they are subjected to in the first experiment. Now we can start the second experiment by first applying the absolute cosine similarity for all three base alignments and later we sort all the words based on the the absolute cosine similarity and thus 3 different sorted rankings for all three different alignments is obtained. The next step is to select both the set H_1, H_2, H_3 of top 50 words of the ranking and the set B_1, B_2, B_3 of bottom 50 words of all three ranking lists. And then, we will check the effectiveness of the alignments on these selected sets. We know that words in the lists have the properties that:

- H_i contains the last 50 words in r_i corresponding to the high absolute cosine similarity value, thus, for words $w \in H_i$, they are the stable ones after applying the alignment W_i.

- L_i contains the first 50 words in r_i corresponding to the low absolute cosine similarity value, thus, for words $w \in L_i$, they are the words that are subjected to a rapid change after applying the alignment W_i

And finally, we can perform analysis based these sets and their intersections with lists H_i and L_i by checking the intersections between the top lists H_i and the test sets and the bottom lists L_i and the test sets. Here, we will report these lists found by all three alignments.

H1
'hands', 'tone', 'knights', **'window'**, 'noise', 'tristram', 'sitting', 'sleep', 'face', 'hair', 'heavy', 'hand', 'night', 'oh', 'between', 'river', 'horses', 'five', 'walls', 'stairs', 'through', 'ye', 'words', 'sun', **'windows'**, **'fire'**, 'steps', 'voice', **'water'**, 'sea', 'church', 'gaheris', 'gate', 'sat', 'hear', 'years', **'god'**, 'horse', **'knight'**, 'beaumains', 'morning', 'floor', 'launcelot', 'evening', 'heard', 'hundred', 'trees', **'door'**, 'miles', 'marhaus'

H2
'horse', 'afternoon', 'hours', 'horses', 'eyes', **'window'**, 'sitting', 'thou', 'tried',

’gude’, ’brownlow’, ’standing’, ’morning’, ’hair’, ’mother’, **’chair’**, ’warm’, ’twenty’, **’knight’**, ’oh’, ’unto’, ’glass’, ’steps’, ’kenwigs’, ’sea’, ’stairs’, ’voice’, ’hundred’, ’church’, ’floor’, ’ye’, ’hear’, ’years’, **’windows’**, ’jesu’, ’launcelot’, ’five’, **’water’**, ’balin’, ’evening’, ’miles’, **’door’**, ’gat’, ’trees’, ’hight’, ’sat’, ’heard’, ’beaumains’, ’gaheris’, ’marhaus’

H3
’horse’, ’horses’, ’hours’, ’afternoon’, ’eyes’, ’window’, **’hair’**, ’tried’, ’gude’, ’sitting’, ’thou’, ’brownlow’, ’standing’, ’morning’, ’mother’, ’chair’, ’twenty’, ’warm’, **’knight’**, ’unto’, ’oh’, ’glass’, ’steps’, ’kenwigs’, ’church’, ’sea’, ’stairs’, ’hundred’, ’voice’, ’floor’, ’ye’, ’hear’, ’years’, ’jesu’, **’windows’**, ’launcelot’, ’five’, **’water’**, ’evening’, ’balin’, ’miles’, **’door’**, ’gat’, ’trees’, ’hight’, ’sat’, ’heard’, ’beaumains’, ’gaheris’, ’marhaus’

L1
’guy’, ’gavin’, ’dow’, ’egyptian’, ’dad’, ’ut’, ’mac’, ’tony’, ’quentin’, **’checking’**, ’nanny’, ’cranford’, ’vis’, ’jan’, ’barnaby’, ’georgie’, ’percy’, ’66’, ’dinah’, ’kenneth’, ’barker’, ’deborah’, **’gypsy’**, ’jessie’, ’overdue’, ’plight’, ’molly’, ’comer’, ’ta’, ’obviously’, ’jean’, ’hermiston’, ’gilbert’, ’comers’, ’wanting’, ’designs’, ’jenny’, ’charlie’, ’catriona’, ’matilda’, ’glen’, ’wight’, ’martha’, ’romances’, ’palmer’, ’waverley’, ’foster’, ’presumed’, ’tho’, ’y’

L2
’guy’, ’dow’, ’tony’, ’quentin’, ’gavin’, ’georgie’, ’vis’, ’egyptian’, **’checking’**, ’mac’, ’ut’, ’barnaby’, ’jan’, ’the’, ’dinah’, **’gypsy’**, ’cranford’, ’comer’, ’catriona’, ’percy’, ’deborah’, ’dad’, ’gilbert’, ’jessie’, ’palmer’, ’jason’, ’kenneth’, ’martha’, ’barker’, ’edith’, ’foster’, ’nanny’, ’ta’, ’overdue’, ’scot’, ’almayer’, ’comers’, ’66’, ’covers’, ’esther’, ’signor’, ’judy’, ’headed’, ’romances’, ’clennam’, ’tho’, ’hilda’, ’urge’, ’amy’, ’jean’

L3
’guy’, ’dow’, ’tony’, ’quentin’, ’gavin’, ’georgie’, ’egyptian’, ’vis’, **’checking’**, ’mac’, ’ut’, ’barnaby’, ’jan’, ’the’, ’dinah’, **’gypsy’**, ’cranford’, ’comer’, ’percy’, ’catriona’,

'deborah', 'dad', 'gilbert', 'palmer', 'jessie', 'jason', 'martha', 'kenneth', 'edith', 'barker', 'foster', 'nanny', 'ta', 'scot', 'overdue', 'almayer', 'comers', '66', 'covers', 'esther', 'signor', 'headed', 'judy', 'romances', 'clennam', 'hilda', 'tho', 'urge', 'weaving', 'plight'

From the dataset above, we represent the intersections between our test sets H, B, S and the corresponding top and bottom lists in bold. Since we have already verified our test set, thus the for set H, since it has a rapid transformation of its meaning, thus, we expect the words in H to be present in set H and probably also in B, but we will never expect it to be in S as those are the stables words and by construction. Similarly, we should expect opposite result for H_i. Our ideal alignment has intersection with only set H if that word comes from list L_i, vice versa, our ideal alignment has intersection with only set S if that word comes from list H_i. We can see that for the words in the bottom lists L_i, all three alignments picked the same words in the test set and all of alignment ignored the set S which means that they correctly separated set S with other sets. Now, let us check the list H_i, all the alignments provided 5 words in the stable set S which mean they correctly labeled those words, then L1 and zero norm method selected the same word in set H, which means that they couldn't separate those words correctly. However, comparing to the Procrustes alignment, we can see that if further made 1 more mistake as it labeled the word 'god' as word that is subjected to a more rapid semantic change. We further made a more detailed report about the number of intersecting words in the following table (6.4):

Set	L_1	H_1	L_2	H_2	L_3	H_3
H	2	3	2	2	2	2
B	1	0	1	0	1	0
S	0	5	0	5	0	5

Table 6.4: Number of intersecting words between test sets and the top and bottom lists.

We further calculated the mean and variance for the top and bottom lists so that we can gain a better quantitative insight. In order to compare them, we will first report the s_E value which corresponds to the case without any alignment and then we will also report the same statistics after applying the corresponding alignments. And these numbers are reported in the tables (6.5) (6.6) below:

Set	s_E
L_1	0.124 ± 0.005
H_1	0.615 ± 0.002
L_2	0.131 ± 0.009
H_2	0.622 ± 0.002
L_3	0.129 ± 0.008
H_3	0.622 ± 0.002

Table 6.5: s_E values for all the top and bottom lists.

Set	$s_1 \pm \sigma_{s_1}$
L_1	0.180 ± 0.004
H_1	0.767 ± 0.0
	$s_2 \pm \sigma_{s_2}$
L_2	0.214 ± 0.005
H_2	0.819 ± 0.0
	$s_3 \pm \sigma_{s_3}$
L_3	0.212 ± 0.005
H_3	0.819 ± 0.0

Table 6.6: Absolute cosine similarity with alignments for all the top and bottom lists.

In order to give a better presentation, we will now report the relevant bar plot (6.3) with for each alignment, the two bars at the left represent the mean value without the alignment and the two bars at the right represent the mean value with the corresponding alignment method.

From both the table and figure above, we can see that for all the top and bottom lists, after the alignments, the absolute cosine similarity increases. Furthermore, we can again notice that L1 and zero norm methods out-performed the Procrustes method. We can also observe that for the words in the bottoms list, their mean values has increased by a small value, on average around 0.05. While as for the words in the top lists, we can see that they have increased at least three times as much as the words in the bottom lists. This phenomenon is especially evident for L1 and zero norm alignments.

Now, we will repeat the same procedures above, but this time, we consider the theta measure

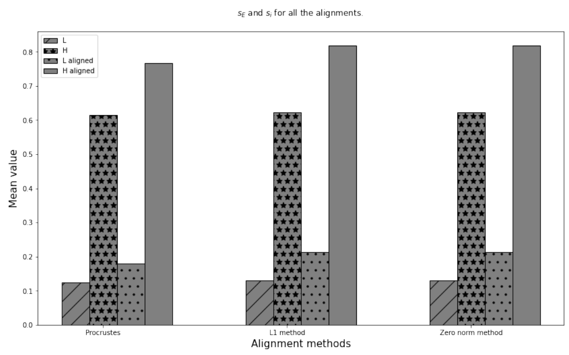


Figure 6.3: Comparison of s_E and s_i values for all alignments.

$\theta(\vec{x}, \vec{y})$ without alignments and theta measure $\theta(W_i\vec{x}, \vec{y})$ for word representation $\vec{x} \in V_{1890}$ and $\vec{y} \in V_{1990}$ and W_i being the corresponding alignment. The final result is reported in the table (6.7) and (6.8)

Set	$\theta(\vec{x}, \vec{y}) \pm \sigma_{\theta(\vec{x}, \vec{y})}$
L_1	0.045 ± 0.0
H_1	0.653 ± 0.0
L_2	0.032 ± 0.0
H_2	0.658 ± 0.0
L_3	0.030 ± 0.0
H_3	0.658 ± 0.0

Table 6.7: Theta measure without alignments for all the top and bottom lists.

From the tables above, we can notice that again our alignments have increased the theta measure of our top and bottom lists. However, we can also notice that even though the value increases for the bottom lists, but this increment is barely significant. On the other hand, we can see the top lists are all aligned well by our alignments, which gives us a result that our alignment can align well the stable words and leave the unstable ones unchanged. Furthermore, unlike the absolute cosine similarity, for the top lists, all three alignments performed equally. We will also report the bar plots in the following figure (6.4). And again, for the same position, the two bars on the left side are the mean values obtained before applying the corresponding alignment and the ones on the right side corresponds to those that are subjected to alignment W_i.

Set	$\theta(W_i\vec{x},\vec{y}) \pm \sigma_{\theta(W_i\vec{x},\vec{y})}$
L_1	0.056 ± 0.0
H_1	0.794 ± 0.0
L_2	0.047 ± 0.0
H_2	0.803 ± 0.0
L_3	0.043 ± 0.0
H_3	0.803 ± 0.0

Table 6.8: Theta measure with different alignments for all the top and bottom lists.

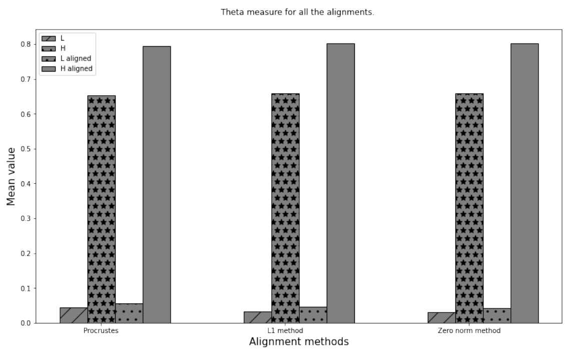


Figure 6.4: Comparison of theta measure for all the alignments.

6.1 Frank-Wolfe and its variants

In this subsection, we will focus on the variants of the Frank Wolfe algorithm and try to implement various variants and with different initialization in order to reduce the computational time and keep the comparable performance at the same time. Before moving further, we will first report the time needed for the calculation of the three base line alignments. And it is reported in the table (6.9).

	Procrustes	L1 method	zero norm
time	< 1 s	24390 s	113773 s

Table 6.9: Time needed for the computation of the baseline alignments.

We can clearly see that the fastest one is the Procrustes method, as even though it uses the SVD for the final construction of the solution, but essentially, even SVD is just simple matrix multiplication, and with numpy, these matrix multiplications can be solve easily, thus giving us a really fast computation. As for the L1 method and the zero norm method, we have to first report that in order to obtain the alignment W_3, we have set the maximum iteration to 5, the Frank-Wolfe algorithm failed to converge in 5 iterations. Now if we take the mean value of each iteration of the Frank-Wolfe problem, we can see that it is similar to the time required to compute with the L1 method. This is reasonable, as L1 and zero norm methods essentially have the same constraints and dimensions, thus they require similar amount of time to process the data. Furthermore, even though the full Frank-Wolfe method didn't converge in 5 iterations but this result is still acceptable as the there isn't a lot of improvement between iterations. In the following table (6.10), we will report the objective value and the gaps for each iteration. Furthermore, we will provide a plot of the value of the objective function.

Objective function	92964.881	92763.364	92761.556	92761.368	92761.315
Gap at each iteration	-638166.272	-181.8135	-1.267	-0.108	-0.025

Table 6.10: Gap and value of the objective function for the full Frank-Wolfe algorithm.

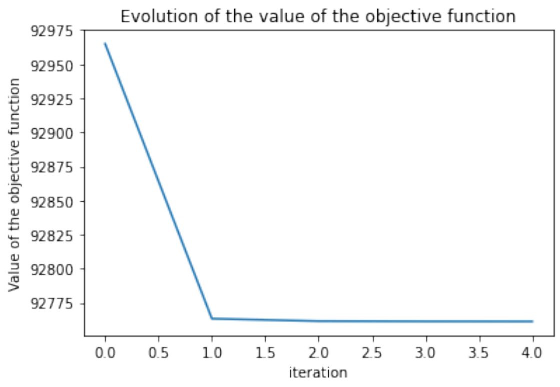


Figure 6.5: Value of the objective function

From both the table and the figure above, we can see that the value of the objective function decreased rapidly for the first Frank-Wolfe iteration and for the other iterations, this improvement is limited. Furthermore, we can also see the gap has been continuously decreasing but again not by a significant amount. For both reasons stated above, we are already close to the minima and if we keep running, the algorithm will keep improving, but this improvement is negligible. Thus, we can conclude that our alignment is relatively good.

Now, we can apply the experiments to the test sets H, B, S with Block-coordinate Frank Wolfe and BCFW initialized with the solution found by the L1 method and perform the standard Frank Wolfe initialized with the projected solution of the Procrustes problem. Again, we will first report in the following table (6.11)the time needed to obtain these alignments and their final value of the objective function.

	Procrustes and FW	L1 and BCFW	BCFW
time	111684 s	24473 s	21506 s
value	92795.8	730102	730103

Table 6.11: Time needed for the computation of various alignments.

From the table above, we can see that the procrustes initialization gives the lowest value of the objective function but in the cost of a long computational time. While the other two methods require way less time, but their value of objective function are way higher than the procrustes initialization. Let us again apply these alignment and apply the same analysis as before to see their performance in bias analysis. The theta measure before and after all these alignments are (6.12):

Set	$\theta(W_4\vec{x},\vec{y}) \pm \sigma_{\theta(W_4\vec{x},\vec{y})}$	$\theta(W_5\vec{x},\vec{y}) \pm \sigma_{\theta(W_5\vec{x},\vec{y})}$	$\theta(W_6\vec{x},\vec{y}) \pm \sigma_{\theta(W_6\vec{x},\vec{y})}$
Top	0.838 ± 0.0	0.846 ± 0.0.	0.840 ± 0.0
Bottom	0.179 ± 0.0	0.171 ± 0.0	0.176 ± 0.0
H	0.472 ± 0.0	0.471 ± 0.0	0.457 ± 0.0
B	0.527 ± 0.0	0.527 ± 0.0	0.523 ± 0.0
S	0.660 ± 0.0	0.655 ± 0.0	0.630 ± 0.0

Table 6.12: Theta measure for all the test sets after applying variants of Frank-Wolfe.

Similarly, we also provide a plot (6.6) for the mean values of the theta measure.

We can notice that, all three methods gives essentially the same result and now let us check the absolute cosine similarity with table (6.13) and plot (6.7).

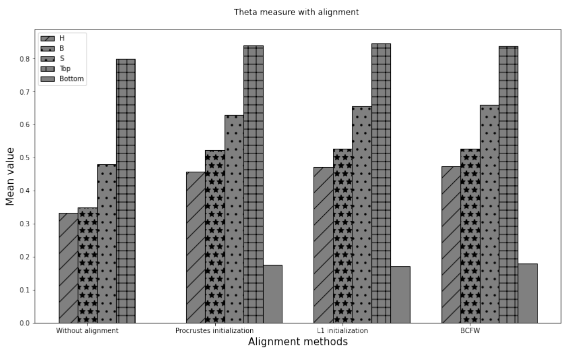


Figure 6.6: Mean value of theta measure with variants of FW.

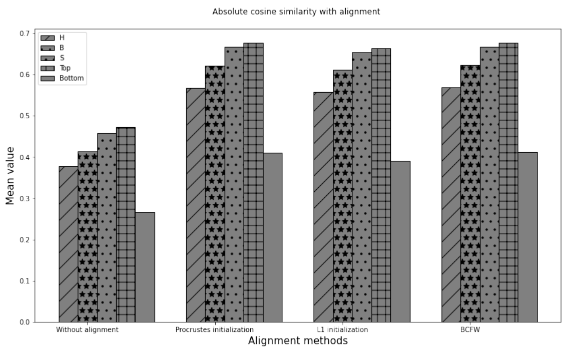


Figure 6.7: Mean value of cosine similarity with variants of FW.

Set	$s_4 \pm \sigma_{s_4}$	$s_5 \pm \sigma_{s_5}$	$s_6 \pm \sigma_{s_6}$
Top	0.678 ± 0.009	0.664 ± 0.012	0.677 ± 0.009
Bottom	0.412 ± 0.015	0.390 ± 0.016	0.411 ± 0.156
H	0.569 ± 0.029	0.558 ± 0.028	0.568 ± 0.026
B	0.622 ± 0.018	0.611 ± 0.022	0.621 ± 0.018
S	0.668 ± 0.009	0.655 ± 0.011	0.667 ± 0.009

Table 6.13: Table of result of absolute cosine similarity for variants of FW.

From the result above, we can see that the L1 initialization has a weaker performance in term of the absolute cosine similarity. The other two methods have similar performance, but we have to remember that the Procrustes initialization takes way longer to calculate which makes it less favorable.

Now, we apply the same method to get the top and bottom lists with different alignments and these lists are:

H_4
'horse', 'afternoon', 'hours', 'horses', 'eyes', 'window', 'sitting', 'thou', 'gude', 'tried', 'brownlow', 'standing', 'morning', 'hair', 'mother', 'chair', 'warm', 'twenty', 'knight', 'unto', 'oh', 'glass', 'steps', 'kenwigs', 'sea', 'stairs', 'voice', 'hundred', 'church', 'ye', 'floor', 'hear', 'years', 'windows', 'jesu', 'launcelot', 'five', 'water', 'balin', 'evening', 'miles', 'door', 'gat', 'trees', 'hight', 'sat', 'heard', 'beaumains', 'gaheris', 'marhaus'

H_5
'eyes', 'strength', 'faces', 'sun', 'voice', 'kenwigs', 'slight', 'god', 'oh', 'standing', 'sleep', 'wife', 'hands', 'steps', 'hair', 'church', 'gude', 'chair', 'tried', 'unto', 'afternoon', 'sitting', 'morning', 'stairs', 'sit', 'warm', 'knight', 'five', 'ye', 'jesu', 'gat', 'balin', 'sea', 'floor', 'miles', 'years', 'mother', 'glass', 'door', 'hear', 'evening', 'windows', 'hight', 'water', 'sat', 'trees', 'beaumains', 'heard', 'gaheris', 'marhaus'

H_6
'horse', 'horses', 'hours', 'afternoon', 'eyes', 'window', 'hair', 'tried', 'gude', 'thou',

’sitting’, ’brownlow’, ’morning’, ’standing’, ’chair’, ’twenty’, ’mother’, ’warm’, ’knight’, ’oh’, ’unto’, ’glass’, ’steps’, ’church’, ’kenwigs’, ’sea’, ’stairs’, ’voice’, ’hundred’, ’ye’, ’floor’, ’hear’, ’years’, ’jesu’, ’windows’, ’launcelot’, ’water’, ’five’, ’evening’, ’balin’, ’miles’, ’door’, ’gat’, ’trees’, ’sat’, ’hight’, ’heard’, ’beaumains’, ’gaheris’, ’marhaus’

L_4
’guy’, ’dow’, ’tony’, ’quentin’, ’gavin’, ’georgie’, ’vis’, ’egyptian’, ’checking’, ’mac’, ’ut’, ’barnaby’, ’jan’, ’the’, ’dinah’, ’gypsy’, ’cranford’, ’comer’, ’catriona’, ’percy’, ’deborah’, ’dad’, ’gilbert’, ’jessie’, ’palmer’, ’jason’, ’kenneth’, ’martha’, ’barker’, ’edith’, ’foster’, ’nanny’, ’ta’, ’overdue’, ’scot’, ’almayer’, ’comers’, ’66’, ’covers’, ’esther’, ’signor’, ’judy’, ’headed’, ’romances’, ’clennam’, ’tho’, ’hilda’, ’urge’, ’amy’, ’jean’

L_5
’guy’, ’georgie’, ’tony’, ’gavin’, ’quentin’, ’vis’, ’egyptian’, ’dow’, ’mac’, ’jan’, ’ut’, ’checking’, ’barnaby’, ’dinah’, ’cranford’, ’the’, ’martha’, ’gilbert’, ’gypsy’, ’nanny’, ’deborah’, ’comer’, ’percy’, ’barker’, ’dad’, ’jessie’, ’kenneth’, ’edith’, ’jason’, ’judy’, ’catriona’, ’66’, ’palmer’, ’foster’, ’scot’, ’almayer’, ’joan’, ’jenny’, ’romances’, ’comers’, ’overdue’, ’jean’, ’esther’, ’amy’, ’covers’, ’matthew’, ’hilda’, ’clennam’, ’signor’, ’charlie’

L_6
’guy’, ’dow’, ’tony’, ’quentin’, ’gavin’, ’georgie’, ’egyptian’, ’vis’, ’checking’, ’mac’, ’ut’, ’barnaby’, ’jan’, ’the’, ’dinah’, ’gypsy’, ’cranford’, ’comer’, ’percy’, ’catriona’, ’deborah’, ’dad’, ’gilbert’, ’jessie’, ’palmer’, ’jason’, ’martha’, ’kenneth’, ’barker’, ’edith’, ’foster’, ’nanny’, ’ta’, ’scot’, ’overdue’, ’almayer’, ’comers’, ’66’, ’covers’, ’esther’, ’signor’, ’headed’, ’judy’, ’romances’, ’clennam’, ’hilda’, ’tho’, ’urge’, ’weaving’, ’plight’

And we will also report the intersection between these sets and test sets H, B, S (6.14).

With the intersection table above, we can see that all the alignments performed well with the bottom list, however, when it comes to the top lists, the alignment with L1 initialization seems like to be poorly performed compared with the other two as more errors are made on set H and

Set	L_4	H_4	L_5	H_5	L_6	H_6
H	2	2	2	3	2	2
B	1	0	1	0	1	0
S	0	5	0	3	0	5

Table 6.14: Number of intersecting words between test sets and the top and bottom lists.

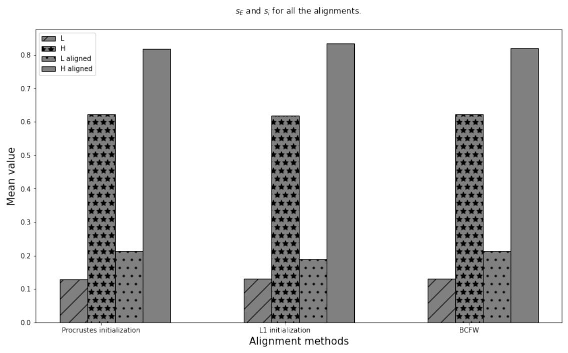


Figure 6.8: Mean value of absolute cosine similarity with variants of FW for top and bottom lists.

less correctly labeled stable words in S. As for the absolute cosine similarity for those lists, we reported the result in the following following plot (6.8).

Again from the plot above, we can see that the alignments performed in a similar manner, thus considering all these data above, it seems like that BCFW is overall the best option for this task as it both requires less time and a relatively effective analysis ability.

6.2 Conclusion and future work

With this thesis work, we have performed the diachronic bias analysis of the word embedding using the alignment techniques. From a performance point of view, it appears that the L1 and zero norm methods performed better compared to the Procrustes method, however, with a much higher computational cost. Thus, for this specific task, if we do not require the precision of the analysis, the Procrustes methods appear to be the best choice as it takes a couple of seconds to get a descent result instead of a few hours. Later, we further exploited the variants of the Frank-Wolfe algorithm and we proposed various starting point for the algorithm. From our analysis we can see that from all the proposed methods performed well in terms of the bias analysis, however, the computational cost of the procrustes initialization is way higher than the other two versions which means that our hypothesis that the solution of the procrustes problem is close to a local minima does not hold true, and in fact, we need the same number of iterations to get to the final result as the full Frank-Wolfe algorithm which makes it useless in terms of the time reduction. On the other hand, the L1 initialization and BCFW methods are way more computationally efficient than the procrustes initialization and also they require less than a quarter of the time for the full Frank-Wolfe method. But in terms of the performance of the bias analysis, even though the result obtained with the L1 initialization methods is reasonable, but it still cannot match up with the BCFW method. This indicates that the solution obtained by the L1 method is indeed close to the local minima, but this minima does not appear to be too close to the global minima of the Frank-Wolfe algorithm. Thus, if we want to achieve a good bias analysis performance while spending reasonable time, then the BCFW seems to be the best option. Furthermore, it is worth mentioning that even if the initialization of the technique can provide us with a reasonable result, but it is still more interesting to study about the BCFW method, as this method generally speaking works for all types of alignment problem. And finally, our result also indicates that the alignments is more sensitive to the stable words by the mean of theta measure. And they also appear to be efficient for the unstable words in term of absolute cosine similarity.

Several future works related to this thesis can be interesting to study such as studying the initialization of this problem that is finding a new way to start close to a better local minima than the one found by L1 method. Another interesting direction can be using the procrustes method to perform the analysis on the evolution of the meaning of words and also coeval political data as these types of tasks require a huge amount of computation and with the procrustes method, a reasonable result can be obtained in a really fast way. Other interesting work can be the implementation of the alignment W_7 introduced in this thesis work which is finding the alignment by adding one word at a time and stop when the change is large enough.

References

[1] M. S. P. P. Simon Lacoste-Julien, Martin Jaggi, *Block-Coordinate Frank-Wolfe Optimization for Structural SVMs*.

[2] J. Y. Z. V. S. Tolga Bolukbasi, Kai-Wei Chang and A. T. Kalai, *Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings*. Advances in Neural information processing systems, 29, 2016.

[3] J. L. William L. Hamilton and D. Jurafsky, *Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change*. arXiv preprint arXiv:1605.09096, 2016.

[4] S. S. B. N. T. Adam Jatowt, Ricardo Campos and A. Doucet, *Every word has its history: Interactive exploration and visualization of word sense evolution*. Proceedings of the 27th ACM In- ternational Conference on Information and Knowledge Management, pages 1899–1902, 2018.

[5] D. Jurafsky and J. Martin, *Speech and Language Processing*. Prentice Hall, 2000.

[6] Z. Yin and Y. Shen, *On the Dimensionality of Word Embedding*. Advances in neural information processing systems, 2018.

[7] W. t. Y. Tomas Mikolov and G. Zweig., "Linguistic regularities in continuous space word representations," 2013a.

[8] Y. G. Omer Levy and I. Dagan, *Improving Distributional Similarity with Lessons Learned from Word Embeddings*. Transactions of the association for computational linguistics, 3:211–225,, 2015a.

[9] A. J. Piotr Bojanowski, Edouard Grave and T. Mikolov, *Enriching word vectors with subword information*. Transactions of the Association for Computational Linguistics, 5:135–146,, 2017.

[10] M. Joos, *Description of language design*. JASA, 1950.

[11] Z. S. Harris, *Distributional structure*. Word, 10:146–162, 1954.

[12] J. R. Firth, *A synopsis of linguistic theory 1930–1955*. In Studies in Linguistic Analysis. Philological Society, 1957.

[13] G. Atienza, *Operations on Word Vectors*. https://github.com/gemaatienza/Deep-Learning-Coursera, 2018.

[14] D. E. Rumelhart and A. A. Abrahamson, *A Model for Analogical Reasoning*. Cognitive Psychology, 5(1):1–28, 1973.

[15] M. L. Aston Zhang, Zachary C. Lipton and A. J. Smola, *Dive into Deep Learning*. arXiv preprint arXiv:2106.11342, 2021.

[16] https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/.

[17] G. C. Tomas Mikolov, Kai Chen and J. Dean, *Efficient Estimation of Word representations in Vector Space*. arXiv preprint arXiv:1301.3781,, 2013b.

[18] R. S. Jeffrey Pennington and C. D. Manning, *Glove: Global Vectors for Word Representation*. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 1532–1543, 2014.

[19] B. Madhukar, *The Continuous Bag Of Words (CBOW) Model in NLP – Hands-On Implementation With Codes*.

[20] A. Kalinowski and Y. An, *A Survey of Embedding Space Alignment Methods for Language and Knowledge Graphs*. arXiv preprint arXiv:2010.13688, 2020.

[21] J. L. William L. Hamilton and D. Jurafsky, *Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change*. arXiv preprint arXiv:1605.09096, 2016.

[22] I. V. Sebastian Ruder and A. Søgaard, *A Survey of Cross-Lingual Word Embedding Models*. Journal of Artificial Intelligence Research, 65: 569–631, 2019.

[23] A. Mogadala and A. Rettinger, *Bilingual Word Embeddings from Parallel and Non-parallel Corpora for Cross-Language Text Classification*. Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 692–702, San Diego, California, 2016.

[24] P. Prettenhofer and B. Stein, *Cross-Language Text Classification Using Structural Correspondence Learning*. Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 1118–1127, Uppsala, Sweden, 2010.

[25] D. C. Will Y. Zou, Richard Socher and C. D. Manning, *Bilingual Word Embeddings for Phrase-Based Machine Translation*. Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 1393–1398, Seattle, Washington, USA, 2013.

[26] Q. V. L. Tomas Mikolov and I. Sutskever, *Exploiting Similarities among Languages for Machine Translation*. arXiv preprint arXiv:1309.4168, 2013c.

[27] P. H. Schönemann, *A Generalized Solution of the Orthogonal Procrustes Problem*. Psychometrika, 31(1):1–10, 1966.

[28] C. L. Chao Xing, Dong Wang and Y. Lin., *Normalized word embedding and orthogonal transform for bilingual word translation*. Proceedings of the 2015 conference of the North American chapter of the association for computational linguistics: human language technologies, pages 1006–1011, 2015.

[29] G. L. Mikel Artetxe and E. Agirre, *Learning principled bilingual mappings of word embeddings while preserving monolingual invariance*. Proceedings of the 2016 conference on empirical methods in natural language processing, pages 2289–2294, 2016.

[30] *OED Oxford English Dictionary*. https://www.oed.com/view/Entry/18564?rskey=xGzzk8result=1isAdva 2021.

[31] K. Crawford, *The trouble with bias*. Proceedings of NeurIPS 2017, 2017.

[32] B. Friedman and H. Nissenbau, *Bias in Computer Systems*. ACM Transactions on information systems (TOIS), 14(3):330–347, 1996.

[33] H. D. I. Su Lin Blodgett, Solon Barocas and H. Wallach, *Language (Technology) is Power: A Critical Survey of" Bias" in NLP*. arXiv preprint arXiv:2005.14050, 2020.

[34] T. C. R. W. Aylin Caliskan, Pimparkar Parth Ajay and M. R. Banaji, *Gender Bias in Word Embeddings: A Comprehensive Analysis of Frequency, Syntax, and Semantics*. arXiv preprint arXiv:2206.03390, 2022.

[35] J. J. B. Aylin Caliskan and A. Narayanan, *Semantics derived automatically from language corpora contain human-like biases*. Science, 356(6334):183–186, 2017.

[36] D. J. Nikhil Garg, Londa Schiebinger and J. Zou, *Word embeddings quantify 100 years of gender and ethnic stereotypes*. Proceedings of the National Academy of Sciences, 115(16):E3635–E3644, 2018.

[37] S. Kiritchenko and S. M. Mohammad, *Examining Gender and Race Bias in Two Hundred Sentiment Analysis Systems*. arXiv preprint arXiv:1805.04508, 2018.

[38] D. B. Thomas Davidson and I. Weber, *Racial bias in hate speech and abusive language detection datasets*. arXiv preprint arXiv:1905.12516, 2019.

[39] M. B. Josh Gordon and J. Matthews, *Studying Political Bias via Word Embeddings*. WWW '20: Companion Proceedings of the Web Conference 2020, pages 760–764, 2020.

[40] P. N. Emily Sheng, Kai-Wei Chang and N. Peng, *Societal Biases in Language generation: Progress and Challenges*. arXiv preprint arXiv:2105.04054, 2021.

[41] L. Bloomfield, *Language*. George Allen UNWIN LTD, 1933.

[42] C. Dictionary, *diachronic*. https://www.collinsdictionary.com/dictionary/english/, 2022a.

[43] E. A. L. J. O. W. B. Yuri Lin, Jean-Baptiste Michel and S. Petrov., *Syntactic Annotations for the Google Books NGram Corpus*. Proceedings of the ACL 2012 system demonstrations, pages 169–174, 2012.

[44] GoogleBooks, *The Google Books Ngram Viewer*. https://storage.googleapis.com/books/ngrams/books/datasetsv3.html, 2013.

[45] M. Davies, *Expanding horizons in historical linguistics with the 400 million word Corpus of Historical American English*. Corpora, 7(2):121–157, 2012, 2012.

[46] Y. G. Omer Levy and I. Dagan, *Improving Distributional Similarity with Lessons Learned from Word Embeddings*. Transactions of the association for computational linguistics, 3:211–225, 2015b.

[47] K. i. K. S. J. Mozhi Zhang, Keyulu Xu and J. Boyd-Graber, *Are Girls Neko or Shojo? Cross-Lingual Alignment of Non-Isomorphic Embeddings with Iterative Normalization*. arXiv preprint arXiv:1906.01622, 2019.

[48] S. R. Anders Søgaard and I. Vulić, *On the Limitations of Unsupervised Bilingual Dictionary Induction*. arXiv preprint arXiv:1805.03620, 2018.

[49] M. R. L. D. Alexis Conneau, Guillaume Lample and H. Jégou, *Word translation without parallel data*. arXiv preprint arXiv:1710.04087, 2017.

[50] K. V. Amaldi, E., *On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems*. Theor. Comput. Sci. 209, 237–260, 1998.

[51] M. S. Francesco Rinaldi and F. Shoen, *Concave programming for minimizing the zero-norm over polyhedral sets*. Comput Optim Appl 46, 2008.

[52] E. D. Casa, *Bias Analysis in Word Embeddings with Alignment Techniques*, 2023.

Acknowledgments

Here we will report the code used in this thesis. The first one is the Frank-Wolf algorithm

```
def FW_alignment(X, Y, grad_in , d, N):
    """
    Solve the Frank-Wolfe alignment at the
    iteration k with the PuLP package.
    Return matrices A, W and zero norm array
    z_new (the output of iteration k and input
    of iteration k + 1 if necessary).
    ------------------------------------------------
    INPUT :
    X : Matrix of the first embedding.
    Y : Matrix of the second embedding.
    grad_k : Array that contains the gradient of the
    function at iteration k.
    z_in : Output zero norm array of the iteration
    k - 1 and input of the iteration k.
    d : Dimension of the embedding, typically equals to 300.
    N : Number of words of the vocabulary.
    ------------------------------------------------
    OUTPUT :
    W : Alignment matrix (d X d)
    A : Matrix that contains the error of the alignment
    "WX - Y" (d X N)
    ------------------------------------------------
    """

    # Initialize dictionaries for the result
    A = {}
```

```
W = {}

# Set up the dictionary of variables
mat_w = [(i, j) for i in range(d) for j in range(d)]
mat_a = [(i, j) for i in range(d) for j in range(N)]

# Set up the variables
dict_a = LpVariable.dicts('A', mat_a, lowBound=0)
dict_w = LpVariable.dicts('W', mat_w)

# Solve d different sub-problem
for j in range(d):
    # solve the j-th row of the matrix W (the j-th sub-problem)

    """
    Superscript ^ represents the ROWS of a matrix
    Subscript _ represents the COLUMNS of a matrix
    """
    # Define the problem and set up the objective function
    """
    argmin Σ_i (g_k)_i * a_i^j        for i = 1, ....., N
    """
    prob_FW = LpProblem('Frank-Wolfe', LpMinimize)

    prob_FW += (
        lpSum(grad_in[j, i] * dict_a[j, i] for i in range(N))
    )

    # Add the constraint to the FW problem
    for i in range(N):
        """
        for n in range(d) since it's a dot product of two vectors
        in R^d for i in range range(N) to access the i-th position
        of the row vector A^j
        """

        prob_FW += (
            - dict_a[j, i] <= lpSum(dict_w[j, n] * X[n, i] for n in
        )

        prob_FW += (
            lpSum(dict_w[j, n] * X[n, i] for n in range(d)) - Y[j,
        )

    # Solve the problem
    prob_FW.solve(GUROBI_CMD(msg = True))

    # print warning when optimal solution is not reached
    if LpStatus[prob_FW.status] != "Optimal":
        print('Warning␣:␣The␣result␣is␣NOT␣optimal!')

    # Store the results in the corresponding dictionary
    for v in prob_FW.variables():
        if v.name.startswith('A'):
            A[v.name[3: -1].replace('_', "")] = v.varValue
        else:
            W[v.name[3: -1].replace('_', "")] = v.varValue
    print(f'The␣row␣{j}␣is␣solved!')
return W, A
```

The second one we will report is the BCFW.

```
def BCFW(X, Y, grad_k, W_in, A_in, d, N, index_sel):
"""
Solve the Frank-Wolfe alignment at the iteration k with
the PuLP package.
Return matrices A, W and zero norm array z_new
(the output of iteration k and input of iteration k + 1 if necessar
-----------------------------------------------------
```

```
INPUT :
X : Matrix of the first embedding.
Y : Matrix of the second embedding.
grad_k : Array that contains the gradient of the function
at iteration k.
z_in : Output zero norm array of the iteration k - 1 and
input of the iteration k.
d : Dimension of the embedding, typically equals to 300.
N : Number of words of the vocabulary.
W_in : Input alignment from the previous iteration.
A_in : Input error matrix from the precious iteration.
index_sel : Random block selected.
--------------------------------------------------------
OUTPUT :
W : Alignment matrix (d X d)
A : Matrix that contains the error of the alignment
"WX - Y" (d X N)
--------------------------------------------------------
"""

# Initialize dictionaries for the result
A = A_in
W = W_in

# Set up the dictionary of variables
mat_w = [(i, j) for i in range(d) for j in range(d)]
mat_a = [(i, j) for i in range(d) for j in range(N)]


# Solve d different sub-problem

for j in range(d):
    # solve the j-th row of the matrix W(the j-th sub-problem)
```

```
"""
Superscript ^ represents the ROWS of a matrix
Subscript _ represents the COLUMNS of a matrix
"""

if j in index_sel:   # Optimize this row
    # Define the problem and set up the objective function
    print(f'Calculating␣row␣{j}...')

    """
    argmin Σ_i (g_k)_i * a_i^j      for i = 1, ....., N
    """
    # Set up the dictionary of variables
    mat_w = [(j, i) for i in range(d)]
    mat_a = [(j, i) for i in range(N)]
    prob_FW = LpProblem('Frank-Wolfe', LpMinimize)

    # Set up the variables
    dict_a = LpVariable.dicts('A', mat_a, lowBound=0)
    dict_w = LpVariable.dicts('W', mat_w)
    prob_FW += (
        lpSum(grad_k[j, i] * dict_a[j, i] for i in range(N))
    )

    # Add the constraint to the FW problem
    for i in range(N):
        """
        for n in range(d) since it's a dot product of two
        vectors in R^d for i in range range(N) to access the i-
        """

        prob_FW += (
            - dict_a[j, i] <= lpSum(dict_w[j, n] * X[n, i] for
        )
```

```
            prob_FW += (
                lpSum(dict_w[j, n] * X[n, i] for n in range(d)) - Y
            )

        # Solve the problem
        prob_FW.solve(GUROBI_CMD(msg = True))

        # print warning when optimal solution is not reached
        if LpStatus[prob_FW.status] != "Optimal":
            print('Warning␣:␣The␣result␣is␣NOT␣optimal!')

        # Store the results in the corresponding dictionary
        for v in prob_FW.variables():
            if v.name.startswith('A'):
                A[v.name[3: -1].replace('_', "")] = v.varValue
            else:
                W[v.name[3: -1].replace('_', "")] = v.varValue

return W, A
```