Mini-project 1

Wikipedia content analysis

# **Aim:**

To perform **Text mining** from a web scrapped data from Wikipedia and find out the

1. Precision, Recall, F-score
2. Form a word cloud
3. Keyword search
4. Content mining

# **Procedure:**

1. Import modules such as tkinter, requests, wordcloud, matplotlib.pyplot, sklearn.feature\_extraction.text, sklearn.decomposition, nltk, collections, sumy.parsers.plaintext, sumy.nlp.tokenizers, sumy.summarizers.lsa, textblob, PIL, sklearn.metrics.pairwise, numpy, gensim.models, gensim.corpora
2. Build a Tkinter user interface.
3. Scrape the data from Wikipedia using BeautifulSoup
4. Return the extracted data and perform topic modeling and display in UI
5. After performing topic modeling find the coherence score and perplexity
6. Simultaneously, pass the extracted content to form a word cloud and show it in the UI
7. Perform content mining by finding the ngrams . which checks for the most occurred words for nth time together
8. Implement keyword search for multiple keywords by getting the keywords from user as comma separated values and find the no of occurrence of the keyword and the place where the keyword is counted from.
9. Then the extracted passage summarized version is displayed.
10. Display all the methods implemented in Tkinter window.

# **Techniques Implemented:**

1. Web scrapping
2. Word Cloud
3. Keyword search
4. Content mining (ngram)
5. Topic modeling along with coherence score and perplexity
6. Summarizing content

# **Code:**

def scrape\_wikipedia\_content(topic):

url = f"https://en.wikipedia.org/wiki/{topic.replace(' ', '\_')}"

response = requests.get(url)

soup = BeautifulSoup(response.content, 'html.parser')

content\_paragraphs = soup.find\_all('p')

content\_text = ' '.join([paragraph.text for paragraph in content\_paragraphs])

return content\_text,url

def generate\_word\_cloud(text,frame):

if text:

wordcloud = WordCloud(width=1000, height=270, background\_color='white').generate(text)

plt.figure(figsize=(7, 2))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis('off')

plt.tight\_layout(pad=0)

plt.savefig('wordcloud.png')

plt.close()

img = tk.PhotoImage(file='wordcloud.png')

wordcloud\_label = tk.Label(frame, image=img)

wordcloud\_label.image = img

wordcloud\_label.grid(row=1, column=1)

def calculate\_perplexity(lda\_model, X):

return lda\_model.perplexity(X)

def perform\_topic\_modeling(texts, num\_topics=5, max\_features=None):

vectorizer = CountVectorizer(stop\_words='english', max\_features=max\_features)

X = vectorizer.fit\_transform(texts)

lda = LatentDirichletAllocation(n\_components=num\_topics, random\_state=42)

lda.fit(X)

dt\_matrix = lda.transform(X)

tw\_matrix = lda.components\_ / lda.components\_.sum(axis=1)[:, np.newaxis]

tokenized\_texts = [text.split() for text in texts]

dictionary = corpora.Dictionary(tokenized\_texts)

feature\_names = vectorizer.get\_feature\_names\_out()

topics = []

for i in range(num\_topics):

topic\_tokens = [feature\_names[idx] for idx in np.argsort(-tw\_matrix[i])[:10]] # Get top 10 tokens for each topic

topics.append(topic\_tokens)

coherence\_model = CoherenceModel(topics=topics, texts=tokenized\_texts, dictionary=dictionary, coherence='u\_mass')

coherence\_score = coherence\_model.get\_coherence()

perplexity = calculate\_perplexity(lda, X)

return lda, vectorizer.get\_feature\_names\_out(), coherence\_score, perplexity

def content\_mining(texts, n=2, top\_n=10):

lemmatizer = WordNetLemmatizer()

stop\_words = set(stopwords.words('english'))

all\_ngrams = []

for text in texts:

words = word\_tokenize(text)

words = [word.lower() for word in words if word.isalnum()]

words = [lemmatizer.lemmatize(word) for word in words if word not in stop\_words]

text\_ngrams = list(ngrams(words, n))

all\_ngrams.extend(text\_ngrams)

ngram\_counts = Counter(all\_ngrams)

common\_ngrams = ngram\_counts.most\_common(top\_n)

return common\_ngrams

def summarize\_content(text):

parser = PlaintextParser.from\_string(text, Tokenizer("english"))

summarizer = LsaSummarizer()

summary = summarizer(parser.document, sentences\_count=2)

return " ".join([str(sentence) for sentence in summary])

def perform\_sentiment\_analysis(text):

blob = TextBlob(text)

sentiment\_score = blob.sentiment.polarity

return sentiment\_score

def search\_keywords(text, keywords):

keyword\_results = {}

for keyword in keywords:

keyword\_count = text.lower().count(keyword.lower())

positions = [pos for pos, char in enumerate(text.lower()) if text.lower().find(keyword.lower(), pos) == pos]

keyword\_results[keyword] = {"count": keyword\_count, "positions": positions}

return keyword\_results

def display\_results():

topic = paper\_name\_entry.get()

content\_text,url = scrape\_wikipedia\_content(topic)

topic\_text.insert(tk.END,f"url: {url}\n")

generate\_word\_cloud(content\_text,wordcloud\_frame)

texts = content\_text.split("\n") # Assuming each document is separated by a newline character

texts = [doc.strip() for doc in content\_text.split('\n') if doc.strip()]

lda\_model, feature\_names, coherence\_score, perplexity = perform\_topic\_modeling(texts)

topics = {}

for topic\_idx, topic in enumerate(lda\_model.components\_):

top\_words\_idx = topic.argsort()[:-10 - 1:-1]

top\_words = [feature\_names[i] for i in top\_words\_idx]

topics[f"Topic {topic\_idx+1}"] = top\_words

for topic, words in topics.items():

topic\_text.insert(tk.END, f"{topic}: {', '.join(words)}\n")

common\_ngrams = content\_mining([content\_text], n=2, top\_n=10)

ngrams\_text.delete('1.0', tk.END) # Clear previous content

for ngram in common\_ngrams:

ngrams\_text.insert(tk.END, f"{ngram}\n")

paper\_summaries = [summarize\_content(content\_text)]

for summary in paper\_summaries:

summary\_text.insert(tk.END, f"{summary}\n")

keywords = keyword\_entry.get().strip().split(',') # Strip extra spaces before splitting

keyword\_results = search\_keywords(content\_text, keywords)

keyword\_result\_text.delete('1.0', tk.END)

for keyword, data in keyword\_results.items():

keyword\_count = data["count"]

keyword\_positions = data["positions"]

positions\_info = ", ".join([str(pos) for pos in keyword\_positions])

keyword\_result\_text.insert(tk.END, f"{keyword}: \nCount: {keyword\_count}, \nPositions: {positions\_info}\n\n")

senti\_score=perform\_sentiment\_analysis(content\_text)

sentiment\_table.insert("", "end", values=("Score", coherence\_score,perplexity))

root = tk.Tk()

root.title("Wikipedia Content Analysis")

main\_frame = tk.Frame(root)

main\_frame.pack(fill=tk.BOTH, expand=True)

paper\_label = tk.Label(main\_frame, text="Enter Topic:")

paper\_label.grid(row=0, column=0)

paper\_name\_entry = tk.Entry(main\_frame, width=50)

paper\_name\_entry.grid(row=0, column=1)

keyword\_label = tk.Label(main\_frame, text="Enter Keywords (comma separated):")

keyword\_label.grid(row=1, column=0)

keyword\_entry = tk.Entry(main\_frame, width=50)

keyword\_entry.grid(row=1, column=1)

results\_button = tk.Button(main\_frame, text="Get Results", command=display\_results)

results\_button.grid(row=2, columnspan=2)

result\_frame\_1 = tk.Frame(main\_frame)

result\_frame\_1.grid(row=3, column=0, columnspan=2, padx=10, pady=5, sticky="nsew")

wordcloud\_frame = tk.Frame(result\_frame\_1, width=200, height=400)

wordcloud\_frame.grid(row=1, column=2, padx=10, pady=5, sticky="w")

topic\_label = tk.Label(result\_frame\_1, text="Topics:")

topic\_label.grid(row=0, column=0, sticky="w")

topic\_text = scrolledtext.ScrolledText(result\_frame\_1, width=80, height=12,wrap="word")

topic\_text.grid(row=1, column=0, padx=10, pady=5)

keyword\_result\_label = tk.Label(result\_frame\_1, text="Keyword Search Results:")

keyword\_result\_label.grid(row=2, column=0, sticky="w")

keyword\_result\_text = scrolledtext.ScrolledText(result\_frame\_1, width=80, height=12,wrap="word")

keyword\_result\_text.grid(row=3, column=0, padx=10, pady=5)

scrollbar = tk.Scrollbar(result\_frame\_1, orient="vertical", command=keyword\_result\_text.yview)

scrollbar.grid(row=3, column=1, sticky="ns")

keyword\_result\_text.configure(yscrollcommand=scrollbar.set)

result\_frame\_2 = tk.Frame(main\_frame)

result\_frame\_2.grid(row=4, column=0, columnspan=2, padx=10, pady=5, sticky="nsew")

ngrams\_text = scrolledtext.ScrolledText(result\_frame\_2, width=80, height=12,wrap="word")

ngrams\_text.grid(row=1, column=1, padx=10, pady=5, sticky="nsew")

summary\_label = tk.Label(result\_frame\_2, text="Article Summaries:")

summary\_label.grid(row=0, column=0, sticky="w")

summary\_text = scrolledtext.ScrolledText(result\_frame\_2, width=80, height=12,wrap="word")

summary\_text.grid(row=1, column=0, padx=10, pady=5, sticky="nsew")

result\_frame\_3 = tk.Frame(main\_frame)

result\_frame\_3.grid(row=5, column=0,rowspan=1, columnspan=2, padx=10, pady=5, sticky="nsew")

style = ttk.Style()

style.configure("Custom.Treeview", font=('Helvetica', 8)) # Set font size

sentiment\_label = tk.Label(result\_frame\_3, text="Sentiment Metrics")

sentiment\_label.pack(side="bottom")

sentiment\_table = ttk.Treeview(result\_frame\_3, columns=("Category", "Coherence", "Perplexity"), style="Custom.Treeview")

sentiment\_table.heading("#0", text="", anchor="w")

sentiment\_table.heading("Category", text="Category")

sentiment\_table.heading("Coherence", text="Coherence")

sentiment\_table.heading("Perplexity", text="perplexity")

sentiment\_table.pack(side="bottom", fill="both", expand=True)

sentiment\_table.config(height=3) # Set the height

main\_frame.columnconfigure(0, weight=1)

main\_frame.columnconfigure(1, weight=1)

main\_frame.columnconfigure(2, weight=1)

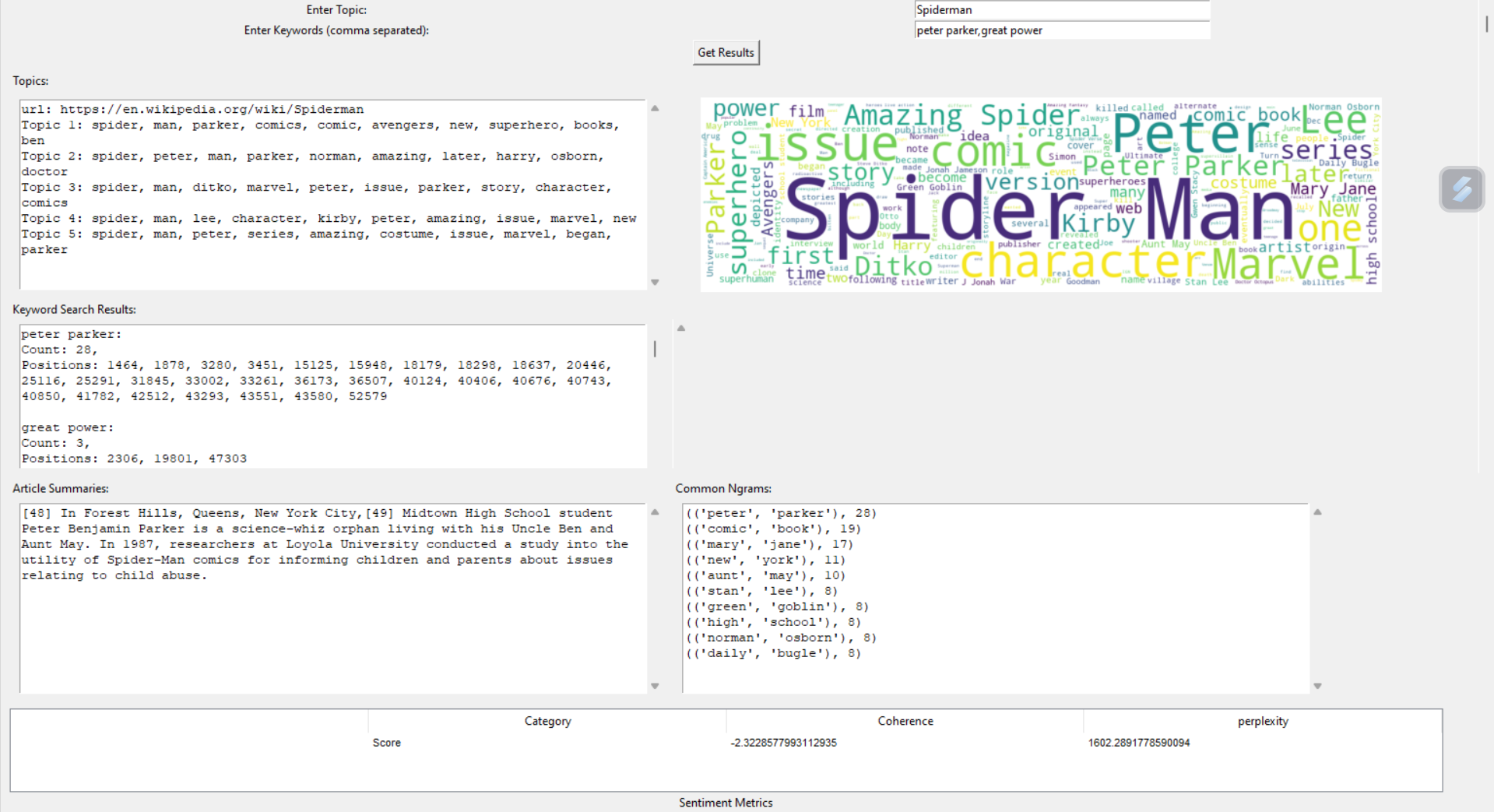
main\_frame.rowconfigure(3, weight=1)

scrollbar = tk.Scrollbar(main\_frame, orient="vertical")

scrollbar.grid(row=0, column=3, rowspan=4, sticky="ns")

root.mainloop()

# **Output:**



**Inference:**

The result of topic and keywords are based on user input. The first frame consists of topic modelling where the significant terms of the first 5 paragraphs or section in Wikipedia website is retrieved. The next frame on the right displays the word cloud of the topic. The next frame has retrieved the keyword count and positions of the user input. The next frame displays the overall summary of the Wikipedia article and common ngrams retrieves the data based on the pair-wise collection of words and their count.

The coherence and perplexity metrics are used to analyze the quality of the data retrieval and the value of coherence of -2.32265 and perplexity of 1682.269, indicates that the topic might be overlapping and requires further tuning with different modelling approaches.