VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

NATURAL LANGUAGE PROCESSING (CO3085)

Assignment report

DATA COLLECTION AND MINING OF INSPIRATIONAL QUOTES FROM FAMOUS PERSONALITIES

Advisor: Bùi Thanh Hùng
Students: Vũ Nguyễn Lan Vi - 2153094

HO CHI MINH CITY, NOVEMBER 2024

Contents

Chapter 1

Methods

1.1 Data Collection

1.1.1 Write code to crawl data from the Quotes to Scrape website, save the results to a corresponding file (kq.txt) and briefly describe the structure of the above website?

- **Implementation**

```
    def scrape_one_page(url):
      response = requests.get(url)
      soup = BeautifulSoup(response.text, 'html.parser')
      quotes = soup.find_all('div', class_='quote')
      return quotes

    def craw_data(base_url, output_file_name):
      page_number = 1
      all_quotes = []
      while True:
        url = f"{base_url}/page/{page_number}/"
        quotes = scrape_one_page(url)
        if len(quotes) == 0:
          break
        all_quotes.extend(quotes)
        page_number += 1
      with open(output_file_name, 'w') as f:
        for quote in all_quotes:
          f.write(str(quote) + '\n')

```

- **Structure diagram of Quotes to Scrape**

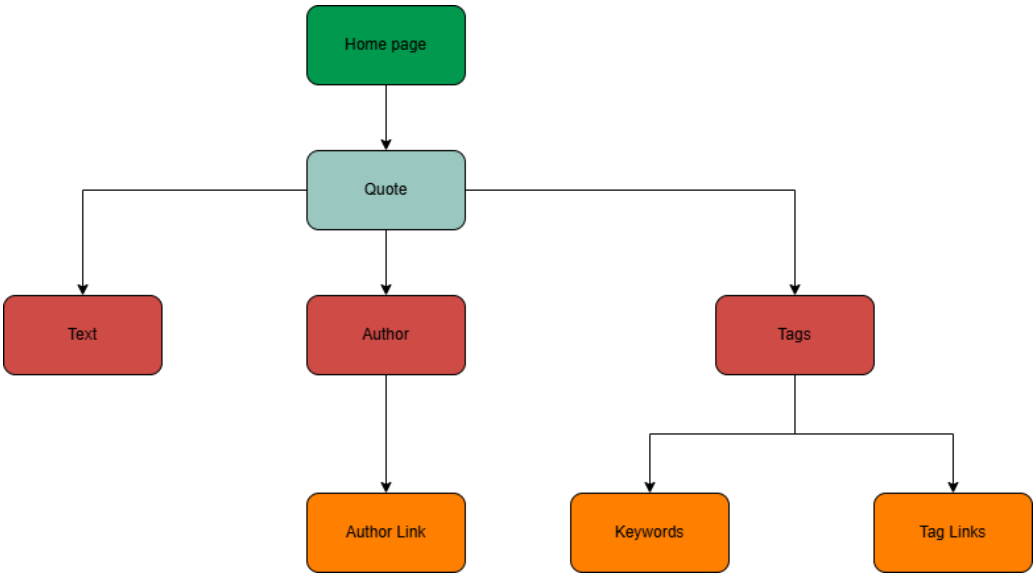


Figure 1.1: Structure diagram of Quotes to Scrape Home page.

1.1.2 With the data you just crawled, please perform the following requests:

Read all the html tags (div) with the class "quote" and save it in the variable "result", display the value of the variable "result" on the screen?

- **Implementation**

```
with open('kq.txt', 'r', encoding='utf-8') as f:
  html_data = f.read()

soup = BeautifulSoup(html_data, 'html.parser')
result = soup.find_all('div', class_='quote')
print(result)

```

- **Results**

Find in the variable 'result', the data containing the label "small" with the class "author" and print the results on the screen?

- **Implementation**

```
def extract_author_info(quotes):
    authors = []
    for quote in quotes:
        author = quote.find('small', class_='author')
        if author:
            authors.append(author.get_text(strip=True))
```

```
[<div class="quote" itemscope="" itemtype="http://schema.org/CreativeWork">
<span class="text" itemprop="text">“The world as we have created it is a process of our thinking. It cannot be changed without changing our
<span>by <small class="author" itemprop="author">Albert Einstein</small>
<a href="/author/Albert-Einstein">(about)</a>
</span>
<div class="tags">
            Tags:
            <meta class="keywords" content="change,deep-thoughts,thinking,world" itemprop="keywords"/>
<a class="tag" href="/tag/change/page/1/">change</a>
<a class="tag" href="/tag/deep-thoughts/page/1/">deep-thoughts</a>
<a class="tag" href="/tag/thinking/page/1/">thinking</a>
<a class="tag" href="/tag/world/page/1/">world</a>
</div>
</div>, <div class="quote" itemscope="" itemtype="http://schema.org/CreativeWork">
<span class="text" itemprop="text">“It is our choices, Harry, that show what we truly are, far more than our abilities.”</span>
<span>by <small class="author" itemprop="author">J.K. Rowling</small>
<a href="/author/J-K-Rowling">(about)</a>
</span>
<div class="tags">
            Tags:
            <meta class="keywords" content="abilities,choices" itemprop="keywords"/>
<a class="tag" href="/tag/abilities/page/1/">abilities</a>
<a class="tag" href="/tag/choices/page/1/">choices</a>
</div>
</div>, <div class="quote" itemscope="" itemtype="http://schema.org/CreativeWork">
...
```

Figure 1.2: Result of all html tags (div) with the class "quote".

```
7               return authors
8
9           authors = extract_author_info(result)
10          for author in authors:
11              print(author)
12
```

- **Results**

Write the function authorLink() to get the content of each author. For each author, print the content on the screen

1. **Author’s name**
2. **Author’s link**
3. **Date of birth**
4. **And the author’s famous quote**

- **Implementation**

```
1           def authorLink(quotes):
2             author_dictionary = {}
3             for quote in quotes:
4               text = quote.find('span', class_='text').get_text()
5               text = text.strip("'\'")
6               author = quote.find('small', class_='author').get_text()
7               if author not in author_dictionary:
```

```
Albert Einstein
J.K. Rowling
Albert Einstein
Jane Austen
Marilyn Monroe
Albert Einstein
André Gide
Thomas A. Edison
Eleanor Roosevelt
Steve Martin
Marilyn Monroe
J.K. Rowling
Albert Einstein
Bob Marley
Dr. Seuss
Douglas Adams
Elie Wiesel
Friedrich Nietzsche
Mark Twain
Allen Saunders
Pablo Neruda
Ralph Waldo Emerson
Mother Teresa
Garrison Keillor
Jim Henson
...
Madeleine L'Engle
Mark Twain
Dr. Seuss
George R.R. Martin
```

Figure 1.3: Result of all data containing the label "small" with the class "author".

```
            author_dictionary[author] = {}
            author_link = f"https://quotes.toscrape.com{quote.find('a')['href']}"
            get_author_infor = requests.get(author_link)
            author_soup = BeautifulSoup(get_author_infor.text, 'html.parser')
            dob = author_soup.find('span', class_='author-born-date').get_text()
            author_dictionary[author]['dob'] = dob
            author_dictionary[author]['link'] = author_link
            author_dictionary[author]['quotes'] = []

        author_dictionary[author]['quotes'].append(text)
    return author_dictionary

author_dictionary = authorLink(result)

for author, infor in author_dictionary.items():
    print(f"Author: {author}")
    print(f"Date of Birth: {infor['dob']}")
    print(f"Link: {infor['link']}")
    print("Quotes:")
    for quote in infor['quotes']:
        print(f"- {quote}")
    print()

```

- **Results**

```
Author: Albert Einstein
Date of Birth: March 14, 1879
Link: https://quotes.toscrape.com/author/Albert-Einstein
Quotes:
- The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking.
- There are only two ways to live your life. One is as though nothing is a miracle. The other is as though everything is a miracle.
- Try not to become a man of success. Rather become a man of value.
- If you can't explain it to a six year old, you don't understand it yourself.
- If you want your children to be intelligent, read them fairy tales. If you want them to be more intelligent, read them more fairy tales.
- Logic will get you from A to Z; imagination will get you everywhere.
- Any fool can know. The point is to understand.
- Life is like riding a bicycle. To keep your balance, you must keep moving.
- If I were not a physicist, I would probably be a musician. I often think in music. I live my daydreams in music. I see my life in terms o
- Anyone who has never made a mistake has never tried anything new.

Author: J.K. Rowling
Date of Birth: July 31, 1965
Link: https://quotes.toscrape.com/author/J-K-Rowling
Quotes:
- It is our choices, Harry, that show what we truly are, far more than our abilities.
- It takes a great deal of bravery to stand up to our enemies, but just as much to stand up to our friends.
- It is impossible to live without failing at something, unless you live so cautiously that you might as well not have lived at all - in wh
- Of course it is happening inside your head, Harry, but why on earth should that mean that it is not real?
- To the well-organized mind, death is but the next great adventure.
- It matters not what someone is born, but what they grow to be.
...
```

Figure 1.4: authorLink() function.

Save the result in question c into the corresponding Quote.csv file, with each author being 1 row of data. You are required to collect at least 40 famous quotes from the above website automatically according to the code of the above ideas?

- **Implementation**

```
1         import csv
2
3         def save_to_csv(author_dictionary, filename):
4             with open(filename, 'w', newline='', encoding='utf-8') as f:
5                 writer = csv.writer(f)
6                 writer.writerow(['Author Name', 'Date of Birth', 'Author Link', 'Famous
      Quotes'])
7                 for author, info in author_dictionary.items():
8                     quotes = ', '.join([f"'{quote}'" for quote in info['quotes']])
9                     writer.writerow([author, info['dob'], info['link'], quotes])
10
11        save_to_csv(author_dictionary, 'Quote.csv')
12
```

- **Results**

	Author Name	Date of Birth	Author Link	Famous Quotes
0	Albert Einstein	March 14, 1879	https://quotes.toscrape.com/author/Albert-Eins...	'The world as we have created it is a process ...
1	J.K. Rowling	July 31, 1965	https://quotes.toscrape.com/author/J-K-Rowling	'It is our choices, Harry, that show what we t...
2	Jane Austen	December 16, 1775	https://quotes.toscrape.com/author/Jane-Austen	'The person, be it gentleman or lady, who has ...
3	Marilyn Monroe	June 01, 1926	https://quotes.toscrape.com/author/Marilyn-Monroe	'Imperfection is beauty, madness is genius and...
4	André Gide	November 22, 1869	https://quotes.toscrape.com/author/Andre-Gide	'It is better to be hated for what you are tha...

Figure 1.5: Save to Quote.csv

Chapter 2

Data Mining

2.1 Data Imputation

- **Introduction**

  The lack of Date of Birth (DOB) data on the Quotes to Scrape website hinders statistical analysis. Without **DOB** and **age**, it becomes challenging to analyze how authors' writing styles reflect their era or to identify whether they belong to modern times. Adding this data makes trend analysis and identifying notable individuals much easier.

- **Approach**

  I crawled additional data from Google by using the `search_wikipedia_url()` function to find an author's Wikipedia link. The `get_wikipedia_info()` function then extracted reliable details like Date of Birth (DOB) and Date of Death (DOD). Age was calculated logically: subtracting the birth year from the death year for deceased authors or from the current year for living ones, ensuring accuracy by considering full dates.

- **Evaluation**

  Information for all 50 authors was successfully retrieved from Wikipedia, and the extracted DOB matched the data available on the Quotes to Scrape website. This method proved to be a simple yet efficient way to gather accurate and reliable data.

- **Discussion**

  This approach resolved the data gaps efficiently, enabling deeper analysis of trends in authors' writing styles over time. Using Wikipedia ensured reliable data, facilitating better statistical insights and historical context.

2.1.1 Some values of the Date of Birth field are not available, please suggest how to fill them in?

- **Implementation**

```
import requests
from bs4 import BeautifulSoup
import re
from datetime import datetime

def extract_date(text):
    date_pattern = re.compile(r'(\d{4}-\d{2}-\d{2})')
    match = date_pattern.search(text)
    if match:
        return match.group(0)
    return None

def get_wikipedia_info(author_name):
  url = f"https://en.wikipedia.org/wiki/{author_name}"
  response = requests.get(url)
  soup = BeautifulSoup(response.text, 'html.parser')
  #find born date and death date if any
  infobox = soup.find('table', class_='infobox')
  if infobox:
    born_th = infobox.find('th', string='Born')
    if born_th:
        born_date = born_th.find_next_sibling('td').get_text(strip=True)
        born_date = extract_date(born_date)
    else:
        born_date = None

    # Find the death date, if available
    death_th = infobox.find('th', string='Died')
    if death_th:
        death_date = death_th.find_next_sibling('td').get_text(strip=True)
        death_date = extract_date(death_date)
    else:
        death_date = None
    return born_date, death_date
  return None, None

```

2.1.2 Please add the Age field and suggest how to fill in the age of the authors?

- **Implementation**

```
for author, infor in author_dictionary.items():
  born_date, death_date = get_wikipedia_info(author)
  dob = infor['dob']
```

```
    dob_date = datetime.strptime(dob, '%B %d, %Y')
    if not death_date:
        today = datetime.now()
        age = today.year - dob_date.year - ((today.month, today.day) < (dob_date.month,
        dob_date.day))
    else:
        death_date = datetime.strptime(death_date, '%Y-%m-%d')
        dob_date = datetime.strptime(dob, '%B %d, %Y')
        age = death_date.year - dob_date.year - ((death_date.month, death_date.day) < (
        dob_date.month, dob_date.day))

    author_dictionary[author]['age'] = int(age)
    df.loc[df['Author Name'] == author, 'age'] = int(age)

```

- **Results**

```
df.head()
```
✓ 0.0s

	Author Name	Date of Birth	Author Link	Famous Quotes	age
0	Albert Einstein	March 14, 1879	https://quotes.toscrape.com/author/Albert-Eins...	'The world as we have created it is a process ...	76.0
1	J.K. Rowling	July 31, 1965	https://quotes.toscrape.com/author/J-K-Rowling	'It is our choices, Harry, that show what we t...	59.0
2	Jane Austen	December 16, 1775	https://quotes.toscrape.com/author/Jane-Austen	'The person, be it gentleman or lady, who has ...	41.0
3	Marilyn Monroe	June 01, 1926	https://quotes.toscrape.com/author/Marilyn-Monroe	'Imperfection is beauty, madness is genius and...	36.0
4	André Gide	November 22, 1869	https://quotes.toscrape.com/author/Andre-Gide	'It is better to be hated for what you are tha...	81.0

2.2 Data Exploration

- **Introduction:** In this study, we analyze a dataset containing authors and their famous quotes. We aim to extract various statistics and visualize patterns to uncover insights related to the authors and the content of their quotes. The analysis involves studying the distribution of quotes, birth years, age statistics, quote lengths, word usage, and relationships between authors. The goal is to provide a deeper understanding of the dataset and explore the similarities in writing styles among the authors.
- **Approach:**
  - **Statistics about the authors and famous quotes in the dataset:** We first visualize the distribution of the number of quotes per author. A bar chart is used to show the frequency of quotes by each author. Additionally, we calculate the total number of authors and quotes to provide a general overview.

- **Statistics about the year of birth and age of the authors:** We visualize the distribution of authors by their birth year using a histogram. The age distribution of the authors is also analyzed, providing insights into the demographic profile of the authors in the dataset.
- **Statistics about famous quotes:** We identify the longest and shortest quotes in the dataset. Additionally, we analyze the number of words in the quotes and visualize the word distribution using histograms.
- **Statistics about the words used in the quotes:** A word cloud is generated to visualize the most frequently occurring words in the quotes. A frequency plot is also created to show the number of occurrences of each word.
- **Analyze, visualize the relationship between the authors and famous quotes:** We create scatter plots to show the relationship between the authors' birth years and the length of their quotes.
- **Analyze, visualize the relationship between the authors with each other:** Using t-SNE, we visualize the similarity in writing styles among authors, based on their quotes. This visualization allows us to observe how authors with similar styles are clustered together.

- **Evaluation:** The effectiveness of our approach is evaluated based on the clarity of the visualizations and the insights gained. We assess how well the visualizations represent the distribution of statistics and the relationships between different attributes of the authors and their quotes.
- **Discussion:** In this section, we discuss the key findings from the analysis, including any trends or patterns identified in the data. We also highlight any challenges encountered during the analysis and propose potential future improvements or directions for further research.

2.2.1 Statistics about the authors and famous quotes in the dataset.

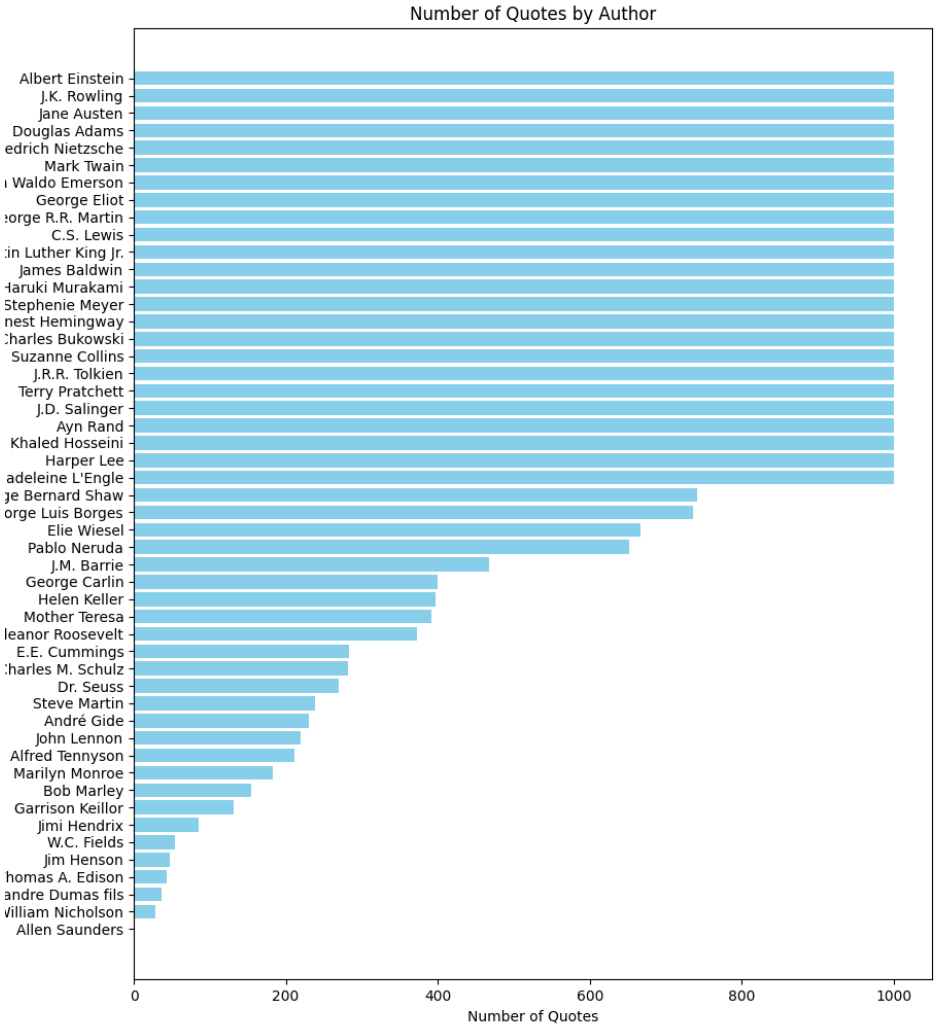
-
-

2.2.2 Statistics about the year of birth and age of the authors.

-

2.2.3 Statistics about famous quotes such as: longest, shortest, number of words, ...

-
-




```
print(f"Number of authors: {len(author_dictionary)}")
```

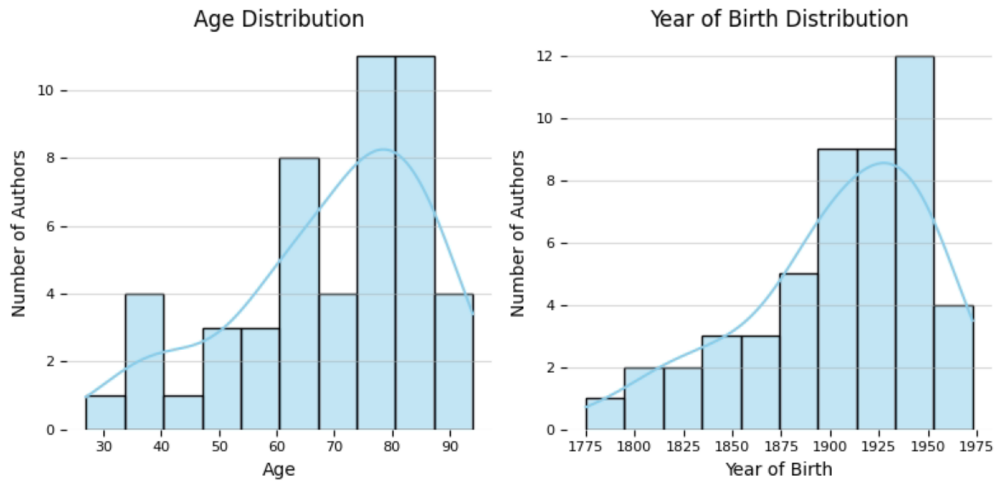
✓ 0.0s

```
Number of authors: 50
```

```
print(f"Total number of quotes: {sum(len(infor['quotes']) for infor in author_dictionary.values())}")
```

✓ 0.0s

```
Total number of quotes: 31323
```




```
print(f"Longest sentence's length: {max(num_words_per_quote)}")
print(f"Shortest sentence's length: {min(num_words_per_quote)}")
```

✓ 0.0s

```
Longest sentence's length: 719
Shortest sentence's length: 1
```

2.2.4 Statistics about the words used in the quotes

-
-

2.2.5 Analyze, visualize the relationship between the authors and famous quotes

-

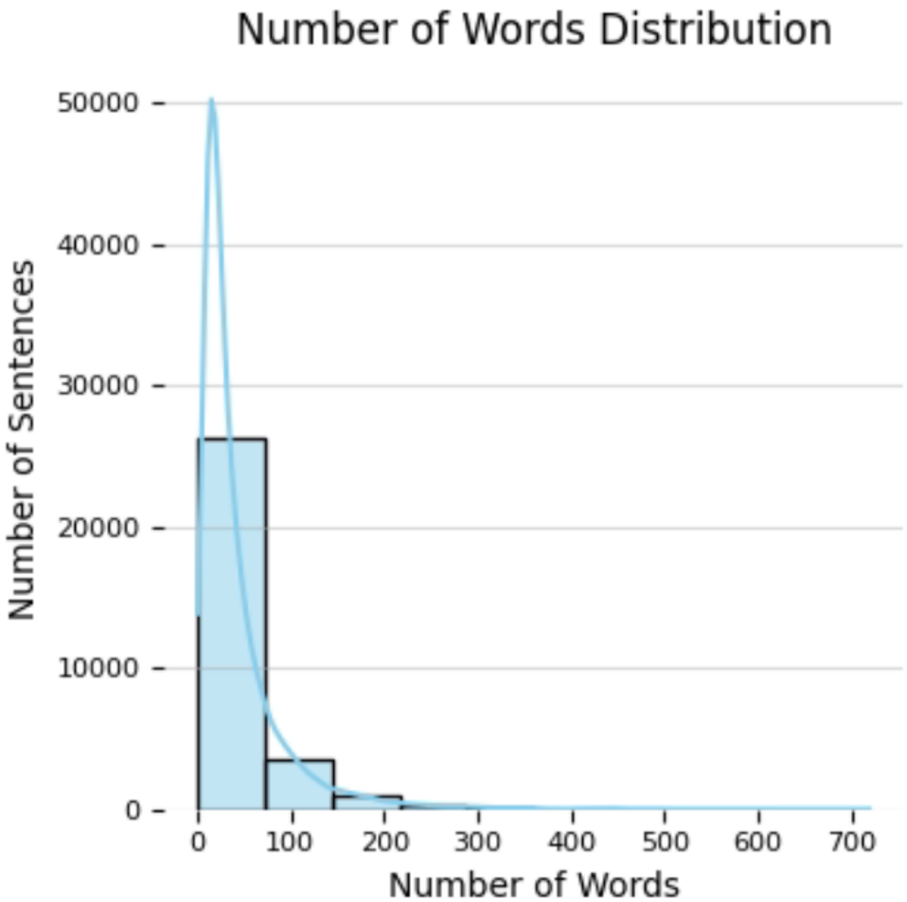
2.2.6 Analyze, visualize the relationship between the authors with each other, ...

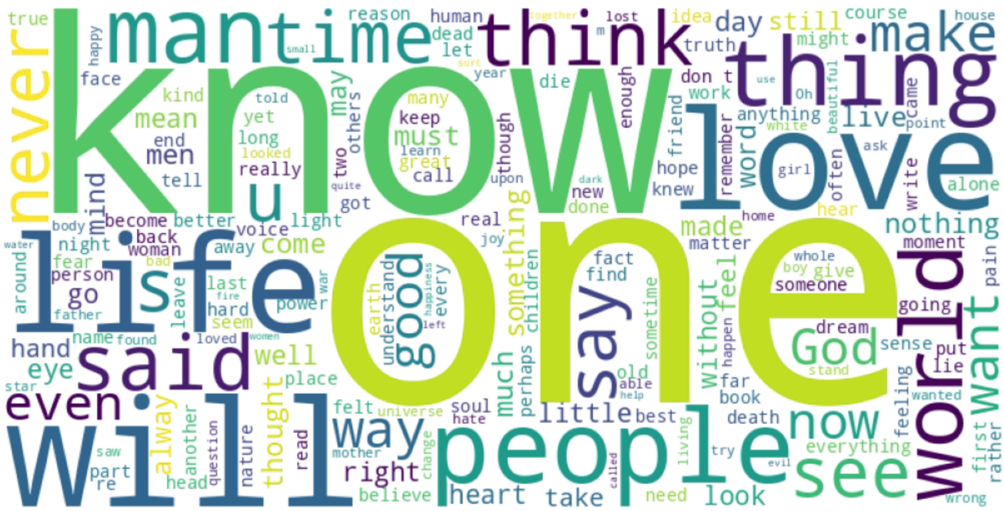
-

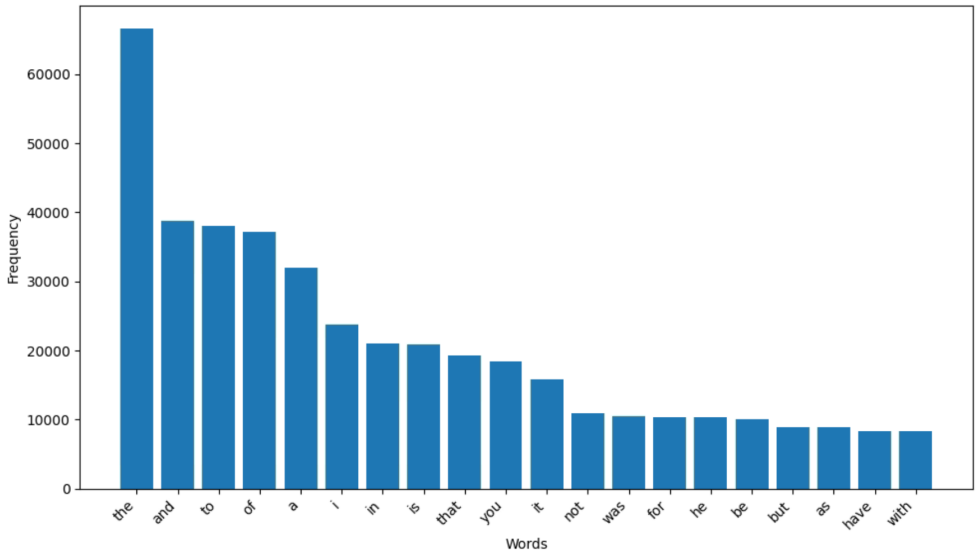
2.3 Feature Extraction

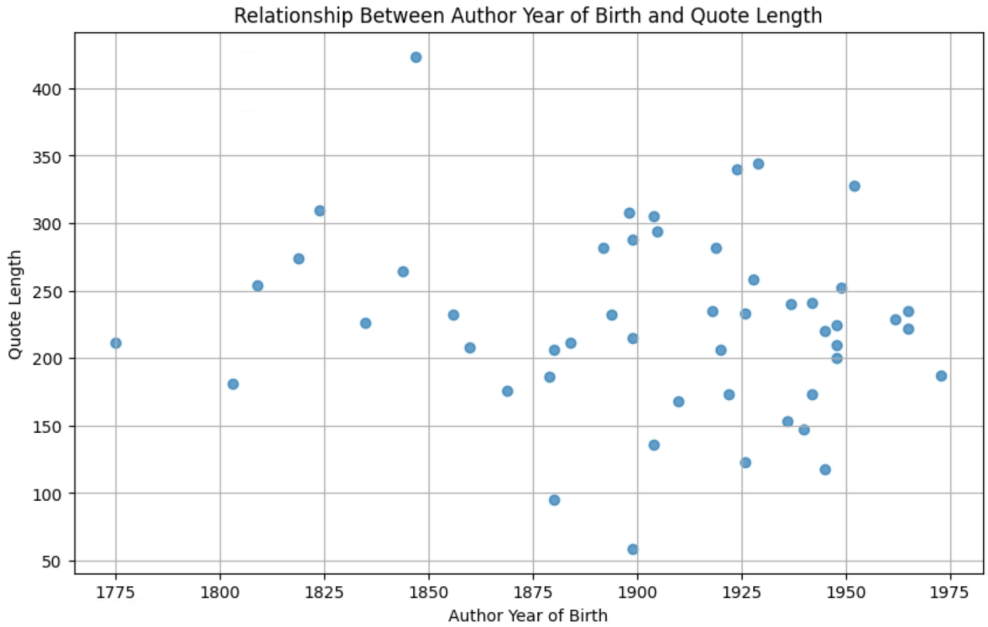
2.3.1 Suggest a way to extract features from the given dataset, provide reasons and explain your approach.

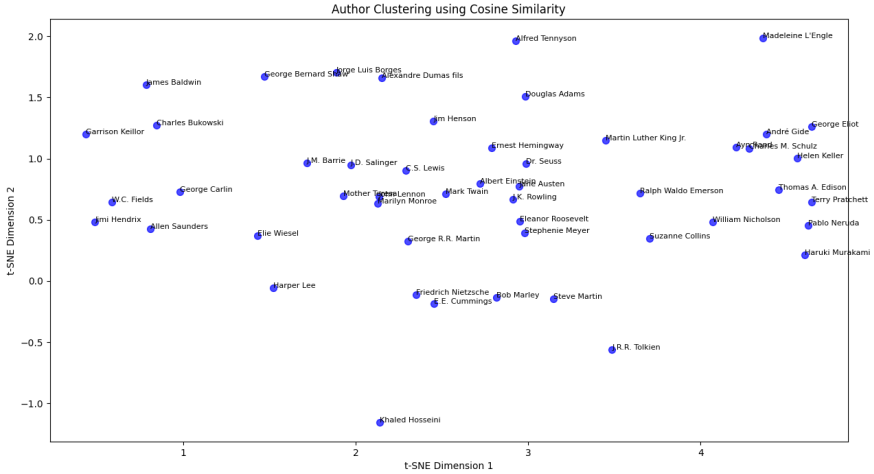
- **Introduction**


Number of Words Distribution
Number of Sentences
50000
40000
30000
20000
10000
0
0
100
200
300
400
500
600
700
Number of Words


know
one
love
life
will
people
thing
man
time
think
never
said
world
make
see
say
good
God
now
even
want
way
little
heart


Frequency
60000
50000
40000
30000
20000
10000
0
the
and
to
of
a
i
in
is
that
you
it
not
was
for
he
be
but
as
have
with
Words


Relationship Between Author Year of Birth and Quote Length
Quote Length
400
350
300
250
200
150
100
50
1775
1800
1825
1850
1875
1900
1925
1950
1975
Author Year of Birth



Feature extraction plays a crucial role in deep learning and machine learning tasks on text data. Since raw text is unstructured, it needs to be transformed into a numerical format for model training. There are various methods to extract these features, such as using classical approaches like TF-IDF or leveraging advanced models like pre-trained language models.

- **Approach**

  In this study, I will explore two distinct approaches. First, I will use traditional machine learning with TF-IDF to extract features from the text. Second, I will apply a pre-trained BERT model to obtain contextual embeddings for a deeper representation of the text.

- **Evaluation**

  To evaluate the effectiveness of the extracted features, I will employ a basic SVM model as a classifier to assess the quality of the features produced by both methods.

- **Discussion**

  The results indicate that TF-IDF features slightly outperform the BERT embeddings (0.5 accuracy vs. 0.49). This could be due to the specific nature of the dataset or the SVM's sensitivity to certain feature types. However, to provide a more robust comparison, I will use a pre-trained BERT model to fine-tune on the task and use the BERT embeddings for training, ensuring a more fair evaluation.

- **Implementation Details**

  - **TF-IDF**

```python
def read_quotes_from_json(filename="quotes.json"):
    with open(filename, 'r', encoding='utf-8') as f:
        data = json.load(f)
```

SVM Model Evaluation (TF-IDF):

	precision	recall	f1-score	support
0	0.34	0.60	0.43	178
1	0.00	0.00	0.00	10
2	0.43	0.09	0.15	34
4	0.60	0.06	0.11	48
5	0.48	0.54	0.51	174
6	0.70	0.21	0.33	33
7	0.49	0.46	0.48	216
8	0.28	0.51	0.36	197
9	0.87	0.23	0.37	56
10	0.57	0.56	0.57	202
11	0.72	0.31	0.43	58
12	0.74	0.24	0.36	59
13	0.63	0.40	0.49	80
14	0.69	0.53	0.60	129
15	0.37	0.48	0.42	195
16	0.41	0.54	0.47	200
17	1.00	0.13	0.23	23
18	0.46	0.37	0.41	139
19	0.52	0.15	0.23	87
20	0.48	0.45	0.46	211
21	0.53	0.58	0.55	213
22	0.70	0.66	0.68	191
...				
accuracy			0.50	6265
macro avg	0.59	0.40	0.42	6265
weighted avg	0.55	0.50	0.50	6265

Figure 2.1: TF-IDF

SVM Model Evaluation (BERT):

	precision	recall	f1-score	support
0	0.40	0.54	0.46	178
1	0.00	0.00	0.00	10
2	0.35	0.53	0.42	34
4	0.26	0.25	0.25	48
5	0.46	0.55	0.50	174
6	0.28	0.33	0.30	33
7	0.39	0.40	0.40	216
8	0.41	0.52	0.46	197
9	0.35	0.43	0.38	56
10	0.53	0.59	0.56	202
11	0.44	0.47	0.45	58
12	0.46	0.44	0.45	59
13	0.38	0.41	0.40	80
14	0.60	0.54	0.57	129
15	0.43	0.43	0.43	195
16	0.39	0.47	0.42	200
17	0.08	0.09	0.08	23
18	0.35	0.33	0.34	139
19	0.32	0.28	0.29	87
20	0.48	0.45	0.46	211
21	0.63	0.64	0.64	213
22	0.54	0.65	0.59	191
...				
accuracy			0.49	6265
macro avg	0.43	0.42	0.42	6265
weighted avg	0.49	0.49	0.49	6265

Figure 2.2: BERT

```
    quotes = []
    authors = []

    for author, author_quotes in data.items():
        quotes.extend(author_quotes)
        authors.extend([author] * len(author_quotes))

    return quotes, authors

quotes, authors = read_quotes_from_json()
tfidf_vectorizer = TfidfVectorizer()
X_tfidf = tfidf_vectorizer.fit_transform(quotes)

```

– BERT

```
            tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertModel.from_pretrained("bert-base-uncased")

def get_bert_embeddings_batch(texts, batch_size=4):
    all_embeddings = []
    for i in range(0, len(texts), batch_size):
        batch_texts = texts[i:i + batch_size]
        inputs = tokenizer(batch_texts, padding=True, truncation=True, max_length
    =128, return_tensors='pt')
        with torch.no_grad():
            outputs = model(**inputs)
```

```
        embeddings = outputs.last_hidden_state.mean(dim=1)
        all_embeddings.append(embeddings.numpy())
    return np.vstack(all_embeddings)

X_bert = get_bert_embeddings_batch(quotes, batch_size=4)

```

– **SVM Evaluation**

```
# Label Encoding
label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(authors)

# Train/Test Split
X_train_tfidf, X_test_tfidf, y_train, y_test = train_test_split(X_tfidf,
    labels_encoded, test_size=0.2, random_state=42)
X_train_bert, X_test_bert = train_test_split(X_bert, test_size=0.2, random_state=42)

# SVM
svm_tfidf = SVC(kernel='linear')
svm_tfidf.fit(X_train_tfidf, y_train)

svm_bert = SVC(kernel='linear')
svm_bert.fit(X_train_bert, y_train)

y_pred_tfidf = svm_tfidf.predict(X_test_tfidf)
print("SVM Model Evaluation (TF-IDF):")
print(classification_report(y_test, y_pred_tfidf))

y_pred_bert = svm_bert.predict(X_test_bert)
print("SVM Model Evaluation (BERT):")
print(classification_report(y_test, y_pred_bert))


```

2.4 Data Mining

2.4.1 Predict the name of the celebrity according to the statement based on the features you extracted above and evaluate on the given dataset with Train/Test ratio and appropriate metrics?

- **Introduction**

  The task involves predicting a celebrity's name from statements in an imbalanced dataset. Two approaches are compared: a traditional SVM using TF-IDF features and a modern BERT-based method.

- **Approach**

Bert-base-uncased was used to generate contextual embeddings for classification, chosen for its robust handling of text patterns and imbalanced data.

- **Evaluation**

  The dataset was split into training and testing sets with a standard 80/20 ratio. The performance was measured using accuracy.

```
Evaluation results: {'eval_accuracy': 0.6011173184357542, 'eval_loss': 1.6731152534484863, 'eval_runtime': 12.0105, 'eval_samples_per
 second': 521.626, 'eval steps per second': 65.276, 'epoch': 3.0}
```

Figure 2.3: Evaluation of Bert Model

The traditional approach with SVM achieved an accuracy of 50%, reflecting the limitations of TF-IDF features in understanding complex text and dealing with imbalanced data. In contrast, the BERT-based approach significantly outperformed, achieving an accuracy of 60%.

The improved results with BERT can be attributed to its ability to capture semantic and syntactic information effectively. Additionally, fine-tuning the BERT model could further enhance performance by tailoring it to the dataset's specific nuances.

- **Discussion**

  The experiment highlights the advantage of leveraging pretrained models like BERT for text classification tasks. While traditional ML techniques like SVM are effective for simpler tasks, their limitations become apparent when dealing with imbalanced datasets or complex text structures.

  The BERT-based approach offers a scalable solution, especially for datasets where contextual understanding is critical. Future work could explore fine-tuning BERT on the dataset to achieve even better results, as well as investigating other advanced techniques to address dataset imbalance.

- **Implementation details**

```
quotes, authors = read_quotes_from_json()
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(authors)
quotes_train, quotes_test, labels_train, labels_test = train_test_split(
    quotes, labels, test_size=0.2, random_state=42
)

train_dataset = Dataset.from_dict({'text': quotes_train, 'label': labels_train})
test_dataset = Dataset.from_dict({'text': quotes_test, 'label': labels_test})

model_name = "bert-base-uncased"
tokenizer = BertTokenizer.from_pretrained(model_name)
def tokenize_function(examples):
    return tokenizer(examples['text'], padding="max_length", truncation=True, max_length
    =128)

train_dataset = train_dataset.map(tokenize_function, batched=True)
```

```
test_dataset = test_dataset.map(tokenize_function, batched=True)
train_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'label'])
test_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'label'])
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=len(set(
    labels)))

def compute_metrics(p):
    predictions, labels = p
    preds = predictions.argmax(axis=1)
    accuracy = accuracy_score(labels, preds)
    return {'eval_accuracy': accuracy}

training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    logging_dir='./logs',
    logging_steps=10,
    load_best_model_at_end=True,
    metric_for_best_model="eval_accuracy",
    greater_is_better=True,
    report_to="none"
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)

trainer.train()
eval_results = trainer.evaluate()
print(f"Evaluation results: {eval_results}")

```

2.4.2 Suggest a way to calculate the similarity of styles between authors and find the authors with the most similar styles?

- **Introduction**

  By analyzing the semantic representation of authors' quotes, we can uncover patterns that define their writing style and explore relationships between their works. In this work, we propose a method to compute

the stylistic similarity between authors based on their quote embeddings. Using precomputed embeddings from a BERT-based model, we calculate a mean embedding for each author and measure pairwise cosine similarity. This approach is efficient, leveraging existing embeddings to provide insights into stylistic relationships.

- **Approach**

  We group quotes by authors and compute a mean embedding for each, using precomputed 768-dimensional embeddings. Pairwise cosine similarity between these mean embeddings quantifies the stylistic similarity between authors.

- **Evaluation**

  We evaluate by identifying the most similar author pairs and qualitatively analyzing their quotes. Stylistic overlaps are assessed alongside tag distributions to validate the results.

```
Top 5 most similar authors:
J.K. Rowling and Jane Austen: similarity = 0.9543
Albert Einstein and Dr. Seuss: similarity = 0.9536
Albert Einstein and J.K. Rowling: similarity = 0.9524
Albert Einstein and Mark Twain: similarity = 0.9503
Albert Einstein and Jane Austen: similarity = 0.9488
```

Figure 2.4: Top 5 authors with the most similar style

- **Discussion**

  Future work could explore more robust methods to address data imbalances and refine results further.

- **Implementation details**

```python
from sklearn.metrics.pairwise import cosine_similarity

authors = list(author_dictionary.keys())
quote_counts = [len(author_dictionary[author]["quotes"]) for author in authors]

author_embeddings = []
start_idx = 0

for count in quote_counts:
    author_quotes_embeddings = X_bert[start_idx:start_idx + count]

    mean_embedding = np.mean(author_quotes_embeddings, axis=0)
    author_embeddings.append(mean_embedding)

```

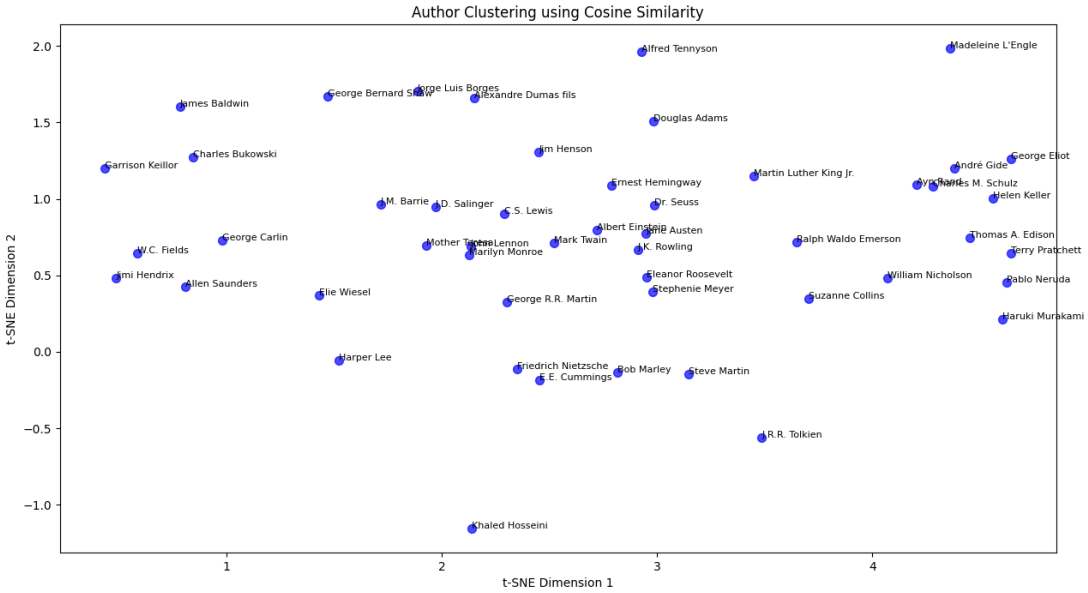


Figure 2.5: tSNE for authors' styles

```
    start_idx += count

author_embeddings = np.array(author_embeddings)

similarity_matrix = cosine_similarity(author_embeddings)

most_similar = []
n_authors = len(authors)
for i in range(n_authors):
    for j in range(i + 1, n_authors):
        similarity = similarity_matrix[i, j]
        most_similar.append((authors[i], authors[j], similarity))

most_similar.sort(key=lambda x: -x[2])
print("Top 5 most similar authors:")
for author1, author2, sim in most_similar[:5]:
    print(f"{author1} and {author2}: similarity = {sim:.4f}")


```