



# Assignment 1

## Experiments with Text Processing Tools

T-725-MALV, Málvinnsla - Natural Language Processing  
Reykjavik University - School of Computer Science, Menntavegi 1, IS-101 Reykjavík, Iceland

Damiano Pasquini  
damiano23@ru.is

20. September 2023

## Contents

<b>1</b>	<b>Python: Gutenberg corpus</b>	<b>2</b>
<b>2</b>	<b>Language Modeling</b>	<b>3</b>
2.1	AWK command . . . . .	3
2.2	Trigram Frequency . . . . .	3
2.3	Informations about <i>eng.sent</i> . . . . .	3
2.4	Maximum Likelihood Estimation . . . . .	4
<b>3</b>	<b>Bias in Word Embeddings</b>	<b>5</b>
3.1	Introduction . . . . .	5
3.2	Biases Research . . . . .	5
3.3	Biases Visualization and Removal . . . . .	7
3.4	Conclusions . . . . .	8
<b>4</b>	<b>Case Study - GPT</b>	<b>10</b>
4.1	Introduction . . . . .	10
4.2	Experiments . . . . .	10
4.3	Conclusions . . . . .	15

# 1 Python: Gutenberg corpus

In this first point, it was asked to write a Python file that, given the name of a specific Gutenberg text obtained from NLTK library [5], prints out some information about the corpus of that particular text. The corpus name asked is *austen-emma.txt* and the following are the results:

Information	Result
Text	austen-emma.txt
Tokens	196546
Types	7738
Types excluding stop words	7609
10 most common tokens	[(',', 12018), (',', 8882), ('to', 5183), ('the', 4844), ...
Long types	['discouragement', 'recommendations', ...
Nouns ending in "ation"	['elevation', 'inspiration', 'deviation', 'exaggeration', ...

Table 1: Results obtained from the Python file *corpusAnalysis.py*. The complete output can be found in *output.txt*.

An example of the code written to get this informations is:

Listing 1: Part of the code from the task 1

```
# Get the 10 most common tokens
fdist = nltk.FreqDist(tokens)
print("10 most common tokens: " + str(fdist.most_common(10)))
f.write("10 most common tokens: "+str(fdist.most_common(10))+"\n")

# Get the long types
long_types = [word for word in types if len(word) > 13]
print("Long types: " + str(long_types))
f.write("Long types: " + str(long_types) + "\n")
```

## 2 Language Modeling

This point asks to write the proper UNIX commands in order to develop an English trigram language model based on the file *eng.sent*<sup>1</sup>. The UNIX commands to use are: **awk**, **head**, **tail**, **paste**, **sort**, **uniq**, **wc**.

### 2.1 AWK command

The first task is to take in input the file *eng.sent*, which is pre-tokenised in the wrong way since the tokens are not in separate lines, and then fix it using **awk** command through UNIX. The input must be called *eng.tok*. Initially, I noticed that the “/” character was used to divide two tags of a single word, but it is always followed by “SYM”, so it should be the tag of the “/” character and this certifies that in *eng.sent* each word (for both tokens and tags) is divided by a space.

Listing 2: Code from the task 2.1

```
awk '{
    for(i=1; i<=NF; i+=2) {
        printf "%s\n", $i
    }
}' eng.sent > eng.tok
```

### 2.2 Trigram Frequency

In this second task is asked to construct a trigram frequency file named *engTri.freq* sorted in descended order of frequency, containing four columns: *frequency*, *word<sub>1</sub>*, *word<sub>2</sub>* and *word<sub>3</sub>*. In the following two lines of code, this is done with the commands **awk**, **uniq** and **sort**.

The **awk** command in the first line reads *eng.tok* and generates trigrams (sets of three consecutive words) from the input file, storing them in *eng.trigrams* (used as an intermediate file).

The **sort** command in line 2 sorts the trigrams alphabetically, and then **uniq -c** counts the frequency of each unique trigram, resulting in a list of trigrams with their frequencies, while the second **sort -nr** command sorts the trigram frequency list in descending order based on the frequency count. Follows the shell code:

Listing 3: Code from the task 2.2

```
awk '{
    prev1 = prev2; prev2 = prev3; prev3 = $0;
    if (NR >= 3) print prev1, prev2, prev3
}' eng.tok > eng.trigrams
sort eng.trigrams | uniq -c | sort -nr > engTri.freq
```

### 2.3 Informations about *eng.sent*

This third task asked to calculate with the commands **awk** and **wc**, through the file *engTri.freq* before the number of trigrams and then the number of distinct trigrams existing in the file *engTri.freq*. This is done as follows:

---

<sup>1</sup>The complete implementation can be seen at the repository [https://github.com/damiano00/T\\_725\\_MALV\\_Natural\\_Language\\_Processing/tree/main/assignment\\_1](https://github.com/damiano00/T_725_MALV_Natural_Language_Processing/tree/main/assignment_1)

Listing 4: Code from the task 2.3

```
# amount of trigrams
awk '{sum += $1} END {print sum}' engTri.freq
# amount of distinct trigrams
awk '{print $2, $3, $4}' engTri.freq | wc -l
```

The output given *engTri.freq* is the following:

```
204564
166451
```

## 2.4 Maximum Likelihood Estimation

This last task asks to calculate, through the file *engTri.freq* containing the trigrams and their frequencies, the **Maximum Likelihood Estimation** (MLE) value given by  $P(\textit{Monday} \mid \textit{said on})$ . In this case, the MLE is the probability that the word *Monday* comes after the words *said on*, and this calculation is possible thanks to the trigrams and their associated frequencies. The resulting MLE is 0.0956938.

The UNIX command used to calculate it is the following:

Listing 5: Code from the task 2.4

```
awk '
BEGIN {
word1 = "said"; word2 = "on"; word3 = "Monday";
count_trigram = 0;
count_said_on = 0;
result = 0;
}
{
    if ($2 == word1 && $3 == word2 && $4 == word3) {
        count_trigram += $1;
    }
    if ($2 == word1 && $3 == word2) {
        count_said_on += $1;
    }
}
END {
    result = count_trigram / count_said_on;
    print result;
} ' engTri.freq
```

The resulting **MLE** is:

```
0.0956938
```

## 3 Bias in Word Embeddings

### 3.1 Introduction

In natural language processing (NLP), a **word embedding** is a representation of a word. The embedding is used in text analysis. Typically, the representation is a **real-valued vector** that encodes the meaning of the word in such a way that words that are closer in the vector space are expected to be similar in meaning. Word embeddings can be obtained using language modelling and feature learning techniques, where words or phrases from the vocabulary are mapped to vectors of real numbers [4].

In this experiment, the aim is to use word embeddings in order to find potential biases of various kinds, e.g., denigration, stereotyping, recognition, under-representation, and more specifically **gender** bias, **sexual orientation** bias, **ethnic** bias and **ageism** bias.

The embeddings used in this experiment came from two datasets: glove-wiki-gigaword-100 and d glove-twitter-100 <sup>2</sup>

### 3.2 Biases Research

In the following tables are reported the biases found. Not both datasets contain the same stereotypes and, as a consequence, words are biased in different ways and sometimes there are no relevant biases in the results.

Listing 6: Python code to perform bias research.

```
import warnings
import gensim.downloader as api
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=DeprecationWarning)

wiki_corpus = api.load('glove-wiki-gigaword-100')
twitter_corpus = api.load('glove-twitter-100')

def find_word(corpus, a, b, x):
    # a is to b as x is to
    a = a.lower()
    b = b.lower()
    x = x.lower()
    print(f"> {a}:{b} as {x}:?")
    top_words = corpus.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 find_words_general(a, b, x):
    print("GloVe Twitter\n")
    find_word(twitter_corpus, a, b, x)
    print("\nGloVe Wikipedia\n")
    find_word(wiki_corpus, a, b, x)

find_words_general("female", "normal", "trans")
```

---

<sup>2</sup>Downloadable here: <https://nlp.stanford.edu/projects/glove/>

Tuple (a is to b as x is to biases)	Dataset	Biases
<b>cis normal intersex</b>	Twitter	(0.862) estranho
	Wikipedia	(0.950) abnormal
<b>male ambitious trans</b>	Twitter	none
	Wikipedia	(1.067) privatization
<b>female normal trans</b>	Twitter	none
	Wikipedia	(0.999) disruption

Table 2: Gender biases discovered

The first category of biases is **gender biases**. In the first case with the tuple “cis normal intersex” we check if the word *intersex* (that is a gender identity), is related to some unusual or offensive word that could be a stereotype or a denigrating adjective. Twitter links this word to “estranho” and Wikipedia with “abnormal”, so in both cases the two datasets are stereotyping the “intersex” gender identity. Instead in the second and third case, the **trans** identity is badly addressed only by Wikipedia and not by Twitter. So *gender stereotypes* are present more in Wikipedia than in Twitter.

Tuple (a is to b as x is to biases)	Dataset	Biases
<b>straight attractive gay</b>	Twitter	(0.938) anormal (0.929) pervertido
	Wikipedia	(0.941) noncitizen
<b>straight attractive queer</b>	Twitter	(0.942) detestable (0.927) unattractive (0.907) deviant (0.907) perverted
	Wikipedia	(0.977) low-class
<b>straight capable gay</b>	Twitter	(0.941) anormal (0.932) incapable
	Wikipedia	(0.913) noncitizen

Table 3: Sexual orientation biases discovered

In Figure 3 are instead reported the **sexual orientation biases** emerged. We can see that both datasets contain offensive biases related to the words “gay” and “queer” with more offensive words from Twitter.

The experiments over **ethnic biases** show that in two cases over four Twitter does not address the word “black” with a derogatory attribute, while Wikipedia in all four cases gives at least one negative bias. On the other hand, Twitter shows more derogatory words concerning Wikipedia.

Figure 5 shows **ageism biases** from both the two datasets. Only one example is reported since all the other attempts to find some interesting bias to report were useless and this one is the most significant.

<b>Tuple</b> (a is to b as x is to biases)	<b>Dataset</b>	<b>Biases</b>
<b>white skilled black</b>	Twitter	(0.801) unskilled
	Wikipedia	(0.959) unskilled (0.879) untrained (0.869) unemployed
<b>white talented black</b>	Twitter	none
	Wikipedia	(0.895) inexperienced
<b>western owner black</b>	Twitter	none
	Wikipedia	(0.905) hired
<b>western boss black</b>	Twitter	(1.036) lil (1.017) bitch (1.011) nigga
	Wikipedia	(0.923) clown

Table 4: Ethnic biases discovered

<b>Tuple</b> (a is to b as x is to biases)	<b>Dataset</b>	<b>Biases</b>
<b>young intelligent old</b>	Twitter	(0.851) weird (0.845) bizarre
	Wikipedia	(0.853) antiquated (0.797) creaky (0.796) fusty

Table 5: Ageism biases discovered

### 3.3 Biases Visualization and Removal

The following are the representations between some of the terms previously discovered as biased in Wikipedia and Twitter, additional some terms were added to test if they are biased only using these graphs<sup>3</sup>.

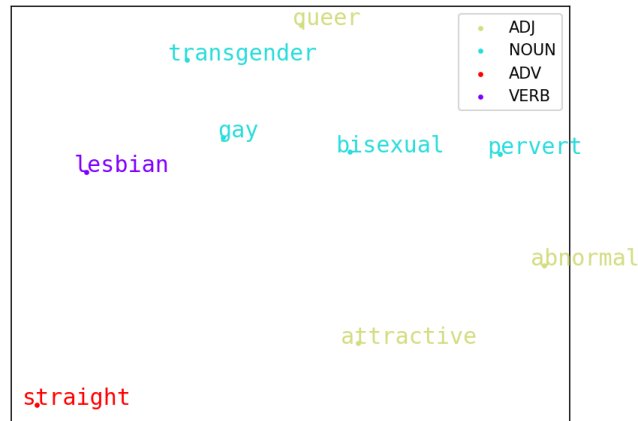


Figure 1: Word embeddings visualization between sexual orientation terms.

<sup>3</sup>These graphs are generated through this website: <http://vectors.nlp.eu/explore/embeddings/en/visual/>

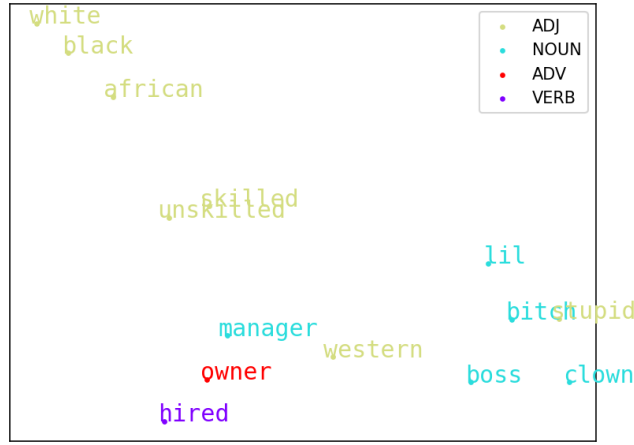


Figure 2: Word embeddings visualization between possible ethnic discrimination terms.

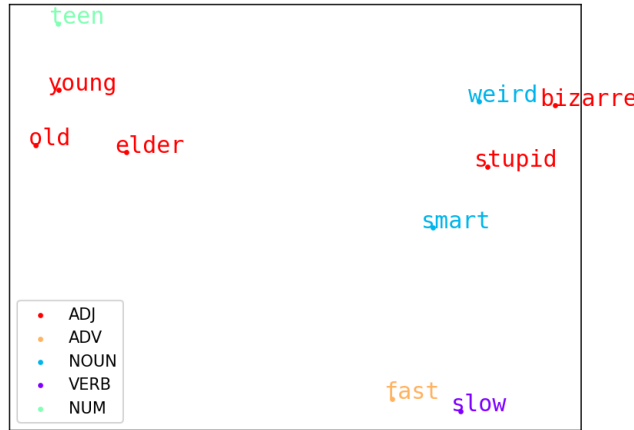


Figure 3: Word embeddings visualization between possible ageism discrimination terms.

Observing these representations from Wikipedia data, we can say that there is no sexual orientation discrimination, but there is some discrimination and stereotypes about the ethnic side and no stereotypes about ageism. Concluding, not all the biases discovered with the previous methods can be discovered through their visualizations. Moreover, some relatively recent research shows how to remove biases in different contexts, like bias removal for hate speech detection [1] or Iterative adversarial removal of gender bias in pre-trained word embeddings [3]. The first research through the Bias Detection Strategy and the Bias Replacement Strategy implements the **De-biased Classifier**.

### 3.4 Conclusions

In conclusion, this experiment investigated the exploration of biases present in word embeddings, specifically focusing on gender, sexual orientation, ethnicity, and ageism biases within the GloVe Twitter and Wikipedia datasets. The analysis revealed varying degrees of biases, with notable differences between the two datasets. Gender stereotypes were more prevalent in Wikipedia, whereas Twitter exhibited fewer stereotypes related to gender identities. Sexual orientation biases were evident in both datasets, with Twitter often containing more offensive associations. Ethnic biases showed disparities, with Twitter being less likely to exhibit derogatory associations compared to Wikipedia. Ageism biases were identified in both datasets, although the presented example was the most significant.

To gain a deeper understanding, visual representations of word embeddings were explored. These



visualizations confirmed the presence of ethnic discrimination but indicated the absence of sexual orientation discrimination and ageism stereotypes in the embeddings.

Furthermore, recent research has highlighted strategies for bias removal in word embeddings, emphasizing the importance of mitigating biases in various contexts.

This experiment underscores the significance of identifying and addressing biases in word embeddings, as they can allow stereotypes and discrimination in natural language processing applications. As the field continues to advance, it is crucial to employ debiasing techniques to ensure fair and unbiased language models and text analysis.

## 4 Case Study - GPT

### 4.1 Introduction

**Generative Pre-trained Transformer 3 (GPT-3)** [2] is a large language model released by OpenAI in 2020. Like its predecessor GPT-2, it is a decoder-only transformer model of deep neural networks, which uses attention in place of previous recurrence and convolution-based architectures [7].

Some examples of the potential applications of a certain model can be **data extraction**, **data generation**, **data transformation**, **code generation and analysis** and last. Specific tasks derived from these are now analyzed more in detail trying to take the most useful capabilities of GPT and its limitations that are actually present in the model.

### 4.2 Experiments

#### Data Extraction: Tweet Classifier

The first capability of GPT is data extraction, and this is done through an example application proposed in OpenAI website as *Tweet classifier*: this example given a tweet detects its sentiment between “negative”, “neutral” and “positive”. Instead of reporting basic cases where the model obviously works based on how single words are classified, it is more interesting to report specific cases where the phrase can be more difficult to classify. Even more interesting is how GPT behaves when the Temperature parameter is changed, so below are reported one basic scenario and a specific scenario with a phrase containing both positive and negative words.

Phrase	Mode	Model	Temp.	OutClass
I loved the new Batman movie!	Chat	GPT-3.5-turbo	0	pos
	Chat	GPT-3.5-turbo	1	pos
	Chat	GPT-3.5-turbo	2	pos

Table 6: Output classification of a phrase quite easy to classify since it contains only positive words, at the changing of *Temperature* parameter.

Phrase	Mode	Model	Temp.	OutClass
Enigmatic mysteries await exploration.	Chat	GPT-3.5-turbo	0	pos
	Chat	GPT-3.5-turbo	1	neutr
	Chat	GPT-3.5-turbo	2	neg

Table 7: Output classification of a phrase that contains neutral words hard to be assigned to one of the three possible classes, at the changing of *Temperature* parameter.

As we can see, given the phrase in Table 6 with easy classifiable words, changing the temperature the model classifies always the same class for all the three inputs. Instead given the phrase in Table 7 with neutral words, changing the temperature the model classifies the phrase each time in a different class. Concluding this first experiment, the hyperparameter “Temperature” is useful since it enables users to tailor the model’s output to suit the specific requirements of a task, balancing between exploration and precision in text generation.

## Data Extraction: Keywords from Text

In a second experiment focused on data extraction, keyword extraction emerged as a highly valuable technique. Its applications encompass a wide range of fields, including automated text summarization [6], contextual relevance, and improved content comprehension. Notably, when we adjust the *presence penalty* parameter, the keywords extracted remain consistent. This is noteworthy because the text includes common words that pose a challenge for extraction when the parameter is increased. This higher value restricts the use of correlations with preceding concepts, making it difficult to differentiate the most relevant keywords. Moreover increasing *temperature* the keywords are meaningless and between them, there are also punctuation characters, so the right temperature parameter remains 0.5, related to the Top P that is 1 and Frequency and Presence penalties that are 0.

## Data Generation

This capability is tested through the proposed example of *Interview questions*: it enables the generation of questions from the user specifications. Important to note is the **System** text box that we can use to customize how the model should answer our questions. For example, when asking a question we specify the topic, but with this text box we can ask the model to ask hard questions with unusual words. Moreover, in this experiment has been changed the **Presence Penalty** parameter, which penalizes new tokens based on whether they appear in the text so far. It increases the model’s likelihood of talking about new topics.

<b>System</b>	You are a helpful assistant
<b>User</b>	Create one question for an interview with a science fiction author.
<b>Assistant</b>	Science fiction often serves as a mirror to society, reflecting our hopes, fears, and aspirations. How do you approach incorporating social commentary into your science fiction stories, and what messages or themes do you hope readers take away from your work?

Table 8: Input and output from the usage of “Interview questions” with **Presence Penalty** set to 0 and using a “**easy interviewer**”

<b>System</b>	You are a hard interviewer who uses unusual words
<b>User</b>	Create one question for an interview with a science fiction author.
<b>Assistant</b>	How do you utilize speculative elements in your science fiction writing to explore philosophical concepts and challenge conventional thinking?

Table 9: Input and output from the usage of “Interview questions” with **Presence Penalty** set to 0 and using a “**hard interviewer**”

Taking some conclusions from this experiment over the data generation feature of GPT, the positive aspects of using such a model are multiple: the possibility to customize presence penalties helps address concerns about inappropriate or biased content, nevertheless, the possibility to specify one more system behaviour in natural language allows users to tailor AI-generated content to meet specific creative objectives as in the example above (Table 10).

<b>System</b>	You are a hard interviewer who uses unusual words
<b>User</b>	Create one question for an interview with a science fiction author.
<b>Assistant</b>	As a wordsmith of the imagination, how do you navigate the delicate balance between crafting a captivating narrative and delving into the depths of scientific plausibility in your science fiction works?

Table 10: Input and output from the usage of “Interview questions” with **Presence Penalty** set to 1 and using a “**hard interviewer**”

## Data Transformation

For the purpose of exploring Data Transformation through a generative model as GPT is, it was chosen as an example *Translation*. The parameters analyzed this time are **Model** and **Top P**. About the first one, the possible choices are “gpt-3.5-turbo” versus “gpt-3.5-turbo-16k” while the second one can be set between 0 and 1. The other parameters remain the same as preset by the platform. Interesting to note that *Top P* is preset at 1. This value is a thresholded probability that allows control of the generation process, but the website does not really explain the reasons why it should be increased or lowered, and in which case. To answer this question also this parameter is changed to evidence some possible differences. For this example, the phrase to be translated is “*My name is Jane. What is yours?*” and the system parameter is “*You will be provided with a sentence in English, and your task is to translate it into French.*”. Processing this phrase by the two models these are the results:

Model	Translation
<b>gpt-3.5-turbo</b>	Je m’appelle Jane. Et toi, comment tu t’appelles ?
<b>gpt-3.5-turbo-16k</b>	Je m’appelle Jane. Et toi, comment t’appelles-tu ?

Table 11: Translation using two different models and the same *Top P* parameter set at 1.

In this result the two translation are correct, but the second one executed using the turbo-16k version propose the formal version of the starting phrase. Running the same experiment but setting Top P at 1, doesn’t change the results.

## Code Generation and Analysis

As a last experiment, the section “Explain code” is taken into consideration. Having set the Playground platform with the preset values, they propose a Python script that is then described in natural language. This experiment tries to test if the model is able to describe code that does not contain easily identifiable variables, so that tries to understand it from its behaviour and not from the variable names.

Listing 7: Python script in input with easily explainable variables.

```
class Log:
    def __init__(self, path):
        dirname = os.path.dirname(path)
        os.makedirs(dirname, exist_ok=True)
        f = open(path, "a+")

        # Check that the file is newline-terminated
        size = os.path.getsize(path)
        if size > 0:
            f.seek(size - 1)
            end = f.read(1)
            if end != "\n":
                f.write("\n")
        self.f = f
        self.path = path

    def log(self, event):
        event["_event_id"] = str(uuid.uuid4())
        json.dump(event, self.f)
        self.f.write("\n")

    def state(self):
        state = {"complete": set(), "last": None}
        for line in open(self.path):
            event = json.loads(line)
            if event["type"] == "submit" and event["success"]:
                state["complete"].add(event["id"])
                state["last"] = event
        return state
```

#### Model output

---

This code defines a Log class that is used to log events to a file and retrieve the state of the log.

The ‘\_\_init\_\_’ method initializes the Log object by creating the directory for the log file if it doesn’t exist, opening the file in append mode, and ensuring that the file ends with a newline character.

The ‘log’ method takes an event as input, adds a unique event ID to the event, and writes the event as a JSON object to the log file. The ‘state’ method reads the log file line by line, parses each line as a JSON object, and checks if the event is of type “submit” and was successful. It keeps track of the set of completed event IDs and the last event in the log. Finally, it returns a dictionary containing the set of completed event IDs and the last event.

Listing 8: Python script in input with variables as acronyms.

```
import random as r, math as m
def main():
    n = 10
    x = r.randint(1, 100)
    y = r.randint(1, 100)

    for i in range(n):
        z = x + y
        w = m.sqrt(z)
        print(f"Result-{i+1}: {w:.2f}")

if __name__ == "__main__":
    main()
```

#### Model output

---

This code defines a ‘main’ function that generates two random numbers ‘x’ and ‘y’ between 1 and 100. It then calculates the sum of ‘x’ and ‘y’, stores it in ‘z’, and calculates the square root of ‘z’, storing it in ‘w’. It then prints the result with two decimal places for each iteration of a loop that runs ‘n’ times. The ‘main’ function is called only if the script is run directly (i.e., not imported as a module).

We can conclude that the model analyzes the code from its behaviour and doesn’t take into consideration how variables are called.

### Parameter Settings and Training Data

OpenAI makes available on its platform an entire section to explore its potentialities and each of them can be tested using different configurations. Most of them are tested in the previous subsection during the experiments. The following is a recap of these parameters:

- Mode: chat, complete or edit
- Model: gpt-3.5-turbo, gpt-3.5-turbo-0301, gpt-3.5-turbo-0613, gpt-3.5-turbo-16k, gpt-3.5-turbo-16k-0613
- Temperature: value from 0 to 2
- Maximum Length: value from 1 to 4096
- Stop Sequences: up to four sequences where the API will stop generating further tokens. The returned text will not contain the stop sequence.
- Top P: from 0 to 1
- Frequence Penalty: from 0 to 2
- Presence Penalty: from 0 to 2

About which **training data** is used in the GPT-3 model, the paper says that they downloaded and filtered a version of CommonCrawl based on similarity to a range of high-quality reference corpora, they performed fuzzy deduplication at the document level, within and across datasets, to prevent redundancy and preserve the integrity of our held-out validation set as an accurate measure of overfitting, and they also added known high-quality reference corpora to the training mix to augment CommonCrawl and increase its diversity[2].

### 4.3 Conclusions

In conclusion, GPT-3 is a formidable tool with far-reaching potential across diverse language-related tasks. However, it is essential to navigate its capabilities and limitations judiciously. The appropriate configuration of parameters and thoughtful context evaluation are crucial for understanding its strengths while being mindful of its boundaries. As we advance in AI and language processing, GPT-3 serves as a testament to the ever-evolving landscape of artificial intelligence, continuously pushing the boundaries of what is possible. These experiments try to explore its features changing parameters and observing its behaviour showing its performances on generating contents, but one possible limitation that emerged is the absence of image generation starting from natural language and this could be something to improve in GPT-3.5 features.

## References

- [1] Pinkesh Badjatiya, Manish Gupta, and Vasudeva Varma. Stereotypical bias removal for hate speech detection task using knowledge-based generalizations. In *The World Wide Web Conference*, pages 49–59, 2019.
- [2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.
- [3] Yacine Gaci, Boualem Benatallah, Fabio Casati, and Khalid Benabdeslem. Iterative adversarial removal of gender bias in pretrained word embeddings. In *Proceedings of the 37th ACM/SIGAPP Symposium On Applied Computing*, pages 829–836, 2022.
- [4] Daniel Jurafsky and James H Martin. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.
- [5] Edward Loper and Steven Bird. Nltk: The natural language toolkit, 2002.
- [6] Juan-Manuel Torres-Moreno. *Automatic text summarization*. John Wiley & Sons, 2014.
- [7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.