Natural Language Processing
Assignment 1

Reykjavik University - School of Computer Science

Samuele Pirani

Table of contents

Tokenization - Section 2 questions

Corpus Choice & Motivations

The corpus used for this project was built from a selection of 5 reviews posted on Google (as illustrated in the picture 1.1) about the Grand Hotel Excelsior (GHE), a historic four-star hotel located in the seaside town of Senigallia, Italy. Every summer, thousands of tourists visit the area, and many choose to stay at the hotel, later sharing their experiences on platforms such as Trivago, TripAdvisor, Booking, and Google. These reviews offer a rich source of textual data, as they reflect the customers' perceptions of various departments within the hotel, including hospitality, dining, cleanliness, and overall service. The decision to use reviews from GHE is also motivated by my personal experience: having worked as a waiter at the hotel for the past three years, I have witnessed firsthand how one of the main challenges lies in the management's difficulty in identifying and resolving issues within specific departments. In this context, NLP techniques, **particularly sentiment analysis**, could offer a valuable solution. By analyzing customer reviews, it becomes possible to detect which departments are most frequently associated with negative feedback, allowing the hotel to respond more quickly and effectively to recurring problems.

Python
```python
corpus = [
"Il voto è la media tra un hotel che strappa una sufficienza stiracchiata e un 'ristorante' fortemente inadeguato, con tovaglioli di carta, acqua del rubinetto filtrata, aceto nelle bustine come al McDonald's (anche nel baretto della spiaggia hanno le bottiglie), un menu da RSA e personale perlopiù gentile ma poco preparato (non tutti). Sono convinto che se ne freghino per scelta di fidelizzare puntando, invece, sul turnover.",

    "Bellissimo hotel molto pulito sono molto cordiali il mangiare e fresco e molto buono il personale e sempre pronto a servirti per qualsiasi tua necessità le pulizie in generale nell hotel sono sempre perfette sono sempre a pulire in modo maniacale,il guardiano dell' hotel e molto educato e pronto a provedere per qualsiasi cosa. Niente da dire che sicuramente non si può lamentarsi per nulla. BEN VOLENTIERI DA RITORNARE A FARE UNA SIGNORA VACANZA.",
    "Siamo stati molto bene, ascolti con molta cordialità,
disponibilità e simpatia. Tutto molto pulito e curato. Colazione
a buffet molto varia, cena molto buona, magari da migliorare le
verdure miste. Parco biciclette da sostituire perché vetusto.
Piscina e servizio spiaggia apprezzati. Tutto sommato,
consigliato.",

    "Siamo stati davvero bene in questo hotel, sia dal punto di
vista della cortesia degli addetti ai lavori, sia per la camera,
molto pulita e ben organizzata. Posizione perfetta difronte al
mare, con servizio spiaggia eccellente.Piscina con idromassaggio
ben curate e rilassanti. Davvero consigliato!!."

"Sono stata benissimo,il cibo era ottimo,la camera molto bella e
spaziosa,in più una cosa bellissima che le animatrici Elena e
Sara sono state eccezionali e hanno fatto divertire i mei figli,
non gli manca nulla e proprio un punto forte,lo consiglio a tutte
le famiglie!"
]
```

Question (a)

What is the fundamental difference between each tokenization method?

There are many differences between the two tokenization methods, but the most important one is the strategy behind the vocabulary creation. WordPiece builds its vocabulary incrementally: it begins with a small set of base tokens and iteratively adds new substrings that most improve the likelihood of the training corpus. This bottom-up approach focuses on refining the vocabulary step by step. Instead, the Unigram method starts with a big vocabulary (made up in different ways such as applying the BPE algorithm to an initial corpus with a large vocabulary size) and shrinking it by removing that contribute least to the overall model performance, in order to reach a desired vocabulary size at the end of the training phase. Furthermore,the two methods differ significantly in how they tokenize input text. In the WordPiece case, a passed word is tokenized by finding the longest subword present in the vocabulary, splitting it and repeating the process for the remaining parts. Unigram, on the other hand, employs a probabilistic model that considers all possible segmentations of a word and selects the one with the highest overall likelihood.

Question (b)

In the WordPiece script, what does ***vocab_size*** variable do? What effect does it have on the output and runtime to increase or decrease it by a factor of 10?

Vocab_size is an essential variable used to indicate to the model what is the desired vocabulary size to reach. After the initialization step in which the base vocabulary (refined by a given corpus passed as an input) has been enlarged with the special character such as [UNK] (used to mark a character or a substring as unrecognized), [CLS], and the rest, algorithm try to add new tokens to the vocabulary by splitting each word of the corpus in its correspondent letters, computing the probability of each letter's pair, finding the pair with the best score and adding it into the vocabulary. This procedure is actually regulated by the vocab_size, which determines how many new tokens are necessary to add in the vocabulary, interrupting the process upon reaching the desired size.

Before discussing how to modify output by changing vocab_size, it is essential to understand how many types compose the given corpus. After the initial phase, the resulting base vocabulary is composed of 80 types coming from corpus and 5 special characters implied by the model, for a total of 85 types. Since the default vocab_size value is equal to 100, the model will perform 15 merge operations, each time adding the new merged pair into the vocabulary, before stopping. Excluding all the previous 85 characters, the expanded vocabulary will be:

```Python
result =

//Base vocabulary (80 types + 5 special characters)

['[PAD]', '[UNK]', '[CLS]', '[SEP]', '[MASK]', '!', '##A', '##C',
'##D', '##E', '##G', '##I', '##L', '##N', '##O', '##R', '##S',
'##T', '##Z', '##a', '##b', '##c', '##d', '##e', '##f', '##g',
'##h', '##i', '##l', '##m', '##n', '##o', '##p', '##q', '##r',
'##s', '##t', '##u', '##v', '##z', '##à', '##é', '##ò', '##ù',
"'", '(', ')', ',', '.', 'A', 'B', 'C', 'D', 'E', 'F', 'I', 'M',
'N', 'P', 'R', 'S', 'T', 'U', 'V', 'a', 'b', 'c', 'd', 'e', 'f',
'g', 'h', 'i', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u',
'v', 'è',

//Merged pairs (15 pairs)

'RS', '##OL', 'VOL', 'RI', 'RIT', 'RITO', '##TI', '##IG', 'VOLE',
'##TIE', '##TIER', '##TIERI', 'RITOR', '##OR', '##RE']
```

If the parameter is increased by 10, the model will continue the iterative merging process for 10 more steps, resulting in 10 additional tokens being added to the vocabulary, as shown in the section below.

```Python
result =

//Base vocabulary (80 types + 5 special characters)

['[PAD]', '[UNK]', '[CLS]', '[SEP]', '[MASK]', '!', '##A', '##C',
'##D', '##E', '##G', '##I', '##L', '##N', '##O', '##R', '##S',
'##T', '##Z', '##a', '##b', '##c', '##d', '##e', '##f', '##g',
'##h', '##i', '##l', '##m', '##n', '##o', '##p', '##q', '##r',
'##s', '##t', '##u', '##v', '##z', '##à', '##é', '##ò', '##ù',
"'", '(', ')', ',', '.', 'A', 'B', 'C', 'D', 'E', 'F', 'I', 'M',
'N', 'P', 'R', 'S', 'T', 'U', 'V', 'a', 'b', 'c', 'd', 'e', 'f',
'g', 'h', 'i', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u',
'v', 'è',

//Merged pairs (25 pairs)

 'RS', '##OL', 'VOL', 'RI', 'RIT', 'RITO', '##TI', '##IG',
'VOLE', '##TIE', '##TIER', '##TIERI', 'RITOR', '##OR', '##RE',
'BE', 'BEN', 'VOLEN', 'VOLENTIERI', 'RITORN', 'UN', '##IGN',
'##IGNOR', '##NZ', 'SIGNOR']
```

Conversely, if *vocab_size* is reduced by 10, the merging process will terminate earlier, and only 5 new tokens will be added beyond the initial 85, as shown in the section below.

```Python
result =

//Base vocabulary (80 types + 5 special characters)

['[PAD]', '[UNK]', '[CLS]', '[SEP]', '[MASK]', '!', '##A', '##C',
'##D', '##E', '##G', '##I', '##L', '##N', '##O', '##R', '##S',
'##T', '##Z', '##a', '##b', '##c', '##d', '##e', '##f', '##g',
'##h', '##i', '##l', '##m', '##n', '##o', '##p', '##q', '##r',
'##s', '##t', '##u', '##v', '##z', '##à', '##é', '##ò', '##ù',
"'", '(', ')', ',', '.', 'A', 'B', 'C', 'D', 'E', 'F', 'I', 'M',
'N', 'P', 'R', 'S', 'T', 'U', 'V', 'a', 'b', 'c', 'd', 'e', 'f',
'g', 'h', 'i', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u',
'v', 'è',

//Merged pairs (5 pairs)

'RS', '##OL', 'VOL', 'RI', 'RIT']
```

Question (c)

In the Unigram/SentencePiece script, what does ***percent_to_remove*** control? What happens if you change that value?

In SentencePiece, the parameter *percent_to_remove* controls how aggressively the candidate vocabulary is pruned during training. At each iteration, the model sorts tokens by likelihood and removes the bottom fraction specified by this parameter. A high value shrinks the vocabulary quickly, leading to faster training, but may discard rare or meaningful tokens too early. Conversely, a low value entails a slower pruning activity, so the training phase requires more time to be completed, but the model preserves more useful candidates and produces a more accurate vocabulary.

To examine the behavior of SentencePiece under varying pruning intensities, I runned a set of experiments on the sentence *"Mi sono trovato molto bene in hotel"*. The parameter *percent_to_remove* was tested across five distinct values: 0.1, 0.5, 0.05 and 0.01. This allowed for a comparative analysis of how progressively more aggressive pruning affects token segmentation. The vocabulary size parameter remained fixed at its default value of 100 throughout all runs, ensuring that observed differences were attributable solely to changes in the pruning rate. The results are illustrated in the section below.

Python

```python
//Command

tokenize("Mi sono trovato molto bene in hotel", model)
```

```
//Test percent_to_remove = 0.1

['_', 'M','i','_sono','_',
't','r','o','v','a','to','_molto','_ben','e',
'_in','_hotel']

//Test percent_to_remove = 0.5

['_','M','i','_sono','_',
 't','r','o','v','a','to','_molto','_ben','e',
 '_','i','n','_hotel']

//Test percent_to_remove = 0.05

['_','M','i','_sono','_',
 't','r','o','v','a','to','_molto','_ben','e',
 '_in','_hotel']


//Test percent_to_remove = 0.01

['_','M','i','_sono','_',
 't','r','o','v','a','to','_molto','_ben','e',
 '_in','_hotel']
```

As it is possible to see from the result section, the segmentation produced by SentencePiece remains consistent across lower values of the *percent_to_remove* parameter, specifically at 0.01, 0.05, and the default setting of 0.1. In these cases, the tokenization preserves key subword units and exhibits no significant structural changes. Alterations in segmentation emerge only when the parameter is increased to higher values, such as 0.5, leading to the decomposition of token *'in'* into *'_'* , *'i'* and *'n'* . This indicates that the pruning mechanism begins to affect token boundaries meaningfully only beyond a certain threshold, while lower values maintain the integrity of the original.

Question (d)

Which tokenization method was the most useful for your corpus and why?

SentencePiece proved to be more effective than WordPiece for tokenizing the corpus. The language used is highly colloquial, with irregular punctuation, contractions, and spelling variations, features that SentencePiece handles more gracefully due to its subword-based approach. Unlike WordPiece, which tends to over-fragment words into less interpretable segments, SentencePiece produces more coherent and semantically meaningful tokens, leading to a more natural representation of the content. Additionally, SentencePiece does not rely on whitespace or predefined word boundaries, which is particularly advantageous for Italian, where compound words and contractions are common.

Bias in Word Embeddings - Section 4 questions

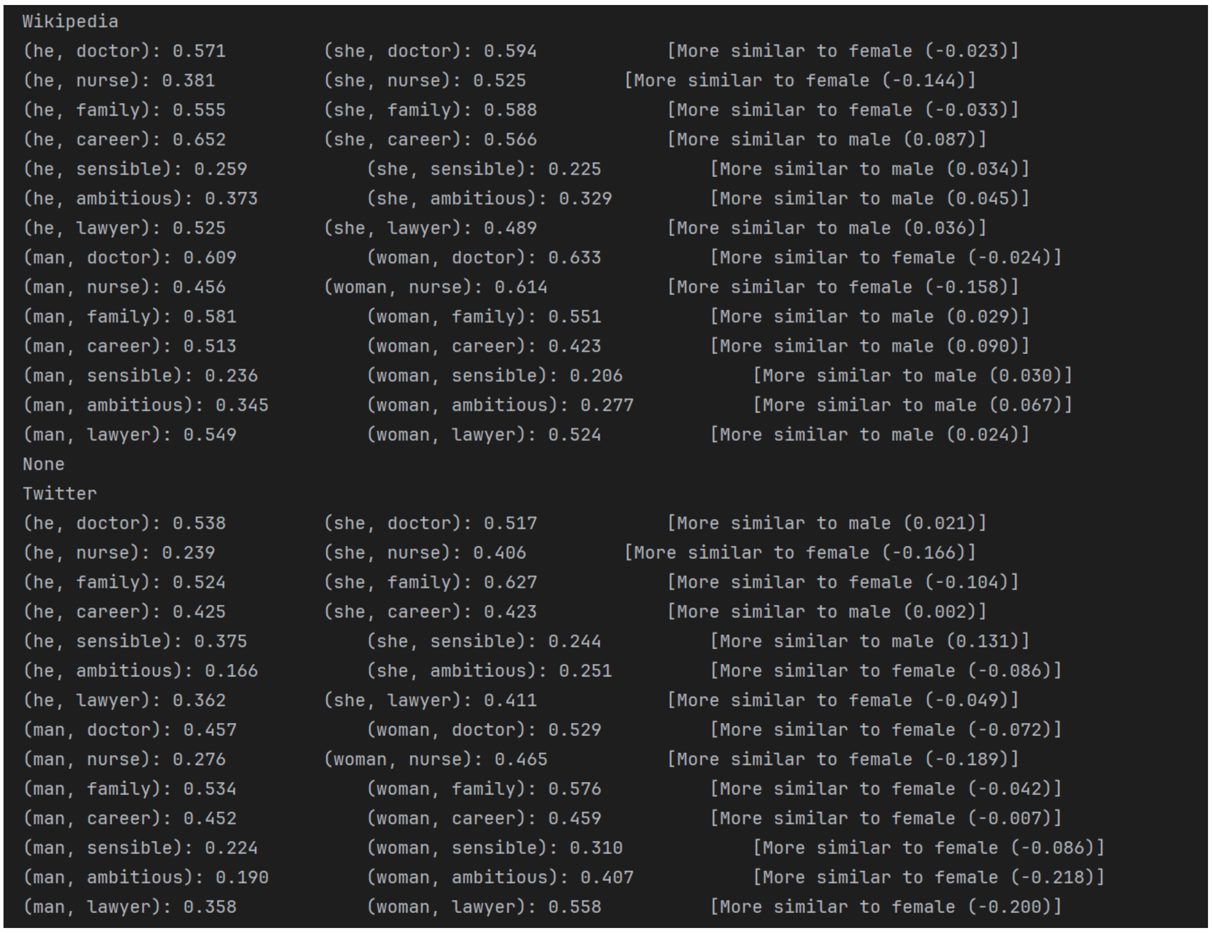
The aim of this section is to find potential bias in the two word embeddings generated from the glove-wiki-gigaword-100 and glove-twitter-100, two datasets respectively containing textual information coming from Wikipedia, Twitter and Gigawords. Despite the effort put into normalizing and checking information before any training phase, some inherent correlation between terms, due to several causes like stereotyping, denigration, underrepresentation and many others, might lead to undesirable effects on the NLP task as well as wrong work.

Starting with the most common, **gender_bias** are those stereotypes implying that some specific activities, sports, jobs, tasks or characteristics are more related to a specific gender than another one. A typical example of this bias is represented by the work distinction, asserting that some jobs are more related to the female (actress, houselady, cleaner, etc..), while other one closer to the man (e.g. engineer, physician, mathematician, etc..). In order to investigate this bias, it has built a list containing eight stereotyped words, three either for the man (*doctor, career* and *ambitious*) and woman(*nurse, family* and *sensible*), and two neutral jobs (*teacher* and *lawyer*) and for each gender (expressed as [*he, man] and [she, women]*) will be computed the degree of similarity through the *similarity* method provided by the **gensim** package. The developed function computes the similarity between passed gender and the stereotype word iteratively, building a dictionary to relate final results to each analyzed pair. The difference between similarities determines if it is more close to a gender than the other one.

```python
results = {}
    for word in words:
        for associated_word in associated_words:
            if associated_word in model and word in model:
                results[f"{word}, {associated_word}"] =
model.similarity(word, associated_word)
```

```
    return results
```

After the testing phase, analysis highlights that gender associations in the embeddings vary across the two models. In both models nursing is consistently aligned with the female dimension, reflecting enduring stereotypes, yet professions such as doctors and lawyers show a sharper male orientation in the Wikipedia embeddings, while in the Twitter model these roles appear more balanced or even slightly closer to the female side. Social and family-related concepts also shift: family is more strongly tied to the female perspective in Twitter, whereas in Wikipedia it often alternates depending on whether pronouns or gendered nouns are used. Career maintains a subtle male orientation across both models, though this bias is more pronounced in Wikipedia than in Twitter. Personality traits reveal further contrasts: sensibility remains closer to male associations in both, but ambition, traditionally linked with masculinity, leans toward male in Wikipedia while in Twitter it tends to align with the female side, suggesting a cultural shift in representation.

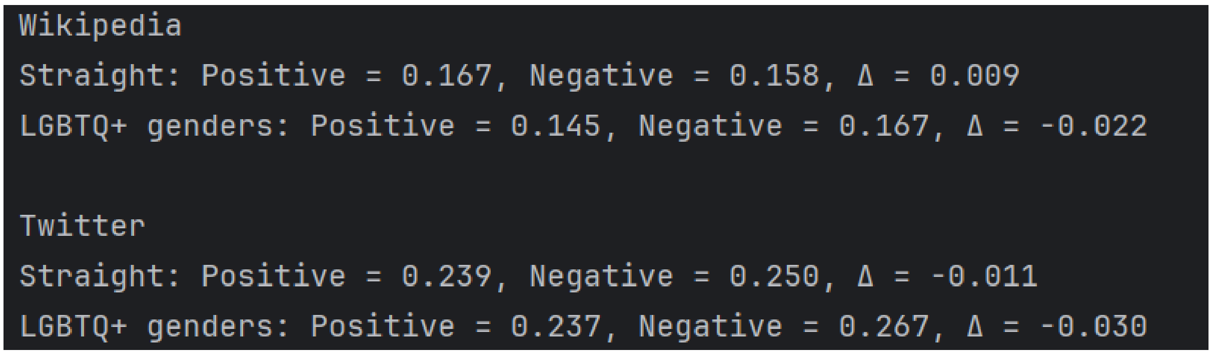


```
Wikipedia
(he, doctor): 0.571         (she, doctor): 0.594            [More similar to female (-0.023)]
(he, nurse): 0.381          (she, nurse): 0.525         [More similar to female (-0.144)]
(he, family): 0.555         (she, family): 0.588            [More similar to female (-0.033)]
(he, career): 0.652         (she, career): 0.566            [More similar to male (0.087)]
(he, sensible): 0.259           (she, sensible): 0.225          [More similar to male (0.034)]
(he, ambitious): 0.373          (she, ambitious): 0.329         [More similar to male (0.045)]
(he, lawyer): 0.525         (she, lawyer): 0.489            [More similar to male (0.036)]
(man, doctor): 0.609            (woman, doctor): 0.633          [More similar to female (-0.024)]
(man, nurse): 0.456         (woman, nurse): 0.614           [More similar to female (-0.158)]
(man, family): 0.581            (woman, family): 0.551          [More similar to male (0.029)]
(man, career): 0.513            (woman, career): 0.423          [More similar to male (0.090)]
(man, sensible): 0.236          (woman, sensible): 0.206            [More similar to male (0.030)]
(man, ambitious): 0.345         (woman, ambitious): 0.277           [More similar to male (0.067)]
(man, lawyer): 0.549            (woman, lawyer): 0.524          [More similar to male (0.024)]
None
Twitter
(he, doctor): 0.538         (she, doctor): 0.517            [More similar to male (0.021)]
(he, nurse): 0.239          (she, nurse): 0.406         [More similar to female (-0.166)]
(he, family): 0.524         (she, family): 0.627            [More similar to female (-0.104)]
(he, career): 0.425         (she, career): 0.423            [More similar to male (0.002)]
(he, sensible): 0.375           (she, sensible): 0.244          [More similar to male (0.131)]
(he, ambitious): 0.166          (she, ambitious): 0.251         [More similar to female (-0.086)]
(he, lawyer): 0.362         (she, lawyer): 0.411            [More similar to female (-0.049)]
(man, doctor): 0.457            (woman, doctor): 0.529          [More similar to female (-0.072)]
(man, nurse): 0.276         (woman, nurse): 0.465           [More similar to female (-0.189)]
(man, family): 0.534            (woman, family): 0.576          [More similar to female (-0.042)]
(man, career): 0.452            (woman, career): 0.459          [More similar to female (-0.007)]
(man, sensible): 0.224          (woman, sensible): 0.310            [More similar to female (-0.086)]
(man, ambitious): 0.190         (woman, ambitious): 0.407           [More similar to female (-0.218)]
(man, lawyer): 0.358            (woman, lawyer): 0.558          [More similar to female (-0.200)]
```

Related to gender bias, **sexual orientation bias** refers to sexual prejudice, a negative attitude towards someone that is based on their sexual orientation. Less frequently in the formal dataset (like Wikipedia) than the informal (as Twitter), the training data often reflects societal prejudices leading the model to associate sexual orientation such as “gay” or “lesbian” to negative concepts. To empirically assess this bias, a similarity-based method is applied using word embeddings. This technique involves computing the average semantic similarity between terms related to sexual orientation (e.g., “gay”, “lesbian”, “transgender”) and the two curated lists of positive and negative concepts. By leveraging cosine similarity, a metric that quantifies how close two word vectors are in the embedding space, it becomes possible to detect implicit associations learned by the model. For instance, if the model consistently shows higher similarity between “gay” and negative terms like “perverted” or “abnormal” than with positive terms such as “natural” or “strong”, this causes an increasing difference between the two classes with a consequently biased representation.

Python
```python
def avg_similarity(group1, group2, model):
    sims = []
    for w1 in group1:
        for w2 in group2:
            if w1 in model and w2 in model:
                sims.append(1 - cosine(model[w1], model[w2]))
    return mean(sims)

def compute_avarage_similarity(group, model):
    pos = avg_similarity(group, positive_traits, model)
    neg = avg_similarity(group, negative_traits, model)
    return  f"{group}: Positive = {pos:.3f}, Negative = {neg:.3f}, Δ = {pos
- neg:.3f}\n"
def run_bias_analysis():
    wiki, twitter = load_wordemeddings_model()

    print("Wikipedia")

    print("Straight: " + compute_avarage_similarity(groups["heterosexual"],
wiki))
    print("LGBTQ+ genders: " +
compute_avarage_similarity(groups["gender_lgbt"], wiki))

    print("\nTwitter")

    print("Straight: " + compute_avarage_similarity("Straight: "+
groups["heterosexual"], twitter))
    print("LGBTQ+ genders: " +
compute_avarage_similarity(groups["gender_lgbt"], twitter))
```

Confirming the previous assumption, the results showed that the bias is more present on the model which contains a higher concentration of informal or colloquial language expressions than the formal one. By comparing Wikipedia and Twitter, two corpora with distinct linguistic registers, the results reveal a consistent pattern: terms related to LGBTQ+ identities are more strongly associated with negative concepts than positive ones, especially in Twitter. In Wikipedia, the difference in semantic similarity (Δ) between positive and negative associations is −0.022 for LGBTQ+ terms, while it is slightly positive (+0.009) for straight-related terms. On Twitter, the bias intensifies (with Δ drops to −0.030 for LGBTQ+ terms, compared to −0.011 for straight terms) indicating that models trained on informal, user-generated content are more likely to reflect societal prejudices.



```
Wikipedia
Straight: Positive = 0.167, Negative = 0.158, Δ = 0.009
LGBTQ+ genders: Positive = 0.145, Negative = 0.167, Δ = -0.022

Twitter
Straight: Positive = 0.239, Negative = 0.250, Δ = -0.011
LGBTQ+ genders: Positive = 0.237, Negative = 0.267, Δ = -0.030
```

Another highly sensitive and socially impactful form of bias in NLP systems is **ethnic bias**. It emerges when models disproportionately link specific ethnic identities to negative concepts, often due to imbalanced or culturally skewed training corpora. Addressing this issue is essential to ensure equitable and responsible language technologies. As in the case of sexual bias, the test implies the usage of the average to discover the degree of similarities between the two ethnic groups, but applied on a single list of negative words referring to stereotype expressions such as “criminal”, “immigrant”, “illegal”, etc… The function returns the mean similarity for each group and the difference (Δ) between them, indicating which group is more closely associated with the negative stereotypes.

```python
Python
def compute_avarage_similarity_between(group1, group2, model):
    group1 = avg_similarity(group1, association_words, model)
    group2 = avg_similarity(group2, association_words, model)
    return f"White ethnicity = {group1:.3f}, Black ethnicity = {group2:.3f},
Δ = {group1 - group2:.3f}"


idef run_bias_analysis():
    wiki, twitter = load_wordemeddings_model()

    print("Wikipedia")
    print(compute_avarage_similarity_between(groups["ethnicity_white"],
groups["ethnicity_black"], wiki))
    print("\nTwitter")
    print(compute_avarage_similarity_between(groups["ethnicity_white"],
groups["ethnicity_black"], twitter))
```

The results of the ethnic bias analysis reveal a consistent pattern across both datasets. In Wikipedia, the average semantic similarity between Black ethnicity terms and negative stereotype words (e.g., “criminal”, “illegal”, “immigrant”) compared to White ethnicity terms, yielding a difference (Δ) of −0.025. This indicates that the model trained on Wikipedia data associates negative concepts more strongly with Black ethnicity than with White ethnicity.

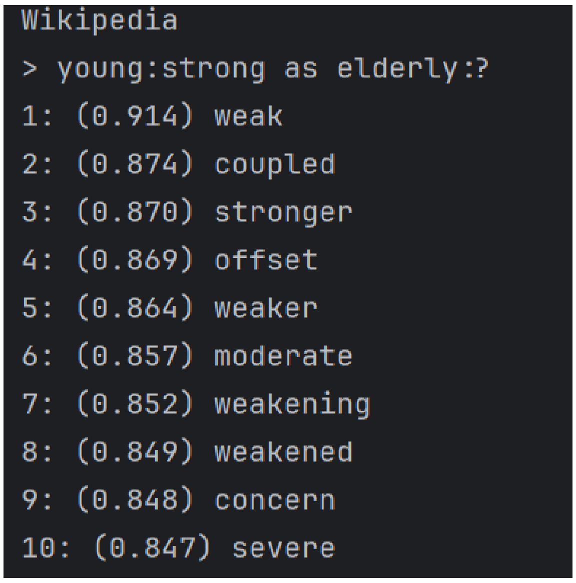
The bias is even more pronounced in the Twitter dataset, where the similarity score for Black ethnicity rises, resulting in a Δ of −0.032. These findings confirm the presence of ethnic bias in both corpora, with a stronger effect in informal datasets.

```
Wikipedia
White ethnicity = 0.265, Black ethnicity = 0.306, Δ = -0.041

Twitter
White ethnicity = 0.322, Black ethnicity = 0.361, Δ = -0.040
```

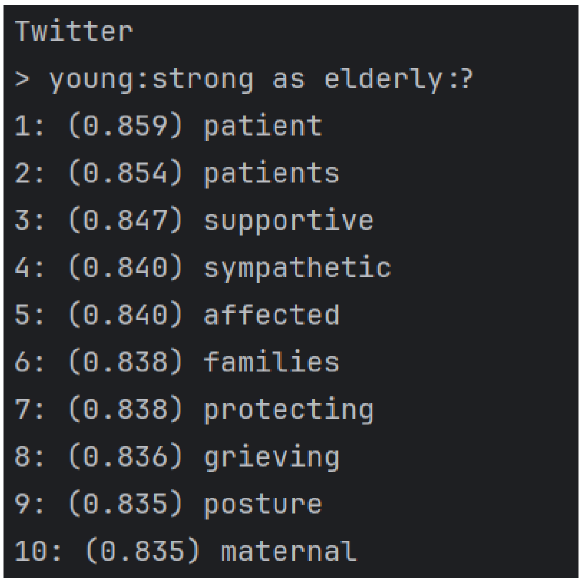
Another socially relevant and often underexplored form of bias is **ageism bias**. This occurs when models implicitly associate older individuals with negative or limiting attributes such as weakness, confusion, or irrelevance, while younger individuals are linked to strength, intelligence, or leadership. To investigate this bias, an analogical reasoning method is applied using word embeddings. Specifically, the *find_word* function is used to solve analogies of the form *“young is to strong as old is to ?”*, allowing us to observe which terms the model semantically links to older age in comparison to younger age. By analyzing the top results returned by the model, we can qualitatively assess whether older age groups are disproportionately associated with negative or stereotypical concepts. This approach offers a powerful lens for uncovering implicit age-related bias embedded in language representations.

Python
```python
def find_word(a, b, x, model):
  a = a.lower()
  b = b.lower()
  x = x.lower()
  print(f"> {a}:{b} as {x}:?")
  top_words = model.most_similar_cosmul(positive=[x, b], negative=[a])
  for num, (word, score) in enumerate(top_words[:10]):
    print(f"{num + 1}: ({score:.3f}) {word}")


def run_bias_analysis():
    wiki, twitter = load_wordemeddings_model()

    print("Wikipedia")
    print(find_word("young","strong","elderly",wiki))
    print(find_word("young", "energetic", "elderly", wiki))
    print(find_word("young", "reckless", "elderly", wiki))

    print(find_word("elderly", "fragile", "young", wiki))
    print(find_word("elderly", "slow", "young", wiki))
    print(find_word("elderly", "wise", "young", wiki))
    print("Twitter")
    print(find_word("", "strong", "elderly", twitter))
    print(find_word("young", "energetic", "elderly", twitter))
    print(find_word("young", "reckless", "elderly", twitter))

    print(find_word("elderly", "fragile", "young", twitter))
    print(find_word("elderly", "slow", "young", twitter))
    print(find_word("elderly", "wise", "young", twitter))
```

The results show how age-related biases manifest differently across corpora. In Wikipedia, elderly individuals are often framed through stereotypical contrasts with youth. Yet, some analogies offer nuance, associating the elderly with traits like “even-tempered” or “family-centered.” Twitter, by contrast, reflects a more emotionally driven and socially contextualized view: elderly people are described as “patient,” “supportive,” and “sympathetic,” though the data is noisier and less semantically consistent due to informal language and cultural variation.

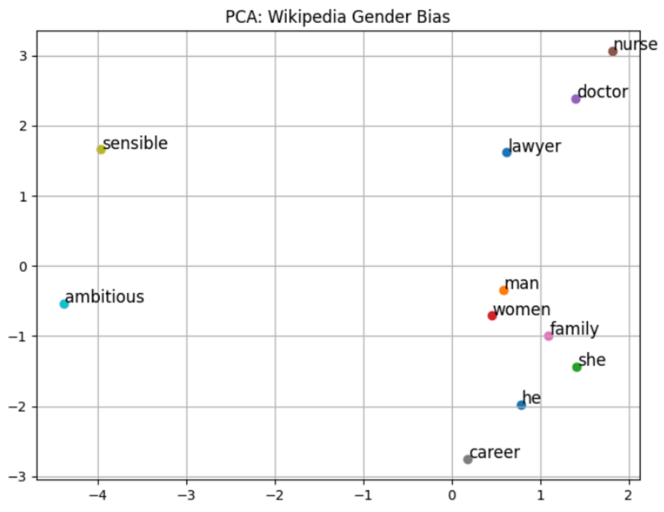


```
Wikipedia
> young:strong as elderly:?
1: (0.914) weak
2: (0.874) coupled
3: (0.870) stronger
4: (0.869) offset
5: (0.864) weaker
6: (0.857) moderate
7: (0.852) weakening
8: (0.849) weakened
9: (0.848) concern
10: (0.847) severe
```

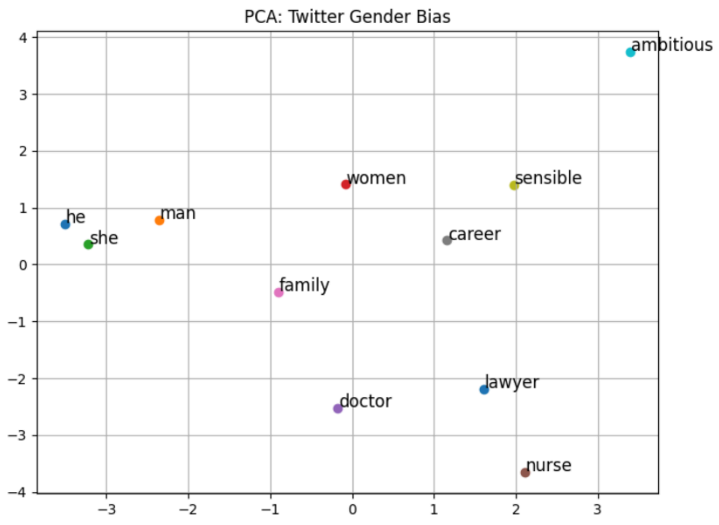


```
Twitter
> young:strong as elderly:?
1: (0.859) patient
2: (0.854) patients
3: (0.847) supportive
4: (0.840) sympathetic
5: (0.840) affected
6: (0.838) families
7: (0.838) protecting
8: (0.836) grieving
9: (0.835) posture
10: (0.835) maternal
```

In the results of the experiment, some biases emerge clearly through semantic analogies (for example, *young:strong as elderly:weak*), while others remain less immediate or even hidden within numerical values. This is why it becomes necessary to adopt visualization and dimensionality reduction techniques, such as **PCA** or **t-SNE**, which allow projecting latent relationships in the embedding space into two dimensions. A concrete example can be observed in the case of gender bias when comparing Wikipedia and Twitter embeddings. In the Wikipedia plot, gendered terms such as *she* and *women* cluster near words like *nurse* and *family*, while *career* and *ambitious* are positioned at a distance, implicitly associating them more with male terms. This distribution reflects more traditional gender stereotypes, where caregiving and domestic contexts are linked to women, and ambition or professional orientation is aligned with men.



In the Twitter plot, the separation is less rigid. Words such as *doctor* and *lawyer* are placed farther from explicit gendered terms, and *ambitious* appears in a more isolated position without strong alignment to either side. This indicates that while bias still exists, it is encoded differently, with weaker clustering around traditional gender-role associations compared to Wikipedia.



Although the recognition of the bias permits discovering them the problem is still there, leading to the necessity to remove them. Even if it is impossible to remove bias fully, existing different solutions are able to mitigate their impact. One of the most influential is **hard debiasing**, which identifies a “bias direction” in the embedding space (for example, the gender axis defined by *he*–*she*) and then adjusts gender-neutral words such as professions so that they are equidistant from this axis.

```python
debiased_vectors_wiki = [model[word] - np.dot(model[word],
gender_direction) * gender_direction for word in stereotypes]
```

This method is effective in removing bias along a clearly defined single dimension, but it also has important limitations: it does not address more complex or multidimensional forms of bias, such as those related to race, ethnicity, religion, or social class. As a result, even after debiasing, terms like *doctor* may remain closer to *he* than to *she*, as it is possible to notice in the following figures. A more flexible alternative is **soft debiasing**, which does not completely eliminate the influence of sensitive attributes but reduces their impact, allowing for a better balance between bias mitigation and preservation of the original semantic structure, even if traces of bias may still persist.

```python
def soft_debias(word_vec, bias_direction, alpha=0.5):
    projection = np.dot(word_vec, bias_direction) /
np.linalg.norm(bias_direction)**2 * bias_direction
    return word_vec - alpha * projection
```

Taken together, these approaches show that while absolute neutrality is unattainable, bias can be significantly reduced, leading to embeddings that are fairer, more transparent, and less likely to reinforce harmful stereotypes.