

MAR 02, 2024

OPEN  ACCESS



Protocol Citation: Zhaoxiang Xu, Mingjian Xie, Yanbo Huang, Qingguo Fang 2024. The public attitude towards ChatGPT on Reddit: A study based on unsupervised learning from sentiment analysis and topic modeling. **protocols.io** <https://protocols.io/view/the-public-attitude-towards-chatgpt-on-reddit-a-st-c674zhqw>

License: This is an open access protocol distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited

Protocol status: Working
We use this protocol and it's working

Created: Jan 09, 2024

Last Modified: Mar 02, 2024

The public attitude towards ChatGPT on Reddit: A study based on unsupervised learning from sentiment analysis and topic modeling

Zhaoxiang Xu^{1,2,3,4,5,6}, Mingjian Xie^{7,8,9,10,6}, Yanbo Huang^{11,12,9,10,6}, Qingguo Fang^{13,8,9,10,6}

¹Department of Data Science; ²School of Computer Science and Engineering; ³Guangzhou Institute of Science and Technology; ⁴Guangzhou; ⁵Guangdong; ⁶China; ⁷Department of Decision Sciences; ⁸School of Business; ⁹Macau University of Science and Technology; ¹⁰Macao; ¹¹Data Science Research Center; ¹²Faculty of Innovation Engineering; ¹³Department of Management

Zhaoxiang Xu: First Author;
Mingjian Xie: Second Author;
Yanbo Huang: Third Author;
Qingguo Fang: Corresponding Author;



Zhaoxiang Xu

DISCLAIMER

The protocol content here is for informational purposes only and does not constitute legal, medical, clinical, or safety advice, or otherwise; content added to [protocols.io](#) is not peer reviewed and may not have undergone a formal approval of any kind. Information presented in this protocol should not substitute for independent professional judgment, advice, diagnosis, or treatment. Any action you take or refrain from taking using or relying upon the information presented here is strictly at your own risk. You agree that neither the Company nor any of the authors, contributors, administrators, or anyone else associated with [protocols.io](#), can be held responsible for your use of the information contained in or linked to this protocol or any of our Sites/Apps and Services.

Our code has THREE main parts:

Data Processing: In terms of data cleaning, for the textual data in the database, special characters, punctuation, links, and unnecessary words were removed from the comment texts. This study applied lowercase conversion (removing uppercase letters) and removed stopwords. The NLTK library provides the English stopword list. It consists of common English words with no semantic or informational value and is typically filtered out in natural language processing to enhance the efficiency and accuracy of text analysis. Stopwords play a crucial role in enhancing text feature quality and reducing the significance of text features. Considering a slight overlap in the data sources, this study also conducted duplicate text filtering. In the end, 23,773 entries were retained, forming the database `Processed_GPT_total.json`.

Sentiment Analysis: The analyzed entries exceeded a total count of 23,773 entries. The two sentiment analysis models, Vader and Textblob, are assigned weights of 0.6 and 0.4 for sentiment classification. The sentiment analysis categorizes the emotional tone of the entries into three distinct parameters: positive, negative, and neutral. This study did histograms and emoji wordclouds of different emotions for analysis. Based on the GPT-3.5 model, ChatGPT was launched by Open AI on November 30, 2022, gaining a growing user base. On March 15, 2023, OpenAI unveiled the new large-scale multimodal model, GPT-4, available for purchase. This study examines daily sentiment trends by comparing the quantity of positive and negative sentiment posts from January to August 2023 (N=23,773). It aims to ascertain whether version updates and evolution have influenced sentiment towards ChatGPT.

Topic modeling: The topic modeling addresses the research question: What are the emerging topics related to ChatGPT? The LDA model could help determine the most suitable number of topics for classification. The topic modeling is performed with LDA, and the perplexity-topic number curve is plotted. Subsequently, the results for each number of topics are analyzed to identify the optimal number of topics based on the highest topic coherence.

Import Packets & Data Preparation

```
import json
import re
import nltk
import random
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from PIL import Image, ImageOps
from wordcloud import WordCloud
from collections import Counter
from nltk.corpus import stopwords
from gensim.models import LdaModel
from gensim.models import CoherenceModel
from nltk.sentiment import SentimentIntensityAnalyzer
from textblob import TextBlob
from gensim import corpora, models
from pprint import pprint
nltk.download('stopwords')
nltk.download('vader_lexicon')
```

The 4 files below are the processed datasets (with the keywords GPT3.0, GPT3.5, GPT4.0 with the full sample files)


 Processed_GPT_total.json 26.2MB


 Processed_GPT4.json 8.7MB


 Processed_GPT35.json 8.5MB

 Processed_GPT3.json 9.4MB

The following 3 files are the original files (with keywords GPT3.0, GPT3.5 & GPT4.0)

 G3-11730.json 13.8MB

 G35-10109.json 10.6MB

 G4-12046.json 12.4MB

Data Processing

2 Data Balancing

```
# Read the JSON file
with open('G3-11730.json', 'r', encoding='utf-8') as f:
    data = json.load(f)

# Total number of samples and target retention number
total_samples = len(data)
desired_samples = 10000

# Perform random sampling
if total_samples <= desired_samples:
    selected_samples = data
else:
    selected_samples = random.sample(data, desired_samples)

# Write the sampled results to a new JSON file
with open('G3.json', 'w', encoding='utf-8') as f:
    json.dump(selected_samples, f, ensure_ascii=False, indent=4)

print(f"Random sampling completed, retained {len(selected_samples)} samples.")
```

Data Merging (GPT3.0, GPT3.5, GPT4.0)

```
# Read data from each JSON file
file_names = ["G3.json", "G35.json", "G4.json"]
all_data = []

for file_name in file_names:
    with open(file_name, 'r', encoding='utf-8') as f:
        data = json.load(f)
        all_data.extend(data)

# Write the combined data to a new JSON file
output_file_name = "GPT_total.json"
with open(output_file_name, 'w', encoding='utf-8') as f:
    json.dump(all_data, f, ensure_ascii=False, indent=4)

print(f"The file merge has been completed and the name of the merged file is {output_file_name}, containing {len(all_data)} samples.")
```

Data Cleaning

```
def has_missing_body(sample):
    return 'body' not in sample

def clean_text(text):
    # Data cleaning: remove special symbols and links
    text = re.sub(r'http\S+', '', text) # Remove links
    text = re.sub(r'\W+', ' ', text) # Remove special symbols
    return text

def preprocess_text(text):
    # Text preprocessing: convert to lowercase, remove punctuation, and
    numbers
    text = text.lower()
    text = re.sub(r'\d+', '', text) # Remove numbers
    text = re.sub(r'^\w\s|$', '', text) # Remove punctuation
    return text

def tokenize_text(text):
    # Text tokenization: split the text by space
    return text.split()

def remove_stopwords(words_list):
    # Stopwords handling: remove stopwords
    stop_words = set(stopwords.words('english'))
    return [word for word in words_list if word.lower() not in stop_words]

def remove_short_words(words_list):
    # Remove words with length less than or equal to 2
    return [word for word in words_list if len(word) > 2]

def process_samples(input_file, output_file):
    with open(input_file, 'r') as file:
        samples = json.load(file)

    processed_samples = []
    unique_samples = set() # Used to record the unique processed sample
    content

    for sample in samples:
        if has_missing_body(sample):
            continue

        body_text = sample['body']

        # Data cleaning
        body_text = clean_text(body_text)
```

```
# Text preprocessing
body_text = preprocess_text(body_text)

# Text tokenization
words_list = tokenize_text(body_text)

# Stopwords handling and removing short words
words_list = remove_stopwords(words_list)
words_list = remove_short_words(words_list)

# Reassemble the text
processed_text = ' '.join(words_list)

# Check if the same sample content already exists
if processed_text in unique_samples:
    continue

# Add to the unique sample set
unique_samples.add(processed_text)

# Update the 'body' field of the sample
sample['body'] = processed_text

processed_samples.append(sample)

with open(output_file, 'w') as file:
    json.dump(processed_samples, file, indent=4)

print(f"Sample size after preprocessing: {len(processed_samples)}")

# Replace with your file paths
input_file_path = 'G35.json'
output_file_path = 'Processed_GPT_35.json'

process_samples(input_file_path, output_file_path)
```

Word Frequency

3 Word frequency bar graph display for GPT_total.json

```
# Replace with the path to the processed file
processed_file_path = 'Processed_GPT_total.json'

with open(processed_file_path, 'r') as file:
    processed_samples = json.load(file)

# Extract the text content from samples and create a large text string
all_text = ' '.join([sample['body'] for sample in processed_samples])

# Remove specific words (e.g., "chatgpt" and "gpt")
exclude_words = ["chatgpt", "gpt", "gt", "amp", "xb", "also", "one"]
all_text = ' '.join([word for word in all_text.split() if word.lower() not
in exclude_words])

from collections import Counter

# Split all words and calculate word frequencies
words = all_text.split()
word_freq = Counter(words)

# Get the top n most common words and their frequencies
top_n = 20 # You can adjust this for the desired number of top words to
display
common_words = [word for word, freq in word_freq.most_common(top_n)]
freq_counts = [freq for word, freq in word_freq.most_common(top_n)]

# Set a dark color palette
colors = sns.color_palette("bright", len(common_words))

# Plot a horizontal bar chart
plt.figure(figsize=(10, 6))

# Add both horizontal and vertical dashed grid lines
plt.grid(axis='both', linestyle='--', alpha=0.3)
bars = plt.barh(common_words, freq_counts, color=colors)
# Set x-axis label positions and font size
plt.xticks([0, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500],
fontsize=12)

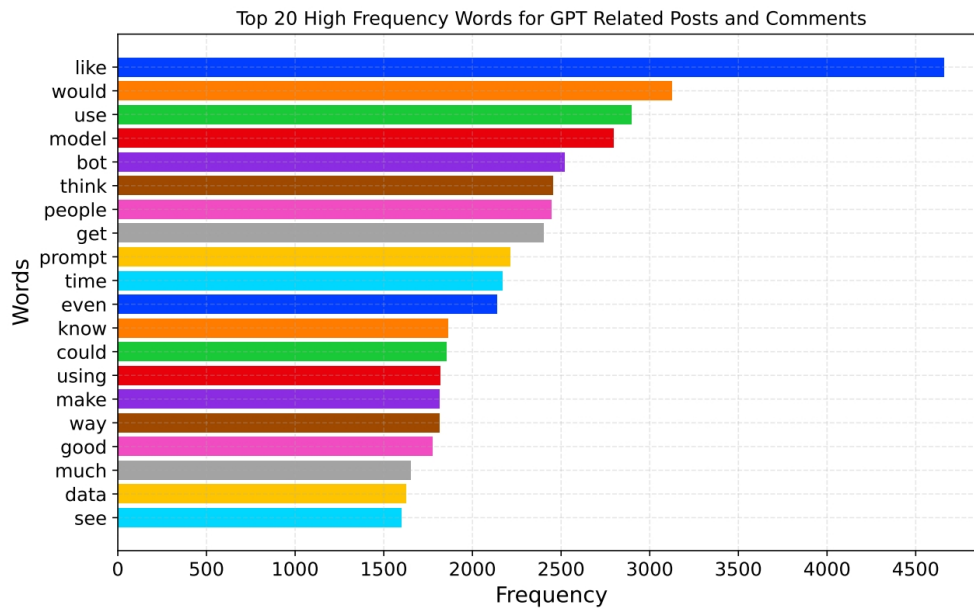
# Set y-axis label font size
plt.yticks(fontsize=12)

# Set axis labels
plt.xlabel('Frequency', fontsize=14)
plt.ylabel('Words', fontsize=14)
```

```
# Set the title
plt.title('Top 20 High Frequency Words for GPT Related Posts and Comments')

# Invert the y-axis to display high-frequency words at the top
plt.gca().invert_yaxis()

# Display the image
plt.show()
```



Sentiment Analysis

4 Sentiment classification (bar chart)


```
# Load the preprocessed file
with open('Processed_GPT_total.json', 'r') as file:
    data = json.load(file)

# Extract text data
texts = []
for item in data:
    texts.append(item['body']) # Choose the desired field based on your
    actual data

# Perform sentiment analysis using TextBlob
textblob_scores = []
for text in texts:
    blob = TextBlob(text)
    sentiment = blob.sentiment.polarity
    textblob_scores.append(sentiment)

# Perform sentiment analysis using VADER
vader_scores = []
analyzer = SentimentIntensityAnalyzer()
for text in texts:
    scores = analyzer.polarity_scores(text)
    vader_scores.append(scores['compound'])

# Weighted aggregation of results
textblob_weight = 0.4
vader_weight = 0.6
weighted_scores = [(textblob_weight * tb + vader_weight * vd) for tb, vd in
zip(textblob_scores, vader_scores)]

# Build a DataFrame
df = pd.DataFrame({'TextBlob': textblob_scores, 'VADER': vader_scores,
'Weighted Score': weighted_scores})

# Plot the visualization results
plt.figure(figsize=(10, 6))

# Add gray horizontal and vertical grids
plt.grid(color='gray', linestyle='--', alpha=0.3, zorder=0) # Set zorder to
place the grid at the bottom layer

# Plot the histogram with zorder set to 1 to place it at the top layer
plt.hist(df['Weighted Score'], bins=20, label='Weighted Score', zorder=1)

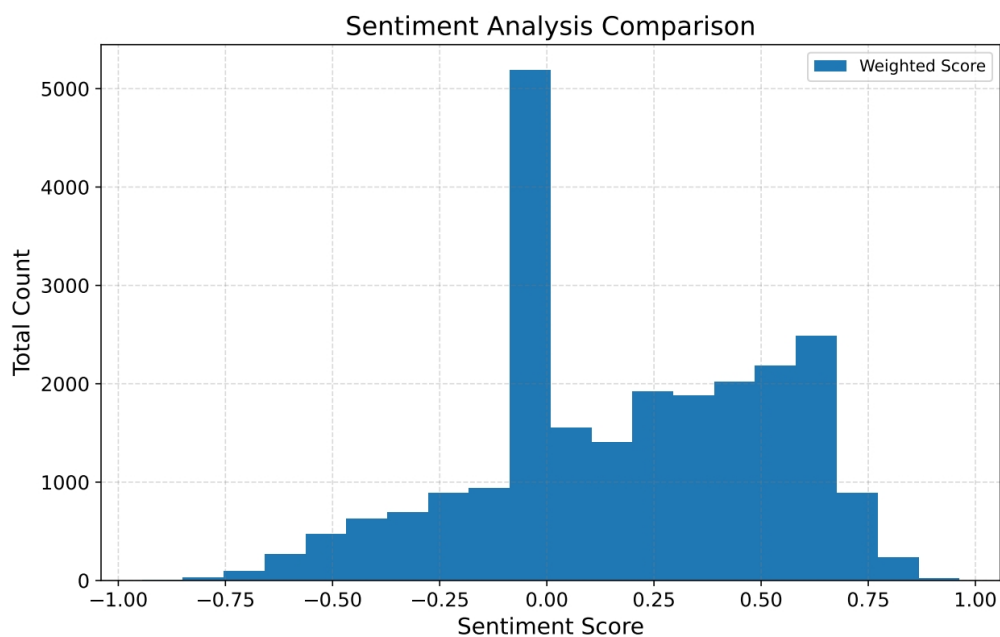
# Increase font size of x and y axis labels
plt.xticks(fontsize=12)
```

```
plt.yticks(fontsize=12)

plt.xlabel('Sentiment Score', fontsize=14) # Increase x-axis label font
size
plt.ylabel('Total Count', fontsize=14) # Increase y-axis label font size

# Add title and legend
plt.title('Sentiment Analysis Comparison', fontsize=16)
plt.legend()

# Display the plot
plt.show()
```



Sentiment classification (pie chart)

```
# Calculate sentiment classification proportions
positive_count = len([score for score in weighted_scores if score > 0])
neutral_count = len([score for score in weighted_scores if score == 0])
negative_count = len([score for score in weighted_scores if score < 0])

# Pie chart data
labels = ['Positive', 'Neutral', 'Negative']
sizes = [positive_count, neutral_count, negative_count]
colors_inner = ['#1f78b4', '#e41a1c', '#33a02c'] # Inner circle colors
colors_outer = ['#67a9cf', '#fb6a4a', '#78c679'] # Outer ring colors
explode = (0.1, 0, 0) # Separate the "Positive" portion

# Plot inner circle pie chart
fig, ax = plt.subplots(figsize=(8, 8))
inner_pie = ax.pie(sizes, explode=explode, colors=colors_inner,
                  autopct='%1.1f%%', pctdistance=0.8,
                  wedgeprops=dict(width=0.4, edgecolor='w'), shadow=True,
                  startangle=140)
ax.axis('equal') # Keep it circular

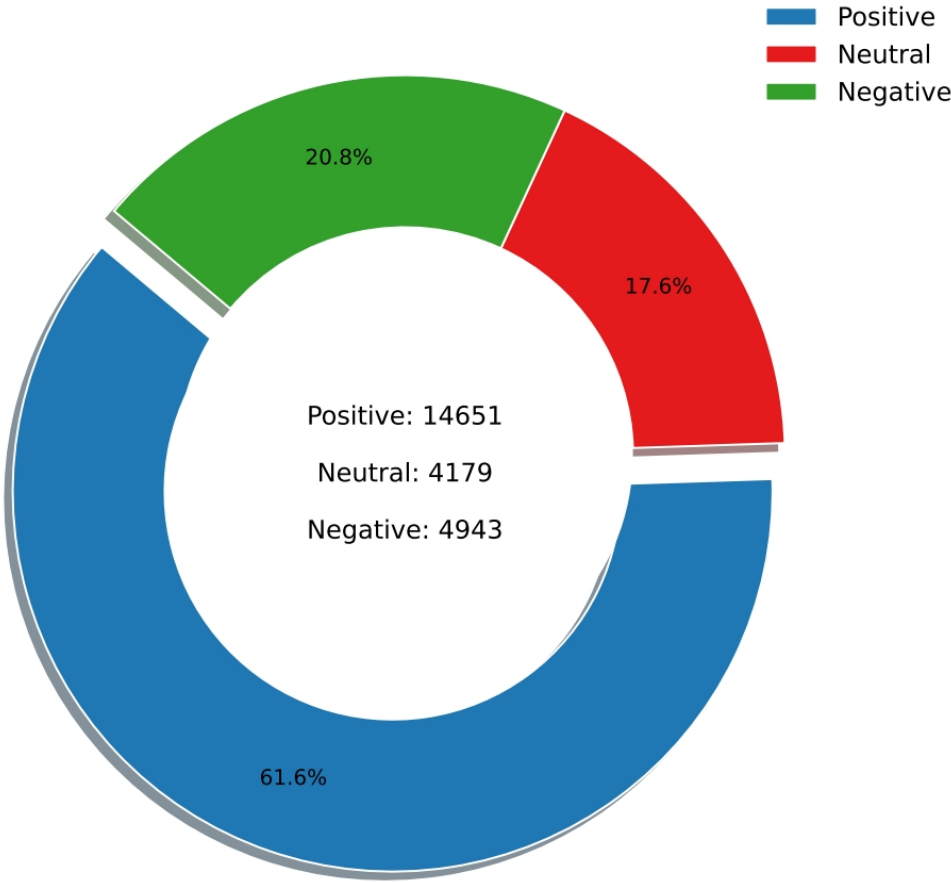
# Plot outer ring pie chart
ax.pie([1], colors=['w'], radius=0.6) # Draw a white circle with a radius
of 0.6, creating a ring

# Add quantity labels
for i, size in enumerate(sizes):
    ax.text(0, 0.1 - i * 0.15, f'{labels[i]}: {size}', ha='center',
           va='center', fontsize=12, color='black')

# Add legend, adjust font size, and remove the border
legend = ax.legend(labels, loc='upper right', bbox_to_anchor=(1.1, 1),
                  fontsize='large', frameon=False)

# Adjust font size for percentage labels
for text in inner_pie[1]:
    text.set_fontsize('large')

plt.show()
```



Changes in Positive and Negative Emotions over Time

```
# Load the preprocessed file with open('Processed_GPT_total.json', 'r') as
file:
    data = json.load(file)

# Extract date and text data
dates = []
texts = []
for item in data:
    if item['createdAt'][:4] >= '2023': # Only keep posts from 2023 and
later
        dates.append(item['createdAt'][:10])
        texts.append(item['body']) # Choose the required field based on the
actual situation# Calculate sentiment scores using TextBlob and VADER
textblob_weight = 0.4
vader_weight = 0.6

textblob_scores = []
vader_scores = []
analyzer = SentimentIntensityAnalyzer()

for text in texts:
    blob = TextBlob(text)
    textblob_scores.append(blob.sentiment.polarity)

    scores = analyzer.polarity_scores(text)
    vader_scores.append(scores['compound'])

# Weight the results
weighted_scores = [(textblob_weight * tb + vader_weight * vd) for tb, vd in
zip(textblob_scores, vader_scores)]

# Construct the data frame
df = pd.DataFrame({'Date': dates, 'Weighted Score': weighted_scores})

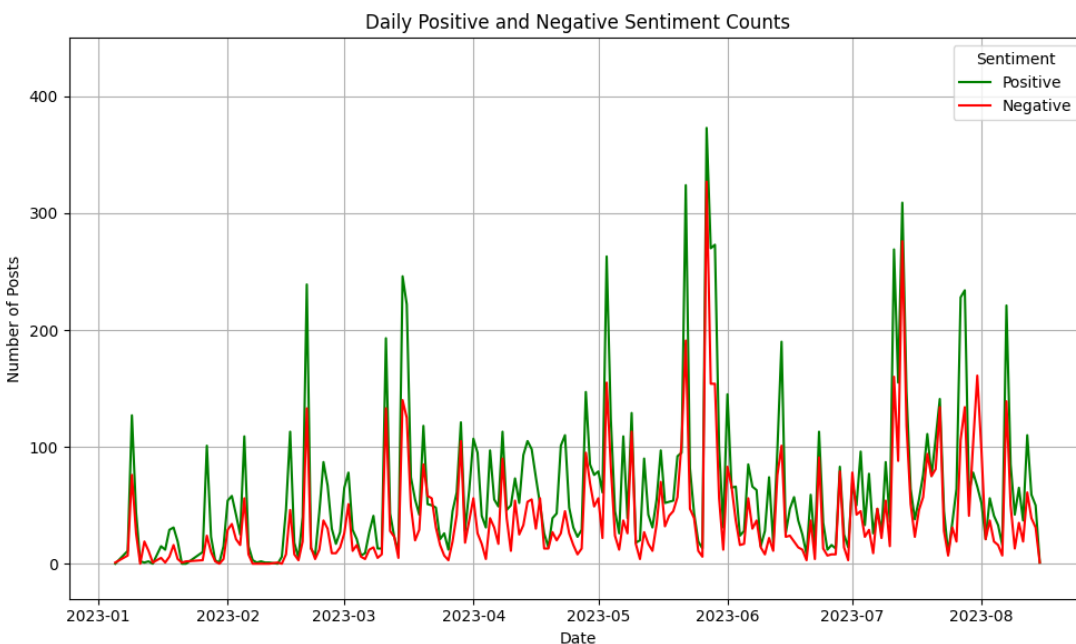
# Convert the Date column to a datetime type
df['Date'] = pd.to_datetime(df['Date'])

# Calculate the number of positive and negative sentiments daily
df['Sentiment'] = df['Weighted Score'].apply(lambda score: 'Positive' if
score > 0 else 'Negative')

# Group by date and sentiment, then calculate daily sentiment counts
daily_sentiment_counts = df.groupby(['Date',
'Sentiment']).size().unstack(fill_value=0)

# Plot the daily positive and negative sentiment counts over time
```

```
plt.figure(figsize=(10, 6))
plt.plot(daily_sentiment_counts.index, daily_sentiment_counts['Positive'],
label='Positive', color='green')
plt.plot(daily_sentiment_counts.index, daily_sentiment_counts['Negative'],
label='Negative', color='red')
plt.title('Daily Positive and Negative Sentiment Counts')
plt.xlabel('Date')
plt.ylabel('Number of Posts')
plt.legend(title='Sentiment')
plt.ylim(-30, 450)
plt.grid()
plt.tight_layout()
plt.show()
```



Topic Modeling

5 LDA topic modeling and plotting perplexity-topic curves

```
# Perform topic modeling
def perform_topic_modeling(samples, num_topics, dictionary, corpus):
    lda_models = []
    perplexity_scores = []

    for i in range(1, num_topics + 1):
        lda_model = models.LdaModel(corpus=corpus, id2word=dictionary,
num_topics=i, passes=10, random_state=1586)
        lda_models.append(lda_model)
        perplexity_scores.append(lda_model.log_perplexity(corpus))

    # Print perplexity scores for each number of topics
    pprint(list(zip(range(1, num_topics + 1), perplexity_scores)))

    return lda_models, perplexity_scores

if __name__ == "__main__":
    # Read the processed JSON file
    processed_file_path = 'Processed_GPT_total.json'
    with open(processed_file_path, 'r') as file:
        processed_samples = json.load(file)

    # Text processing
    documents = [sample['body'] for sample in processed_samples]

    filtered_texts = []
    for document in documents:
        words = re.findall(r'\w+', document.lower())
        filtered_words = [word for word in words if word not in ['che',
'sisters', 'brothers', 'harry', 'una', 'con', 'los', 'wtf', 'nier', 'one',
'reddit', 'chatgpt', 'comment', 'post', 'bro', 'per', 'fed', 'gpt',
'models', 'conscious', 'companies', 'model', 'using', 'apps', 'mai', 'fuck',
'dude', 'jesus', 'quran', 'dan', 'que', 'sally', 'non']]
        filtered_texts.append(filtered_words)

    texts = filtered_texts

    # Build a bag-of-words model
    dictionary = corpora.Dictionary(texts)
    corpus = [dictionary.doc2bow(text) for text in texts]

    # Perform topic modeling and obtain perplexity scores
    num_topics = 20 # Set the upper limit for the number of topics
    lda_models, perplexity_scores =
perform_topic_modeling(processed_samples, num_topics, dictionary, corpus)
```

```
# Plot the perplexity-number of topics curve
plt.figure(figsize=(10, 6))
plt.plot(range(1, num_topics + 1), perplexity_scores, marker='o')

# Add both horizontal and vertical grids
plt.grid(True, linestyle='--', alpha=0.7)

# Increase x and y axis label font size
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

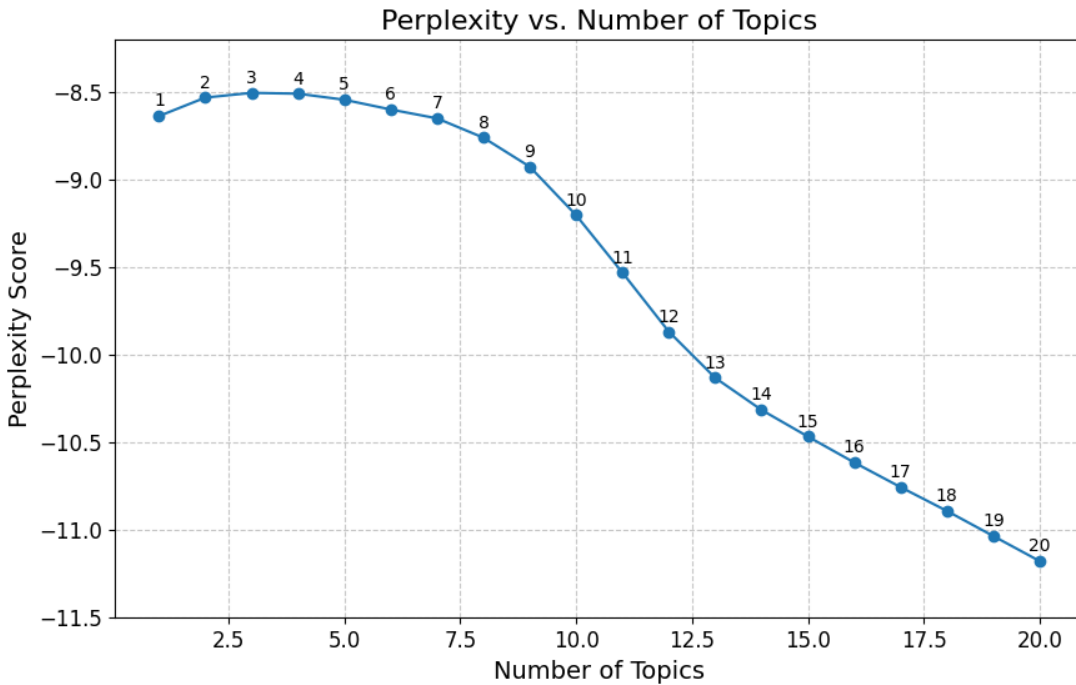
# Add annotation text on each scatter point
for i, txt in enumerate(range(1, num_topics + 1)):
    plt.annotate(txt, (i + 1, perplexity_scores[i]), textcoords="offset
points", xytext=(0, 6), ha='center', fontsize=10)

# Set x and y axis labels
plt.xlabel('Number of Topics', fontsize=14)
plt.ylabel('Perplexity Score', fontsize=14)

# Set the title
plt.title('Perplexity vs. Number of Topics', fontsize=16)

plt.ylim(-11.5, -8.2)

# Display the image
plt.show()
```

```
# Execute topic modeling
num_topics = 7
lda_model = perform_topic_modeling(processed_samples, num_topics,
dictionary, corpus)

# Get the top 10 keywords for each topic
topic_keywords = get_topic_keywords(lda_model, num_topics, num_words=10)
```

```
[(0,
'0.014*"like" + 0.010*"people" + 0.009*"think" + 0.008*"would" + 0.008*"get" '
'+ 0.008*"even" + 0.006*"use" + 0.006*"time" + 0.006*"much" + 0.006*"good"'),
(1,
'0.064*"bot" + 0.023*"please" + 0.022*"prompt" + 0.021*"link" + 0.019*"open" '
'+ 0.013*"questions" + 0.013*"amp" + 0.013*"message" + 0.012*"action" + '
'0.011*"free"'),
(2,
'0.019*"consciousness" + 0.015*"conscious" + 0.010*"agi" + 0.008*"humans" + '
'0.008*"brain" + 0.007*"god" + 0.007*"human" + 0.007*"believe" + '
'0.006*"intelligence" + 0.005*"self"'),
(3,
'0.010*"like" + 0.008*"code" + 0.008*"use" + 0.007*"would" + 0.007*"prompt" '
'+ 0.006*"text" + 0.005*"data" + 0.005*"language" + 0.005*"also" + '
'0.005*"make"'),
(4,
```

```
'0.009*"man" + 0.006*"music" + 0.005*"art" + 0.005*"state" + 0.005*"top" + '  
'0.004*"health" + 0.004*"home" + 0.004*"quantum" + 0.004*"menu" + '  
'0.004*"fire"),  
(5,  
'0.008*"amp" + 0.008*"world" + 0.005*"market" + 0.004*"people" + '  
'0.004*"life" + 0.004*"human" + 0.004*"game" + 0.004*"new" + 0.003*"jobs" + '  
'0.003*"impact"),  
(6,  
'0.005*"gif" + 0.005*"admit" + 0.004*"giphy" + 0.004*"wide" + 0.004*"care" + '  
'0.003*"forgot" + 0.003*"trump" + 0.003*"president" + 0.003*"capital" + '  
'0.002*"spider"]]
```

```
# Execute topic modeling  
num_topics = 8  
lda_model = perform_topic_modeling(processed_samples, num_topics,  
dictionary, corpus)  
  
# Get the top 10 keywords for each topic  
topic_keywords = get_topic_keywords(lda_model, num_topics, num_words=10)
```

```
[(0,  
'0.015*"link" + 0.009*"lmao" + 0.006*"fire" + 0.006*"years" + '  
'0.005*"windows" + 0.005*"coal" + 0.005*"range" + 0.005*"countries" + '  
'0.005*"waiting" + 0.005*"air"),  
(1,  
'0.023*"consciousness" + 0.006*"care" + 0.006*"universe" + '  
'0.006*"intelligence" + 0.006*"awareness" + 0.005*"life" + 0.005*"dumber" + '  
'0.004*"self" + 0.004*"humans" + 0.004*"women"),  
(2,  
'0.015*"like" + 0.009*"would" + 0.009*"think" + 0.009*"people" + '  
'0.008*"even" + 0.006*"data" + 0.006*"time" + 0.006*"way" + 0.005*"know" + '  
'0.005*"good"),  
(3,  
'0.058*"bot" + 0.032*"open" + 0.024*"source" + 0.024*"please" + '  
'0.024*"prompt" + 0.016*"amp" + 0.015*"message" + 0.015*"openai" + '  
'0.014*"questions" + 0.014*"image"),  
(4,  
'0.014*"use" + 0.013*"code" + 0.010*"api" + 0.009*"prompt" + 0.009*"chat" + '  
'0.008*"get" + 0.008*"like" + 0.007*"text" + 0.006*"answer" + 0.006*"also"),  
(5,  
'0.009*"market" + 0.005*"impact" + 0.005*"content" + 0.005*"gpu" + '  
'0.004*"trade" + 0.004*"government" + 0.004*"nvidia" + 0.003*"voice" + '
```

```
'0.003*"apple" + 0.003*"cost"),
(6,
'0.009*"amp" + 0.005*"new" + 0.005*"pay" + 0.005*"company" + 0.004*"people" '
'+ 0.004*"game" + 0.004*"world" + 0.004*"help" + 0.004*"story" + '
'0.003*"business"),
(7,
'0.005*"wolfram" + 0.005*"electric" + 0.004*"sample" + 0.004*"censored" + '
'0.003*"didnt" + 0.003*"repetition" + 0.003*"para" + 0.003*"sin" + '
'0.003*"russia" + 0.003*"wont"]]
```

```
# Execute topic modeling
num_topics = 9
lda_model = perform_topic_modeling(processed_samples, num_topics,
dictionary, corpus)

# Get the top 10 keywords for each topic
topic_keywords = get_topic_keywords(lda_model, num_topics, num_words=10)
```

```
[(0,
'0.046*"link" + 0.006*"windows" + 0.005*"state" + 0.005*"electric" + '
'0.005*"delete" + 0.005*"drive" + 0.005*"teams" + 0.004*"face" + '
'0.004*"poem" + 0.004*"tiktok"),
(1,
'0.011*"women" + 0.008*"god" + 0.008*"care" + 0.007*"sentient" + 0.007*"men" '
'+ 0.006*"bullshit" + 0.005*"biological" + 0.005*"planet" + 0.004*"holy" + '
'0.004*"bag"),
(2,
'0.018*"like" + 0.011*"would" + 0.010*"think" + 0.008*"even" + '
'0.008*"people" + 0.007*"know" + 0.007*"good" + 0.006*"way" + '
'0.006*"something" + 0.006*"could"),
(3,
'0.070*"bot" + 0.030*"open" + 0.029*"please" + 0.027*"prompt" + '
'0.021*"source" + 0.018*"message" + 0.017*"image" + 0.017*"questions" + '
'0.015*"free" + 0.015*"amp"),
(4,
'0.015*"bing" + 0.015*"thank" + 0.011*"bard" + 0.010*"answer" + '
'0.009*"youtube" + 0.009*"thanks" + 0.008*"dumb" + 0.008*"question" + '
'0.007*"turbo" + 0.007*"web"),
(5,
'0.017*"use" + 0.016*"api" + 0.013*"text" + 0.012*"data" + 0.010*"user" + '
'0.010*"code" + 0.009*"openai" + 0.008*"content" + 0.008*"prompt" + '
'0.007*"chat"),
(6,
```

```
'0.009*"get" + 0.008*"people" + 0.008*"time" + 0.006*"like" + 0.006*"pay" + '  
'0.005*"work" + 0.005*"make" + 0.005*"company" + 0.005*"years" + '  
'0.005*"use"),  
(7,  
'0.006*"wolfram" + 0.006*"willing" + 0.006*"elon" + 0.005*"cap" + '  
'0.004*"fictional" + 0.004*"percentile" + 0.004*"sample" + 0.004*"dollar" + '  
'0.004*"censored" + 0.004*"repetition"),  
(8,  
'0.007*"game" + 0.007*"amp" + 0.006*"world" + 0.005*"language" + '  
'0.005*"prompt" + 0.004*"character" + 0.004*"story" + 0.004*"response" + '  
'0.004*"provide" + 0.004*"knowledge"]]
```

Robustness testing

Using the sampling model set up in Part II, use the model from the original dataset

"Processed_GPT_total.json" and set `random.seed(61)` to take 10,000 samples as a subset and name it "`subsets_GPT_total.json`"

```
if __name__ == "__main__":
    # First dataset
    processed_file_path_1 = 'Processed_GPT_total.json'
    with open(processed_file_path_1, 'r') as file:
        processed_samples_1 = json.load(file)

    # Second dataset
    processed_file_path_2 = 'subsets_GPT_total.json'
    with open(processed_file_path_2, 'r') as file:
        processed_samples_2 = json.load(file)

    # Text processing
    documents_1 = [sample['body'] for sample in processed_samples_1]
    documents_2 = [sample['body'] for sample in processed_samples_2]

    filtered_texts_1 = []
    filtered_texts_2 = []
    filtered_words = [word for word in words if word not in ['che',
'sisters', 'brothers', 'harry', 'una', 'con', 'los', 'wtf', 'nier', 'one',
'reddit', 'chatgpt', 'comment', 'post', 'bro', 'per', 'fed', 'gpt',
'models', 'conscious', 'companies', 'model', 'using', 'apps', 'mai', 'fuck',
'dude', 'jesus', 'quran', 'dan', 'que', 'sally', 'non']]

    for document in documents_1:
        words = re.findall(r'\w+', document.lower())
        filtered_words = filtered_words
        filtered_texts_1.append(filtered_words)

    for document in documents_2:
        words = re.findall(r'\w+', document.lower())
        filtered_words = filtered_words
        filtered_texts_2.append(filtered_words)

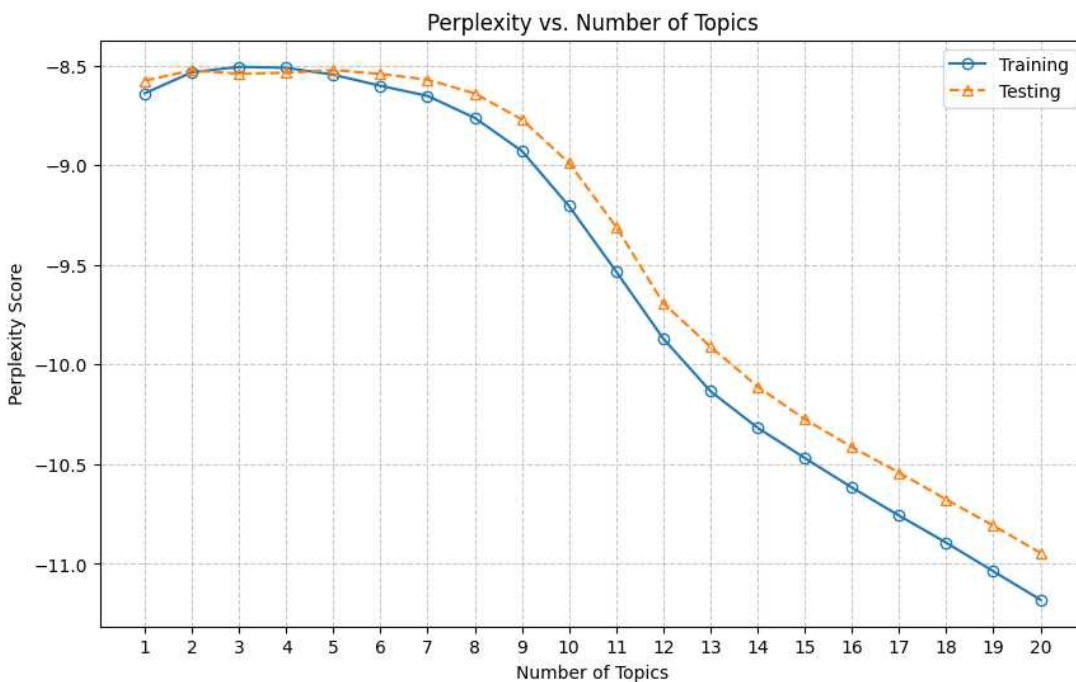
    texts_1 = filtered_texts_1
    texts_2 = filtered_texts_2

    # Build the dictionary and corpus
    dictionary_1 = corpora.Dictionary(texts_1)
    dictionary_2 = corpora.Dictionary(texts_2)
    corpus_1 = [dictionary_1.doc2bow(text) for text in texts_1]
    corpus_2 = [dictionary_2.doc2bow(text) for text in texts_2]

    # Perform topic modeling and get perplexity scores
    num_topics = 20 # Set the upper limit of the number of topics
    lda_models_1, perplexity_scores_1 =
perform_topic_modeling(processed_samples_1, num_topics, dictionary_1,
```

```
corpus_1)
lda_models_2, perplexity_scores_2 =
perform_topic_modeling(processed_samples_2, num_topics, dictionary_2,
corpus_2)

# Plot the perplexity-number of topics curve
plt.figure(figsize=(10, 6))
plt.plot(range(1, num_topics + 1), perplexity_scores_1, marker='o',
markerfacecolor='none', linestyle='-', label='Training')
plt.plot(range(1, num_topics + 1), perplexity_scores_2, marker='^',
markerfacecolor='none', linestyle='--', label='Testing')
plt.grid(True, linestyle='--', alpha=0.7)
plt.xticks(range(1, 21, 1))
plt.xlabel('Number of Topics')
plt.ylabel('Perplexity Score')
plt.title('Perplexity vs. Number of Topics')
plt.legend()
plt.savefig('Two_Curves.pdf', format='pdf')
plt.show()
```



```
lda_model_2 = lda_models_2[num_topics_7 - 1]
top_words_2 = lda_model_2.show_topics(num_topics=num_topics_7, num_words=10,
formatted=False)
print("\nTop 10 words for dataset 2 with {} topics:".format(num_topics_7))
for topic_id, topic_words in top_words_2:
    print("Topic {}: {}".format(topic_id, [word[0] for word in
topic_words]))
```

Topic 0: ['art', 'people', 'life', 'industry', 'market', 'pig', 'sister', 'business', 'technology', 'profit']

Topic 1: ['use', 'api', 'code', 'prompt', 'openai', 'text', 'get', 'chat', 'content', 'version']

Topic 2: ['tom', 'youtube', 'also', 'tone', 'blog', 'words', 'use', 'copyright', 'pay', 'battery']

Topic 3: ['like', 'would', 'know', 'good', 'get', 'people', 'really', 'time', 'even', 'something']

Topic 4: ['bot', 'prompt', 'please', 'open', 'questions', 'discord', 'automatically', 'free', 'subreddit', 'message']

Topic 5: ['houses', 'link', 'district', 'red', 'para', 'winter', 'governed', 'blue', 'sol', 'fewer']

Topic 6: ['think', 'human', 'could', 'would', 'people', 'even', 'time', 'make', 'like', 'way']

```
# Baseline data
baseline = np.array([[ -8.53, -8.50, -8.51, -8.54, -8.60, -8.65, -8.76]])

# Other five runs data (numeric results from running with randomly sampled
subsets)
data = np.array([
    [-8.52, -8.54, -8.53, -8.52, -8.54, -8.57, -8.63],
    [-8.49, -8.45, -8.46, -8.48, -8.50, -8.52, -8.61],
    [-8.49, -8.46, -8.47, -8.50, -8.53, -8.55, -8.63],
    [-8.53, -8.48, -8.50, -8.52, -8.54, -8.55, -8.63],
    [-8.48, -8.47, -8.49, -8.52, -8.55, -8.56, -8.66]])

# Calculate differences
differences = data - baseline

# Plot heatmap
plt.figure(figsize=(8, 6))
plt.imshow(differences, cmap='coolwarm', interpolation='nearest')
plt.colorbar(label='Difference from Baseline')
plt.title('Difference Heatmap from Baseline (Topic Numbers = 8)')
plt.xlabel('Topic Number')
plt.ylabel('Number of subsets')
plt.xticks(np.arange(7), np.arange(2, 9))
plt.yticks(np.arange(5), np.arange(1, 6))
plt.savefig('Heatmap_Differences.pdf', format='pdf')
plt.show()
```

