To what extent can different methods of debiasing word embeddings be used to reduce bias within natural language processing models?

Word count: 3809

Personal code: kmt024

Section 1 – Introduction:

As we progress into the future more and more artificial intelligence (AI) becomes a larger part of our lives. With the recent introduction of OpenAI's Natural language processing (NLP) AI, ChatGPT (OpenAI, 2022) to the world the versatility and applicability of NLP models have become clear to everyone (Nielsen, 2022). However, these models are not perfect and as such are very vulnerable to bias.

Consider the following example: as a currently unemployed person John is looking for a job. John asks an NLP algorithm for a job recommendation and using what it knows about John (his qualifications, age etc.) it returns that he should consider a Job in computer science. Jane is also unemployed and currently in search of a Job. When she asks the NLP algorithm for a job recommendation it, also using what it knows about her, returns homemaker. While this example may sound far-fetched it is based on a real-life example (Bolukbasi et al., 2016) in which an NLP algorithm stated that a man was to a computer scientist as a woman was to a homemaker. This clear display of bias is completely unethical and leads to societal problems such as job inequality. The algorithm was unintentionally created such that despite John and Jane having the same qualifications they would be recommended different jobs based on gender. This example of bias within a machine learning algorithm can be used as an allegory for how I will define bias: The unjustified display of preference for one group over another. This, however, begs the question of where does bias come from in a machine-learning algorithm? There are 5 ways of classifying where bias in an NLP algorithm may come from (Hovy & Prabhumoye, 2021). Bias from data, annotations, input representations, models, and research design:

- Bias from data is the one which is mostly thought of as the primary cause of the bias within any algorithm and is inherently caused by poor data selection to train a model on.
- Bias from annotations occurs in supervised learning algorithms when the labels for data contain bias. This is usually due to human error.
- Bias from models is effectively the effect of pre-existing bias within a model that is exponentially increased as the model receives new data and reinforces those beliefs.
- Bias from research design is like bias from data in that it is caused by neglecting data which would represent minority groups therefore not considering them when presenting solutions to problems.

The focus of this investigation will be bias from input representations. Bias from this is caused by how the computer stores words or phrases and the relationships between those words and phrases. The example with John and Jane could likely be described as an input relationship bias as the algorithm has decided that computer science was a male job, and that homemaker was a female job, therefore, shifting the likelihood of one or the other being the primary suggestion to the user depending on their gender.

The process of the investigation will begin with understanding how word embeddings work and how the words are represented after which I will explain the debiasing methods that I will be evaluating followed by an analysis of the results to find out how effective the debiasing methods have been.

Section 2 – Background Theory

2.1 Word embeddings

A common method of input representations for NLP algorithms is word embeddings (Hu, 2022). It is of encoding words for a computer that turns words into floating point vectors meaning that computers can use ‘words’ for equations discerning relationships

2.1.1 OneHotEncoding

A very simple example of word embedding is OneHotEncoding. This is a method commonly used in regression to transform discrete variables into a numerical interpretation. This is done with vectors. By creating a new dimension in a vector for every new variable; if a word j is the nth new word to appear it is encoded as 0 for all other dimensions but 1 for the nth dimension of the vector.

Feature for:	Dog	Cat	Kettle	...	Pen

One-hot vector representation for words:

Dog	1	0	0	...	0
Kettle	0	0	1	...	0
Cat	0	1	0	...	0
Pen	0	0	0	...	1

etc... [1]

[1] (Borisov, 2021)

2.1.2 Context-based encoding

One of the many problems of OneHotEncoding is that it provides no context for words. It merely encodes them into vectors for the computer to apply arithmetic too. This is useful for something such as regression where an understanding of the word is not needed, however for NLP it is the exact opposite; we need the context of the word to properly use it. However, there exist methods of encoding words such that instead of encoding the word in a corpus of text based on itself, it is done based on the context surrounding that word. The context is defined as a range of other words around the desired word. There are a variety of ways to decide the context itself, one such is if I have a corpus of text:

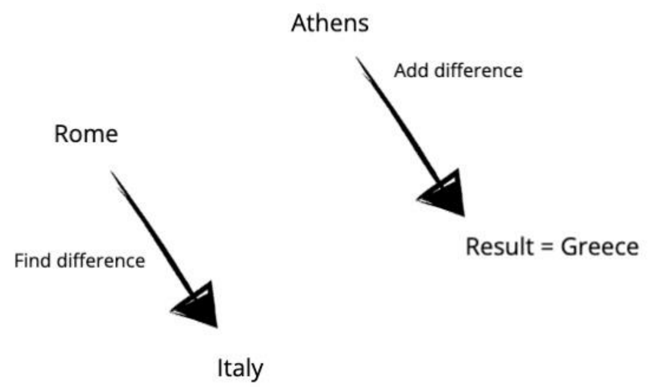
"I like concise code.

I like concise fast code."

If I wanted to encode the word "code" I could look at the previous 2 words. For each new word in the previous 2 on each line, I would add a new dimension. Using this method, the word: "code" would encode to $\begin{matrix} Concise & fast & like \\ 2 & 1 & 1 \end{matrix}$, where the bottom line is the vector, each dimension representing the number of occurrences of the word above it. By encoding words like this, when the computer is analysing a word vector it now knows a variety of other words that are associated to it which allows for it to have an improved understanding of what the word means.

2.1.3 Word2Vec embedding model

Word2Vec (Sharma, 2021) is a popular context-based word embedding model. It takes a corpus of text and turns them all into vectors. As it is a context-based word embedding model it does this by defining words as vectors relative to the words around them. There are two training methods to do this: continuous bag of words (CBOW) and skip-gram. CBOW involves using the context to predict a target word. For example: “Rome is the capital of Italy, Athens is the capital of ____” it will use the context to predict that the best match is Greece. The Skip-gram training method involves using an input word to predict a target context. For example, given input word “Rome” it might predict that the context is “the capital of Italy”. CBOW usually leads to more accurate word embeddings in larger text corpora due to the larger availability of contexts. If a word vector does not accurately fit a context, then the values of the dimensions in the vector are adjusted to fit that context more accurately. This has the effect of causing word vectors that can be applied in similar contexts to be grouped more closely together. One of the consequences of embedding words like this is that the embeddings subliminally convey more information than it appears they should. Going back to the example of “Rome is the capital of Italy; Athens is the capital of ____”, we can find this out by subtracting the word vector for Italy from the word vector for Rome and then Adding the word vector for Athens. This will result in the word vector for Greece.

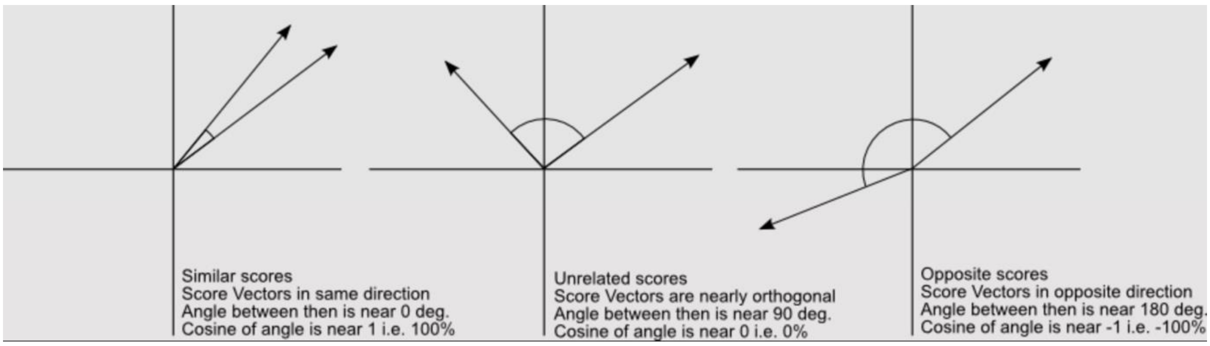


2.2 Vector space

The space where the words are stored can be visualised as a vector space. This just means that all the words can be thought of as arrows from the origin travelling a set distance along each of the axis (dimension). This has the effect of resulting in words that are of similar meaning getting grouped. The primary ways of measuring how similar words are involved using the cosine similarity and Euclidian distance of two vectors. The cosine similarity of two vectors A and B is given by:

$$\cos(\mathrm{x}) = \frac{A \cdot B}{|A||B|}$$

Which is visually represented as the angle between the two vectors. A cosine similarity that is closer to 1 show that the vectors are similar and closer to 0 shows that they are not similar.



[2]

And the Euclidian distance between two vectors is given by:

$$d = \sqrt{(A_1 - B_1)^2 - (A_2 - B_2)^2 + \ldots + (A_i - B_i)^2}$$

(Where A_i represents the value of the ith dimension)

---

[2] Mk, S. (2020, October 5)

Which is visually represented as the distance between the two vectors. It is denoted ||a-b||

2.3 Bias

Thus, given a context-based word embedding model such as Word2Vec, how does bias occur? Bias occurs when a model encodes the vectors in such a way that a vector X is closer, in terms of cosine similarity or Euclidian distance, to A than B even though it should be equidistant from both. If we consider the example of bias posed in the introduction paragraph again, this was a result of the “computer scientist” vector having a greater cosine similarity or Euclidian distance to the “male” vector than the “female” vector, and the converse being true for the “homemaker” vector. Debiasing is therefore the process of trying to find a way to offset this bias within the vector space.

2.4 Analogies for evaluating bias

It is important to establish a method of measuring bias. The method I will be using will be the word embedding association test (WEAT) which is an algorithm to measure bias in analogies generated from the embedding model. Analogies will be statements of the form “[MALE WORD] is to x as [FEMALE WORD] is to y”.

Firstly, we need a way to create analogies. For this, I will give two target words, one male and one female, as input. This pair will be called a pair of seed words, vectors a and b. From this pair of seed words, a direction $(a - b)$ will be determined and then we will look to find another pair of words from the original word embeddings such that:

$$S_{a,b}(x,y) = cos(a-b, x-y), \quad if\ |x-y| \leq \delta, 0\ else$$

Where the optimal words x and y will be whichever inputs yield the largest positive $S_{a,b}$ scores (Bolukbasi et al., 2016). This will be the word which fits the analogy “a is to x as b is to y” the best. Where the delta (δ) is a maximum measurement of how close the vectors x and y can be to be considered a pair. If they are not sufficiently close, then they are not considered valid pairs. The thought behind this is that we want an analogy pair which is of similar distance to the seed pair. We set δ based on the distance between a and b. If a dataset has more elements with higher dimensionality, a higher threshold can be used as there is more variation in the meanings of the words. This holds for the inverse as well. Furthermore, it may also depend on the application of the word embeddings. If we are undergoing a task that requires more fine-tuning, such as sentiment analysis, a lower threshold can ensure that the pairs generated are more nuanced and specific to the context. We can set the threshold for similarity to 1, which indicates that the two words are closer to each other than they are to the origin. To reduce redundancy, multiple analogies which share the same word x will not be output.

After generating these analogies, we can evaluate them using WEAT. WEAT is an evaluation method that takes four parameters, each one an array. An array of each of the target vectors and an array of each of the attribute vectors. The target vectors array are the arrays of the words that were used as the seed pairs, and the attribute vectors array are the arrays of the words that were generated as being the best analogy to a corresponding seed pair. It is mathematically represented by the formulae (Kurita et al., 2019):

$$s(t, A_1, A_2) = mean_{a \in A_1}\big(sim(t, a)\big) - mean_{b \in A_2}\big(sim(t, b)\big)$$

$$S(T_1, T_2, A_1, A_2) = [mean_{x \in T_1}(s(x, A_1, A_2)) - mean_{y \in T_2}(s(y, A_1, A_2))]$$

The first equation is used to measure the average difference in cosine similarity between a specific target vector and each of the Attribute groups. The second equation is finding the average difference of the first function between the two target vectors. This returns the test statistic which is the score representing how much the attribute vectors seem to favour one target over another. If the vectors were truly unbiased, we might expect to see a result of 0 showing that there is no partiality. We also measure the effect size. The effect size in WEAT analysis is a measurement which tells us the strength of association between the two target sets of words and the attribute words set. If the effect size is larger, it means that more words are associated with the target words. In our investigation, we want a smaller effect size as we don't want non-gendered words to end up associated with gender. It is denoted by the equation:

$$d = \frac{S(T_1, T_2, A_1, A_2)}{std_{t \in T_1 \cup T_2} s(t, A_1, A_2)}$$

Where std is the standard deviation function. The test statistic will be used to compare how effective the debiasing method was and the effect size will be used to compare how the debiasing method has affected the relationships between the words.

2.5 Identifying the gender subspace

The gender subspace is what we call the dimensions of the vectors which are most strongly associated with gender. This is where words get wrongly identified as being more partial to one gender than another. By identifying a gender subspace, we can recognise words that should not have values in that subspace and remove them from that subspace. We find the space by first finding all words which should be gendered, such as man, woman, mother, and father, and then subtracting the mean female embedding from the mean male embedding. This results in a vector that captures the direction of the gender bias in the embedding space.

2.6 Model to be used

The model that I will use during this investigation will be an online available model called “w2v_gnews_text_small” (*NLP-Word2Vec-Embeddings (Pretrained)*, 2018). This is an embedding model trained on a corpus of google news articles. It contains over 32 000 words and was chosen as it would not be an unjustified to assume that a model trained on such a corpus should be relatively free from bias in the embeddings.

Section 3 – presenting solutions

Before describing at the methods to be evaluated for gender debiasing, I ought to describe the generic steps before implementing each debiasing method. Firstly, we ought to identify the gender subspace. To identify the gender subspace, we take the mean of the vector embeddings for the words associated with male and the mean of the word embeddings associated with female, pre-defined by external dataset, using the equations (Bolukbasi et al., 2016):

$$m_f = \frac{1}{n_f} * \sum S_i \ \ (if\ S_i = female\ word)$$

$$m_m = \frac{1}{n_m} * \sum S_i \ \ (\ if\ S_i = male\ word)$$

Where n_f and n_m are the numbers of female and male labelled vectors respectively. The gender direction is then defined as the vector difference between the two mean embeddings:

$$g = m_m - m_f$$

If we plot our gender-neutral words onto the gender direction axis we will see that there is a large amount of bias in some words as they are leaning more to the side of either male or female despite being gender neutral.

[3]

While some of these words are justified in leaning more towards male/female such as boyhood/queen, others are not, such as command/browsing. This is a clear demonstration of bias.

[3] (Bolukbasi et al., 2016)

3.1 Hard debiasing

The way that hard debiasing is performed is by performing a linear transformation to the word embeddings that reduces the gender dimension in the gender-neutral words. This is done by projecting the word vectors onto a new space that is orthogonal to the gender subspace. By performing this transformation, it removes the gendered information from the gender-neutral words. To de-bias the embeddings in subset N we first need to compute a projection matrix which is a matrix describing the transformation of the vectors from the space W to space W' – the space without bias. This is given by the equation (Bolukbasi et al., 2016):

$$P = I - \frac{g * g^T}{||g||^2}$$

The transformation P is the linear transformation that will be applied to the matrix of gendered neutral words **N**. By taking the outer product of the gender dimension with the transpose of itself, we have a matrix that captures the direction of the gender bias in the embedding space. We divide it by the squared norm of the gender subspace to normalise the outer product matrix. By subtracting this from the identity matrix, we create a matrix that projects and vector onto a space that is orthogonal to the gender subspace.

Now that we have defined the transformation, we simply need to apply it using matrix multiplication. This is done:

$$N' = P * N$$

This results in all our gender-neutral words no longer having a gendered dimension, therefore removing gender-based associations from the embeddings.

3.2 Soft debiasing

Soft debiasing involves generating a matrix S of all gendered word embeddings where each row corresponds to the embedding vector for a single word. Let us refer to x_i as the embedding vector for the ith word.

To de-bias the embeddings in subset N, we will subtract the gendered direction from the gender-neutral words. This is done with the equation (Bolukbasi et al., 2016):

$$x_i^{db} = x_i - proj_g(x_i)$$

$$proj_g(x_i) = \left(\frac{x_i^T \cdot g}{g^T \cdot g}\right) * g$$

$proj_g(x_i)$ is a projection function which maps the vector x_i onto the gendered direction. To do this it takes the dot product of the transpose of x_i with the gendered direction and then normalises the length to a unit length. Afterwards, it is multiplied by the gendered direction so that it can be plotted on the gendered axis.

x_i^{db} represents the unbiased x_i vector.

But by doing this we encounter a problem of losing some of the original information of the embedding. To counteract this problem, we introduce a minimisation problem with a hyperparameter lambda to keep the unbiased embedding as close to the original embedding as possible. This means we should instead use the equation (Bolukbasi et al., 2016):

$$x_i^{db} = \min_{x}(\left||x - x_i|\right|^2 + \lambda((x - x_i)^T \cdot W(x - x_i)))$$

This presents an optimisation problem which will return the vector x which can maintain as much information from the original embedding, while also removing the gender bias. W is a weight matrix that mathematically presents the importance of each dimension within my word embeddings. As I am looking to remove the gender dimension it will consist mainly of small values for non-gender dimensions and larger floating-point values for the gender dimension. By applying this equation to all the x_i^{th} vectors in the original embedding we will have a new embedding which has reduced the gender bias while maintaining associations that some words will have to gender.

Section 4 – Results + Analysing solutions

After programming these models (Appendix. A) and using the methods aforementioned, I created a CSV file to show the different analogies that the different embedding models led to. Here are the results (Table. 1) (Appendix. B):

Given the analogy format x is to a, as y is to b, the column title corresponds to x or y and the elements in the column to a or b. The columns marked as (Bias) denote the analogies generated from the original embedding model, the columns marked (Hard) are analogies from the hard debiased model and the columns marked (Soft) the analogies from the soft debiased model.

She(Bias)	He(Bias)	She(Hard)	He(Hard)	She(Soft)	He(Soft)
she	he	woman	man	she	he
herself	himself	sorority	fraternity	herself	himself
her	his	twin_sister	twin_brother	her	his
woman	man	ladies	gentlemen	woman	man
daughter	son	mare	gelding	daughter	son
businesswoman	businessman	estrogen	testosterone	businesswoman	businessman
girl	boy	sister	brother	girl	boy
actress	actor	grandma	grandpa	actress	actor
chairwoman	chairman	congresswoman	congressman	chairwoman	chairman
heroine	hero	aunt	uncle	heroine	hero
mother	father	spokeswoman	spokesman	mother	father
spokeswoman	spokesman	gals	dudes	spokeswoman	spokesman
sister	brother	daughter	son	sister	brother
girls	boys	moms	dads	girls	boys

sisters	brothers	councilwoman	councilman	sisters	brothers
queen	king	mothers	fathers	gal	guy
niece	nephew	sisters	brothers	queen	king
councilwoman	councilman	grandmother	grandfather	niece	nephew
motherhood	fatherhood	queens	kings	councilwoman	councilman
women	men	granddaughter	grandson	motherhood	fatherhood
petite	lanky	ovarian_cancer	prostate_cancer	women	men

Table 1

I will be analysing these methods of debiasing both mathematically and by human analysis. The mathematical evaluation will be an implementation of a WEAT wherein I will compare the effect size, and test statistics of the various models to each other. A smaller test statistic indicates that there is less systematic bias as the mean cosine similarity between the target vectors and the attribute vectors is small. A smaller effect size indicates that the strength of association between the target and attribute words is smaller and therefore implies less bias, and a larger effect size indicates that the strength of association between the sets of words is stronger. An ideal solution would have a small test statistic and a small effect size.

In the human analysis I will pass three words into each model, say: ‘man’, ‘king’, ‘woman’, which will be equivalent to an analogy of form x is to a as y is to b. The computer will then return a predicted b. It will output the word which it thinks is most like ‘woman’ by the degree that ‘king’ is like ‘man’. Then I will rate these analogies either 1 or 0 based on whether I find them biased or not (1 being biased, 0 being not biased); then find the mean human evaluated bias for them.

Mathematical analysis

The mathematical evaluations displayed (Table. 2) show us that the most overall effective method for removing bias is hard debiasing. While soft debiasing does reduce the strength of association between the target and attribute words, the number of words associated with gender (Effect size) is significantly higher than it is for the Hard debiased model. This means that while the words do not lean towards male or female specifically, they are more often associated with male and female which is not ideal. Hard debiasing on the other hand reduced both the effect size and the test statistic showing us that it has reduced the association of female and male to words as well as reducing the preferential treatment some words will have for males and others for females as seen in below (Table. 2).

	Original model	Hard debiased model	Soft debiased model
Effect size	0.937182	0.713975	0.902294
Test statistic	-0.135519	-0.112795	-0.115080

Table 2

Positive/Negative test statistics indicate in which direction the words are on average more biased.

Human analysis

Analysis of the analogies was done by deciding whether I believed that the analogy seemed to favour one gender, e.g.: she is to a registered nurse as he is to a physician, and whether the analogy had been chosen on an unreasonable basis of gender, e.g.: she is to netball as he is to rugby, I acquired these mean ‘bias scores’:

Average bias score for original model	Average bias score for hard debiased model	Average bias score for soft debiased model
0.35616348	0.194029851	0.2564103

Table 3

Section 5 – Conclusion + Final thoughts

To conclude it is undeniable that hard debiasing is the most effective method of reducing bias in word embeddings. Due to having the lowest human bias score, lowest effect size and lowest test statistic, it is the most effective. The low-test statistic demonstrates there are fewer words related to she/he than there are in the other models which are more likely to reduce bias. This also means that words are much more likely to make contextual sense.

Looking at the soft debiasing method, we can see that it also had a low test statistic but the effect size did not massively decrease from the original model. This is because by trying to maintain as much of the original association as we could we include more words that are associated with gender. While this may make sense in one context it is important to note that humans do not perceive bias in the same way that a machine does. The machine has measured bias by the average difference in cosine similarity between male and the attribute words and female and the attribute words. This means it is given a floating-point value; humans simply see things as biased or not. Therefore, by maintaining more words associated with gender that do not necessarily need to be, a human observer is more likely to perceive a model as being more biased.

Therefore, I reason that it can be stated that hard debiasing is effective at reducing bias in word embeddings to a great extent, however, this would not remove all bias from an NLP model as it is important to recall that there are multiple sources of bias. Not only can bias manifest within the word embeddings, but also in parts of the creative process such as research design. Especially with the debiasing methods that were used in this investigation, there is a heavy focus on the two genders male and female. This means that people who identify as other genders are not aptly included within the model's understanding of gender.

To truly reduce all bias, it would be essential to approach the problem in a variety of ways wherein looking at the word embeddings would only be one of them.

Bibliography

Bolukbasi, T., Chang, K.-W., Zou, J., Saligrama, V., & Kalai, A. (2016). *Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings*. https://arxiv.org/pdf/1607.06520

Borisov, O. (2020, December 15). *Word Embeddings: Intuition behind the vector representation of the words*. Medium. https://towardsdatascience.com/word-embeddings-intuition-behind-the-vector-representation-of-the-words-7e4eb2410bba

Google Code Archive - Long-term storage for Google Code Project Hosting. (2019). Google.com. https://code.google.com/archive/p/word2vec/

Hovy, D., & Prabhumoye, S. (2021). Five sources of bias in natural language processing. *Language and Linguistics Compass*, *15*(8). https://doi.org/10.1111/lnc3.12432

hu, jiawei. (2020, March 6). *An Overview for Text Representations in NLP*. Medium. https://towardsdatascience.com/an-overview-for-text-representations-in-nlp-311253730af1

Kurita, K., Vyas, N., Pareek, A., Black, A. W., & Tsvetkov, Y. (2019, August 1). *Measuring Bias in Contextualized Word Representations*. ACLWeb; Association for Computational Linguistics. https://doi.org/10.18653/v1/W19-3823

Mk, S. (2020, October 5). *sagarmk/Cosine-similarity-from-scratch-on-webpages*. GitHub. https://github.com/sagarmk/Cosine-similarity-from-scratch-on-webpages

Nielsen, L. (2021, August 4). *10 Use-Cases in everyday business operations using NLP*. MLearning.ai. https://medium.com/mlearning-ai/10-use-cases-in-everyday-business-operations-using-nlp-af49b9650d8f

NLP-Word2Vec-Embeddings(pretrained). (2018, February 1). Www.kaggle.com. https://www.kaggle.com/datasets/pkugoodspeed/nlpword2vecembeddingspretrained?select=GoogleNews-vectors-negative300.bin

OpenAI. (2022, November 30). *ChatGPT: Optimizing Language Models for Dialogue*. OpenAI. https://openai.com/blog/chatgpt

Sharma, Y. (2021, June 17). *Gensim Word2Vec - A Complete Guide - AskPython*. https://www.askpython.com/python-modules/gensim-word2vec

Tolga. (2023, March 3). *Debiaswe: try to make word embeddings less sexist*. GitHub. https://github.com/tolga-b/debiaswe

Appendix

Appendix A:

Code written by me and used from (Tolga, 2023).

Tools.py:

```
from __future__ import print_function, division
import re
import sys
import numpy as np
import scipy.sparse
from sklearn.decomposition import PCA
if sys.version_info[0] < 3:
    import io
    open = io.open
else:
    unicode = str
"""
Tools for debiasing word embeddings
Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings
Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai
2016
"""

DEFAULT_NUM_WORDS = 27000
FILENAMES = {"g_wiki": "glove.6B.300d.small.txt",
             "g_twitter": "glove.twitter.27B.200d.small.txt",
             "g_crawl": "glove.840B.300d.small.txt",
             "w2v": "GoogleNews-word2vec.small.txt",
             "w2v_large": "GoogleNews-word2vec.txt"}


def dedup(seq):
    seen = set()
    return [x for x in seq if not (x in seen or seen.add(x))]
```

```
def safe_word(w):
    # ignore words with numbers, etc.
    # [a-zA-Z\.'_\- :;\(\)\]] for emoticons
    return (re.match(r"^[a-z_]*$", w) and len(w) < 20 and not re.match(r"^_*$", w))


def to_utf8(text, errors='strict', encoding='utf8'):
    """Convert a string (unicode or bytestring in `encoding`), to bytestring in
utf8."""
    if isinstance(text, unicode):
        return text.encode('utf8')
    # do bytestring -> unicode -> utf8 full circle, to ensure valid utf8
    return unicode(text, encoding, errors=errors).encode('utf8')


class WordEmbedding:
    def __init__(self, fname):
        self.thresh = None
        self.max_words = None
        self.desc = fname
        print("*** Reading data from " + fname)
        if fname.endswith(".bin"):
            import gensim.models
            model =gensim.models.KeyedVectors.load_word2vec_format(fname,
binary=True)
            words = sorted([w for w in model.vocab], key=lambda w:
model.vocab[w].index)
            vecs = [model[w] for w in words]
        else:
            vecs = []
            words = []

            with open(fname, "r", encoding='utf8') as f:
                for line in f:
                    s = line.split()
                    v = np.array([float(x) for x in s[1:]])
                    if len(vecs) and vecs[-1].shape!=v.shape:
                        print("Got weird line", line)
                        continue
    #                  v /= np.linalg.norm(v)
                    words.append(s[0])
                    vecs.append(v)
        self.vecs = np.array(vecs, dtype='float32')
        print(self.vecs.shape)
        self.words = words
        self.reindex()
        norms = np.linalg.norm(self.vecs, axis=1)
        if max(norms)-min(norms) > 0.0001:
            self.normalize()

    def reindex(self):
```

```
        self.index = {w: i for i, w in enumerate(self.words)}
        self.n, self.d = self.vecs.shape
        assert self.n == len(self.words) == len(self.index)
        self._neighbors = None
#        print(self.n, "words of dimension", self.d, ":", ", ".join(self.words[:4] +
["..."] + self.words[-4:]))

    def v(self, word):
        try:
            return self.vecs[self.index[word]]
        except KeyError:
            return np.array([0])
    def w(self,vector):
        return self.words[np.where(self.vecs==vector)[0]]
    def diff(self, word1, word2):
        v = self.vecs[self.index[word1]] - self.vecs[self.index[word2]]
        return v/np.linalg.norm(v)

    def normalize(self):
        self.desc += ", normalize"
        self.vecs /= np.linalg.norm(self.vecs, axis=1)[:, np.newaxis]
        self.reindex()

    def shrink(self, numwords):
        self.desc += ", shrink " + str(numwords)
        self.filter_words(lambda w: self.index[w]<numwords)

    def filter_words(self, test):
        """
        Keep some words based on test, e.g. lambda x: x.lower()==x
        """
        self.desc += ", filter"
        kept_indices, words = zip(*[[i, w] for i, w in enumerate(self.words) if
test(w)])
        self.words = list(words)
        self.vecs = self.vecs[kept_indices, :]
        self.reindex()

    def save(self, filename):
        with open(filename, "w") as f:
            f.write("\n".join([w+" " + " ".join([str(x) for x in v]) for w, v in
zip(self.words, self.vecs)]))
#        print("Wrote", self.n, "words to", filename)

    def save_w2v(self, filename, binary=True):
        with open(filename, 'wb') as fout:
            fout.write(to_utf8("%s %s\n" % self.vecs.shape))
            # store in sorted order: most frequent words at the top
            for i, word in enumerate(self.words):
                row = self.vecs[i]
                if binary:
```

```
                    fout.write(to_utf8(word) + b" " + row.tostring())
                else:
                    fout.write(to_utf8("%s %s\n" % (word, ' '.join("%f" % val for
val in row))))

    def remove_directions(self, directions): #directions better be orthogonal
        self.desc += ", removed"
        for direction in directions:
            self.desc += " "
            if type(direction) is np.ndarray:
                v = direction / np.linalg.norm(direction)
                self.desc += "vector "
            else:
                w1, w2 = direction
                v = self.diff(w1, w2)
                self.desc += w1 + "-" + w2
            self.vecs = self.vecs - self.vecs.dot(v)[:,
np.newaxis].dot(v[np.newaxis, :])
        self.normalize()

    def compute_neighbors_if_necessary(self, thresh, max_words):
        thresh = float(thresh) # dang python 2.7!
        if self._neighbors is not None and self.thresh == thresh and self.max_words
== max_words:
            return
#       print("Computing neighbors")
        self.thresh = thresh
        self.max_words = max_words
        vecs = self.vecs[:max_words]
        dots = vecs.dot(vecs.T)
        dots = scipy.sparse.csr_matrix(dots * (dots >= 1-thresh/2))
        from collections import Counter
        rows, cols = dots.nonzero()
        nums = list(Counter(rows).values())
#       print("Mean:", np.mean(nums)-1)
#       print("Median:", np.median(nums)-1)
        rows, cols, vecs = zip(*[(i, j, vecs[i]-vecs[j]) for i, j, x in zip(rows,
cols, dots.data) if i<j])
        self._neighbors = rows, cols, np.array([v/np.linalg.norm(v) for v in vecs])

    def neighbors(self, word, thresh=1):
        dots = self.vecs.dot(self.v(word))
        return [self.words[i] for i, dot in enumerate(dots) if dot >= 1-thresh/2]

    def more_words_like_these(self, words, topn=50, max_freq=100000):
        v = sum(self.v(w) for w in words)
        dots = self.vecs[:max_freq].dot(v)
        thresh = sorted(dots)[-topn]
        words = [w for w, dot in zip(self.words, dots) if dot>=thresh]
        return sorted(words, key=lambda w: self.v(w).dot(v))[-topn:][::-1]
```

```
    def best_analogies_dist_thresh(self, v, thresh=1, topn=500, max_words=50000):
        """Metric is cos(a-c, b-d) if |b-d|^2 < thresh, otherwise 0
        """
        vecs, vocab = self.vecs[:max_words], self.words[:max_words]
        self.compute_neighbors_if_necessary(thresh, max_words)
        rows, cols, vecs = self._neighbors
        scores = vecs.dot(v/np.linalg.norm(v))
        pi = np.argsort(-abs(scores))

        ans = []
        usedL = set()
        usedR = set()
        for i in pi:
            if abs(scores[i])<0.001:
                break
            row = rows[i] if scores[i] > 0 else cols[i]
            col = cols[i] if scores[i] > 0 else rows[i]
            if row in usedL or col in usedR:
                continue
            usedL.add(row)
            usedR.add(col)
            ans.append((vocab[row], vocab[col], abs(scores[i])))
            if len(ans)==topn:
                break

        return ans


def viz(analogies):
    print("\n".join(str(i).rjust(4)+a[0].rjust(29) + " | " + a[1].ljust(29) +
(str(a[2]))[:4] for i, a in enumerate(analogies)))


def text_plot_words(xs, ys, words, width = 90, height = 40, filename=None):
    PADDING = 10 # num chars on left and right in case words spill over
    res = [[' ' for i in range(width)] for j in range(height)]
    def rescale(nums):
        a = min(nums)
        b = max(nums)
        return [(x-a)/(b-a) for x in nums]
    print("x:", (min(xs), max(xs)), "y:",(min(ys),max(ys)))
    xs = rescale(xs)
    ys = rescale(ys)
    for (x, y, word) in zip(xs, ys, words):
        i = int(x*(width - 1 - PADDING))
        j = int(y*(height-1))
        row = res[j]
        z = list(row[i2] != ' ' for i2 in range(max(i-1, 0), min(width, i +
len(word) + 1)))
        if any(z):
            continue
```

```
        for k in range(len(word)):
            if i+k>=width:
                break
            row[i+k] = word[k]
    string = "\n".join("".join(r) for r in res)
#      return string
    if filename:
        with open(filename, "w", encoding="utf8") as f:
            f.write(string)
        print("Wrote to", filename)
    else:
        print(string)


def doPCA(pairs, embedding, num_components = 10):
    matrix = []
    for a, b in pairs:
        center = (embedding.v(a) + embedding.v(b))/2
        matrix.append(embedding.v(a) - center)
        matrix.append(embedding.v(b) - center)
    matrix = np.array(matrix)
    pca = PCA(n_components = num_components)
    pca.fit(matrix)
    # bar(range(num_components), pca.explained_variance_ratio_)
    return pca


def drop(u, v):
    return u - v * u.dot(v) / v.dot(v)
```

Main.py

```
import numpy as np
import pandas as pd
from tools import WordEmbedding
import learn_gender_specific
import data
import json
import debias


#constants
delta = 1
lambd = 0.2
analogies = []
hard_analogies = []
soft_analogies = []

bias_model = WordEmbedding("w2v_gnews_small.txt")
hard_model = WordEmbedding("w2v_gnews_small.txt")
soft_model = WordEmbedding("w2v_gnews_small.txt")
```

```python
#Load jobs
professions = data.load_professions()
professions_words = [p[0] for p in professions]
#Find gender direction of subspacea
gender_Dir = bias_model.diff('she', 'he')
#Create analogies between gender and professions
gender_analogies = bias_model.best_analogies_dist_thresh(gender_Dir)
for (a,b,c) in gender_analogies:
    analogies.append([a, b])


with open('./data/definitional_pairs.json', "r") as f:
    defs = json.load(f)

with open('./data/equalize_pairs.json', "r") as f:
    equalize_pairs = json.load(f)

with open('./data/gender_specific_seed.json', "r") as f:
    gender_specific_words = json.load(f)

debias.debias(hard_model, gender_specific_words, defs, equalize_pairs)


#Soft debiasing

vectors = soft_model.vecs

def debias_single_word(embedding, W, lambd):
    x_i = embedding
    I = np.eye(len(x_i))

    # Calculate debiased embedding using the equation
    x_i_db = np.linalg.inv(I + lambd * W) @ x_i

    return x_i_db

W = np.outer(gender_Dir, gender_Dir)
for count, i in enumerate(vectors):
    print(count)
    soft_model.vecs[count] = debias_single_word(i, W, lambd)


def cosine_similarity(a, b):
    return np.dot(a,b)/(np.linalg.norm(a)*np.linalg.norm(b))

#Define method for returning the effect size
def weat_value(target1, target2, attribute1, attribute2, permutations=False):


    # Compute the mean embeddings for the target and attribute words
    t1_embedding = np.mean([word for word in target1], axis=0)
    t2_embedding = np.mean([word for word in target2], axis=0)
```

```
    a1_embedding = np.mean([word for word in attribute1], axis=0)
    a2_embedding = np.mean([word for word in attribute2], axis=0)


    # Compute the cosine similarities between the target and attribute embeddings
    sim_t1_a2 = cosine_similarity(t1_embedding, a2_embedding)
    sim_t2_a1 = cosine_similarity(t2_embedding, a1_embedding)
    sim_t2_a2 = cosine_similarity(t2_embedding, a2_embedding)
    sim_t1_a1 = cosine_similarity(t1_embedding, a1_embedding)

    # Compute the means and standard deviations of the cosine similarities
    mean_t1_a = np.mean([sim_t1_a1, sim_t1_a2])
    mean_t2_a = np.mean([sim_t2_a1, sim_t1_a2])
    std_t1_a = np.std([sim_t1_a1, sim_t1_a2])
    std_t2_a = np.std([sim_t2_a1, sim_t2_a2])

    # Compute the effect size using Cohen's d
    effect_size = (mean_t1_a - mean_t2_a) / np.sqrt((std_t1_a ** 2 + std_t2_a ** 2)
/ 2)

    value = mean_t1_a-mean_t2_a
    return effect_size, value




equalize_pairs = np.array(equalize_pairs)
female_vectors = [bias_model.v(word) for word in equalize_pairs[:,0] if
np.linalg.norm(bias_model.v(word)) != 0]
male_vectors = [bias_model.v(word) for word in equalize_pairs[:,1] if
np.linalg.norm(bias_model.v(word)) != 0]


bias_analogy1_vectors = [bias_model.v(word[0]) for word in analogies]
bias_analogy2_vectors = [bias_model.v(word[1]) for word in analogies]
analogies = np.array(analogies)


#Find gender direction of subspace
gender_Dir = hard_model.diff('she', 'he')
#Create analogies between gender and professions
hard_gender_analogies = hard_model.best_analogies_dist_thresh(gender_Dir)
for (a,b,c) in hard_gender_analogies:
    hard_analogies.append([a, b])

hard_analogy1_vectors = [hard_model.v(word[0]) for word in hard_analogies]
hard_analogy2_vectors = [hard_model.v(word[1]) for word in hard_analogies]
hard_analogies = np.array(hard_analogies)


#Find gender direction of subspace
gender_Dir = soft_model.diff('she', 'he')
```

```python
#Create analogies between gender and professions
soft_gender_analogies = soft_model.best_analogies_dist_thresh(gender_Dir)
for (a,b,c) in soft_gender_analogies:
    soft_analogies.append([a, b])

soft_analogy1_vectors = [soft_model.v(word[0]) for word in soft_analogies]
soft_analogy2_vectors = [soft_model.v(word[1]) for word in soft_analogies]
soft_analogies = np.array(soft_analogies)

data = {"She(Bias)":analogies[:152,0], "He(Bias)":analogies[:152,1],
"She(Hard)":hard_analogies[:,0], "He(Hard)":hard_analogies[:,1],
"She(Soft)":soft_analogies[:152,0], "He(Soft)":soft_analogies[:152,1]}

df = pd.DataFrame(data)
df.to_csv("Results.csv")
WEAT_hard = weat_value(female_vectors, male_vectors, hard_analogy1_vectors,
hard_analogy2_vectors)
WEAT_soft = weat_value(female_vectors, male_vectors, soft_analogy1_vectors,
soft_analogy2_vectors)
WEAT_bias = weat_value(female_vectors, male_vectors, bias_analogy1_vectors,
bias_analogy2_vectors)

weats = {"WEAT for bias (Effect size | Weat)":WEAT_bias, "WEAT for hard (Effect
size | Weat)":WEAT_hard, "WEAT for soft (Effect size | Weat)":WEAT_soft}
print(pd.DataFrame(weats))
```

Data.py

```python
import json
import os

"""
Tools for data operations

Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings
Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai
2016
"""
PKG_DIR = os.path.dirname(os.path.abspath(__file__))


def load_professions():
    professions_file = os.path.join(PKG_DIR, 'data', 'professions.json')
    with open(professions_file, 'r') as f:
        professions = json.load(f)

    return professions
```

Debias.py

```python
rom __future__ import print_function, division
```

```python
import tools
import json
import numpy as np
import argparse
import sys
if sys.version_info[0] < 3:
    import io
    open = io.open
"""
Hard-debias embedding

Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings
Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai
2016
"""


def debias(E, gender_specific_words, definitional, equalize):
    gender_direction = tools.doPCA(definitional, E).components_[0]
    specific_set = set(gender_specific_words)
    for i, w in enumerate(E.words):
        if w not in specific_set:
            E.vecs[i] = tools.drop(E.vecs[i], gender_direction)
    E.normalize()
    candidates = {x for e1, e2 in equalize for x in [(e1.lower(), e2.lower()),
                                                     (e1.title(), e2.title()),
                                                     (e1.upper(), e2.upper())]}

    for (a, b) in candidates:
        if (a in E.index and b in E.index):
            y = tools.drop((E.v(a) + E.v(b)) / 2, gender_direction)
            z = np.sqrt(1 - np.linalg.norm(y)**2)
            if (E.v(a) - E.v(b)).dot(gender_direction) < 0:
                z = -z
            E.vecs[E.index[a]] = z * gender_direction + y
            E.vecs[E.index[b]] = -z * gender_direction + y
    E.normalize()


if __name__ == "__main__":

    parser = argparse.ArgumentParser()
    parser.add_argument("embedding_filename", help="The name of the embedding")
    parser.add_argument("definitional_filename", help="JSON of definitional pairs")
    parser.add_argument("gendered_words_filename", help="File containing words not
to neutralize (one per line)")
    parser.add_argument("equalize_filename", help="???.bin")
    parser.add_argument("debiased_filename", help="???.bin")

    args = parser.parse_args()
```

```
    print(args)

    with open(args.definitional_filename, "r") as f:
        defs = json.load(f)
    print("definitional", defs)

    with open(args.equalize_filename, "r") as f:
        equalize_pairs = json.load(f)

    with open(args.gendered_words_filename, "r") as f:
        gender_specific_words = json.load(f)
    print("gender specific", len(gender_specific_words),
gender_specific_words[:10])

    E = tools.WordEmbedding(args.embedding_filename)

    print("Debiasing...")
    debias(E, gender_specific_words, defs, equalize_pairs)

    print("Saving to file...")
    if args.embedding_filename[-4:] == args.debiased_filename[-4:] == ".bin":
        E.save_w2v(args.debiased_filename)
    else:
        E.save(args.debiased_filename)

    print("\n\nDone!\n")
```

Appendix B:

Table of analogies

She(Bias)	He(Bias)	She(Hard)	He(Hard)	She(Soft)	He(Soft)
she	he	woman	man	she	he
herself	himself	sorority	fraternity	herself	himself
her	his	twin_sister	twin_brother	her	his
woman	man	ladies	gentlemen	woman	man
daughter	son	mare	gelding	daughter	son
businesswoman	businessman	estrogen	testosterone	businesswoman	businessman
girl	boy	sister	brother	girl	boy
actress	actor	grandma	grandpa	actress	actor
chairwoman	chairman	congresswoman	congressman	chairwoman	chairman
heroine	hero	aunt	uncle	heroine	hero

mother	father	spokeswoman	spokesman	mother	father
spokeswoman	spokesman	gals	dudes	spokeswoman	spokesman
sister	brother	daughter	son	sister	brother
girls	boys	moms	dads	girls	boys
sisters	brothers	councilwoman	councilman	sisters	brothers
queen	king	mothers	fathers	gal	guy
niece	nephew	sisters	brothers	queen	king
councilwoman	councilman	grandmother	grandfather	niece	nephew
motherhood	fatherhood	queens	kings	councilwoman	councilman
women	men	granddaughter	grandson	motherhood	fatherhood
petite	lanky	ovarian_cancer	prostate_cancer	women	men
ovarian_cancer	prostate_cancer	ex_boyfriend	ex_girlfriend	petite	lanky
Anne	John	granddaughters	grandsons	ovarian_cancer	prostate_cancer
schoolgirl	schoolboy	girl	boy	Anne	John
granddaughter	grandson	mother	father	schoolgirl	schoolboy
aunt	uncle	husbands	wives	granddaughter	grandson
matriarch	patriarch	filly	colt	aunt	uncle
twin_sister	twin_brother	queen	king	matriarch	patriarch
mom	dad	schoolgirl	schoolboy	Carrie	Greg
lesbian	gay	chairwoman	chairman	twin_sister	twin_brother
husband	younger_brother	princess	prince	mom	dad
gal	dude	she	he	Sarah	Brett
lady	gentleman	herself	himself	gals	guys
sorority	fraternity	girls	boys	lesbian	gay
mothers	fathers	daughters	sons	husband	younger_brother
grandmother	grandfather	female	male	Keisha	Jermaine
blouse	shirt	females	males	lady	gentleman
soprano	baritone	convent	monastery	sorority	fraternity
queens	kings	her	his	mothers	fathers
Jill	Greg	motherhood	fatherhood	hers	theirs

daughters	sons	mom	dad	grandmothe r	grandfather
grandma	grandpa	niece	nephew	blouse	shirt
volleyball	football	women	men	soprano	baritone
diva	superstar	businesswo man	businessm an	queens	kings
mommy	kid	actress	actor	daughters	sons
Sarah	Matthew	gal	dude	grandma	grandpa
hairdresser	barber	lesbian	gay	sassy	brash
softball	baseball	compatriot	countryma n	volleyball	football
goddess	god	husband	younger_br other	diva	superstar
Aisha	Jamal	heroine	protagonist	chick	dude
waitress	waiter	actresses	actors	mommy	kid
princess	prince	housewife	homemake r	hairdresser	barber
filly	colt	waitress	waiter	softball	baseball
mare	gelding	aunts	uncles	goddess	god
ladies	gentlemen	feminism	feminist	Aisha	Jamal
childhood	boyhood	mustache	beard	waitress	waiter
interior_desi gner	architect	hers	theirs	princess	prince
nun	priest	kid	guy	filly	colt
wig	beard	fella	gentleman	mare	gelding
granddaught ers	grandsons	nieces	nephews	ladies	gentlemen
girlfriends	buddies	teenage_girls	teenagers	Emily	Matthew
gals	dudes	nun	monk	childhood	boyhood
aunts	uncles	stepdaughter	stepson	interior_desi gner	architect
congresswo man	congressman	childhood	boyhood	nun	priest
feminism	conservatism	mommy	daddy	wig	beard
bitch	bastard	me	him	granddaught ers	grandsons
hers	yours	goddess	god	mammogra m	prostate
bra	pants	viagra	cialis	girlfriends	buddies
moms	dads	diva	superstar	aunts	uncles
nurse	surgeon	fillies	colts	congresswo man	congressman
heiress	magnate	brides	bridal	feminism	conservatism
feminine	manly	matriarch	patriarch	heiress	billionaire

glamorous	flashy	maid	housekeeper	bitch	bastard
actresses	actors	hostess	bartender	bra	pants
registered_nurse	physician	vagina	penis	moms	dads
cupcakes	pizzas	mama	fella	nurse	surgeon
blond	burly	teenage_girl	teenager	feminine	manly
babe	fella	stepmother	eldest_son	glamorous	flashy
mums	blokes	ballerina	dancer	actresses	actors
gorgeous	magnificent	maternity	midwives	registered_nurse	physician
compatriot	countryman	grandmothers	grandparents	cupcakes	pizzas
fabulous	terrific	compatriots	countrymen	blond	burly
breast	prostate	witch	witchcraft	babe	fella
starlet	youngster	boyfriend	stepfather	mums	blokes
Laurie	Brett	uterus	intestine	gorgeous	magnificent
kids	guys	menopause	puberty	compatriot	countryman
sewing	carpentry	heiress	socialite	fabulous	terrific
kinda	guy	bride	wedding	starlet	youngster
headscarf	turban	lesbians	gays	sewing	carpentry
siblings	elder_brother	eldest	elder_brother	headscarf	turban
charming	affable	politician	statesman	siblings	elder_brother
sassy	snappy	maids	servants	charming	affable
cosmetics	pharmaceuticals	dictator	strongman	cosmetics	pharmaceuticals
estrogen	testosterone	youngster	lad	estrogen	testosterone
handbag	briefcase	nuns	priests	handbag	briefcase
housewife	shopkeeper	maternal	infant_mortality	housewife	shopkeeper
fillies	colts	hubby	pal	sexy	macho
nieces	nephews	blokes	bloke	fillies	colts
whore	coward	lady	waitress	Jill	Todd
boyfriend	pal	soprano	baritone	nieces	nephews
salon	barbershop	girlfriends	buddies	whore	coward
Latonya	Leroy	boyfriends	girlfriend	boyfriend	pal
vagina	penis	facial_hair	beards	salon	barbershop
breast_cancer	lymphoma	womb	fetus	Latonya	Leroy
vocalist	guitarist	businesspeople	businessmen	vagina	penis

me	him	fiance	roommate	breast_cancer	lymphoma
children	youngsters	beau	lover	vocalist	guitarist
adorable	goofy	salesperson	salesman	me	him
giggling	grinning	witches	vampires	children	youngsters
cheerful	jovial	estranged_husband	estranged	adorable	goofy
lovely	brilliant	counterparts	brethren	giggling	grinning
giggle	chuckle	bastard	chap	netball	sevens
bras	trousers	widow	deceased	cheerful	jovial
wedding_dress	tuxedo	obstetrics	pediatrics	lovely	brilliant
singer	frontman	spokespeople	spokesmen	Laurie	Neil
netball	rugby	friendship	brotherhood	middle_blocker	redshirt_freshman
rebounder	playmaker	hens	chickens	giggle	chuckle
vocalists	trumpeter	hen	cock	bras	trousers
nude	shirtless	replied	sir	wedding_dress	tuxedo
beautiful	majestic	colon	prostate	singer	frontman
feisty	mild_mannered	mistress	prostitute	rebounder	playmaker
feminists	socialists	stallion	stud	vocalists	trumpeter
nanny	chauffeur	manly	macho	nude	shirtless
females	males	wife	cousin	beautiful	majestic
pediatrician	orthopedic_surgeon	ma	na	feisty	mild_mannered
teenage_girls	youths	carpenter	handyman	feminists	socialists
pink	red	bulls	bull	nanny	chauffeur
convent	monastery	widows	families	females	males
midwife	doctor	salespeople	salesmen	pediatrician	orthopedic_surgeon
feminist	liberal	girlfriend	friend	teenage_girls	youths
gown	blazer	suitor	takeover_bid	pink	red
blonde	blond	gaffer	lads	convent	monastery
stepdaughter	stepson	semen	saliva	midwife	doctor
wonderful	great	elephants	lions	gown	blazer
breasts	genitals	suitors	bidders	blonde	blond
luscious	crisp	fiancee	married	stepdaughter	stepson

judgmental	arrogant	guys	fellas	wonderful	great
skirts	shorts	hair_salon	barbershop	breasts	genitals
middle_aged	bearded	elephant	lion	luscious	crisp
spokespeople	spokesmen	colts	mares	judgmental	arrogant
female	male	pa	mo	skirts	shorts
beauty	grandeur	footy	blokes	middle_aged	bearded
salesperson	salesman	aldermen	councilmen	spokespeople	spokesmen
witch	demon	monks	monasteries	female	male
male_counterparts	counterparts	widower	widowed	beauty	grandeur
violinist	virtuoso	bachelor	bachelor_degree	salesperson	salesman
practicality	durability	sperm	embryos	witch	demon
boobs	ass	deer	elk	male_counterparts	counterparts
dolls	replicas	residence_halls	fraternities	violinist	virtuoso
husbands	wives	wedlock	fathered	practicality	durability
ponytail	mustache	penis	genitals	boobs	ass
sexism	racism	princes	royals	dolls	replicas