UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

WEFE: THE WORD EMBEDDINGS FAIRNESS EVALUATION FRAMEWORK

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS,
MENCIÓN COMPUTACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN
COMPUTACIÓN

PABLO FERNANDO BADILLA TORREALBA

PROFESOR GUÍA:
FELIPE BRAVO MÁRQUEZ
PROFESOR GUÍA 2:
JORGE PÉREZ ROJAS

MIEMBROS DE LA COMISIÓN:
RICARDO BAEZA-YATES
AIDAN HOGAN
ELIANA SCHEIHING GARCIA

Este trabajo ha sido parcialmente financiado por el programa Iniciativa Científica Milenio
Código ICN17_002.

SANTIAGO DE CHILE
2020

Resumen

En el último tiempo, diversos estudios han mostrado que los modelos de *word embeddings* exhiben sesgos estereotipados de género, raza y religión, entre otros criterios. Varias métricas de equidad se han propuesto para cuantificar automáticamente estos sesgos. Aunque todas las métricas tienen un objetivo similar, la relación entre ellas no es clara. Dos problemas impiden una comparación entre sus resultados: la primera es que operan con parámetros de entrada distintos, y la segunda es que sus salidas son incompatibles entre sí. Esto implica que un modelo de *word embedding* que muestra buenos resultados con respecto a una métrica de equidad, no necesariamente mostrará los mismos resultados con una métrica diferente.

En esta tesis proponemos WEFE, *the Word Embeddings Fairness Evaluation framework*, un marco teórico para encapsular, evaluar y comparar métricas de equidad. Nuestro marco toma como entrada una lista de modelos de *word embeddings* pre-entrenados y un conjunto de pruebas de sesgo agrupadas en distintos criterios de equidad (género, raza, religión, etc...). Luego ranquea los modelos según estos criterios de sesgo y comprueba sus correlaciones entre los rankings.

Junto al desarrollo del marco, efectuamos un estudio de caso que mostró que rankings producidos por las métricas de equidad existentes tienden a correlacionarse cuando se mide el sesgo de género. Sin embargo, esta correlación es considerablemente menor para otros criterios como la raza o la religión. También comparamos los rankings de equidad generados por nuestro estudio de caso con rankings de evaluación de desempeño de los modelos de *word embeddings*. Los resultados mostraron que no hay una correlación clara entre la equidad y el desempeño de los modelos. Finalmente presentamos la implementación de nuestro marco teórico como librería de Python, la cual fue publicada como software de código abierto.

Abstract

Word embeddings are known to exhibit stereotypical biases towards gender, race, religion, among other criteria. Several *fairness metrics* have been proposed in order to automatically quantify these biases. Although all metrics have a similar objective, the relationship between them is by no means clear. Two issues that prevent a clean comparison is that they operate with different inputs, and that their outputs are incompatible with each other. This implies that one method exhibiting good results with respect to one fairness metric does, not necessarily exhibits the same results with respect to a different metric.

In this thesis we propose WEFE, *the Word Embeddings Fairness Evaluation framework*, to encapsulate, evaluate and compare fairness metrics. Our framework needs a list of pretrained *word embeddings* models and a set of bias tests grouped into different fairness criteria (gender, race, religion, etc...) and it is based on ranking the models according to these bias criteria and checking the correlations between the resulting rankings.

We conduct a case study showing that rankings produced by existing fairness methods tend to correlate when measuring gender bias. This correlation is considerably less for other biases like race or religion. We also compare the fairness rankings with an embedding benchmark showing that there is no clear correlation between fairness and good performance in downstream tasks. We finally present the implementation of our theoretical framework as a Python library that is publicly released as open source software.

Dedicado a mi familia y amigos,

como también a mi perro Don Mota,

que en paz descanse.

Agradecimientos

Primero que nada, le agradezco infinitamente y de todo corazón a mi madre Jesica y a mi padre Claudio por todo el cariño y apoyo incondicional que me han entregado, no solo durante este largo y valioso paso por la universidad, si no que durante toda mi vida. Sin ustedes no sería lo que me he convertido. Muchas, pero muchas gracias.

Agradezco de igual manera a mis hermanos Antonio y Claudia, que siempre han estado de alguna u otra forma presentes y que espero que sea siempre así. Mención especial para Don Mota, nuestro querido perro que estuvo a meses de verme titulado. ¡Descansa en paz Motoso!

Doy las gracias también a mis abuelos maternos Arnoldo e Inés, como también a mi Abuela paterna Olivia, por cuidarme y acogerme, por sus divertidas anécdotas y por las ardientes discusiones políticas. Muchas gracias a ellos y a toda mi familia por su cariño, experiencia y conocimiento.

Quiero agradecer en igual medida a mis profesores guía Felipe Bravo y Jorge Pérez por la gran oportunidad y la confianza que me entregaron para desarrollar este trabajo. Fueron excelentes tutores y me introdujeron espectacularmente en el mundo de la ciencia. ¡Muchas gracias! También, a todos los profesores y profesoras que confiaron en mi para ser profesor auxiliar de sus ramos. Fueron excelentes oportunidades para complementar mi aprendizaje.

Agradezco con todo mi cariño también a mis queridos amigos. Qué hubiese sido mi vida sin compartir con ustedes el colegio y la universidad; sin los innumerables viernes en la tarde en 850 (y ocasionalmente los Bella) y sábados por la noche deambulando por CDLV; sin los partidos ni los trekkings, sin los asados, sin las pizzas después de las clases, sin las protestas, sin los viajes a la playa ni las salidas al cerrito; sin permitirme experimentar en la cocina, sin pudrirnos juntos en los juegos por la noche (sobre todo en cuarentena) y en general, sin la gran cantidad de espléndidos momentos y compañía (tanto en las buenas como en las malas) que ustedes me han entregado a lo largo de todos estos años.

Por último, hay un sinfín de personas que me han permitido llegar hasta aquí: Angélica, Marcia, Jaime, profesores y amigos del colegio Manquecura CDLV, personal del DCC y muchos mas que no he nombrado pero que de todas formas los considero: muchas gracias por su apoyo, cariño y confianza.

¡Muchas gracias a todas y todos ustedes por acompañarme y permitirme llegar hasta aquí!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Data-driven models consist of a wide range of techniques that focus on inferring patterns from data. Such inference is achieved through the training process, which consists of analyzing immense amounts of data to learn their intrinsic relationships. Nowadays, such models are widely used both in industry and academia. Clear examples of the application of these models are automatic translators, chatbots and personalized web content search.

Natural language processing (NLP) is a good example of an area of study that has followed the data driven approach in recent years. Its main objective is to design methods and algorithms capable of understanding or producing unstructured natural language. Commonly, these methods are focused on solving narrow, well-defined tasks. An example of this is the Question Answering task: it takes a question as input and gives an answer to it as output, all in natural language.

One of the fundamental problems in NLP is how to represent words as mathematical structures that can be operated by a computer. A popular approach is to create the representations based on the use of the distributional hypothesis. This hypothesis states that *words occurring in the same contexts tend to have similar meanings* [22], or in simpler words, *a word is characterized by the company it keeps* [17]. This strongly suggests that since contexts can define words, we can create representations of words based on their context.

Word embeddings are a set of models that use the distributional hypothesis approach to represent words. Embedding models captures the meaning of words in dense, low-dimensional continuous vectors derived from the training of neural networks. The training process of the networks is executed by using large collections of documents, commonly called "corpora". The enormous amount of data used in the training process allows word embedding to effectively capture the semantics of words. The outstanding performance in capturing semantics has led to word embeddings becoming key components in many NLP systems.[19].

One of the advantages of using word embeddings is the ability to operate them mathematically. The calculation of the distances between the representations allows the study of the association between different words, as well as the finding of similar words. Another common operation is the calculation of analogies: given the representations of words $\vec{a}, \vec{b}, \vec{x}$, find the

representation $\vec{y}$ such that *"$\vec{a}$ is to $\vec{b}$ as $\vec{x}$ is to $\vec{y}$"*. The most iconic example of is shown by Mikolov et al. [35] in which for the equation $\overrightarrow{Germany} - \overrightarrow{Berlin} \approx \overrightarrow{France} - \overrightarrow{x}$, the vector $x = \overrightarrow{Paris}$ is the most suitable one as a solution.

A widely reported shortcoming of word embeddings is that they are prone to inherit stereotypical social biases (regarding gender, ethnicity, religion, as well as other dimensions) exhibited in the corpora on which they are trained [7, 18, 48]. These biases usually show some attributes (e.g., professions, attitudes, traits) being more strongly associated with one particular social group than another. An illustrative example is the vector analogy between the representations of words "man" and "woman" being similar to the relationship between words "computer programmer" and "homemaker" [5].

Although the previously mentioned works have detected different biases in word embeddings, their measurement methods are poorly formalized. Most exploratory work on bias in embeddings, to the best of our knowledge, has only studied models trained in the English language and none of them have been used to compare and rank embeddings according to their biases.

In this thesis we propose a new approach to solve these pitfalls by first formalizing the bias measurement processes in a theoretical framework and then, by implementing and using this framework to conduct a case study comparing various embedding models according to their bias.

Even though our case study does not cover the measurement of bias in other language embedding models, we hope that the implementation and further publication of our framework as a Python library enables other teams to carry out these studies as seamlessly as possible.

1.1 Prior Definitions

Before continuing, it is necessary to summarize and more formally define some of the concepts introduced in the previous section, as well as to introduce additional key concepts used in this thesis.

Word Embeddings Word Embeddings are a set of techniques for representing natural language words in dense real number vectors. These representations aim to capture the context of each word.

Social Group We define Social group as a group of people that are grouped together according to a certain *criterion*. This criterion can be any character or trait that distinguishes groups of people, such as gender, social class, religion, ethnicity, among others. For instance, *Asians, Europeans, Hispanics* are groups of people defined by the criterion of *ethnicity*.

Bias In our context, we define bias as the act of treating individuals belonging to different social groups unequally, usually in a way that is considered unfair.

Target Set A target word set corresponds to a set of words intended to denote a particulars social group, which is defined by a certain criterion. For example, if the criterion is *ethnicity*, we can define the groups of people according to their surnames: a set of target words representing the *Hispanic* social group could contain words like *"gonzález", "rodríguez", "pérez"* and a set of target words representing the *Asian* social group could contain words like *"wong", "wu" and "chen"*.

Attribute Set An attribute word set is a set of words representing some attitude, characteristic, trait, occupational field, etc. that can be associated with individuals from any social group. For example, we can define the set of *intelligence* attribute words with the words *"smart", "ingenious", "clever"*.

Query A query is a collection of target and attribute sets. It is intended to contain the sets of words whose relationships will be studied by a fairness metric. For example, a query that studies the relationship between *Hispanics* and *Asians* with respect to *intelligence* would be composed of the word sets described in the previous point.

Fairness Metric A fairness metric is a function that quantifies the degree of association between target and attribute words in a word embedding model. Further details and examples of metrics will be given later in the corresponding chapter.

1.2 Research Problem

The problem of how to quantify the mentioned biases is currently an active area of research [7, 15, 18, 20, 32, 48, 49, 52], and several different *fairness metrics* have been proposed in the literature in the past few years. These metrics compare the representations of several word sets through vector operations and calculate a numerical score associated with the bias they were able to detect. Commonly, these comparisons are made between sets of words that are associated with social groups (target words) and words of attributes that can be associated (unfairly or not) with people (attribute words).

Although all metrics have a similar objective, the relationship between them is by no means clear. Two issues that prevent a clean comparison is that they operate with different inputs (pairs of words, sets of words, multiple sets of words, and so on), and that their outputs are incompatible with each other (reals, positive numbers, $[0, 1]$ range, etc.).

Moreover, fairness metrics are usually proposed coupled with a specific debias method [5, 32]. This implies that one debias method exhibiting good results with respect to one fairness metric does not necessarily exhibit the same results with respect to a different metric.

Currently there are many pre-trained word embeddings circulating on the web. Many of the NLP systems use these models as key components. Most of these models have never been subjected to fairness studies: research has tended to focus on studying very specific biases of a few popular set of embedding models (word2vec [35], fasttext [4], glove [37], conceptnet [45]). Even worse, to the best of our knowledge there are no simple methods to compare the bias exhibited by different models of embeddings. In order to prevent these

biases from being inherited in such systems, it would be extremely useful to provide some mechanism to compare and rank these models according to their unfairness.

Gender bias is arguably the criterion that has received the most attention by the research community. Most of the metrics and tests have been specially designed for studying this type of bias. This also provides evidence that more work is needed to propose new experiments able to consistently test and rank embeddings for criteria beyond gender, such as ethnicity, religion, social class, political position, among others types of biases.

1.3 Research Hypothesis

Based on the observable differences in the approaches to measuring fairness mentioned above, we propose that it is possible to formulate a framework to standardize fairness measures and through its use, propose a methodology to rank models of embeddings according to different fairness metrics and bias criteria.

1.4 Results

As mentioned in the introduction, the results achieved by the development of this thesis can be summarized into the three following points:

1. The creation of a theoretical framework that standardizes the main building blocks and process of bias measuring in word embeddings models.
2. The carrying out of a case study where we were able to measure and rank the embeddings according to the gender, ethnic and religious bias that these models presented.
3. The develop of an open-source library that implements the framework and enables the execution of the case study.

The first result involved the formalization of what a fairness metric is, the parameters necessary for its operation (a query and an embedding model), its expected outcomes and the adaptation of the metrics presented in the literature to our proposal. Since it was also necessary to compare different models of embeddings on different bias tests, this point also included the development of an aggregation and ranking method based on the framework.

The second result was the execution of a case study that evaluated and ranked gender, ethnic and religious biases exposed by seven publicly available pre-trained word embedding models. The evaluation of these biases was carried out using three fairness metrics proposed in the literature: WEAT [7], RND [18], and RNSB [48]. This point also considers checking whether the rankings obtained in assessing word embeddings bias match the ratings derived from the common tasks for assessing the performance of embedding models (by using the Word Embedding Benchmark [24] library). The scores and rankings obtained allowed to elucidate which are the least biased models in the three evaluated areas, the effects of a debias process on a model with respect to the original and the relationship between model biases and model performance.

The last result included the implementation of the framework and aggregation mechanisms as a Python library. We considered that it would be very relevant to publish the library as an open source toolkit in order to allow anyone to replicate the case study and conduct their own bias experiments. This is why the library was designed to fulfill the standards required to be used and extended by the community (conventions, testing, documentation, code-style, among others).

1.5 Research Outcome

As a result of this work, we were able to publish a paper describing our proposed framework and our experimental results derived from our case study at the International Joint Conference on Artificial Intelligence (IJCAI)[2]. Derived from our paper, we published a blog in KDNuggets [1], a page dedicated to the publication of news in Machine Learning [1]. We are also preparing a paper to publish our library in the open-source software track of the Journal of Machine Learning Research (JMLR). Furthermore, the library is available from its official website[2] and its implementation code in the github repository[3].

1.6 Outline

The rest of the thesis is organized as follows:

In Chapter 2 we give a brief introduction to the scientific disciplines involved in our work (Section 2.1) and the necessary background to understand word representations and present their most preferred models (Section 2.2). Next, we provide an overview of the research work on machine learning bias (Section 2.3). and then, we continue with a review of recent work on bias evaluation in embedding models (Section 2.4). The chapter closes with a discussion of the problems presented by the previous works on bias measuring and a proposal to solve them. (Section 2.5).

In Chapter 3 we describe our framework and the execution of the case study in detail. We start by showing the WEFE main building blocks (Section 3.1); then we explain how we use WEFE to compare and rank different models of embeddings according to their bias (Section 3.2). Next, we show the parameters used in our case study (Section 3.3). We close this chapter by providing an extensive analysis of the results produced by the case study (Section 3.3.4).

In Chapter 4 we describe the implementation of the library. We begin by giving a brief motivation about the importance of implementing our framework as an open-source library (Section 4.1). Then we continue by detailing the main classes and functions of the library (Section 4.2) and showing the different bias assessment processes that can be executed by our software (Section 4.3).

[1] `https://www.kdnuggets.com/2020/08/word-embedding-fairness-evaluation.html`
[2] `https://wefe.readthedocs.io`
[3] `https://github.com/dccuchile/wefe/`

The last chapter of this work (Chapter 5) enumerates the conclusions derived from our work (Section 5.1) and provides a comprehensive list of topics for future work (Section 5.2).

Chapter 2

Background and Related Work

This chapter is composed of two main topics. First, we present the background which briefly introduces the reader to the area of knowledge in which the thesis is developed and the methods associated with it. Then, we go through the related work specific to the topic of biases in word embeddings. Its structure is as follows: background in the area of natural language processing and machine learning, word representations, word embeddings, fairness in machine learning and fairness in word embeddings.

Each of these points is detailed in the following outline: We begin by briefly reviewing the fields of Natural Language Processing (Section 2.1.1) and Machine Learning (Section 2.1.2.

Next, we introduce the word representations (Section 2.2) along with the distributional hypothesis (Section 2.2.2). Later, we make a brief overview of the models prior to word embeddings: one-hot representations (Section 2.2.1) and word-context matrices (Section 2.2.3).

Then, we review word embeddings: their formulation (Section 2.2.4, the tasks they can solve (Section 2.2.4), how to obtain these models (Section 2.2.4), how to evaluate this models (Section 2.2.4) and their limitations (Section 2.2.4). This is followed by a presentation of the related work on fairness (Section 2.3). In this part we will discuss biases in the data (Section 2.3.1) as well as in the algorithms (Section 2.3.2).

The following section presents the related work of fairness in word embeddings (Section 2.4). We then review both the assessment (Section 2.4.1) and bias mitigation in these models (Section 2.4.2). Finally, in consideration of the aforementioned issues, we provide a small discussion arguing for the need to develop a framework to standardize previous work (Section 2.5).

2.1 Scientific Disciplines

This section intends to briefly contextualize the scientific disciplines in which our work is developed: natural language processing and machine learning.

2.1.1 Natural Language Processing

Natural Language Processing (NLP) is an area of study of computer science, artificial intelligence and linguistics. There are multiple definitions that vary slightly by author. Golberg, in his book Neural Network Methods for Natural Language Processing [19] proposes the following definition: *NLP is a collective term referring to automatic computational processing of human languages includes both algorithms that take human-produced text as input, and algorithms that produce natural looking text as outputs.* Eisenstein [12] in his notes sets out his definition of NLP as: *Natural language processing is the set of methods for making human language accessible to computers.* On other hand, Johnson [25] claims that *NLP develops methods for solving practical problems involving language.*

In practical terms, NLP focuses its efforts on solving well-defined and limited tasks such as translation, topic-classification, part-of-speech tagging, among several others. Each of these tasks is a subfield of this area and has particular objectives, methodologies and evaluation methods. Although the resolution of these tasks requires the understanding of language, NLP does not include the study of it among its objectives. On the contrary, this is accomplished by another area of study: Computational Linguistics. Although both areas share common problems and their intersections are broad, their goals are different.

The tasks that NLP aims to solve are commonly grouped into three main groups:

- Text Classification: Given a document, assign it a class. In this group we find tasks such as topic classification (designating a defined topic to a document), spam filtering and sentiment analysis (defining a feeling or mood to a document), among others.
- Sequence Labeling: Given a sentence, assign a class to each of its words. In these we find part of speech tagging (POS) and named entity recognition (NER) in the words that compose the documents, among others.
- Sequence to Sequence: Given some sentence or document, generate another from it. In this category we find the most challenging tasks, such as translation, question answering, summarization and chatbots, among others.

The resolution of these tasks presents complex challenges. Its primary challenge is the very nature of natural language: generally it is very ambiguous, it is constantly changing and evolving, and presents great difficulty in formalizing the rules that govern it. Goldberg proposes three properties that make the computational processing of natural language challenging [19]:

- The first is that language is *discrete*. The most basic elements of written language are characters. These can be combined to represent objects, concepts, events, actions and ideas. Each way in which the characters can be combined is a discrete object. This property states that we cannot always infer the relationship between two words from the characters that compose them. Take the example of the words dog and cat: we

know that they represent words to designate pets, however, their composition does not indicate any apparent relationship between them or with pets.

- The second is that language is *compositional*. The combination of multiple words can produce phrases or sentences, each with a specific meaning and goal. This property states that the meaning of a sentence goes beyond the individual words that compose them. This implies that in order to understand sentences we need to analyze not only the words and letters that compose them, but the sentence as a whole.
- The third is that language is *sparse*. This indicates that the way we can combine characters and words is in practice infinite, which implies, that we will also have infinite meanings. This fact prevents us from building a definite set of sentences that generalize to all the others. And even if we manage to establish all the valid sentences up to this point, we will probably in the future observe cases that we did not consider before. These reasons make it complex to learn solutions to the example tasks.

The classic approach used to solve these tasks was based on designing very complex systems of rules defined manually by linguists and programmers. However, due to the difficulty of their creation and maintenance, as well as their performance limitations, their use began to decline. Recently, the main focus of solving these tasks changed radically to the intensive use of methods based on statistics and supervised machine learning, especially the subarea of deep learning. These techniques have better performance than their predecessors and their creation is simpler.

2.1.2 Machine Learning

Machine Learning is the field of computer science and artificial intelligence focused on the study of algorithms that are capable of learning from previous observations (sample data) to make predictions [19]. The main objective of these algorithms is the building of mathematical models able to make decisions without being explicitly programmed. The models are generated through the training process. In this process, the algorithm tries to infer as many patterns and regularities as possible from the sample data, trying to internalize the knowledge in the model in the best possible way. The resulting model must be able to generalize the learned knowledge in order to process samples that it has never observed.

The learning technique varies according to the model to be created: it is supervised when the algorithm needs labelled data to train and unsupervised when the algorithm is able to infer knowledge from the non-labelled data.

Although these methods generally have better results than rule-based systems, they also have difficulties: models expressing complex systems in reality require a massive amount of data to function properly, the costs and time of tagging large amounts of data are very high, and the models are often task-specific.

There are many different algorithms for creating such models. Each of these has different approaches and complexities. Within these are the Neural Networks (NN). Neural Networks is a family of learning techniques that was historically inspired by the way computation works in the brain, and which can be characterized as learning of parameterized differentiable mathematical functions [19]. The basic unit of the networks are the neurons, which are

stacked in layers. The production of networks with several layers is known as Deep Learning (DL). Currently, the great learning capabilities offered by deep learning models and their associated high performance make them the preferred choice when solving the majority of NLP tasks. We will not cover in detail their definition and function in this thesis. However, excellent references can be found in books by Goodfellow [21] or Golderg [19].

2.2 Word Representations

A fundamental issue in NLP is how we represent words as mathematical objects that we are able to manipulate. In simpler words, we need to convert the words that compose the documents we work with into vectors so that ML algorithms are able to process them. Representations will vary depending on the method used to create them. In the following sections we present different models of vector representations of documents and words. We start by showing the classic bag of words model and its semantic limitations. Then, we show how distributional representations solve some of these problems.

2.2.1 One Hot Representations

One hot representation is one of the most basic word and document (set of words) representations. Its original objective is to model documents through vectors that contain the count of their words. Its formulation is founded on a very simple representation of words: a one-hot vectors.

In this model, the words are represented in vectors of the size of the vocabulary used. Each dimension of the vector will be related to a specific word in the vocabulary. The appearance of a word in the document is represented as a a one-hot vector, in other words, a vector whose components are only zeros, except in the dimension to which the word corresponds, where its value is one.

Representing documents in this model is relatively simple: first calculate the vector representation of each word and then average them into a single vector. Formally, let a document be a set of words $\{w_1, w_2, ..., w_n\} \in D$ Suppose we have a dataset of documents $\{\mathrm{d}_1, \mathrm{d}_2, ..., \mathrm{d}_m\} \in \mathcal{D}$ with $|V|$ different words or tokens (which we will call the vocabulary). The model consist of vectors $\boldsymbol{x} \in \mathbb{R}^{|V|}$, where each dimension of the vector is one-hot encoded word vector. We can see how the word *cat* would be represented within the model in the one-hot vectors shown in formula 2.1:

$$\boldsymbol{x} = \begin{bmatrix} ant \\ \vdots \\ casual \\ cat \\ catacombs \\ \vdots \\ zoom \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.1}$$

Then, to represent the documents, simply average the one-hot vectors for each of the words in the document. This is shown in the formula 2.2:

$$\boldsymbol{x} = \frac{1}{|D|} \sum_{x_{\mathrm{i}} \in D}^{|D|} \boldsymbol{x}_{\mathrm{i}} \tag{2.2}$$

One of the major advantages is that documents with different lengths and words reside in the same vector space. However, this type of representation poses two major difficulties. First, they ignore the meaning of words. Second, for very big vocabularies (most real cases) the created representations are of high dimensionality, which prevents the correct operation of the ML algorithms. Possible solutions are detailed in the next part.

2.2.2 Distributional Hypothesis and Distributional Representations

One hot representation is presented as a simple but powerful option to represent the documents in a vector space where we can easily work with them. However, it has a major shortcoming: the representations do not contain the meaning of the words. This makes it impossible to calculate any type of relationship between words.

For example, the one-hot representation of the word *cat* is as different from that of a *dog* as that of a *pizza*: they all have different dimensions associated with them. This implies that although we know that the representations of dog and cat should not be totally different (since they are both animals), this model ignores any relationship between them. We see that this fact prevents us from relating words by their meaning, which limits their possible uses.

An option that emerged for this problem is to use the Distributional Hypothesis. The *Distributional Hypothesis* states that words that occur in the same context tend to have similar meanings [22] or equivalently, a word is characterized by the company it keeps.

Using the previous idea, we can create representations that capture the meaning of words by capturing their context in vectors. These models are known as Distributional Representations. The first method to implement Distributed Representations is the creation of word vectors through the use of word-context matrices.

2.2.3 Word Context Matrices

Word-context matrices are models that attempt to capture the distributional properties of words by using the co-occurrences that occur between them. In practice, each row i represents a word and each column j represents a context word. Thus, each value in the (i, j) matrix quantifies the strength of association between a word and its context.

There are several ways to calculate correlation matrices. Below we will present two.

	I	like	cats	pizza	enjoy	sleeping
I	0	2	0	0	0	0
like	2	0	1	1	0	0
cats	0	1	0	0	0	0
pizza	0	1	0	0	0	0
enjoy	1	0	0	0	0	1
sleeping	0	0	0	0	1	0

Table 2.1: An example of a word-context matrix.

Co-ocurrence counts

The first option to calculate the force of association consists of counting of the co-occurrences between a target word w_{i} and the words of its context c_j over all the documents of the corpus. The context is defined as windows of words surrounding w_{i} and its length k is a parameter of the model. If the vocabulary of the context is equal to the vocabulary of the target words, then the size of the matrix will be $|V| \times |V|$. In other words, each word is represented as a sparse vector in high dimensional space, encoding the weighted bag of contexts in which it occurs [19].

For example, for the following documents, the word-context matrix is represented in the table 2.1:

- I like cats.
- I like pizza.
- I enjoy sleeping.

In table 2.1 one can observe the similarity between vectors that appear in the same contexts: the ones of cat and pizza. Although they do not represent the same thing, it is understood that both characterize an entity with the quality of being liked.

The count-based word-context matrix presents an improvement over one-hot by including semantics in word representations . However, word count is not a very good method of achieving this. Context words that have a very high count frequency will have very unbalanced vectors. For example, word-context pairs *a cat* and *the cat* will receive a greater importance in the model than *cute cat* and *small cat*, although paradoxically, the latter contexts of cats are more descriptive than the former ones. [19] The following method attempts to solve this problem.

Positive Point-Wise Mutual Information

The correlations can be calculated by using another metric that also captures the association between a target word and its context: the pointwise mutual information (PMI).

$$\text{PMI}(x, y) = \log_2 \left(\frac{P(x, y)}{P(x)P(y)} \right) \tag{2.3}$$

Formula 2.3 calculates the probability that the context target word pair will occur together with respect to the probability that both will appear separately in the training data set. These probabilities are empirically calculated from the number of target and context words that appear in the dataset. The formula 2.4 shows how to empirically obtain these distributional vectors:

$$\text{PMI}(w, c) = \log_2 \left(\frac{P(w, c)}{P(w)P(c)} \right) = \log_2 \left(\frac{\text{count}(w, c) \times |\mathcal{D}|}{\text{count}(w) \times \text{count}(c)} \right) \tag{2.4}$$

The above formulation presents two problems. First, it may be that there are word-context pairs in the model that occur with much less frequency than chance. In these cases, the PMI values will be negative. Second, word-context pairs that do not exist in the dataset will have a value of $-\infty$.

To solve these issues of values that represent pairs with very low or no frequencies, we use only the maximum between the positive value of PMI or zero, which is called Positive Point-Wise Mutual Information (PPMI). Its definition is presented in the formula 2.5:

$$\text{PPMI}(w, c) = \max(\text{PMI}(w, c), 0) \tag{2.5}$$

Problems of word-context matrices

Although word-context matrices create better representations than one-hot vectors by including semantics in the model, these continue to have a large dimensionality (of the size of the vocabulary).

This presents two challenges for this type of representation: the first is that storing and working with these matrices is memory intensive. The second is that self-learning classification models do not work well with such high dimensional inputs.

As we said before, this implies that various ML methods will not be able to function properly. This problem can simply be solved by using some type of dimensionality reduction technique such as Singular Value Decomposition (SVD) on the generated representations. This will not be covered in this thesis. However, an excellent reference to this method can be found in Goldberg's book [19].

With the rise of neural networks, the community has preferred to take another direction: using distributed representations. These are introduced below.

2.2.4 Distributed Representations or Word Embeddings

Distributed representations or word embeddings consist of a set of models that capture the semantics of the words within dense continuous vectors of small dimensionality. Like the previous models, these vectors are based on the distributional hypothesis. In other words, they represent words based on the context in which they occur. This implies that words that appear in similar contexts tend to be represented by similar vectors.

Word embeddings are trained from large corpora of documents using neural networks. This process distributes the semantic of the words across the dimensions of the vectors (which gives it the name of distributed representations). Because of the learning method, the dimensions of its vectors are not interpretable. The previous fact represents a disadvantage with respect to the models previously seen. However, these models are usually more powerful than the count-based ones. Thanks to their improved performance, they have become central components in most current NLP systems.

Word Embedding Tasks

The embeddings can perform a multitude of practical tasks. We describe some of them in the following lines.

- **Word Similarity**. This similarity can be quantified through some measure of vector similarity. The higher the value of the similarity between two vectors, the more similar their meanings will be. This leads us to the first task that can be solved by embeddings: to calculate the similarity between words.

  As an example, let us suppose we have a set of animals (*dog*, *lizard*, *mountain lion*, ...) and *cat* representations, and a similarity function, such as cosine similarity. We can find the most similar animal to *cat* (according to the model) by using the similarity function. The procedure is the following: First, the similarity of the cat representation to the representations of all other animals is calculated. Then, we order the words in a descending order. The words at the beginning of the ordered list are the most similar words according to the model. The same procedure can be used to calculate the similarity between any set of word representations.

- **Finding similar words**. We can use the same approach as above to find the words most similar to a given word. The procedure in general is quite simple: we calculate the similarity of the chosen word representation with respect to all others and then we order them according to their obtained values. The words with the highest similarity scores will be the most similar words, according to the model.

- **Word Analogies**. Let us define the word analogy task as an operation between words *a,b,x* and *y* in the form *"a is to b as x is to y"* [14]. The word analogy task consists of solving analogies between different words through arithmetic operations of their embedding representations. The example par excellence is the one observed by Mikolov et al [35], where they shows that if they operates the embeddings of $\vec{king} - \vec{man} + \vec{woman}$ the most similar representation to this resulting vector is $\vec{queen}$.

- **Clustering**. This task consists simply in using some clustering algorithm (like K-means) to find cluster of words according to their representations. These clusters are commonly expected to group words according to some coherent criterion or category.

- **Other Tasks**. While the above tasks are the ones that have been given the most attention, there are many more that can be directly solved using embeddings. These include odd one out, short document similarity, similarity to a group of words, among others. A complete reference can be found in the *Using Word Embeddings* chapter of Goldberg's book [19] .

Obtaining Word Embedding Models

There are two main approaches for obtaining word embedding models:

1. Training an embedding layer in a task-specific neural network from labeled examples. If there is enough training data, then the network training procedure will create good embeddings. The model created with this method captures the information of the task it solves. For example, if you train in solving sentiment analysis, then its word embeddings will contain information about the sentiment of the words.
2. To use a pre-trained word embedding model. The idea behind this is to train a model using an auxiliary task that does not require tagged data. This task must be able to capture the meaning of the words in a proper way. Note that different iterations of the learning method using different parameters (corpus, architecture of the network, training method, dimensions, etc...) will create different representations.

There are many pre-trained word embedding models circulating in the network. Among the most popular we can find Word2vec [35, 34], Fasttext [4], Glove [37]. On the other hand, there are some models worth mentioning such as Conceptnet [45] and Lexvec [40, 41].

Some of these models are described in detail in the following subsections.

Word2vec Word2vec is a software package that implements two word embedding architectures, skip-gram (SG) and continuous bag of words (CBOW), and two optimization techniques: negative sampling (NS) and hierarchical softmax [35, 34]. In this work, we only detail the skip-gram model with negative sampling. CBOW and hierarchical softmax can be found directly in their original references [35, 34].

Note that the term Word2vec is commonly used indistinctly to refer to both the software and the pre-trained models made available by its authors. The version of pre-trained Word2vec embeddings is based on skip-gram with negative sampling (SG-NS).

Skip-Gram The first model, skip-gram, consists of training a shallow neural network, (a network with a single hidden layer with no activation function) to solve the following task: given a central word that shifts along the training corpus, predict its context words (i.e., surrounding words in a context window). In the training process, the network learns co-occurrence statistics about central and context words. Once the training is completed, the resulting weights are used as embeddings.

Figure 2.1[1] provides a high-level description of the skip-gram architecture. Next, we detail both the architecture and the training process of the skip-gram model.

The central word and the surrounding words (k sized window) are the respective inputs and outputs of the network. Each input is a word that belongs to a vocabulary of size 10,000. Both are represented as one-hot encoded vectors. Therefore, the size of the input and output layers are the size of the corpus vocabulary $|V|$ (in this case, $|V| = 10,000$). The hidden layer has 300 neurons. Each neuron has 10,000 parameters (one for each possible input word). Like

[1]Image taken from http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

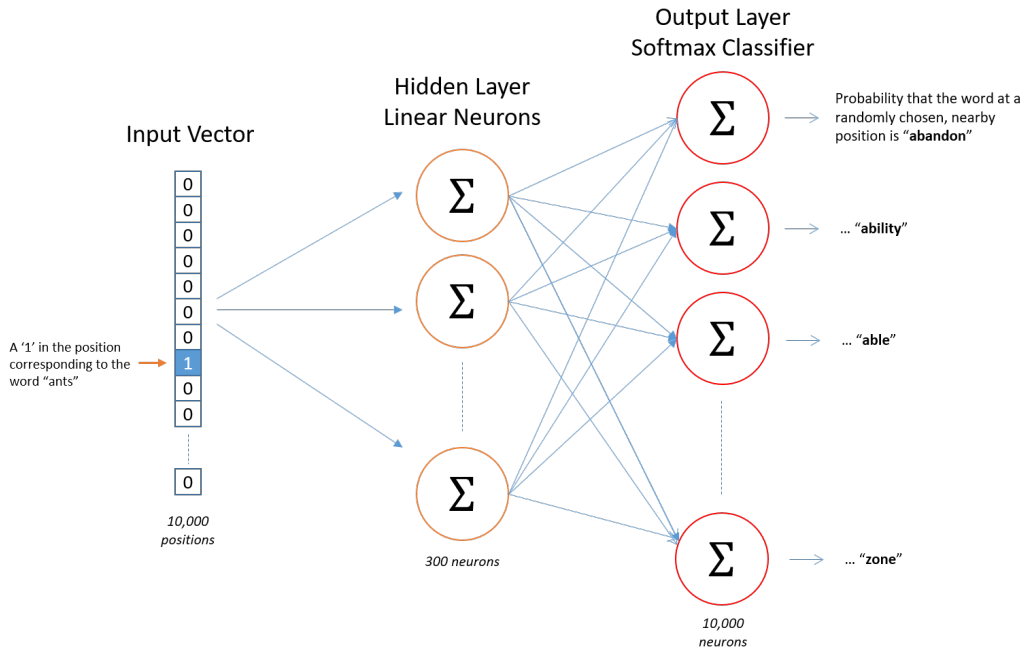

Figure 2.1: Architecture of a skip-gram neural network.

we said before, the output layer has one neuron per word in the vocabulary. Therefore, each neuron has be 300 weights. Each output neuron uses a softmax activation function.

During the training process, the entire training corpus is gone through at least once. At each training step, a central word is selected from the corpus and its context window. Then, the network predicts from the central word and the information contained in the hidden layer, the context words. This result is compared with the words in the window and the weights are adjusted according to backpropagation. This training process is repeated until the training loss is minimized as much as possible.

Once the training is over, the matrix containing the hidden layer weights is used as word embeddings. The matrix with the output weights will also contain contextual information of the words. However, these are not used in this model.

Formally, let us suppose we have a document corpus formed by a sequence of words $w_1, w_2, w_3, ..., w_t$ and a window of size k. We denote the word target/center with the letter w and those in context with the letter c. The words in the context $c_{1:k}$ of the letter w_t then correspond to $(w_{t-k/2}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+k/2})$ (if we assume that k is even). The objective of the Skip-gram model is to maximize average log probability of the context words given the target words.

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{c\in c_{1:k}}\log P(c|w_t)$$

Let C be the set of all possible context words, commonly the same as the vocabulary. We model the conditional probability of a context word c appearing next to the central word using a softmax as:

$$P(c|w) = \frac{\exp^{\vec{c}\cdot\vec{w}}}{\sum_{c' \in C} \exp^{\vec{c'}\cdot\vec{w}}}$$

Note that both $\vec{c}$ and $\vec{w}$ are model parameters. However, only $\vec{w}$ is occupied as the representation of the word.

Let D be the set of correct word-context pairs (the pairs that exist in the training corpus) Then, the optimization goal is to maximize the conditional log-likelihood of the context c:

$$\arg\max_{\vec{c},\vec{w}} \sum_{(w,c)\in D} \log P(c|w) = \sum_{(w,c)\in D} (\log \mathrm{e}^{\vec{c}\cdot\vec{w}} - \sum_{c' \in C} \mathrm{e}^{\vec{c'}\cdot\vec{w}})$$

Although, it is assumed that maximizing this function will result in good embeddings, this formulation presents a serious computational problem: the calculation of $P(c|w)$ is computationally very expensive because summation $\sum_{c' \in C} \mathrm{e}^{\vec{c'}\cdot\vec{w}}$ runs over all context words, which is usually a large number. There are two possible variations of the skip-gram model that address this problem: 1) hierarchical softmax and 2) negative sampling. Negative sampling will be described as follows, but hierarchical softmax will not be explained in this thesis.

Skip-Gram with Negative Sampling Negative Sampling is a variation of the previous problem where a different function is optimized. This new objective function intends to solve the above problem by being more efficient. It consists of maximizing the probability that a word-context pair (w, c) comes from the set of word-context pairs existing in corpus D using a sigmoid function. In simpler words, this network is trained to be able to distinguish the "good" word-context pair from the "bad" ones. Its formulation is as follows:

$$P(D = 1|w, c_{\mathrm{i}}) = \frac{1}{1 + \exp^{-\vec{w}\cdot\vec{c_{\mathrm{i}}}}}$$

If we assume that all context words are independent form each other, we can treat each target word context pair $(w, c_1), \ldots, (w, c_k)$ as an independent training sample. Therefore, the function to be optimized for all words in the context of the target word is:

$$P(D = 1|w, c_{1:k}) = \prod_{\mathrm{i}=1}^{k} P(D = 1|w, c_{\mathrm{i}}) = \prod_{\mathrm{i}=1}^{k} \frac{1}{1 + \exp^{-w\cdot c_{\mathrm{i}}}}$$

This leads to the following target function:

$$\arg\max_{\vec{c},\vec{w}} \log P(D=1|w,c_{1:k}) = \sum_{\mathrm{i}=1}^{k} \log \frac{1}{1+\mathrm{e}^{-\vec{w}\cdot\vec{c_{\mathrm{i}}}}}$$

In this state, this optimization function will result in an undesired trivial solution: If we set P(D=1 |w,c) = 1 for all pairs (w,c), then, training will tend to set $\vec{w} = \vec{c}$. The way the authors of Word2vec solve this, is through the use of negative samples: pairs that do not appear in the corpus and whose probability should be very low in the optimization function. The negative samples $\tilde{D}$ are generated from the following process: for each good pair $(w, c) \in D$, sample m words $w_{1:m}$ and add each of (w_{i}, c) as a negative example to $\tilde{D}$.

Then, the optimization function is transformed to:

$$\arg\max_{\vec{c},\vec{w}} \sum_{(w,c)\in D} \log P(D=1|w,c_{1:k} + \sum_{w,c\in\tilde{D}} \log P(D=0|w,c_{1:k})$$

The negative words are sampled from a smoothed version of the corpus frequencies:

$$\frac{\#(w)^{0.75}}{\sum_{w'} \#(w')^{0.75}}$$

This gives more relative weight to less frequent words.

Glove The Glove (global vectors) algorithm, proposed by Pennington et al. [37] is based on the construction of a word-context matrix to train the embeddings. Its goal is that the model be able to predict the co-occurrence ratios between words. Through this mechanism (unlike Word2vec that only takes local contexts into account) Glove tries to take advantage of the global word counts within its embeddings.

Formally, let us suppose that we build a word-context matrix from a certain corpus. We can access its values through the function $\#(w, c)$. In addition, let the word vectors $\vec{w}$ and context vectors $\vec{c}$ and b_w and b_c be their associated biases. The Glove algorithm attempts to satisfy the following equality constraint:

$$w \cdot c + b_w + b_c = \log \#(w,c) \quad \forall (w,c) \in D$$

The optimization objective is a weighted least-squares loss. This loss assigns more weight to the correct reconstructions of frequent terms co-occurrences. It also prevents very common co-occurrences (such as "it is") from dominating the loss values [29].

A new feature of Glove is that it uses the addition of the word and context embeddings generated by the training of network.

Lexvex Lexical vectors (LexVec) [41, 40] is a method for creating embeddings that combines PPMI and singular value decomposition methods with skip-gram and negative sampling. Its results in several tests outperform those obtained by the two models mentioned above [41]. The full detail of its implementation are given in the original publications of the method works [41, 40].

Fasttext Fasttext [4] extends the skip-gram model to take into account the internal structure of words. The main idea is that the model not only learn representations for the whole words, but also the n-grams that compose it.

Specifically, each word is represented as a bag of n-grams. The model learns embeddings for these n-grams, and the final representation of a word corresponds to the sum of the embeddings of the n-grams. For example, taking n=3, the representation of the word 'Coffee' would be composed of the n-grams: *"<co", "cof", "off", "ffe", "fee", "ee>"* (where '<' and '>' are the symbols for the beginning and end of the word respectively).

The embedding models seen in the previous points assign every concept to a different vector. This makes the vectors completely unrelated to each other, causing them to be unable to share their meanings. In contrast, in this model words that have n-grams in common contain the same vectors in their representations. This allows words that share some sub-words to share meanings. The above-mentioned fact is quite useful for morphologically rich languages where words are generated from common stems. A clear example of this would be comparing the embeddings of the words "amazing" and "amazingly" in Word2vec and Fasttext. While in Word2vec, both are unrelated vectors, in Fasttext both words are represented by almost the same sum of n-grams.

Fasttext also has the advantage of being able to deal with out-of-vocabulary words as it is able to generate a representation for them by summing its composing n-grams.

Conceptnet Numberbatch Conceptnet numberbatch [45] goes beyond distributional semantics by incorporating knowledge graph relationships into the learning process. Its knowledge is collected from resources like WordNet, Wiktionary and DBpedia, as well as commonsense knowledge from crowd-sourcing and games with a purpose.

The Conceptnet training process is performed in several stages. First, it creates a term-term matrix, where each value represents the sum of the weights of the path from one word to another in the knowledge graph. Then, a PPMI matrix is calculated and its dimensionality is reduced using single value decomposition (SVD) [31]. At this point, the initial embeddings have been created. Next, retrofitting [16] is used to adjust the embeddings according to the information of the knowledge graph. Finally, the PPMI embeddings are merged with Word2vec and Glove to infer additional information. The process of inference and merging is based on the technique described in the following paper [51].

Evaluation

There are multiple ways to evaluate the performance of embeddings in different tasks. Each of these has a particular objective: from testing their effectiveness in downstream NLP tasks to exploring the semantics captured by the model. For the above reason, there is no consensus within the community about which method or methods are the most suitable to evaluate the performance of embeddings [3].

Schnabel et al. [42] classified the evaluation methods into two categories: intrinsic and extrinsic methods. A brief explanation of both follows.

Intrinsic evaluation methods Intrinsic evaluation methods are based on measuring the quality of embeddings based on tasks inherent to embeddings models, such as those seen in subsection 2.2.4. This type of evaluation is independent of the applications of the model in downstream tasks. Commonly all data used in these evaluations are created by human experts or crowd souring. Among these methods we can find:

- Word semantic similarity: Measure the distances between words that humans might consider similar (explained in subsection 2.2.4). Commonly an assessor chooses scores to relate the words. These are then compared to the distances of their representations.
- Analogy: It consists of evaluating the analogy operation (explained in subsection 2.2.4) on a dataset of questions created manually.
- Categorization: cluster the words in different categories (task explained in subsection 2.2.4) and compare them with hand-tagged sets.

Extrinsic evaluation methods Extrinsic evaluations measure the contribution of word embedding models to a specific NLP downstream task. In this case, the evaluation metric of the specific task is the metric that evaluates the quality of the embedding. In general, any NLP task that uses embeddings can be considered as an evaluation method. Within these tasks one can find text classification, sentiment analysis, named entity recognition among many others.

A comprehensive list of intrinsic and extrinsic evaluation methods, as well as the problems and limitations that still exist in these can be found in the work published by Barkarov [3].

Word Embedding Benchmark The word embedding benchmark [24] is a software toolkit that bundles a series of standardized intrinsic tests: analogy, similarity and categorization tasks. It allows for easy comparison of the performance of various pre-trained embedding models. The project is open-source, hence the source code and all the tests are publicly available in a repository [2]. In addition to this, the authors also published a list with the evaluation of various popular word embeddings models [3].

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

Limitations

As we mentioned above, models based on the distributional hypothesis (word-context matrices and word embeddings) manage to capture the semantics of words according to their contexts. However, thanks to their very formulation, they also have several limitations and problems.

It is very important that these limitations are exposed and taken into account, considering the consequences they can cause when used in more complex systems.

Section 10.7 of Goldberg's book [19] enumerates many relevant shortcomings of these models. In this subsection, we show some of them.

Definition of Similarity The similarity in distributional models is given only by the context of the words. There is no control over the type of similarity that can be applied to representations.

For example, consider the words *dog*, *cat* and *tiger*. If we wanted to find similarity with respect to pets within the model, *cat* would be more similar to *dog* than to *tiger*. On the other hand, if we are looking for similarity of *cat* with respect to both being felines, *tiger* would be much more similar than *dog*. However, in the embedding models, this cannot be controlled. This poses the first of the limitations.

Black Sheep When we communicate, we often skip details that we take for granted, but add them when we assume they are not obvious. Goldberg [19] points out a very clear example: we never say *white sheep*, because *sheep* is already associated with a white animal. However, when *black sheep* is communicated, it is always accompanied by the color. As a result of this, models will tend to learn non-obvious relationships as the most common ones to the detriment of the obvious ones.

Antonyms This is one of the most known limitations of this type of model. We know that antonyms represent words with opposite meanings. But like synonyms, the antonyms of a word tend to appear in the same contexts.

Take, for example, the sentences "it's hot today" and "it's cold today". Both are extremely common phrases and their context is exactly the same. However, their meaning is opposite.

This implies that antonyms, having similar contexts, are assigned similar representations within the model. Obviously, this fact can lead to an undesirable behavior in some NLP tasks such as sentiment analysis.

Lack of Context While distributional models base their learning on contexts, the representations learned are independent of them. This implies that when they are used, they cannot obtain information from their context. This is a major problem because many words tend to vary in meaning according to the context around them.

This problem can be clearly illustrated in the polysemic words: words that can have multiple meanings. Consider the example of the word *mouse*. On the one hand, it can represent a rodent. On the other hand, it can represent the pointer controller in a computer.

In a common distributional model, the representation of both concepts is contained in a single vector. This implies that when using a *mouse* representation, the meaning it renders will be that of both. And this cannot be selected according to the context.

Biases Several studies have shown the tendency of word embeddings models to inherit stereotypical social biases (regarding gender, ethnicity, religion, as well as other dimensions) that occur in the corpus on which they are trained. These biases can induce unintended or even harmful behavior in NLP systems dependent on such representations. For this reason it is extremely important to study them and try to mitigate them.

As we already mentioned in the introduction (Chapter 1), the research of this phenomenon is the main topic of study in this thesis. In order to cover it in a more comprehensive way, it is necessary to first introduce the field of fairness in machine learning. Once completed, we will revisit the previous work regarding the analysis and mitigation of bias in word embeddings.

2.3 Fairness in Machine Learning

Machine learning (ML) algorithms are widely used by businesses, governments and other organizations to make decisions that have a great impact on individuals and society [36]. These algorithms have the power to control the information we consume, the opportunities we are able to access and the ones we can not and can strongly condition the decisions we make. There are many examples of decisions made by ML that cover many areas of our daily lives: movie recommendations, personalized advertising and purchase suggestions, high-stakes decisions in loan applications, appointments, hiring, among many others. [33]

The use of ML systems responds to the multiple advantages they present with respect to humans: they do not get tired and can take into account an extremely large amount of information and factors when making a decision. However, machine learning systems present an essential problem when performing their functions: just like humans, these systems are prone to rendering *unfair decisions*.

Different definitions of bias and fairness applicable to AI systems have been presented in previous work. In the words of Mehrabi et al. 2019 [33], *fairness is the absence of any prejudice or favoritism toward an individual or a group based on their inherent or acquired characteristics. Thus, an unfair algorithm is one whose decisions are skewed toward a particular group of people.* On the other hand, Ntoutsi et al. 2020 [36] defined the bias as a *inclination or prejudice of a decision made by an AI system which is for or against one person or group, especially in a way considered to be unfair.*

With the massive expansion of ML systems in our daily life and the concern that they exhibit unfair behaviors, the study of fairness in IA models has become a very relevant research topic. The main objective of this field of study is to detect and analyze the possible sources of bias and prevent these systems from unfairly treating or harming any social group.

There are several studies showing that AI systems make biased decisions:

- The Correctional Offender Management Profiling for Alternative Sanctions (COMPAS) system for predicting the risk of re-offending was found to predict higher risk values for black defendants (and lower risk for white ones) than their actual risk [26]
- Richardson et. al [38] showed shown that police prediction systems in jurisdictions with extensive histories of illicit police practices presented high risks of dirty data leading to erroneous or illicit predictions, which in turn risk perpetuating additional harm via feedback loops.
- Google's Ads tool for targeted advertising was found to serve significantly fewer ads for high paid jobs to women than to men [10].
- The study of an ML system that judges beauty contest winners showed that it is biased against darker-skinned contestants [30].
- A facial recognition software in digital cameras that overpredicts Asians as blinking [39].

Mehrabi et al.[33] indicates the existence of two sources of unfairness in machine learning models: *those arising from biases in the data and those arising from the algorithms.*

2.3.1 Bias in Data

Machine learning relies heavily on learning from enormous amount of data generated by humans or collected via systems created by humans. These datasets are commonly heterogeneous, i.e., they are generated by different social groups, each with its own characteristics and behaviors and biases. Therefore, any bias that exists in these human-based datasets is learned by these systems and worse, may even be amplified. Bias in data can exist in many shapes and forms, some of which can lead to unfairness in different downstream learning tasks. The following is a description of the most general ones discussed in [47].

- **Historical Bias**: This is the type of bias that exists in the real world *as it is or was*, which, even with perfect attribute selection and sampling, is impossible to eliminate. An example of this is the search for images of CEOs on the web. Statistics show that only 5% of U.S. CEOs are women, which causes search engines to reflect the same proportion in their results [47]. While these results reflect reality, it is important to question whether this type of behavior is indeed desired.
- **Representation Bias**: This type of bias arises when the target population is underrepresented in the data. This type of bias can happen mainly for two reasons: the samples are not representative of the entire population or the population of interest has changed during the development of the model. Such is the case of *ImageNet*, a public dataset of images of approximately 1.2 million tagged images widely used in ML. Shankar et al. [43] showed that 45% of their images were taken in the United States, with the majority remaining in North America and Europe. They finish their work by showing that models trained with this data and then tested with images of underrepresented countries perform poorly with respect to the images of the countries contained in the dataset.

- **Measurement Bias**: It happens when we choose, measure and calculate the features that will then be used in a classification problem. The chosen set of features and labels may leave out important factors or introduce group or input-dependent noise. There are three main reasons for this type of bias: the measurement process varies across the observed groups, the quality of the groups varies across the observed groups, and the classification task that use the collected data is an oversimplification of reality. As a example, Suresh and Guttag [47] shows that within predictive policing applications, the proxy variable "arrest" is often used to measure "crime" or some underlying notion of "riskiness". Because minority communities are often more highly policed and have higher arrest rates, there is a different mapping from crime to arrest for people from these communities.
- **Aggregation Bias** This type of bias occurs when different populations are inappropriately combined. In most applications, the populations of interest are heterogeneous, which means that a single model is not sufficient to describe well all the different groups within the population. This can lead to no group being well represented or one group dominating the others.
- **Evaluation Bias** This occurs when the evaluation method does not represent the target population. The causes of this are that the measurement method does not equally represent the population and when the performance metric is external (designed to evaluate the performance of other tasks) or not appropriate for measuring the model performance.
- **Deployment Bias** This bias occurs when the model developed with a certain objective is incorrectly used to solve another problem. An example of this is risk prediction algorithms focused on predicting the probability of a person committing a crime. These algorithms have the ability to be used beyond their original formulation: they can predict prison time. Collins [9] showed that the indiscriminate use of such systems outside their original formulation can lead to distortions in the length of sentences and increase the use of imprisonment over other types of sanctions.

While different resources identify many more sources of bias, we only cover those mentioned above.

2.3.2 Algorithmic Fairness

Mehrabi et al. [33] considers a definition of fairness as *the absence of any prejudice or favoritism towards an individual or a group based on their intrinsic or acquired traits in the context of decision-making.*

Many applications of the design and implementation of fair algorithms involve classification problems. This is mainly due to the potential direct impact that the unfair behavior of these systems can have on society. Since classification bias is not the main objective of this thesis, we not go into further detail. However, the above mentioned survey [33] is a proper reference of this topic as it contains several definitions of fairness in classification.

The fact that modern text classification systems are heavily dependent on word embeddings has attracted many researchers to study the fairness exhibited by them. Next, we review some of the work that has focused on the study of fairness in word embeddings.

2.4 Fairness in Word Embeddings

Word embeddings models, as we mentioned in the section 2.2.4, became a core component in solving downstream tasks of natural language processing. Unfortunately, like all ML models, the embeddings are also susceptible to bias. Recent findings have led to claim that word embeddings are prone to perpetuate biases and prejudices contained in the corpora on which they are trained. The effects of this are at first sight alarming: the extensive use of biased embeddings in NLP systems cause them to render unfair decisions.

The study of fairness in word embeddings models has focused on two objectives: 1) measuring the bias present in these models, and 2) trying to mitigate it. In the following sections we show a set of studies that, through different metrics and tests, evidence different types of bias (gender, ethnic) in the embedding models as well as other efforts that have been made to mitigate it.

2.4.1 Works on Bias Measurement in Word embeddings

Below, we disucss three important works about the measurement of bias in word embeddings. Our contribution is strongly based on the proposals of these papers.

Semantics derived automatically from language corpora necessarily contain human biases

This work published by Caliskan et al. 2017 [7] is the first to show the existence of bias in word embeddings. The authors present a study on gender and ethnic biases in embeddings through the adaptation of the Implicit Association Test (IAT), (a bias test developed in psychology): The **Word Embedding Association Test (WEAT)**.

The IAT consists of measuring the differences in response times when subjects are asked to match concepts that they find similar, as opposed to two concepts that they find different. Studies using this methodology have reported significant differences in response time when running these tests. Response times are generally much shorter when the target concepts are likely to be associated. For example, subjects are much quicker to label insects as unpleasant and flowers as pleasant than if they were asked to do the opposite [7].

WEAT adapts IAT by quantifying the degree of association between the representations of two sets of target words (word representing social groups) and two sets of attribute words (words representing some attitude, characteristic, trait, occupational field). Since a word representation can contain different meanings, WEAT uses word sets that define the tested categories. The metric is calculated by performing arithmetic operations between the embeddings vectors of the words from each set. The result of this metric is a numerical result which the more positive the value, the more target 1 will be related to attribute 1 and target 2 to attribute 2. On the other hand, the more negative the value, the more target 1 will be related to attribute 2 and target 2 to attribute 1. The ideal score is 0.

The results of this study, based on previous IAT works, reveal biases regarding ethnicity (in relation to pleasantness) and gender (in relation to occupations). The Table 2.2 shows the tests performed in the study.

Target words	Attribute words	NT	NA	d	p
Flowers vs insects	Pleasant vs unpleasant	25×2	25×2	1.50	10^{-7}
Instruments vs weapons	Pleasant vs unpleasant	25×2	25×2	1.53	10^{-7}
Eur.-American vs Afr.-American names	Pleasant vs unpleasant	32×2	25×2	1.41	10^{-8}
Eur.-American vs Afr.-American names	Pleasant vs unpleasant	16×2	25×2	1.50	10^{-4}
Eur.-American vs Afr.-American names	Pleasant vs unpleasant	16×2	8×2	1.28	10^{-3}
Male vs female names	Career vs family	8×2	8×2	1.81	10^{-3}
Math vs arts	Male vs female terms	8×2	8×2	1.06	.018
Science vs arts	Male vs female terms	8×2	8×2	1.24	10^{-2}
Mental vs physical disease	Temporary vs permanent	6×2	7×2	1.38	10^{-2}
Young vs old people's names	Pleasant vs unpleasant	8×2	8×2	1.21	10^{-2}

Table 2.2: WEAT original results using word2vec model [7]. In the figure: N_T is the size of the target sets, N_A is the size of the attribute sets, d is the value of the of the metric and p is the p-value of the test.

In the first two tests, the authors present a set of baseline tests to explain how the metric behaves. They compare types of *flowers* target word set with *pleasant* attribute words and *insects* target word sets with *unpleasant* attribute words. Then, they repeat this same test for *musical instruments* and *weapons* with respect to *pleasant* and *unpleasant* attribute word sets. The results obtained in both tests serve as a reference to indicate what values WEAT should take when detecting a strong relationship between targets and attributes.

The following three experiments explore *ethnicity bias* using the same criteria: names commonly associated with *white people* are related to *pleasant terms* and names commonly associated with *black people* are related to *unpleasant terms*. The results are conclusive: the models detect a positive association between the sets and therefore, a marked ethnic bias against the black people.

Afterwards, *gender bias* tests are performed by comparing *feminine terms* with *family and arts* and *masculine terms* with *career, mathematics and science*. Once again, we can observe positive relationships between the groups, with the *career and family* experiment being the strongest detected in the whole study.

The words used in each test as well as the full results of this study can be found in the appendix of the original article [7].

Word Embeddings Quantify 100 Years of Gender and Ethnic Stereotypes

In a similar way, Garg et al. 2018 [18] develop a framework to quantify changes in stereotypes and attitudes toward women and ethnic minorities in the 20th and 21st centuries in the United States. The **Relative Norm Distance (RND)** metric was proposed to accomplish that goal. This work uses a different terminology: group words represent social groups (e.g., gender, ethnicity) and neutral words correspond to words that are not intrinsically related to any social group (e.g, firefighter, doctor). RND compares the embeddings of two sets of social group words against a single set of neutral words.

The results revealed that certain adjectives and occupations became more closely related to certain social groups over time. Thus, the authors showed that word embeddings are a powerful lens through which we can systematically quantify common stereotypes and other historical trends. The complete study and its results can be found in the original publication [18]. Additionally, the authors made available the code and words used in the experiments of this study on Github [4].

A Transparent Framework for Evaluating Unintended Demographic Bias in Word Embeddings

Sweeney and Najafian [48] present a framework and a metric for evaluating discrimination across protected groups via the relative negative sentiment associated with demographic identity terms. The proposed metric, the **Relative Negative Sentiment Bias (RNSB)** relies on a sentiment lexicon of positive and negative words for measuring bias. The main assumption is that in a fair model, a representation of a protected (e.g., "Indian") group term should have no difference in the probability of being classified as negative with respect to another protected group term (e.g. "Italian"). Therefore, in a fair model, the probability distribution of belonging to the negative sentiment class should be the same for all the terms tested.

The approach trains a logistic regression on the word embeddings matching the words of the lexicon, which is then applied to a set of protected group identity terms.[5] such as American, Mexican, and Canadian. Then, the metric is calculated as the Kullback-Leibler (KL) divergence between the negative sentiment probability of the identity terms (after normalization) and a uniform distribution.

The novelties of this metric are:

- It can be directly applied to more than 2 social groups.
- It is used to compare different pre-trained embedding models according to fairness. See [44] and Table 2.3.

Figure 2.2 shows the sentiment distribution of the demographic identity terms tested in their case study. The bottom histogram is the uniform distribution of negative sentiment in a perfectly fair scenario: there is no representation with more or less negative sentiment than the others. The top left shows the glove distribution. In this one it can be noted that terms such as Mexican, Indian or Russian are more likely to be negative than the other terms. The notorious difference between this distribution and the fair one implies the existence of a significant ethnic bias in this model. The top right distribution corresponds to conceptnet, which shows a more even and therefore fairer distribution than the previous one.

Table 2.3 shows the comparison between the values of different models of embeddings according to two case studies. In both cases, conceptnet is the least biased according to the metric result; then it is followed by word2vec with results close to the previous one, and finally glove with significantly higher scores than the other models tested.

[4] `https://github.com/nikhgarg/EmbeddingDynamicStereotypes`

[5] In their original work, a single word per social group

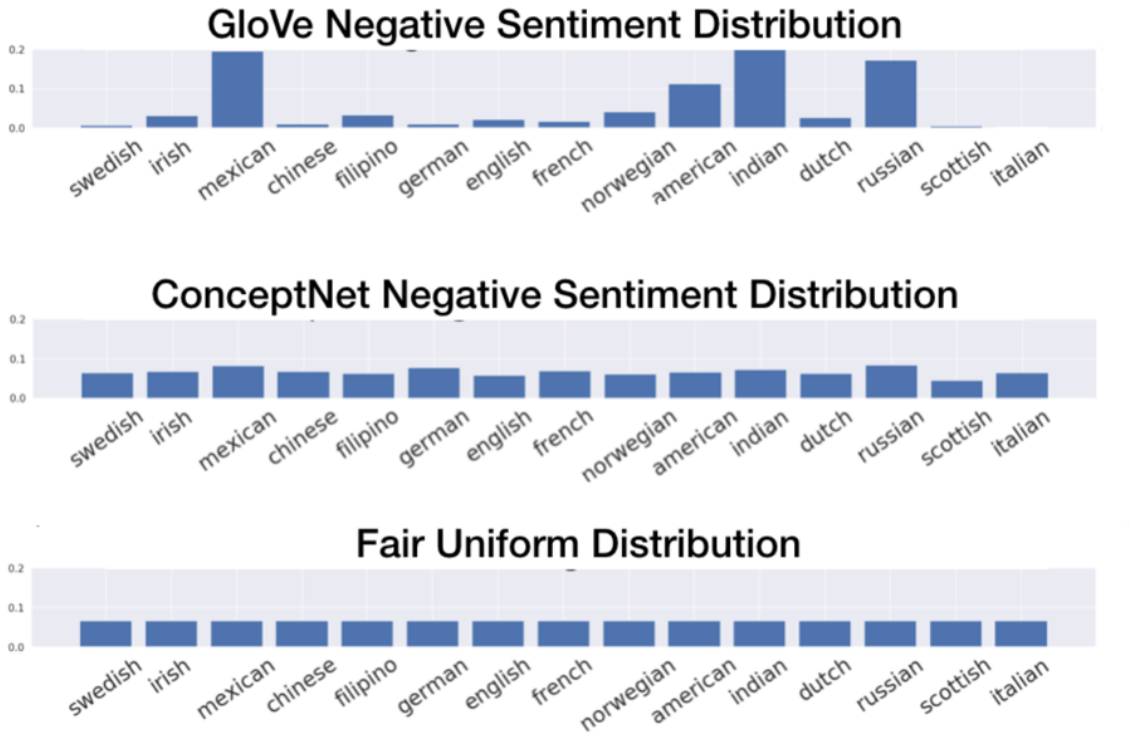

Figure 2.2: National origin identity terms sentiment distribution.

Case Study	Glove	Word2vec	Conceptnet
National Origin Identity	0.6225	0.1945	0.0102
Religion Identity	0.3692	0.1026	0.0291

Table 2.3: RNSB KL-divergence Case Study results.

2.4.2 Bias Mitigation of Word Embeddings

There have also been attempts to automatically mitigate bias (this process is commonly referred to as debiasing) in pre-trained word embeddings. Bolukbasi et al. [5] observed that there is one direction in the embedding space that largely captures gender. The proposed debiasing approach sets gender neutral words (e.g., occupations) to zero in the subspace generated by the gender direction.

On the other hand, Zhao et al [50] propose training debiased embeddings by changing the loss function. The change is targeted at concentrating all the gender-specific information to a particular dimension which can be later discarded.

However, Gonen and Goldberg [20] have argued that those approaches only hide the bias but do not eliminate it completely.

Several other debiasing methods have been proposed in recent years. [8, 11, 27, 28, 32, 50, 52].. Each of those proposes a different way to achieve this goal. However, we will not go into detail on these because it falls outside of the focus of this work. However, a comprehensive survey of methods focused on gender bias mitigation can be found in the work proposed by Sun et al. [46].

2.5 Discussion

As already discussed in Section 2.2.4, there are many pre-trained word embeddings models that can be freely obtained from the Web. Although previous studies have begun to measure bias in these models, they are limited in both the measured bias criteria (gender, ethnicity) and the tested models. Moreover, each study proposes its own metric with particular features. Every metric operates with different inputs (pairs of words, sets of words, multiple sets of words) and produces outputs that are incompatible with each other (reals, positive numbers, $[0, 1]$ range, etc.). As a consequence, the relationship between these metrics is unclear.

The fact that each metric provides a different way to measure bias can be useful in understanding bias from multiple points of view. It can also be very valuable in understanding which metrics are most likely to agree or disagree with each other. The possibility of using these metrics to measure and rank the bias of different word embedding models in a more standardized way is the main motivation of this thesis. To carry out such a study, we must first develop a common basis for unifying these different views of the same problem. With this goal in mind, we created a theoretical framework that aims to formalize the main building blocks for measuring bias in word embedding models: the Word Embedding Fairness Evaluation framework (WEFE). In the following chapter we present a detailed description of our proposal, as well as a case study in which our framework is put into practice.

Chapter 3

WEFE Design

In this chapter we describe the design of our main contribution to the area: WEFE: the Word Embeddings Fairness Evaluation framework, as well as its application for our case study. The structure of this chapter is described in the following lines:

The first section (Section 3.1) describes the building blocks of the theoretical framework. In this section we find the definition of target (Section 3.1.1) and attribute (Section 3.1.2) word sets, the queries (Section 3.1.3), the templates (Section 3.1.4) and the fairness metrics (Section 3.1.5).

The second section describes the process needed to run rank the embeddings using multiple bias tests and metrics (Section 3.2). First we detail how to run bias tests for the same criteria (gender, ethnicity or religion) on multiple embedding models and store the obtained scores in a matrix (Section 3.2.1). Then, we show how we convert the scores matrix into rankings (Section 3.2.2) and finally, how we gather several rankings all into a single ranking matrix (Section 3.2.3.

Section 3.3 defines the parameters used in the development of the case study: the embedding models (Section 3.3.1), the queries (Section 3.3.2) and metrics (Section 3.3.3) used.

The last section (Section 3.3.4) gives an exhaustive analysis of the results obtained by running the case study.

3.1 Building Blocks

WEFE is a framework that standardizes the bias measurement in word embeddings in a unified theoretical framework. In this section we formally define its main building blocks.

WEFE works over pretrained word embeddings. We assume that a word embedding model $\mathbf{M}$ is simply a function mapping a word w to a vector $\mathbf{M}(w)$ in $\mathbb{R}^{\mathrm{d}}$, where d is called the *dimension* of the embedding. For the rest of this article, and when the embedding model is clear from the context, words will not be explicitly distinguished from their corresponding embedding vectors.

3.1.1 Target Set

A target word set (denoted by T) corresponds to a set of words intended to denote a particular social group, which is defined by a certain criterion. This criterion can be any character, trait or origin that distinguishes *groups of people* from each other e.g., gender, social class, age, and ethnicity. For example, if the criterion is gender we can use it to distinguish two groups, *women* and *men*. Then, a set of target words representing the *women* social group could contain words like "*she*", "*woman*", "*girl*", etc. Analogously, the target words for the *men* social group could include "*he*", "*man*", "*boy*", etc. It should be noticed that constructing target sets of words that represent groups of people is a subjective procedure.

3.1.2 Attribute Set

An attribute word set (denoted by A) is a set of words representing some attitude, characteristic, trait, occupational field, etc. that can be associated with *individuals* from any social group. For example, the set of *science* attribute words could contain words such as "*technology*", "*physics*", "*chemistry*", while the *art* attribute words could have words like "*poetry*", "*dance*" and "*literature*". As for the case of target words, constructing attribute sets of words is a subjective procedure.

3.1.3 Query

A *query* is a pair $Q = (\mathcal{T}, \mathcal{A})$ in which $\mathcal{T}$ is a set of target word sets, and $\mathcal{A}$ is a set of attribute word sets. That is $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ where every T_i is a target word set, and $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$ where every A_j is an attribute word set. For example, consider the target word sets

$$\begin{aligned} T_{\text{women}} &= \{\text{she}, \text{woman}, \text{girl}, \ldots\}, \\ T_{\text{men}} &= \{\text{he}, \text{man}, \text{boy}, \ldots\}, \end{aligned}$$

and the attribute word sets

$$\begin{aligned} A_{\text{science}} &= \{\text{math}, \text{physics}, \text{chemistry}, \ldots\}, \\ A_{\text{art}} &= \{\text{poetry}, \text{dance}, \text{literature}, \ldots\}. \end{aligned}$$

Then the following is a query in our framework

$$Q = (\{T_{\text{women}}, T_{\text{men}}\}, \{A_{\text{science}}, A_{\text{art}}\}). \tag{3.1}$$

Queries are the main building blocks used by fairness metrics to measure bias of word embedding models. But before we explain how fairness metrics work in our context, we need to introduce some further technicalities.

3.1.4 Templates and Subqueries

A *query template* is simply a pair $(t, a) \in \mathbb{N} \times \mathbb{N}$. We say that query $Q = (\mathcal{T}, \mathcal{A})$ satisfies a template (t, a) if $|\mathcal{T}| = t$ and $|\mathcal{A}| = a$. For example, the query in equation (3.1) above, satisfies the template $(2, 2)$.

A template can also be used to produce all subqueries that satisfy the template. Formally, given query $Q = (\mathcal{T}, \mathcal{A})$ and template $s = (t, a)$, we denote by $Q(s)$ as the set of all queries $Q' = (\mathcal{T}', \mathcal{A}')$ such that $\mathcal{T}' \subseteq \mathcal{T}$, $\mathcal{A}' \subseteq \mathcal{A}$, and Q' satisfies template s, that is $|\mathcal{T}'| = t$ and $|\mathcal{A}'| = a$. For example, given the query Q in equation (3.1) above, the template $(2, 1)$ produces two subqueries

$$\begin{aligned} Q_1 &= (\{T_{\text{women}}, T_{\text{men}}\}, \{A_{\text{science}}\}) \\ Q_2 &= (\{T_{\text{women}}, T_{\text{men}}\}, \{A_{\text{art}}\}) \end{aligned}$$

and then $Q(s) = \{Q_1, Q_2\}$. As we later show, templates can be used to solve the *input mismatch* of fairness metrics.

3.1.5 Fairness Metrics

Intuitively, a fairness metric is a function that quantifies the degree of association between target and attribute words in a word embedding model. In our framework, every fairness metric is defined as a function that has a query and a model as input, and produces a real number as output. As we have mentioned in Section 2.4.1, several fairness metrics have been proposed in the literature. But, using our terminology, not all of them share a common input template for queries. Thus, we assume that every fairness metric comes with a template that essentially defines the shape of the input queries supported by the metric. For instance, a metric such as WEAT [7] has a $(2, 2)$ template, while the RND metric [18] has a $(2, 1)$ template.

Formally, let F be a fairness metric with template $s_F = (t_F, a_F)$. Given an embedding model $\mathbf{M}$ and a query Q that satisfies s_F, the metric produces the value $F(\mathbf{M}, Q) \in \mathbb{R}$ that quantifies the degree of bias of $\mathbf{M}$ with respect to query Q.

We still have the problem of how to interpret the value $F(\mathbf{M}, Q)$. Although it depends on every particular metric, we assume that the metric is equipped with a total order relation $\leq_F$ that establishes what is to be considered as *less biased*. That is, if we fix a query Q and we consider two different models $\mathbf{M_1}$ and $\mathbf{M_2}$, then $F(\mathbf{M_1}, Q) \leq_F F(\mathbf{M_2}, Q)$ states that model $\mathbf{M_1}$ *is less biased than model* $\mathbf{M_2}$ when measuring bias with respect to query Q. Notice that with this order relation, we can avoid having to actually interpret the value given by F, and just use it to compare embedding models, which is exactly what the ranking part of WEFE does.

3.2 WEFE Ranking Process

Next, we will show how to rank by fairness the embeddings models using multiple queries and multiple fairness metrics. Our starting point is composed of three sets:

- a set $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_r\}$ of predefined queries where each Q_i represents a particular bias test over a certain criterion,
- a set $\mathcal{M} = \{\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n\}$ of pre-trained word embedding models, and
- a set $\mathcal{F} = \{F_1, \ldots, F_m\}$ of fairness metrics, where every F_i comes with its particular template $s_\text{i} = (t_\text{i}, a_\text{i})$ and order relation $\leq_{F_\text{i}}$.

3.2.1 Creating the Scores Matrix

Lets fix a fairness metric $F \in \mathcal{F}$ and assume that $s = (t, a)$ is its associated query template. The first step is to update $\mathcal{Q}$ by adding all subqueries that satisfy the template. Formally, we create the new set

$$\mathcal{Q}_F = Q_1(s) \cup Q_2(s) \cup \cdots \cup Q_r(s)$$

where $Q_{\mathrm{i}}(s)$ is the set of all subqueries of Q_{i} that satisfy template $s = (t, a)$. We note that $\mathcal{Q}_F$ is usually bigger than $\mathcal{Q}$ (they coincide if the template of the metric is satisfied by all the original queries in $\mathcal{Q}$).

Now for a fixed embeddings model $\mathbf{M} \in \mathcal{M}$ we can compute the value $F(\mathbf{M}, Q)$ for every $Q \in \mathcal{Q}_F$. We can think of these values as a row vector of fairness scores, where every component of the vector corresponds to a different query. We repeat this process for every model $\mathbf{M}_{\mathrm{i}} \in \mathcal{M}$ to construct a scores matrix associated to the fairness metric F. This matrix is of dimensions $|\mathcal{M}| \times |\mathcal{Q}_F|$.

3.2.2 Creating the Rankings

The next step is to create the ranking. First, we aggregate each of the scores by embedding model (for each row). To do this, we need to choose an aggregation function that is consistent with the metric F. In particular, we need to ensure that the aggregation satisfies the following monotonicity property with respect to $\leq_F$. Let x, y, x' and y' be arbitrary values in $\mathbb{R}$, and assume that $x \leq_F y$ and $x' \leq_F y'$. Then it must hold that $\mathrm{agg}(x, x') \leq_F \mathrm{agg}(y, y')$. For most of the metrics that we use in our case study, an aggregation function such as the mean of the absolute values of the scores would satisfy this property. But for more complicated metrics deciding on a good aggregation function might not be a trivial matter.

After aggregating the scores we end up with a column vector of size $|\mathcal{M}|$ over whose we can use $\leq_F$ to construct a ranking for all the embeddings in $\mathcal{M}$. For us, this ranking is represented by another column vector which values are a permutation of the values in $\{1, 2, \ldots, |\mathcal{M}|\}$ stating the index for each embedding model in the generated ranking. This ranking is generated in an ascending way, that is, smaller scores get the top positions.

3.2.3 Gathering Rankings in a Final Matrix

Finally, we can repeat the previous process for each one of the fairness metrics in $\mathcal{F}$ to obtain a final matrix of size $|\mathcal{M}| \times |\mathcal{F}|$ containing the ranking indexes of every embedding model for every metric. In our case study, we use this matrix to study correlations among the fairness rankings produced by different metrics and for different sets of queries.

There are several aspects of the process that should be noticed. First, the dimensions of the final matrix ($|\mathcal{M}| \times |\mathcal{F}|$) is independent of the queries used to define the bias that we are considering. Moreover, every column in this matrix represents a fairness metric as a permutation of the same set of integers ($\{1, 2, \ldots, |\mathcal{M}|\}$).

These two aspects allow us to effectively compare all different fairness metrics even though they can receive different forms of queries as inputs, and produce different scores as outputs.

Name	Number of dimensions	Training Corpus	References
Conceptnet	300	Conceptnet, word2vec, Glove, and OpenSubtitles 2016	[45]
Fasttext	300	Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (16B tokens)	[4]
Glove-twitter	200	Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased)	[37]
Glove-wikipedia	300	Wikipedia 2014 + Gigaword 5 (6B tokens, uncased)	[37]
Lexvec	300	Common Crawl	[41, 40]
Word2vec	300	Google News (about 100 billion words)	[35, 34]
Gender Hard Debiased Word2vec	300	Google News (about 100 billion words)	[35, 34, 5]

Table 3.1: Details of the embeddings used in our case study

We also notice that we can compare all metrics without needing to actually change any of its particularities.

Finally, any other meaningful ranking of embeddings can be added to this matrix and the correlations and comparisons can still be computed. In our case study, we add a performance ranking obtained from the Word Embedding Benchmark [24].

3.3 Case Study

In this section we instantiate our framework to conduct a case study in which seven publicly available word embedding models are compared according to four fairness metrics. These metrics are described in detail in the next section. We first briefly describe the embedding models and queries.

3.3.1 Embedding models

Table 3.1 shows a complete detail of the pre-trained word embeddings models used in our case study. Note that we added a word2vec gender debiased model in order to check the effects of this procedure according to our queries. Most of the models were obtained using Gensim library interface[1]. Lexvec and Gender Hard Debiased Word2vec were obtained from their original sources.[2][3]

[1] `https://github.com/RaRe-Technologies/gensim-data`
[2] `https://github.com/alexandres/lexvec`
[3] `https://github.com/tolga-b/debiaswe`

3.3.2 Queries and Query Sets

We consider a total of 25 queries satisfying the $(2, 2)$ template, all of them built upon previous work.

From them we construct three query sets $\mathcal{Q}_{\text{gender}}$ with 7 queries, $\mathcal{Q}_{\text{ethnicity}}$ with 9 queries, and $\mathcal{Q}_{\text{religion}}$ with 9 queries. For the sake of space we cannot describe the content of each query, but we next list the previous work from which we form all of them. We take the attribute word sets *pleasant*, *unpleasant*, *math* and *arts* from [7]; the target sets *ethnicity-surnames*, *male* and *female*, and attribute words related to *intelligence*, *appearance*, *sensitive* and *occupations* were taken from [18]; the attribute word set *religion* was taken from [32]; *positive* and *negative* sentiment attribute words were taken from the Bing Liu lexicon [23].

The full list of queries with their respective sets of words are available at queries annex B.

3.3.3 Specific Fairness Metrics

Next, we describe the four fairness metrics we consider in this case study from the point of view of WEFE.

Word Embedding Association Test (WEAT)

Proposed by Caliskan et al. [7] the WEAT metric receives two sets T_1 and T_2 of target words, and two sets A_1 and A_2 of attribute words. Thus, in our terminology, it always expects a query of the form $Q = (\{T_1, T_2\}, \{A_1, A_2\})$ and then its associated template is $s_{\text{WEAT}} = (2, 2)$. Its objective is to quantify the strength of association of both pair of sets through a permutation test. Given a word embedding w, WEAT defines first the measure $\text{d}(w, A_1, A_2)$ as

$$\big(\text{mean}_{x \in A_1} \cos(w, x)\big) - \big(\text{mean}_{x \in A_2} \cos(w, x)\big)$$

where $\cos(w, x)$ is the cosine similarity of the word embedding vectors. Then for a query $Q = (\{T_1, T_2\}, \{A_1, A_2\})$ the WEAT metric is defined as

$$F_{\text{WEAT}}(\mathbf{M}, Q) = \sum_{w \in T_1} \text{d}(w, A_1, A_2) - \sum_{w \in T_2} \text{d}(w, A_1, A_2)$$

The idea is that the more positive the value given by F_{WEAT}, the more target T_1 will be related to attribute A_1 and target T_2 to attribute A_2. On the other hand, the more negative the value, the more target T_1 will be related to attribute A_2 and target T_2 to attribute A_1. The ideal score is 0. This would imply that there is no unjustified relationship between the target sets and the attribute sets. Thus, the order induced by WEAT is such that $x \leq_{F_{\text{WEAT}}} y$ iff $|x| \leq |y|$.

WEAT Effect Size (WEAT-ES)

This metric represents a normalized measure that quantifies how far apart the two distributions of association between targets and attributes are. It also receives queries with template $s_{\text{WEAT-ES}} = (2, 2)$. Then $F_{\text{WEAT-ES}}(\mathbf{M}, Q)$ is computed as:

$$\frac{\text{mean}_{w \in T_1} \, \text{d}(w, A_1, A_2) - \text{mean}_{w \in T_2} \, \text{d}(w, A_1, A_2)}{\text{std}_{w \in T_1 \cup T_2} \, \text{d}(w, A_1, A_2)}$$

Since the ideal is also 0, we define $\leq_{F_{\text{WEAT-ES}}}$ just as $\leq_{F_{\text{WEAT}}}$.

Relative Norm Distance (RND)

Proposed by Garg et al. [18], it receives queries with template $s_{\text{RND}} = (2, 1)$. Given a query $Q = (\{T_1, T_2\}, \{A\})$ the metric $F_{\text{RND}}(Q)$ is computed as

$$\sum_{x \in A} \left(\| \, \text{avg}(T_1) - x \|_2 \; - \; \| \, \text{avg}(T_2) - x \|_2 \right)$$

where $\| \cdot \|_2$ represents the Euclidean norm, and $\text{avg}(T)$ is the vector resulting from averaging all the vectors in T. That is, RND averages the embeddings of each target set, and then for each one of the attribute words, it computes the norm between the words and the targets averages and then subtracts the norms. The more positive (negative) that the relative norm distance is, the more associated the attribute set is towards group two (one). The optimal value here is 0, and thus as for WEAT we let $x \leq_{F_{\text{RND}}} y$ iff $|x| \leq |y|$.

Relative Negative Sentiment Bias (RNSB)

We consider a straightforward generalization of this metric [48][4]. RNSB receives as input queries with two attribute sets A_1 and A_2 and two or more target sets, and thus has a template of the form $s = (N, 2)$ with $N \geq 2$. Given a query $Q = (\{T_1, T_2, \ldots, T_n\}, \{A_1, A_2\})$ and an embedding model $\mathbf{M}$, in order to compute the metric $F_{\text{RNSB}}(\mathbf{M}, Q)$ one first constructs a binary classifier $C_{(A_1, A_2)}(\cdot)$ using set A_1 as training examples for the negative class, and A_2 as training examples for the positive class. After the training process, this classifier gives for every word w a probability $C_{(A_1, A_2)}(w)$ that can be interpreted as the degree of association of w to attribute A_2 (value $1 - C_{(A_1, A_2)}(w)$ is the degree of association to A_1). Now, we construct a probability distribution $P(\cdot)$ over all the words w in $T_1 \cup \cdots \cup T_n$, by computing $C_{(A_1, A_2)}(w)$ and normalizing it to ensure that $\sum_w P(w) = 1$. The main idea behind RNSB is that the more that $P(\cdot)$ resembles a uniform distribution, the less biased the word embedding model is. Thus, one can compute $F_{\text{RNSB}}(\mathbf{M}, Q)$ as the distance between $P(\cdot)$ and the uniform distribution $U(\cdot)$. RNSB uses the KL-divergence to compute that distance. As before, the optimal value is 0. Since it cannot deliver negative values, we let $x \leq_{F_{\text{RNSB}}} y$ iff $x \leq y$.

[4] In the original RNSB proposal, attribute sets of words are always associated to positive and negative lexicons, and in the experiments they only consider singletons as target sets.

Queries set by criteria Model name	Gender WEAT	WEAT-ES	RND	RNSB	Ethnicity WEAT	WEAT-ES	RND	RNSB	
conceptnet-numberbatch 19.08-en dim=300	2 (0.37)	2 (0.20)	2 (0.01)	**1 (0.02)**	**1 (0.46)**	**1 (0.14)**	2 (0.03)	**1 (0.03)**	
fasttext-wiki-news dim=300	5 (0.71)	4 (0.47)	3 (0.02)	2 (0.02)	3 (0.49)	5 (0.20)	3 (0.06)	2 (0.04)	
glove-twitter dim=200	3 (0.50)	3 (0.41)	5 (0.13)	5 (0.23)	6 (0.75)	6 (0.42)	5 (0.16)	5 (0.07)	
glove-wiki-gigaword dim=300	4 (0.66)	7 (0.84)	6 (0.18)	6 (0.29)	7 (1.00)	7 (0.58)	6 (0.26)	4 (0.07)	
lexvec-commoncrawl W+C dim=300	6 (0.79)	5 (0.71)	7 (0.33)	7 (0.32)	2 (0.47)	2 (0.15)	7 (0.73)	7 (0.17)	
word2vec-gender-hard-debiased dim=300	**1 (0.16)**	**1 (0.08)**	**1 (0.00)**	3 (0.03)	4 (0.52)	3 (0.19)	**1 (0.03)**	3 (0.05)	
word2vec-google-news dim=300	7 (0.90)	6 (0.82)	4 (0.08)	4 (0.14)	5 (0.53)	4 (0.19)	4 (0.15)	6 (0.12)	
Queries set by criteria Model name	Religion WEAT	WEAT-ES	RND	RNSB	Overall WEAT	WEAT-ES	RND	RNSB	WEB
conceptnet-numberbatch 19.08-en dim=300	4 (0.96)	**1 (0.11)**	2 (0.05)	2 (0.07)	2 (0.61)	**1 (0.15)**	2 (0.03)	2 (0.04)	**1**
fasttext-wiki-news dim=300	**1 (0.84)**	3 (0.16)	3 (0.13)	**1 (0.04)**	3 (0.68)	3 (0.26)	3 (0.07)	**1 (0.03)**	2
glove-twitter dim=200	2 (0.84)	2 (0.15)	6 (0.44)	5 (0.18)	4 (0.71)	4 (0.32)	5 (0.25)	5 (0.15)	7
glove-wiki-gigaword dim=300	7 (1.18)	7 (0.27)	5 (0.33)	3 (0.10)	7 (0.97)	7 (0.54)	6 (0.26)	4 (0.14)	6
lexvec-commoncrawl W+C dim=300	3 (0.94)	6 (0.21)	7 (0.89)	6 (0.22)	5 (0.73)	5 (0.33)	7 (0.65)	7 (0.23)	4
word2vec-gender-hard-debiased dim=300	6 (1.05)	5 (0.19)	**1 (0.03)**	4 (0.17)	**1 (0.61)**	2 (0.16)	**1 (0.02)**	3 (0.09)	5
word2vec-google-news dim=300	5 (1.04)	4 (0.19)	4 (0.20)	7 (0.31)	6 (0.82)	6 (0.37)	4 (0.15)	6 (0.19)	3

Table 3.2: Final matrices obtained after applying our framework for several metrics, embedding models, and three different query sets.

3.3.4 Results

Using the WEFE ranking process described in Section 3.2 together with the three query sets, the four fairness metrics and the seven embedding models described above, we obtained three scores matrices that are shown in Table 3.2, one for each query set $\mathcal{Q}_{\text{gender}}$, $\mathcal{Q}_{\text{ethnicity}}$ and $\mathcal{Q}_{\text{religion}}$. Additionally, we created a fourth matrix (Overall in Table 3.2) by applying our framework to query set $\mathcal{Q} = \mathcal{Q}_{\text{gender}} \cup \mathcal{Q}_{\text{ethnicity}} \cup \mathcal{Q}_{\text{religion}}$ containing all our queries using an aggregation function that performs a weighted average of the different query sets (the weights correspond to the cardinality of each query set).

We add an additional column to this last matrix obtained by running the Word Embedding Benchmark (WEB) on our embedding models. WEB rankings are obtained by adding up the rankings produced by all the metrics implemented by the benchmark. Notice that WEB metrics are ranked in descending order unlike the metrics evaluated in WEFE.

In addition, we generate correlation matrices between the rankings using Spearman's rank correlation coefficient (Figure 3.1). These allow us to state whether or not the rankings are aligned with each other according to the criteria evaluated. Such agreement would allow us to state more strongly that the rankings obtained are more reliable.

If we focus on the gender results in Table 3.2, we can observe a clear tendency for word2vec-gender-hard-debiased and conceptnet to be at the top of the ranking. We can also observe that the debiased version of word2vec outperforms the non-debiased version across all metrics. Another noteworthy result derived from Figure 3.1, are the high correlations observed between all metrics for gender. This implies a consistency between metrics with respect to gender bias.

For the case of ethnicity, although conceptnet consistently outperforms other models (with rankings, 1, 1, 2, and 1) the differences in terms of absolute scores with the closest competitor is very short. Moreover, for ethnicity the correlations between each ranking is almost totally lost (Figure 3.1).

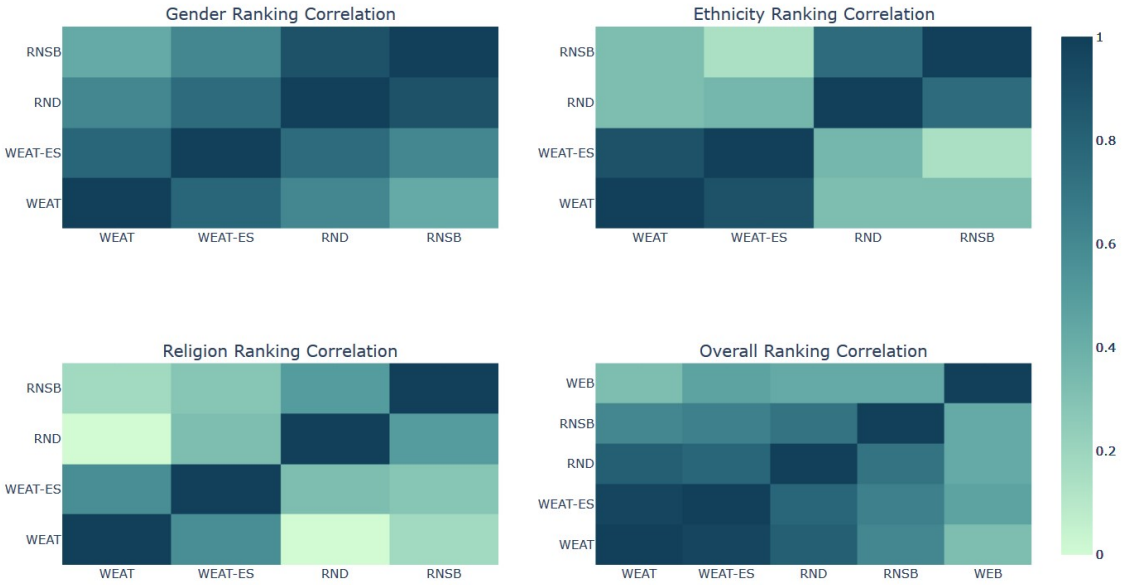

Figure 3.1: Spearman correlation matrix of rankings by different measures.

Something similar happens for the case of religion in which not only the correlation between the rankings is lost but in which it is not clear at all what method is the best. Another observation worth reporting is that for the case of both ethnicity and religion the differences between word2vec and word2vec-gender-hard-debiased in terms of absolute values are considerably less notable compared with the gender case.

Other results that are somewhat consistent across the three tables and the four metrics, are that models, glove-twitter, glove-wikipedia, lexvec and word2vec-google-news, are rarely found in top ranking positions.

Unlike the results for ethnicity and religion, the overall matrix shows a more consistent behavior (Table 3.2). Conceptnet and fastext take the first two places in all metrics. In addition, the low scores obtained by the above-mentioned models are maintained. Similarly to the gender matrix, the four fairness metrics exhibit clear positive ranking correlations (Figure 3.1) in the overall matrix.

In relation to the rankings obtained from the Word Embedding Benchmark (WEB), although conceptnet and fasttext maintain their leading positions (Table 3.2), there is no clear correlation between WEB and WEFE rankings (Figure 3.1). For example, lexvec, which is poorly ranked among WEFE scores, is in the middle of the WEB ranking. In the case of word2vec and its gender debiased variation, their positions in WEB and WEFE rankings are swapped. This suggests that the gender debiasing method proposed in [5] can affect the performance of the embedding model in word similarity and analogy tests.

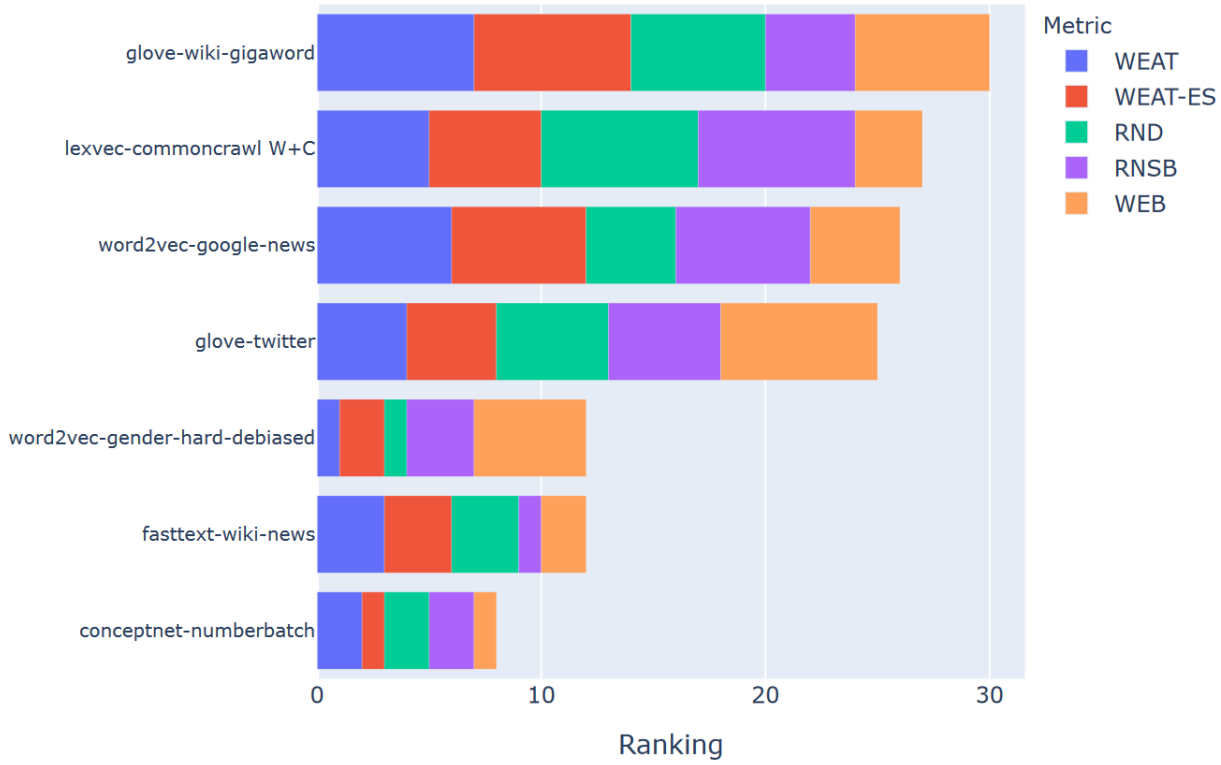

Figure 3.2: Accumulated rankings by metric for the overall results plus WEB.

These misalignments can be further analyzed in Figure 3.2. The figure displays the rankings obtained by the overall WEFE rankings and WEB results using cumulative graph bars (i.e., the larger the size of a bar the lower its position in the corresponding ranking). The figure allows for easy detection of models with good WEFE rankings and poor WEB rankings, such as word2vec gender-hard-debiased version and glove-twitter, as well as the opposite effect: high bias and good WEB performance, such as word2vec and lexvec.

Chapter 4

WEFE Library

The WEFE library is an open source implementation of our framework for measuring bias in word embedding models. The library is currently published and can be found at the following link[1]. It was developed in the Python programming language and is integrated with various of the popular data science libraries available for this language. It can be installed using the `pip`[2] or `conda`[3] package manager tools. The project was released under the *BSD 3-Clause* license and is publicly hosted on Github[4]. The source code was structured according to the design and code patterns recommended by sickit-learn community. It has an extensive documentation which contains several tutorials, the *API* definition, a guide explaining how to contribute to the project, how to execute the tests and how to compile the documentation.

The library was designed to meet the following objectives:

- Encapsulating existing fairness metrics from previous work and designing new ones.
- Encapsulating the test words used by fairness metrics into standard objects called queries.
- Computing a fairness metric on a given pre-trained word embedding model using user-given queries.

It also provides more advanced features for:

- Running several queries on multiple embedding models and returning a DataFrame with the results.
- Plotting those results on a barplot.
- Based on the above results, calculating a bias ranking for all embedding models. This allows the user to evaluate the fairness of the embedding models according to the bias criterion (defined by the query) and the metric used.
- Plotting the rankings on a barplot.

[1] `https://wefe.readthedocs.io/en/latest/about.html`
[2] `https://pypi.org/project/wefe/`
[3] `https://anaconda.org/pbadilla/wefe`
[4] `https://github.com/dccuchile/wefe`

- Correlating the rankings. This allows the user to see how the rankings of the different metrics or evaluation criteria are correlated with respect to the bias presented by the models.

In the next section we provide details of the library and its implementation. We start presenting a brief motivation of why we implemented and published this library. Then, we present the design of the main components and the typical processes involved in measuring bias. We end by showing the advanced processes of the library such as the calculation of several queries on various embeddings, the aggregation of these results, their subsequent conversion to rankings and finally, the calculation of correlations between rankings.

In addition, the tutorials containing the user guide and the replication of the case studies from previous papers can be found in Annex C.

4.1 Motivation

One of the main requirements to conduct our case study was the implementation of the fairness metrics and queries used in previous studies.

We tried to reuse all publicly available resources (open source code, word sets) and to recreate the missing ones. However, when the development of our theoretical framework was completed, it became evident that it would not be possible to unify directly their implementations. Each code and resource was developed to meet the requirements of each particular work. This severely constrained its reuse and extension.

These reasons led us to re-implement the metrics and queries under the guidelines of our framework. The high degree of formalization provided by WEFE allowed us not only to implement the metrics, but to develop them into a well-designed and highly extensible code.

Moreover, the lack of standardized tools to measure bias in embeddings, the limitations in our case study and the high-level development we were carrying out led to the idea of publishing our code as an open library available to any member of the research community.

4.2 Components

In this Section we show the design and implementation of each building block of our framework shown in Chapter 3.

4.2.1 Target and Attribute Sets

As we explained in the previous chapters, each set of target words T is a set of words that denotes a particular social group. In code, each word is represented by a string. Therefore, the most direct way to represent these sets is by using a list of string arrays.

The pattern applies for the A attribute word sets (words that represent attributes, traits, occupations, among others). Each attribute word is represented in the code as a string. Thus, a set of attribute words is represented by a list of string arrays.

Furthermore, a collection of attribute word sets is simply represented as a list containing these sets. The same applies for a collection of attribute sets.

The decision not to create particular classes for these two main components is due to the simplicity that Python offers to represent them. We allow the logic that checks and processes these sets to reside in the class that will contain both sets: `Query`.

4.2.2 Query

As a reminder, in our framework a *query* is a pair $Q = (\mathcal{T}, \mathcal{A})$ in which $\mathcal{T}$ is a set of target word sets, and $\mathcal{A}$ is a set of attribute word sets. In simpler words, this component is intended to be the container for of target and attribute sets.

Our query objects are implemented by the `Query` class. When a Query object is created, it must receive as parameters lists with the targets and attribute sets. Additionally, it can receive the names of the sets, which allows the user to define a name for the query. All these parameters are stored as attributes. In addition, the the method responsible for creating the object automatically calculates the query template and stores it as an additional attribute.

4.2.3 Word Embedding Model

The `WordEmbeddingModel` class is a container class focused on storing the embedding models on which the bias measurements will be performed.

The mandatory parameters of this class are the model (gensim's KeyedVectors or gensim's api loaded models) and the name of the model used. For some embedding models such as conceptnet [45] words can contain a prefix that indicates their language before the word. For instance, the English word `cat` in conceptnet is associated with the token `/c/en/cat`. This is why there is the possibility of providing a parameter prefix that will be used to find the representations of each word.

4.2.4 Metric

A fairness metric is a function that quantifies the degree of association between targets and attribute words through vectorial operations in our framework. The library expands this definition even further: each metric is implemented in a different class which inherits the `BaseMetric` class and its operations are implemented in a particular method.

The `BaseMetric` class contains functions that are common for all metrics. The two main functionalities of this class are: 1) validate the inputs and, 2) transform the word of each set of the query into embeddings.

Every fairness metric class must implement the `run_query` method. This method is the standard interface for calculating a fairness score. The method receives as input a query, an embeddings model and other configuration parameters, and returns the calculated score. The common procedure that this class performs is described below:

1. The first step consists of checking the input parameters and transforming words into embeddings (using the `BaseMetric`).
2. The second step consists of performing vectorial operations of the embeddings of the words associated with the query.
3. The final step consists of returning the calculated score.

4.2.5 Utils

Our library also implements a series of utility functions that facilitate the execution of several processes. The three main functions are listed below:

- `Run_queries`: a function to test multiple queries on various word embedding models in a single call. It has a large number of functionalities that can be configured through the parameters. Among the most important is the aggregation by embeddings, which allows to add the results of the queries by model. These aggregations are relevant to compare and subsequently rank the bias that the models present according to the queries evaluated.
- `Create_ranking`: this function takes the aggregated results by model, typically generated by the previous function, and turns them into rankings.
- `Calculate_ranking_correlations`: this function takes a list of arrays containing fairness rankings calculated with the previous function and calculates the Spearman correlation between them.

4.3 Bias Measurement Processes

In this section we describe the most important processes implemented by the library:

1. The execution of a simple query.
2. The execution of several queries in one call.
3. The aggregation of their results and their subsequent conversion to ranking.
4. The calculation of the correlation between different rankings.

4.3.1 Simple Query Creation and Execution

The simplest process that can be executed is to create and run a query. The diagram in Figure 4.1 shows and example in which a gender bias test using the WEAT metric is executed. The representations used are obtained from the word2vec [35] embedding model.

The blue rounded rectangular boxes from the first column show the process's inputs: The first box contains the sets of target and attribute words that will be used. The second contains the given word embedding model. The third contains the metric that will calculate the final fairness scores in the next steps.

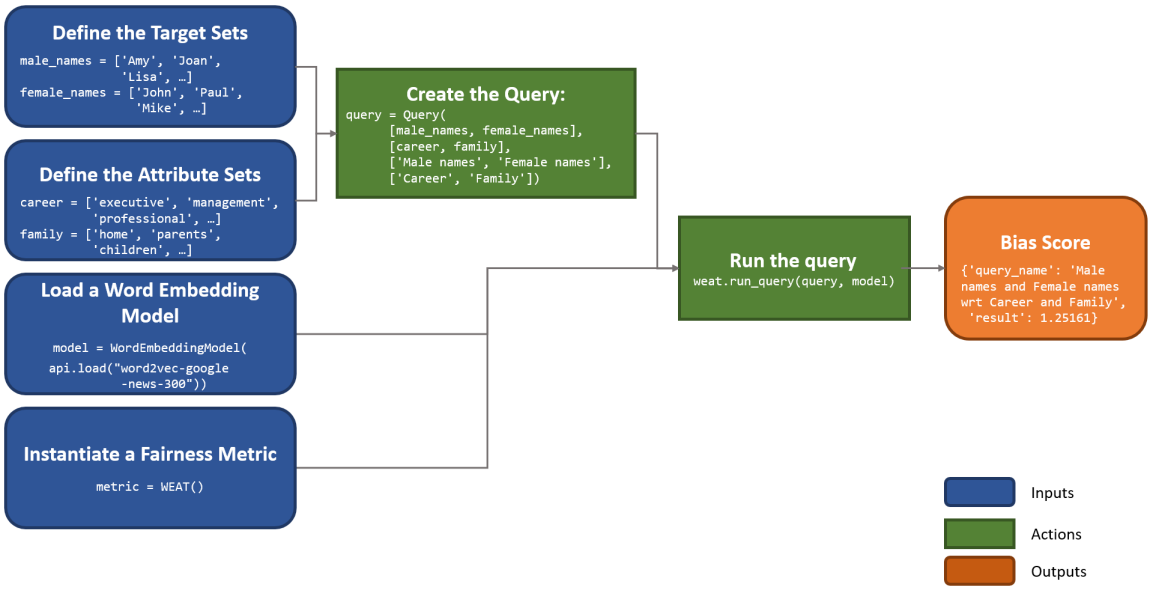

Figure 4.1: Creation and execution of a gender query on word2vec using WEAT.

The green rectangles from the second and thrid columns indicate the actions of the process. The upper box shows the creation of the query using the word sets defined in the above step. Then, we execute the measurement by passing the query and the word embedding model to the `run_query` function. Finally, the score obtained when executing the previous step is the output of the process, which is represented by the orange rounded rectangle.

4.3.2 Runners

We usually want to test several queries on several embedding models to study the biases generally. Trying to execute `run_query` on each pair embedding-query can be complex and will require extra work to implement. For that reason the library also implements a function to test multiple queries on various word embedding models in a single call: the `run_queries` util. The diagram contained in the figure 4.2 shows a typical case of execution of `run_queries`.

The first column with blue rounded rectangular figures shows the inputs: word sets, word embeddings and a fairness metric. The second column with green rectangles shows the actions performed. First the queries are created and then `WEAT` is executed for each query-model pair using `run_queries`. The results are the fairness scores obtained. They are represented by the orange rounded rectangle.

4.3.3 Aggregating Results and Calculating Rankings

The execution of `run_queries` commonly delivers many scores. While each studies a specific bias, these alone do not provide information on the overall bias of the embeddings. We can provide a good indicator of the bias contained in the models by aggregating the scores. We can require that when we execute `run_queries` we calculate the aggregations in the same call by setting the `aggregate_results` parameter as `True`. This is shown in Figure 4.3.

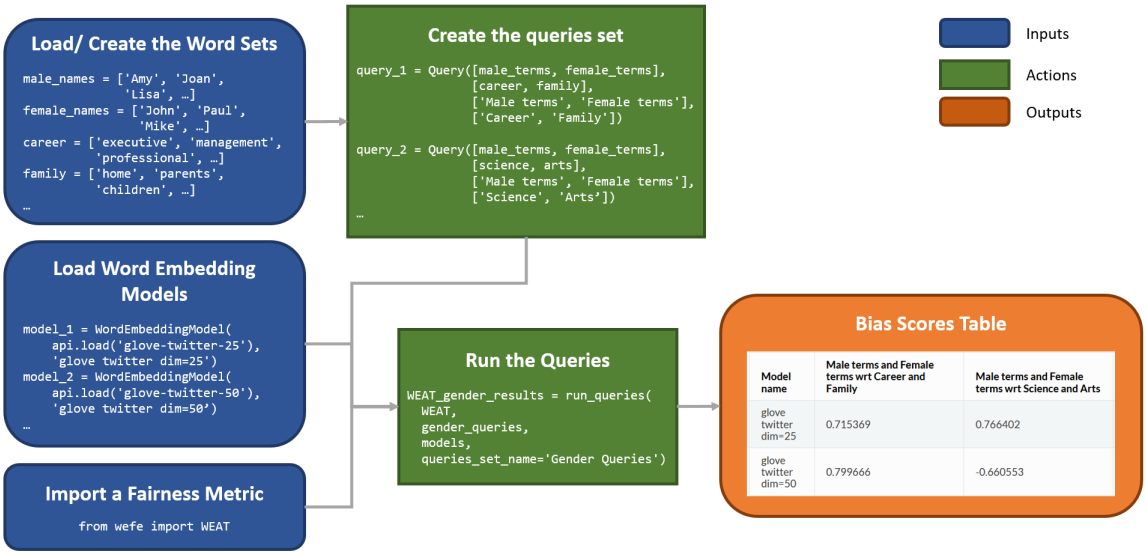

Figure 4.2: Creation and execution of several gender queries on various embedding models using WEAT.

Additionally, a ranking that is calculated from these aggregated results may be easier to interpret when looking for the less biased models. For that reason, the Figure 4.3 also shows how to calculate rankings through the `create_ranking` function.

As in the previous process, the first column with blue rounded rectangular figures shows the entries: word sets, word embeddings and a metric. The second column with green rectangles shows the actions performed. First the queries are created and then `WEAT` is executed for each query-model pair using `run_queries`. The difference now is that the runner function can be configured with the parameters `aggregate_results` and `return_only_aggregation` so that it only returns the aggregated scores per embedding (row). The following column shows the outputs of this process process: 1) the aggregated results (which can be used directly to describe the tested bias) and 2) the rankings associated with the previous scores (that are computed using the `create_ranking` function).

4.3.4 Ranking Correlations

A set of queries subject to some criteria and a set of embeddings can be tested several times using different metrics. Each metric will return different results according to the operations it performs. Therefore, each metric will generate a different ranking.

As we explained our case study of Section 3.3, similar rankings will imply greater reliability in the results. We can verify this agreement of the rankings through correlation matrices. In our software, these matrices can be calculated using the `calculate_ranking_correlations` function.

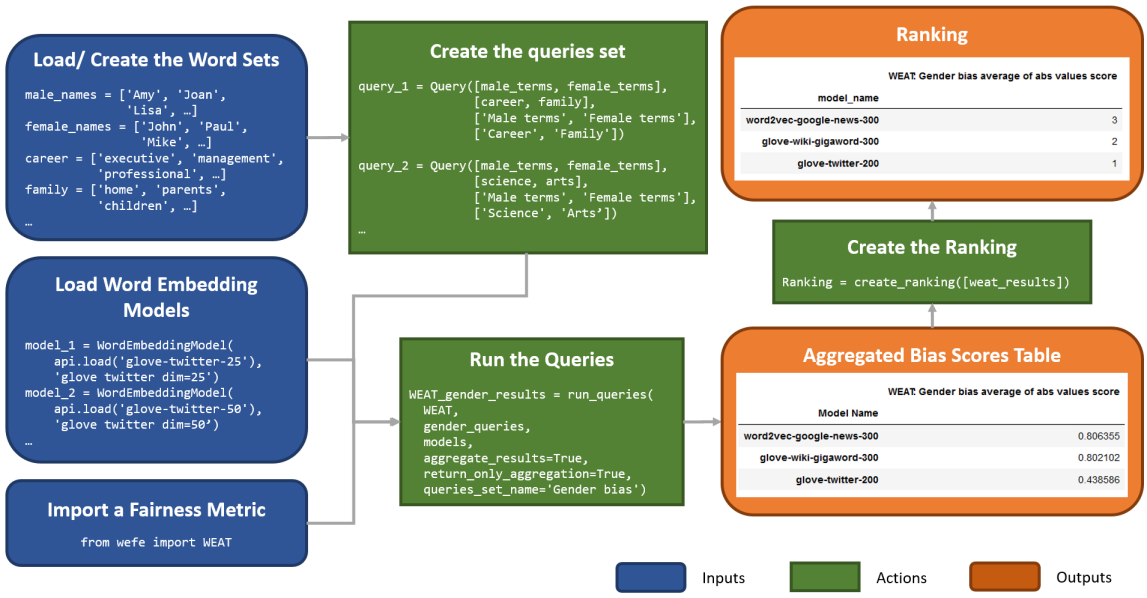

Figure 4.3: Gender bias ranking over various models of embeddings using WEAT.

The diagram contained in Figure 4.4 shows the common process for calculating correlations. The blue boxes show the process inputs. Each of these boxes represents the aggregation of scores obtained from `run_queries`. In each case, `run_queries` is executed with the same set of queries and embedding but varying the metrics. The green rectangles represent the actions of the process. First, a ranking is calculated from the inputs. Then, these rankings are used to calculate the correlations. Finally, the orange box is the output of the process. An example of a correlation matrix derived from the above process is shown in the figure.

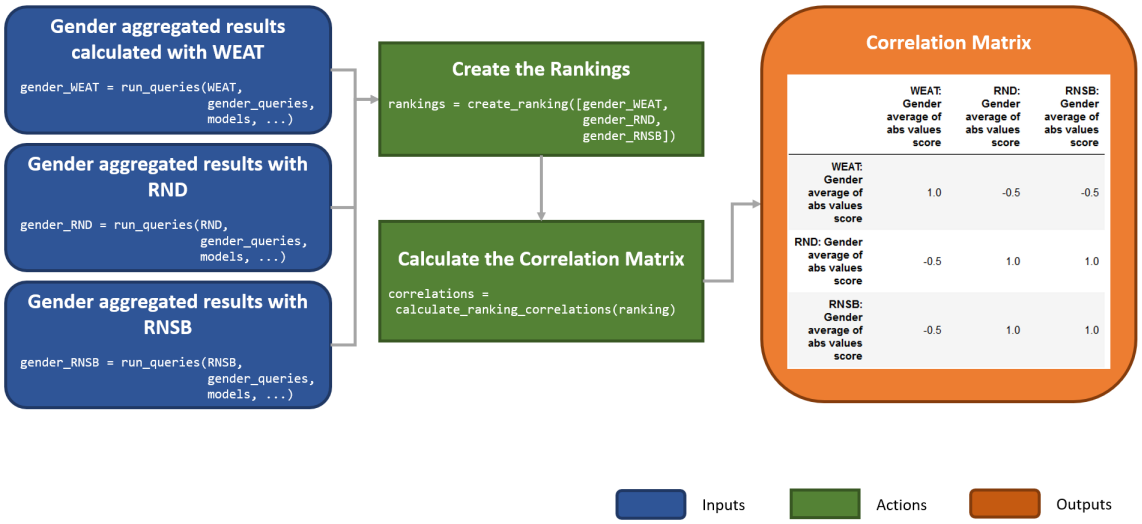

Figure 4.4: Calculation of correlations between rankings obtained from the evaluation of biases on different metrics.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis we presented the Word Embedding Fairness Evaluation framework (WEFE), a theoretical framework that formalizes the main building blocks for measuring bias in word embedding models. Using this framework, we conducted a case study where we evaluated and ranked different publicly available pre-trained word embedding models under different fairness metrics and bias criteria. We also implemented and released our framework as an open-source library along with instructions to replicate the case study previously mentioned.

Our framework allows for a clean comparison of the bias measurements by abstracting away several components such as target and attribute sets, queries, templates, fairness scores, and order relations among those scores.

When applying our framework to specific pre-trained embeddings using various fairness metrics, we were able to spot some differences among these metrics. In particular, we showed that the most widely used fairness metrics are not always correlated beyond the gender dimension. This gives evidence of the difficulty of measuring bias for according to other criteria such as religion or ethnicity, and reveals the necessity of more research in that direction.

In addition, we were able to observe that there is no direct correlation between the performance of the embeddings and the bias they contain. This tells us that while a model may be among the best according to its standard performance evaluation, it will not necessarily be the least biased. This is precisely because these metrics neglect the fairness component. For this reason, when implementing an embeddings-dependent NLP system, it is very important to consider both the performance and the bias of the chosen model. Otherwise, there is a risk of inheriting totally avoidable unfair behavior in these systems.

One important subjective aspect in our framework is the design of the queries (target and attributes) used to test bias. We followed the previous work closely when selecting the words composing every query, but this selection may definitely impact the rankings obtained. Although we were able to propose a small but well formulated set of queries, we left out many very important criteria that have not been considered until now, such as political views, social

class, gender discrimination against LGBT+ people, among many others. It also excludes several groups within the tested categories. Such is the case for ethnicity, which considers a very small subset of social groups with respect to the whole spectrum of social groups within our population.

Another limitation of existing metrics is that they are not able to identify the source of the bias. As suggested by the distributional hypothesis, these biases may come from the corpus on which the embeddings are trained. This is probably the reason why conceptnet [45], which is built from a knowledge graph apart from a corpus, produced better results. However, the bias may also come in part from the specific algorithm used to obtain the word embeddings or even from the hyperparameters used to train them. On the other hand, we do not know if the observed biases are mere statistical coincidence or are actually under the influence of the above-mentioned variables. Thus it is still not clear why some models are less biased than others. Further experimental studies are needed to analyze the origin of the bias in the embedding models considering parameters such as the corpus and the training algorithms.

Regarding the debiased model, we detected a decrease in gender bias in the case study scores. However, our results did not include the tests proposed by Gonen and Goldberg [20], which questioned the effectiveness of debias methods and metrics for measuring it. Therefore, although our results reflected a decrease in gender bias in the debiased model with respect to the original, we cannot conclude with certainty that the mitigation process is actually effective. It is also important to note that the study failed to clarify the effects of the bias mitigation methods on model performance and the effect of this process on the other social groups evaluated.

On another subject, the literature and our efforts to measure and mitigate the problems of bias in embeddings been focused only on the discrimination detected in the English speaking countries and its culture. As far as we know, there are a few studies that address this issue [52]. In the near future, this could become a big problem considering that AI is rapidly expanding all over the world. It is therefore necessary to promote fairness research in other languages in order to minimize the potential damage that these systems may cause.

We also have released WEFE as an open source library[1] for easily performing bias measurements in word embedding models along with tutorials to reproduce the experiments conducted in the case study as well as other previous studies.

5.2 Future Work

It is important to note that our study is limited to the fairness metrics and embeddings we used. This area of study has advanced after the selection of the works on which our framework is based. This is why in the future it is important to expand our study with more recent metrics, such as those proposed in [32, 49, 6], as well as broadening the set of proven embeddings.

[1] `https://wefe.readthedocs.io/en/latest/`

It is also necessary to broaden the criteria of bias assessed so far, as well as to assess whether the queries used in this and previous studies are valid and correct for assessing bias. This implies that in the future, it is of vital importance not only to create more and better queries, but also to design a standardized methodology to generate, validate or refute them.

Another point of interest is to address the origin of the bias in this type of models. As previously mentioned, it is not possible through our case study to identify the origin of the bias. The main impediment to carrying out an objective comparison is the fact that all the models we tested were trained using different corpora, training algorithms and their hyperparameters.

An alternative strategy is to perform the same case study with embeddings trained with a fixed corpus, but varying their training algorithm and hyper parameters. For example, one could use the Google news corpus to train several instances of glove, word2vec and fastext with different hyperparameters (dimension, window size, etc...) and then rank their biases using the same queries of the case study. On the other hand, to test the training corpus effect on model biases, the reverse process can be performed: train models with fixed training algorithms and hyperparameters, but vary the corpus and then rank them according to the detected bias.

Since each training procedure is a random process, each model may exhibit different biases, even under the same hyperparameters. For this reason, it is advisable to carry out these procedures several times and perform corresponding statistical tests on their results to provide greater robustness.

With regard to the debias methods: we still need to include more metrics and queries to verify the real impact of those methods over the models. We also know that there are currently more methods of debiasing and embedding proposed, so it is also a task for the future to include them [8, 11, 27, 28, 32, 50, 52]. On the other hand, models debiased according to criteria different from gender, such as ethnicity and religion, were not considered in our study. This presents and important research opportunity that we plan to explore.

We emphasize that it is imperative to investigate the bias of word embeddings in other languages apart from English, since these models are widely used across many languages. We identified a series of questions that will be very interesting for the development of this area. These are listed below:

- What happens to gender biases in grammatically gendered languages?
- What criteria of bias will be universal and which will be irrelevant when changing language?
- Are the biases purely cultural and/or language-related? To illustrate this question, we put the case of the ethnic bias against the Black population: Will the results be repeated in cultures where this type of bias is considerably less or non-existent?
- Are the queries used in the case study valid for other languages or will they have to be adapted and validated for each language and context?
- What influence does the training corpus have on the detected biases?

Finally, it is worth mentioning that although our software is capable of implementing our entire case study, it is still in a relatively initial version. To achieve the completeness we want, we need to integrate new metrics, implement new methods for loading word embeddings models, create new queries and integrate popular debias methods. Moreover, taking into account the popularity that contextualized word embedding models have gained [13], it would also be useful to study the feasibility of including the work on measuring bias in these models within our framework.

Bibliography

[1] Pablo Badilla and Felipe Bravo-Marquez. Word embedding fairness evaluation blog on kduggets.

[2] Pablo Badilla, Felipe Bravo-Marquez, and Jorge Pérez. Wefe: The word embeddings fairness evaluation framework. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence and the 17th Pacific Rim International Conference on Artificial Intelligence (IJCAI-PRICAI 2020)*, 2020.

[3] Amir Bakarov. A survey of word embeddings evaluation methods. *arXiv preprint arXiv:1801.09536*, 2018.

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[5] Tolga Bolukbasi, Kai-Wei Chang, James Y Zou, Venkatesh Saligrama, and Adam T Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In *Advances in neural information processing systems*, pages 4349–4357, 2016.

[6] Marc-Etienne Brunet, Colleen Alkalay-Houlihan, Ashton Anderson, and Richard Zemel. Understanding the origins of bias in word embeddings. In *International Conference on Machine Learning*, pages 803–811, 2019.

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

[8] Kaytlin Chaloner and Alfredo Maldonado. Measuring gender bias in word embeddings across domains and discovering new gender bias word categories. In *Proceedings of the First Workshop on Gender Bias in Natural Language Processing*, pages 25–32, 2019.

[9] Erin Collins. Punishing risk. *Geo. LJ*, 107:57, 2018.

[10] Amit Datta, Michael Carl Tschantz, and Anupam Datta. Automated experiments on ad privacy settings: A tale of opacity, choice, and discrimination. *Proceedings on privacy enhancing technologies*, 2015(1):92–112, 2015.

[11] Sunipa Dev and Jeff Phillips. Attenuating bias in word vectors. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 879–887, 2019.

[12] Jacob Eisenstein. *Introduction to natural language processing*. MIT press, 2019.

[13] Kawin Ethayarajh. Bert, elmo, gpt-2: How contextual are contextualized word representations?, Mar 2020.

[14] Kawin Ethayarajh, David Duvenaud, and Graeme Hirst. Towards understanding linear word analogies. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3253–3262, 2019.

[15] Kawin Ethayarajh, David Duvenaud, and Graeme Hirst. Understanding undesirable word embedding associations. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1696–1705, Florence, Italy, July 2019. Association for Computational Linguistics.

[16] Manaal Faruqui, Jesse Dodge, Sujay K Jauhar, Chris Dyer, Eduard Hovy, and Noah A Smith. Retrofitting word vectors to semantic lexicons. *arXiv preprint arXiv:1411.4166*, 2014.

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

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

[19] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.

[20] Hila Gonen and Yoav Goldberg. Lipstick on a pig: Debiasing methods cover up systematic gender biases in word embeddings but do not remove them. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 609–614, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

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

[23] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM, 2004.

[24] Stanislaw Jastrzkebski, Damian Lesniak, and Wojciech Marian Czarnecki. How to evaluate word embeddings? on importance of data efficiency and simple supervised tasks.

CoRR, abs/1702.02170, 2017.

[25] Mark Johnson. Introduction to computational linguistics and natural language processing (slides). 2014 Machine Learning Summer School, 2014.

[26] Surya Mattu Julia Angwin, Jeff Larson and Lauren Kirchner. *Machine Bias. There's software used across the country to predict future criminals. And it's biased against blacks.*, 2016 (accessed July 6, 2020).

[27] Masahiro Kaneko and Danushka Bollegala. Gender-preserving debiasing for pre-trained word embeddings. *CoRR*, abs/1906.00742, 2019.

[28] Saket Karve, Lyle Ungar, and João Sedoc. Conceptor debiasing of word representations evaluated on weat. *arXiv preprint arXiv:1906.05993*, 2019.

[29] Keita Kurita. Paper dissected: "glove: Global vectors for word representation" explained, May 2018.

[30] Sam Levin. *A beauty contest was judged by AI and the robots didn't like dark skin.*, 2016 (accessed July 6, 2020).

[31] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

[32] Thomas Manzini, Lim Yao Chong, Alan W Black, and Yulia Tsvetkov. Black is to criminal as caucasian is to police: Detecting and removing multiclass bias in word embeddings. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 615–621, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[33] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *arXiv preprint arXiv:1908.09635*, 2019.

[34] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[35] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[36] Eirini Ntoutsi, Pavlos Fafalios, Ujwal Gadiraju, Vasileios Iosifidis, Wolfgang Nejdl, Maria-Esther Vidal, Salvatore Ruggieri, Franco Turini, Symeon Papadopoulos, Emmanouil Krasanakis, et al. Bias in data-driven artificial intelligence systems—an introductory survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Dis-*

covery, 10(3):e1356, 2020.

[37] Jeffrey Pennington, Richard Socher, and Christopher 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.

[38] Rashida Richardson, Jason M Schultz, and Kate Crawford. Dirty data, bad predictions: How civil rights violations impact police data, predictive policing systems, and justice. *NYUL Rev. Online*, 94:15, 2019.

[39] Adam Rose. *Are Face-Detection Cameras Racist?*, 2010 (accessed July 6, 2020).

[40] Alexandre Salle, Marco Idiart, and Aline Villavicencio. Enhancing the lexvec distributed word representation model using positional contexts and external memory. *arXiv preprint arXiv:1606.01283*, 2016.

[41] Alexandre Salle, Aline Villavicencio, and Marco Idiart. Matrix factorization using window sampling and negative sampling for improved word representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 419–424, Berlin, Germany, August 2016. Association for Computational Linguistics.

[42] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 298–307, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[43] Shreya Shankar, Yoni Halpern, Eric Breck, James Atwood, Jimbo Wilson, and D. Sculley. No classification without representation: Assessing geodiversity issues in open data sets for the developing world, 2017.

[44] Robyn Speer. Conceptnet numberbatch 17.04: better, less-stereotyped word vectors. `http://blog.conceptnet.io/posts/2017/conceptnet-numberbatch-17-04-better-less-stereotyped-word-vectors/`, 2017 (accessed July 6, 2020).

[45] Robyn Speer, Joshua Chin, and Catherine Havasi. ConceptNet 5.5: An open multilingual graph of general knowledge. pages 4444–4451, 2017.

[46] Tony Sun, Andrew Gaut, Shirlyn Tang, Yuxin Huang, Mai ElSherief, Jieyu Zhao, Diba Mirza, Elizabeth Belding, Kai-Wei Chang, and William Yang Wang. Mitigating gender bias in natural language processing: Literature review. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1630–1640, Florence, Italy, July 2019. Association for Computational Linguistics.

[47] Harini Suresh and John V. Guttag. A framework for understanding unintended consequences of machine learning, 2019.

[48] Chris Sweeney and Maryam Najafian. A transparent framework for evaluating unin-

tended demographic bias in word embeddings. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1662–1667, 2019.

[49] Nathaniel Swinger, Maria De-Arteaga, Neil Thomas Heffernan IV, Mark DM Leiserson, and Adam Tauman Kalai. What are the biases in my word embedding? In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 305–311, 2019.

[50] Jieyu Zhao, Yichao Zhou, Zeyu Li, Wei Wang, and Kai-Wei Chang. Learning gender-neutral word embeddings. *arXiv preprint arXiv:1809.01496*, 2018.

[51] Kai Zhao, Hany Hassan, and Michael Auli. Learning translation models from monolingual continuous representations. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1527–1536, 2015.

[52] Pei Zhou, Weijia Shi, Jieyu Zhao, Kuan-Hao Huang, Muhao Chen, Ryan Cotterell, and Kai-Wei Chang. Examining gender bias in languages with grammatical gender. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5275–5283, Hong Kong, China, November 2019. Association for Computational Linguistics.

Appendix A

Word Sets and Queries

The following appendix lists all the word sets used for the construction of the queries in our case study.

The word sets were collected from different previous works. To maintain order, the word sets are grouped according to their source in different subsections. Each subsection contains the source work of the word sets and a table detailing the name of the sets and the words belonging to them. Note that the word sets shown below were collected from previous works and in no way represent the opinions or thoughts of the author.

A.1 WEAT Word Sets

The source of the following word sets is the work published by Caliskan et al. 2017 [7]. Most of these word sets come from psychology studies. The detail of each of these sources is documented in their work.

Table A.1: WEAT word sets

Set name	Words
Pleasant 5	caress, freedom, health, love, peace, cheer, friend, heaven, loyal, pleasure, diamond, gentle, honest, lucky, rainbow, diploma, gift, honor, miracle, sunrise, family, happy, laughter, paradise, vacation.
Unpleasant 5	abuse, crash, filth, murder, sickness, accident, death, grief, poison, stink, assault, disaster, hatred, pollute, tragedy, divorce, jail, poverty, ugly, cancer, kill, rotten, vomit, agony, prison.
European American names 5	Adam, Harry, Josh, Roger, Alan, Frank, Justin, Ryan, Andrew, Jack, Matthew, Stephen, Brad, Greg, Paul, Jonathan, Peter, Amanda, Courtney, Heather, Melanie, Sara, Amber, Katie, Betsy, Kristin, Nancy, Stephanie, Ellen, Lauren, Colleen, Emily, Megan, Rachel.

Continued on next page

Table A.1: WEAT word sets

Set name	Words
African American names 5	Alonzo, Jamel, Theo, Alphonse, Jerome, Leroy, Torrance, Darnell, Lamar, Lionel, Tyree, Deion, Lamont, Malik, Terrence, Tyrone, Lavon, Marcellus, Wardell, Nichelle, Shereen, Ebony, Latisha, Shaniqua, Jasmine, Tanisha, Tia, Lakisha, Latoya, Yolanda, Malika, Yvette.
European American names 7	Brad, Brendan, Geoffrey, Greg, Brett, Jay, Matthew, Neil, Todd, Allison, Anne, Carrie, Emily, Jill, Laurie, Kristen, Meredith, Sarah.
African American names 7	Darnell, Hakim, Jermaine, Kareem, Jamal, Leroy, Rasheed, Tremayne, Tyrone, Aisha, Ebony, Keisha, Kenya, Latonya, Lakisha, Latoya, Tamika, Tanisha.
Pleasant 9	joy, love, peace, wonderful, pleasure, friend, laughter, happy.
Unpleasant 9	agony, terrible, horrible, nasty, evil, war, awful, failure.
Male names	John, Paul, Mike, Kevin, Steve, Greg, Jeff, Bill.
Female names	Amy, Joan, Lisa, Sarah, Diana, Kate, Ann, Donna.
Career	executive, management, professional, corporation, salary, office, business, career.
Family	home, parents, children, family, cousins, marriage, wedding, relatives.
Math	math, algebra, geometry, calculus, equations, computation, numbers, addition.
Arts	poetry, art, dance, literature, novel, symphony, drama, sculpture.
Male terms	male, man, boy, brother, he, him, his, son.
Female terms	female, woman, girl, sister, she, her, hers, daughter.
Science	science, technology, physics, chemistry, Einstein, NASA, experiment, astronomy.
Arts 2	poetry, art, Shakespeare, dance, literature, novel, symphony, drama.
Male terms 2	brother, father, uncle, grandfather, son, he, his, him.
Female terms 2	sister, mother, aunt, grandmother, daughter, she, hers, her.

A.2 RND Word Sets

The source of the following word sets is the work published by Garg et al. 2018 [18]. The sets were collected from a repository published by their authors [1].

Table A.2: RND word sets

Set name	Words
Adjectives appearance	alluring, voluptuous, blushing, homely, plump, sensual, gorgeous, slim, bald, athletic, fashionable, stout, ugly, muscular, slender, feeble, handsome, healthy, attractive, fat, weak, thin, pretty, beautiful, strong.

Continued on next page

[1] https://github.com/nikhgarg/EmbeddingDynamicStereotypes

Table A.2: RND word sets

Set name	Words
Adjectives otherization	devious, bizarre, venomous, erratic, barbaric, frightening, deceitful, forceful, deceptive, envious, greedy, hateful, contemptible, brutal, monstrous, calculating, cruel, intolerant, aggressive, monstrous.
Adjectives sensitive	inhibited, complacent, sensitive, mellow, solemn, studious, intelligent, brilliant, rational, serious, contemplative, cowardly, timid, shy, passive, delicate, gentle, soft, quiet, working.
Asian surnames	cho, wong, tang, huang, chu, chung, ng, wu, liu, chen, lin, yang, kim, chang, shah, wang, li, khan, singh, hong.
Black surnames	harris, robinson, howard, thompson, moore, wright, anderson, clark, jackson, taylor, scott, davis, allen, adams, lewis, williams, jones, wilson, martin, johnson.
Chinese surnames	chung, liu, wong, huang, ng, hu, chu, chen, lin, liang, wang, wu, yang, tang, chang, hong, li.
Hispanic surnames	ruiz, alvarez, vargas, castillo, gomez, soto, gonzalez, sanchez, rivera, mendoza, martinez, torres, rodriguez, perez, lopez, medina, diaz, garcia, castro, cruz.
Russian surnames	gurin, minsky, sokolov, markov, maslow, novikoff, mishkin, smirnov, orloff, ivanov, sokoloff, davidoff, savin, romanoff, babinski, sorokin, levin, pavlov, rodin, agin.
White surnames	harris, nelson, robinson, thompson, moore, wright, anderson, clark, jackson, taylor, scott, davis, allen, adams, lewis, williams, jones, wilson, martin, johnson.
Christianity related words	baptism, messiah, catholicism, resurrection, christianity, salvation, protestant, gospel, trinity, jesus, christ, christian, cross, catholic, church.
Islam related words	allah, ramadan, turban, emir, salaam, sunni, koran, imam, sultan, prophet, veil, ayatollah, shiite, mosque, islam, sheik, muslim, muhammad.
Terrorism related words	terror, terrorism, violence, attack, death, military, war, radical, injuries, bomb, target, conflict, dangerous, kill, murder, strike, dead, violence, fight, death, force, stronghold, wreckage, aggression, slaughter, execute, overthrow, casualties, massacre, retaliation, proliferation, militia, hostility, debris, acid, execution, militant, rocket, guerrilla, sacrifice, enemy, soldier, terrorist, missile, hostile, revolution, resistance, shoot.
Male occupations	physician, doctor, laborer, conservationist, proprietor, operator, mechanic, surveyor, physicist, machinist, architect, photographer, optometrist, millwright, tradesperson, sales, upholsterer, smith, manager, statistician, doorkeeper, athlete, bailiff, typesetter, clerk, boilermaker, cabinetmaker, official, conductor, porter, bookbinder, chemist, inspector, professor, salesperson, lawyer, farmer, electrician, sailor, mailperson, geologist, setter, lumberjack, instructor, plasterer, judge, toolmaker, sheriff, surgeon, scientist, jeweler, compositor, engineer, carpenter, artist, soldier, clergy, painter, shoemaker, plumber, mason, cook, draftsperson, pilot, chiropractor, dentist, fireperson, police, gardener, driver, guard, welder.

Continued on next page

Table A.2: RND word sets

Set name	Words
Female occupations	bankteller, cashier, bartender, teacher, baker, dancer, nutritionist, broker, dietitian, author, entertainer, economist, nurse, secretary, clerical, therapist, technician, veterinarian, attendant, janitor, weaver, musician, waitstaff, psychologist, designer, pharmacist, librarian, accountant, newsperson, administrator, housekeeper.
Common White occupations	plasterer, janitor, porter, cook, shoemaker, laborer, guard, doorkeeper, baker, gardener, cashier, attendant, clerk, mason, upholsterer.
Common Black occupations	farmer, veterinarian, pilot, optometrist, physicist, dentist, chiropractor, geologist, statistician, plasterer, surveyor, author, architect, setter, toolmaker.
Common Asian occupations	plasterer, conductor, boilermaker, millwright, mason, fireperson, conservationist, setter, toolmaker, plumber, upholsterer, farmer, bookbinder, cabinetmaker, carpenter.
Common Hispanic occupations	optometrist, veterinarian, physicist, geologist, pharmacist, chiropractor, statistician, millwright, toolmaker, setter, author, scientist, dentist, lawyer, judge.
Male terms	he, son, his, him, father, man, boy, himself, male, brother, sons, fathers, men, boys, males, brothers, uncle, uncles, nephew, nephews.
Female terms	she, daughter, hers, her, mother, woman, girl, herself, female, sister, daughters, mothers, women, girls, females, sisters, aunt, aunts, niece, nieces.
Adjectives intelligence	precocious, resourceful, inquisitive, sagacious, inventive, astute, adaptable, reflective, discerning, intuitive, inquiring, judicious, analytical, luminous, venerable, imaginative, shrewd, thoughtful, sage, smart, ingenious, clever, brilliant, logical, intelligent, apt, genius, wise.

A.3 Debias Word Embeddings Word Sets

Table A.3: Debias Word Embeddings Word sets

Set name	Words
Male terms	woman, girl, she, mother, daughter, gal, female, her, herself, Mary.
Female terms	man, boy, he, father, son, guy, male, his, himself, John.
Male related words	monastery, spokesman, catholic_priest, dad, men, councilman, grandpa, grandsons, prostate_cancer, testosterone, uncle, wives, father, grandpa, he, boy, boys, brother, brothers, businessman, chairman, colt, congressman, dad, dads, dudes, ex_girlfriend, father, fatherhood, fathers, fella, fraternity, gelding, gentleman, gentlemen, grandfather, grandson, he, himself, his, king, kings, male, males, man, men, nephew, prince, schoolboy, son, sons, twin_brother.

Continued on next page

Table A.3: Debias Word Embeddings Word sets

Set name	Words
Female related words	convent, spokeswoman, nun, mom, women, councilwoman, grandma, granddaughters, ovarian_cancer, estrogen, aunt, husbands, mother, grandma, she, girl, girls, sister, sisters, businesswoman, chairwoman, filly, congresswoman, mom, moms, gals, ex_boyfriend, mother, motherhood, mothers, granny, sorority, mare, lady, ladies, grandmother, granddaughter, she, herself, her, queen, queens, female, females, woman, women, niece, princess, schoolgirl, daughter, daughters, twin_sister.

A.4 Debias Multiclass Words Sets

Table A.4: Debias multiclass word sets

Set name	Words
Male terms	he, his, son, father, male, boy, uncle.
Female terms	she, hers, daughter, mother, female, girl, aunt.
Male roles	manager, executive, doctor, lawyer, programmer, scientist, soldier, supervisor, rancher, janitor, firefighter, officer.
Female roles	secretary, nurse, clerk, artist, homemaker, dancer, singer, librarian, maid, hairdresser, stylist, receptionist, counselor.
Black related terms	Africa, African, black.
White related terms	America, Caucasian, Europe, white.
Asian related terms	Asia, Asian, china.
Black related words	slave, musician, runner, criminal, homeless.
White related words	manager, executive, redneck, hillbilly, leader, farmer.
Asian related words	doctor, engineer, laborer, teacher.
Judaism related terms	jew, Judaism, rabbi, synagogue, Torah.
Christianity terms	bible, christian, Christianity, church, priest.
Islam terms	imam, Islam, mosque, Muslim, Quran.
Greed terms	greedy, cheap, hairy, liberal.
Conservative terms	judgmental, conservative, familial.
Terrorism terms	violent, terrorist, dirty, uneducated.

A.5 Bing Liu Sentiment Lexicon

The complete sentiment lexicon produced by bing-liu [23] contains around 6800 words between the positive and negative word sets. For the sake of space we only give some examples of this one. The complete sets can be found at the footnote.[2]

Table A.5: Bing Liu sentiment lexicon examples

Set name	Words
Positive words	a+, abound, abounds, abundance, abundant, accessable, accessible, acclaim, acclaimed, acclamation, accolade, accolades, accommodative, accomodative, accomplish, accomplished, accomplishment, accomplishments, accurate, accurately, achievable, achievement, achievements, achievible, acumen, adaptable, adaptive, adequate, adjustable, admirable, admirably, admiration, admire, admirer, admiring, admiringly, adorable, adore, adored, adorer, adoring, adoringly, adroit, adroitly, adulate, adulation, adulatory, advanced, advantage, advantageous, advantageously, advantages, adventuresome, adventurous, advocate, advocated, advocates, affability, affable, affably, affectation, affection, affectionate, affinity, affirm, affirmation, affirmative, affluence, affluent, afford, affordable, affordably, afordable, agile, agilely, agility, agreeable, agreeableness, agreeably, all-around, alluring, alluringly, altruistic, altruistically, amaze, amazed, amazement, amazes, amazing, amazingly, ambitious, ...
Negative words	2-faced, 2-faces, abnormal, abolish, abominable, abominably, abominate, abomination, abort, aborted, aborts, abrade, abrasive, abrupt, abruptly, abscond, absence, absent-minded, absentee, absurd, absurdity, absurdly, absurdness, abuse, abused, abuses, abusive, abysmal, abysmally, abyss, accidental, accost, accursed, accusation, accusations, accuse, accuses, accusing, accusingly, acerbate, acerbic, acerbically, ache, ached, aches, achey, aching, acrid, acridly, acridness, acrimonious, acrimoniously, acrimony, adamant, adamantly, addict, addicted, addicting, addicts, admonish, admonisher, admonishingly, admonishment, admonition, adulterate, adulterated, adulteration, adulterier, adversarial, adversary, adverse, adversity, afflict, affliction, afflictive, affront, afraid, aggravate, aggravating, aggravation, aggression, aggressive, aggressiveness, aggressor, aggrieve, aggrieved, aggrivation, aghast, agonies, agonize, agonizing, agonizingly, agony, aground, ail, ailing, ailment, aimles, ...

[2] http://www.cs.uic.edu/~liub/FBS/opinion-lexicon-English.rar

Appendix B

Queries

The following tables provides details of the queries sets used in our case study. The sets of queries are arranged in three different tables. The first table (Table B.1) contains the details of the queries used to study gender bias, the second table (Table B.2) contains the queries that study ethnicity bias and the third table (Table B.3) contains the queries that study religion bias.

All queries were built using a template size (2.2). Through the mechanisms of sub-series generation (Section 3.1.4), the original queries were adapted to work with metrics compatible with smaller sizes.

Since there are duplicate names between sets of words, we detail their source between parentheses. *WEAT* refers to the sets in the table A.1, *RND* to the sets in the table A.2, *Debias multiclass* refers to the sets in the table A.4 and *Bing Liu* refers to the sets in the table A.5.

B.1 Gender Queries

Table B.1: Gender queries

Target set 1	Target set 2	Attribute set 1	Attribute set 2
Male terms (RND)	Female terms (RND)	Career (WEAT)	Family (WEAT)
Male terms (RND)	Female terms (RND)	Math (WEAT)	Arts (WEAT)
Male terms (RND)	Female terms (RND)	Science (WEAT)	Arts (WEAT)
Male terms (RND)	Female terms (RND)	Intelligence (RND)	Appearance (RND)
Male terms (RND)	Female terms (RND)	Intelligence (RND)	Sensitive (RND)
Male terms (RND)	Female terms (RND)	Positive words (Bing Liu)	Negative words (Bing Liu)
Male terms (RND)	Female terms (RND)	Man Roles (Debias multiclass)	Woman Roles (Debias multiclass)

B.2 Ethnicity Queries

Table B.2: Ethnicity queries

Target set 1	Target set 2	Attribute set 1	Attribute set 2
White last names (RND)	Black last names (RND)	Pleasant 5 (WEAT)	Unpleasant 5 (WEAT)
White last names (RND)	Asian last names (RND)	Pleasant 5 (WEAT)	Unpleasant 5 (WEAT)
White last names (RND)	Hispanic last names (RND)	Pleasant 5 (WEAT)	Unpleasant 5 (WEAT)
White last names (RND)	Black last names (RND)	Occupations white (RND)	Occupations black (RND)
White last names (RND)	Asian last names (RND)	Occupations white (RND)	Occupations Asian (RND)
White last names (RND)	Hispanic last names (RND)	Occupations white (RND)	Occupations Hispanic (RND)
White last names (RND)	Black last names (RND)	Positive words(Bing Liu)	Negative words (Bing Liu)
White last names (RND)	Asian last names (RND)	Positive words (Bing Liu)	Negative words (Bing Liu)
White last names (RND)	Hispanic last names (RND)	Positive words (Bing Liu)	Negative words (Bing Liu)

B.3 Religion Queries

Table B.3: Religion queries

Target set 1	Target set 2	Attribute set 1	Attribute set 2
Christianity terms (Debias multiclass)	Islam terms (Debias multiclass)	Pleasant 5 (WEAT)	Unpleasant 5 (WEAT)
Christianity terms (Debias multiclass)	Judaism terms (Debias multiclass)	Pleasant 5 (WEAT)	Unpleasant 5 (WEAT)
Islam terms (Debias multiclass)	Judaism terms (Debias multiclass)	Pleasant 5 (WEAT)	Unpleasant 5 (WEAT)
Christianity terms (Debias multiclass)	Islam terms (Debias multiclass)	Conservative (Debias multiclass)	Terrorism (Debias multiclass)
Christianity terms (Debias multiclass)	Jew terms (Debias multiclass)	Conservative (Debias multiclass)	Greed (Debias multiclass)
Islam terms (Debias multiclass)	Jew terms (Debias multiclass)	Terrorism (Debias multiclass)	Greed (Debias multiclass)
Christianity terms (Debias multiclass)	Islam terms (Debias multiclass)	Positive words (Bing Liu)	Negative words (Bing Liu)
Christianity terms (Debias multiclass)	Jew terms (Debias multiclass)	Positive words (Bing Liu)	Negative words (Bing Liu)
Islam terms (Debias multiclass)	Jew terms (Debias multiclass)	Positive words (Bing Liu)	Negative words (Bing Liu)

Appendix C

WEFE Library Tutorial

The following pages present a comprehensive user guide on how to use the WEFE library presented in Chapter 4. Below:

- We introduce how to run a simple query using some embedding model.
- We present how to run multiple queries on multiple embeddings.
- We show how to compare the results obtained from running multiple sets of queries on multiple embeddings using different metrics through ranking calculation.
- We describe how to calculate the correlations between the rankings obtained.

Please note that the ongoing development of this library may cause the following to be deprecated in the future. To see the documentation of the most recent deploy of the library, visit the link contained in the following footnote.[1]

C.1 Run a Query

The following code explains how to run a gender query using Glove embeddings and the Word Embedding Association Test (WEAT) as the fairness metric. Below we present the three usual steps for performing a query in WEFE:

- Load a word embeddings model as a `WordEmbeddingModel` object.

  First, we load the word embedding pretrained model using the gensim library and then we create a `WordEmbeddingModel` instance. For this example, we will use a 25-dimensional Glove embedding model trained from a Twitter dataset.

```
# Load the modules
from wefe.query import Query
from wefe.word_embedding_model import WordEmbeddingModel
from wefe.metrics.WEAT import WEAT
from wefe.datasets.datasets import load_weat
import gensim.downloader as api
```

[1] https://wefe.readthedocs.io/en/latest/

```
twitter_25 = api.load('glove-twitter-25')
model = WordEmbeddingModel(twitter_25, 'glove twitter dim=25')
```

- Create the query using a `Query` object

  Define the target and attribute words sets and create a `Query` object that contains them. Some well-known word sets are already provided by the package and can be easily loaded by the user. Users can also set their own custom-made sets. For this example, we will create a query with gender terms with respect to family and career. The words we will use will be taken from the set of words used in the WEAT paper [7] (included in the package).

  ```
  # load the weat word sets
  word_sets = load_weat()
  query = Query([word_sets['male_terms'], word_sets['female_terms']],
                [word_sets['career'], word_sets['family']],
                ['Male terms', 'Female terms'], ['Career', 'Family'])
  ```

- Instantiate the Metric

  Instantiate the metric that you will use and then execute `run_query` with the parameters created in the previous steps. In this case we will use the `WEAT` metric.

  ```
  weat = WEAT()
  result = weat.run_query(query, model)
  print(result)
  ```

  The results obtained are as follows:

  ```
  {'query_name': 'Male Terms and Female Terms wrt Arts and Science',
   'result': -0.010003209}
  ```

C.2 Running Multiple Queries

We usually want to test several queries that study some criterion of bias: *gender, ethnicity, religion, politics, socioeconomic, among others*. Let us suppose you have created 20 queries that study gender bias on different models of embeddings. Trying to `run_query` on each pair embedding-query can be a bit complex and will require extra work to implement.

This is why the library also implements a function to test multiple queries on various word embedding models in a single call: the `run_queries` util.

The following code shows how to run various gender queries on different Glove embedding models trained from the Twitter dataset. The queries will be executed using the Effect size variant of WEAT.

1. Load the models:

   Load three different Glove Twitter embedding models. These models were trained using the same dataset varying the number of embedding dimensions.

   ```
   from wefe.query import Query
   from wefe.datasets import load_weat
   from wefe.word_embedding_model import WordEmbeddingModel
   ```

```
from wefe.metrics import WEAT, RNSB
from wefe.utils import run_queries, plot_queries_results

import gensim.downloader as api
```

2. Load the word sets:

   Now, we will load the WEAT word set and create three queries. The first query is intended to measure gender bias and the other two are intended to measure ethnicity bias.

```
# Load the WEAT word sets
word_sets = load_weat()

# Create gender queries
gender_query_1 = Query([word_sets['male_terms'], word_sets['female_terms']],
                       [word_sets['career'], word_sets['family']],
                       ['Male terms', 'Female terms'], ['Career', 'Family'])
gender_query_2 = Query([word_sets['male_terms'], word_sets['female_terms']],
                       [word_sets['science'], word_sets['arts']],
                       ['Male terms', 'Female terms'], ['Science', 'Arts'])
gender_query_3 = Query([word_sets['male_terms'], word_sets['female_terms']],
                       [word_sets['math'], word_sets['arts_2']],
                       ['Male terms', 'Female terms'], ['Math', 'Arts'])

gender_queries = [gender_query_1, gender_query_2, gender_query_3]
```

3. Run the queries on all Word Embeddings using WEAT Effect Size.

   Now, to run our list of queries and models, we call the function `run_queries`. The mandatory parameters of the function are 3:

   (a) a metric,

   (b) a list of queries, and,

   (c) a list of embedding models.

   It is also possible to provide a name for the criterion studied in this set of queries through the parameter `queries_set_name`.

   Notice that one can pass the metric's parameters using a dict object in the `metric_params` parameter. In this case, we specify that `WEAT` returns its Effect size variant as a result by delivering the following parameter to

```
run_queries: metric_params{'return_effect_size': True}
```

```
# Run the queries
WEAT_gender_results = run_queries(WEAT,
                                  gender_queries,
                                  models,
                                  metric_params={
                                      'return_effect_size': True
                                  },
                                  queries_set_name='Gender Queries')
WEAT_gender_results
```

Model name	Male terms and Female terms wrt. Career and Family	Male terms and Female terms wrt. Science and Arts	Male terms and Female terms wrt. Math and Arts
glove twitter dim=25	0.715369	0.766402	0.121468
glove twitter dim=50	0.799666	-0.660553	-0.589894
glove twitter dim=100	0.681933	0.641153	-0.399822

Table C.1: Results of the execution of gender queries evaluated on three models of embeddings using WEAT..

The results of this execution can be found in the table C.1. If more than 20% (by default) of the words from any of the word sets of the query are not included in the word embedding model, the metric will return `NaN`. This behavior can be changed using a float number parameter called `lost_vocabulary_threshold`.

4. Plot the results in a barplot: The following code shows how to graph the above results using the utility `plot_queries_results`. The graph is displayed in the figure C.1

```
plot_queries_results(WEAT_gender_results).show()
```

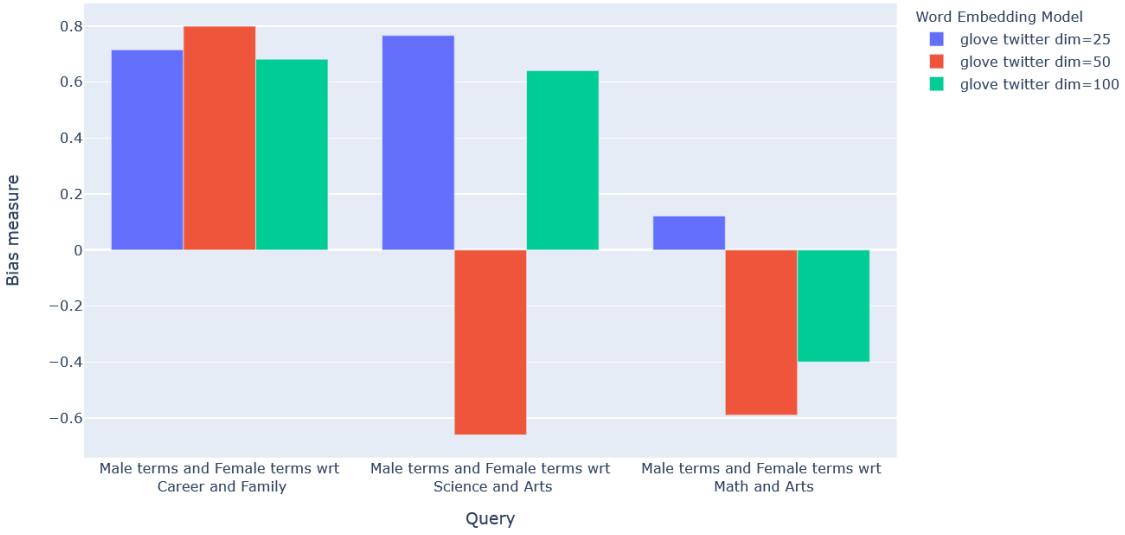

Figure C.1: Graphed results of the execution of gender queries evaluated on three models of embeddings using WEAT.

5. Aggregating Results:

   The execution of `run_queries` in the previous step gave us many results evaluating the gender bias in the tested embeddings. However, these do not tell us much about the overall fairness of the embedding models with respect to the criteria evaluated. Therefore, we would like to have some mechanism that allows us to aggregate the results directly obtained in `run_query` so that we can evaluate the bias as a whole.

   For this, when using `run_queries`, one must set the `aggregate_results` parameter as `True`. This default value will activate the option to aggregate the results by averaging

Model name	Male terms and Female terms wrt. Career and Family	Male terms and Female terms wrt. Science and Arts	Male terms and Female terms wrt. Math and Arts	WEAT: Gender Queries average of abs values score
glove twitter dim=25	0.715369	0.766402	0.121468	0.534413
glove twitter dim=50	0.799666	-0.660553	-0.589894	0.683371
glove twitter dim=100	0.681933	0.641153	-0.399822	0.574303

Table C.2: Aggregated results of the execution of gender queries evaluated on three embedding models using WEAT.

the absolute values of the results and put them in the last column.

This aggregation function can be modified through the `aggregation_function` parameter. Here one can specify a string that defines some of the aggregation types that are already implemented, as well as provide a function that operates in the results dataframe.

The aggregation functions available are (the parameter is listed in monospaced font):

- Average: `avg`.
- Average of the absolute values: `abs_avg`.
- Sum: `sum`.
- Sum of the absolute values: `abs_sum`.

Notice that some functions are more appropriate for certain metrics. For metrics returning only positive numbers, all the previous aggregation functions would be okay. In contrast, for metrics returning real values (e.g., `WEAT`, `RND`), aggregation functions such as sum would make different outputs cancel each other out.

Let us aggregate the results from previous example by the average of the absolute values:

```
WEAT_gender_results_agg = run_queries(WEAT,
                                      gender_queries,
                                      models,
                                      metric_params={'return_effect_size': True},
                                      aggregate_results=True,
                                      aggregation_function='abs_avg',
                                      queries_set_name='Gender Queries')
WEAT_gender_results_agg
```

Finally, we can ask the function to return only the aggregated values (through `return_only_aggregation` parameter) and then plot them.

```
WEAT_gender_results_agg = run_queries(WEAT,
                                      gender_queries,
                                      models,
                                      metric_params={'return_effect_size': True},
                                      aggregate_results=True,
                                      aggregation_function='abs_avg',
                                      return_only_aggregation=True,
                                      queries_set_name='Gender Queries')
```

```
WEAT_gender_results_agg
plot_queries_results(WEAT_gender_results_agg).show()
```

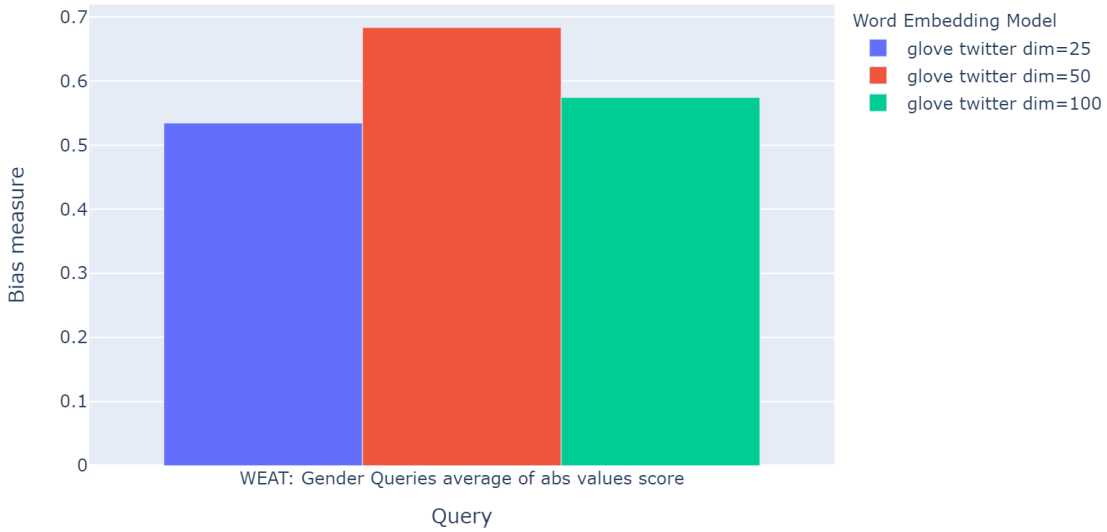

Figure C.2: Graph of the aggregated results of the execution of gender queries evaluated on three models of embeddings using WEAT.

C.3 Rankings

When we want to measure various criteria of bias in different embedding models, two major problems arise:

1. One type of bias can dominate the other because of significant differences in magnitude.
2. Different metrics can operate on different scales, which makes them difficult to compare.

To show that, suppose we have two sets of queries: one that explores gender biases and another that explores ethnicity biases, and we want to test these sets of queries on 3 Twitter Glove models of 25, 50 and 100 dimensions each, using both WEAT and Relative Negative Sentiment Bias (RNSB) as bias metrics.

C.3.1 Differences in Magnitude Using the Same Fairness Metric

The first problem that can occur is that the bias scores obtained from one set of queries are much higher than those from the other set, even when the same metric is used. To prove this, we execute the gender and ethnicity queries using WEAT and the 3 models mentioned above. As can be seen in Table C.3, the results of ethnicity bias are much greater than those of gender.

Model Name	WEAT: Gender Queries average of abs values score	WEAT: Ethnicity Queries average of abs values score
glove twitter dim=25	0.210556	2.64632
glove twitter dim=50	0.292373	1.87431
glove twitter dim=100	0.225116	1.78469

Table C.3: Gender and ethnicity aggregated WEAT results comparison.

Model Name	WEAT: Gender Queries average of abs values score	RNSB: Gender Queries average of abs values score
glove twitter dim=25	0.210556	0.032673
glove twitter dim=50	0.292373	0.049429
glove twitter dim=100	0.225116	0.0312772

Table C.4: WEAT and RNSB gender aggregated results comparison.

C.3.2 Differences in Magnitude Using Different Fairness Metrics

The second problem that can occur is that one metric delivers results with different orders of magnitude to another, even though both measure the same queries. To test this, we execute the gender queries using WEAT and RNSB metrics and the 3 models mentioned above. As we can see in the Table C.4, differences between the results of both metrics of an order of magnitude.

C.3.3 Calculate Rankings

Rankings allow us to focus on the relative differences reported by the metrics (for different models) instead of focusing on the absolute values. The following code loads the models and create the queries:

```
from wefe.query import Query
from wefe.datasets.datasets import load_weat
from wefe.word_embedding_model import WordEmbeddingModel
from wefe.metrics import WEAT, RNSB
from wefe.utils import run_queries, create_ranking,
      plot_ranking, plot_ranking_correlations

import gensim.downloader as api

# Load the models
model_1 = WordEmbeddingModel(api.load('glove-twitter-25'),
                             'glove twitter dim=25')
```

```
model_2 = WordEmbeddingModel(api.load('glove-twitter-50'),
                             'glove twitter dim=50')
model_3 = WordEmbeddingModel(api.load('glove-twitter-100'),
                             'glove twitter dim=100')

models = [model_1, model_2, model_3]


# Load the WEAT word sets
word_sets = load_weat()

# Create gender queries
gender_query_1 = Query([word_sets['male_terms'], word_sets['female_terms']],
                       [word_sets['career'], word_sets['family']],
                       ['Male terms', 'Female terms'], ['Career', 'Family'])
gender_query_2 = Query([word_sets['male_terms'], word_sets['female_terms']],
                       [word_sets['science'], word_sets['arts']],
                       ['Male terms', 'Female terms'], ['Science', 'Arts'])
gender_query_3 = Query([word_sets['male_terms'], word_sets['female_terms']],
                       [word_sets['math'], word_sets['arts_2']],
                       ['Male terms', 'Female terms'], ['Math', 'Arts'])

# Create ethnicity queries
ethnicity_query_1 = Query([word_sets['european_american_names_5'],
                           word_sets['african_american_names_5']],
                          [word_sets['pleasant_5'], word_sets['unpleasant_5']],
                          ['European Names', 'African Names'],
                          ['Pleasant', 'Unpleasant'])

ethnicity_query_2 = Query([word_sets['european_american_names_7'],
                           word_sets['african_american_names_7']],
                          [word_sets['pleasant_9'], word_sets['unpleasant_9']],
                          ['European Names', 'African Names'],
                          ['Pleasant 2', 'Unpleasant 2'])

gender_queries = [gender_query_1, gender_query_2, gender_query_3]
ethnicity_queries = [ethnicity_query_1, ethnicity_query_2]
```

Now, we run the queries with `WEAT` and `RNSB`: First, run the WEAT queries:

```
WEAT_gender_results = run_queries(WEAT,
                                  gender_queries,
                                  models,
                                  aggregate_results=True,
                                  return_only_aggregation=True,

                                  queries_set_name='Gender Queries')
```

Model Name	WEAT: Gender Queries average of abs values	WEAT: Ethnicity Queries average of abs values	RNSB: Gender Queries average of abs values	RNSB: Ethnicity Queries average of abs values
glove twitter dim=25	1	3	3	3
glove twitter dim=50	3	2	2	1
glove twitter dim=100	2	1	1	2

Table C.5: WEAT and RNSB rankings calculated from the aggregated results.

```
WEAT_ethnicity_results = run_queries(WEAT,
                                     ethnicity_queries,
                                     models,
                                     aggregate_results=True,
                                     return_only_aggregation=True,
                                     queries_set_name='Ethnicity Queries')
```

Then, run the queries using RNSB:

```
RNSB_gender_results = run_queries(RNSB,
                                  gender_queries,
                                  models,
                                  aggregate_results=True,
                                  return_only_aggregation=True,
                                  queries_set_name='Gender Queries')

RNSB_ethnicity_results = run_queries(RNSB,
                                     ethnicity_queries,
                                     models,
                                     aggregate_results=True,
                                     return_only_aggregation=True,
                                     queries_set_name='Ethnicity Queries')
```

To create the ranking we use the `create_ranking` function. This function takes all the `DataFrames` containing the calculated scores and uses the last column to create the ranking. It assumes that the scores are already aggregated.

```
ranking = create_ranking([
    WEAT_gender_results, WEAT_ethnicity_results, RNSB_gender_results,
    RNSB_ethnicity_results
])
print(ranking)
```

Finally, we plot the rankings using the `plot_ranking` function. The function can be used in two ways:

- With facet by Metric and Criteria. This figure shows the rankings separated by each bias criteria and metric (i.e, by each column). Each bar represents the position of the

embedding in the corresponding criterion-metric ranking.

```
plot_ranking(ranking, use_metric_as_facet=True)
```

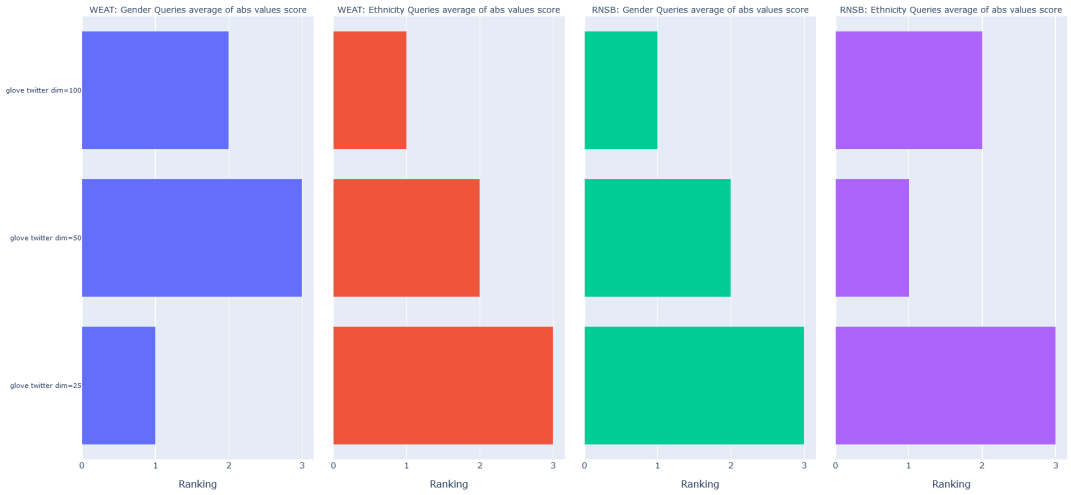

Figure C.3: Bar chart of the rankings using the metric and bias criteria as a separator (facet).

- Without facet. This figure shows the accumulated rankings for each embedding model. Each bar represents the sum of the rankings obtained by each embedding. Each color within a bar represents a different criterion-metric ranking.

Figure C.4: Bar chart of the aggregated rankings by embedding model.

C.3.4 Ranking Correlations

We can see how the rankings obtained in the previous section correlate to each other by using a correlation matrix. To do this we provide the `calculate_ranking_correlations`

	WEAT: Gender Queries average of abs values score	WEAT: Ethnicity Queries average of abs values score	RNSB: Gender Queries average of abs values score	RNSB: Ethnicity Queries average of abs values score
WEAT: Gender Queries average of abs values score	1	-0.5	-0.5	-1
WEAT: Ethnicity Queries average of abs values score	-0.5	1	1	0.5
RNSB: Gender Queries average of abs values score	-0.5	1	1	0.5
RNSB: Ethnicity Queries average of abs values score	-1	0.5	0.5	1

Table C.6: Correlations between rankings.

function. This function takes the rankings as input and returns the Spearman correlation between them.

```
from wefe.utils import calculate_ranking_correlations, plot_ranking_correlations
correlations = calculate_ranking_correlations(ranking)
correlations
```

Finally, we also provide a function to graph the correlations. This function enables us to visually analyze how the rankings correlate to each other.

```
correlation_fig = plot_ranking_correlations(correlations)
correlation_fig.show()
```

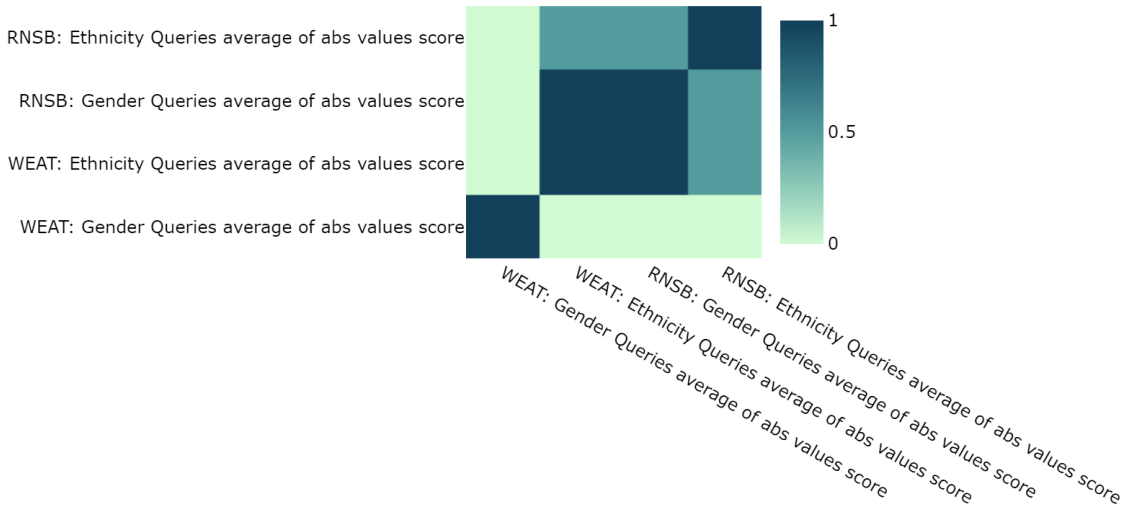

Figure C.5: Heatmap of the correlations between the rankings.