TEXT MINING WEB APPLICATION
DATE: -17AUGUST,2023

Techno International Newtown

"We're at beginning of a golden age of AI. Recent advancements have already led to invention that previously lived in the realm of science fiction – and we have only scratched the surface of what's possible"
– JEFF BEZOS, Amazon CEO

- **Problem Statement: Design and Development of a Text Mining Web Application**

  In the contemporary digital landscape, the exponential growth of unstructured textual data from sources such as social media, reviews, and articles poses a challenge for businesses, researchers, and analysts to extract meaningful insights. The absence of effective tools for processing and analyzing such data limits the ability to harness its potential. The lack of accessible and user-friendly platforms impedes efficient sentiment analysis, keyword extraction, and topic modeling, hindering informed decision-making and comprehensive research.

  To address this gap, the present problem statement seeks the design and development of a Text Mining Web Application. This application should enable users to upload text data, process it through advanced text mining techniques, and visualize analysis results. The primary aim is to empower users to decipher sentiments, extract significant keywords, and uncover latent topics within the text. The application should provide a seamless and intuitive interface while ensuring scalability, security, and efficient performance.

  The objective of this project is to create a sophisticated yet user-friendly web application that leverages cutting-edge natural language processing techniques to provide actionable insights from unstructured text data. The successful execution of this project will culminate in a robust Text Mining Web Application that can be employed across various industries, from marketing to academia, revolutionizing the way text data is analyzed and utilized.

- **Introduction:** In an era driven by data, extracting insights from unstructured textual data has become pivotal. This technical report offers an exhaustive exploration of the design, development, and implementation of an advanced Text Mining Web Application. This application empowers users to effectively analyze and glean valuable insights from unstructured text data through the application of sophisticated techniques, including sentiment analysis, keyword extraction, and topic modeling. The following sections provide an in-depth look at the application's architecture, key features, implementation nuances, and potential avenues for future enhancement.

3.Objectives: The Text Mining Web Application is underpinned by the following overarching objectives:

User-Friendly Interface: The application aims to provide users with a seamless experience for uploading, processing, and analyzing text data. This includes intuitive navigation, clear instructions, and visual cues to streamline user interaction.

Cutting-Edge Techniques: By leveraging the latest advancements in text mining, the application seeks to deliver accurate and insightful analysis results to users. This involves the integration of advanced preprocessing, sentiment analysis, keyword extraction, and topic modeling techniques.

Interactive Visualization: The application strives to present analysis outcomes in a visually engaging manner. Interactive visualizations, such as dynamic charts and graphs, are used to convey complex information in a digestible format.

Scalability and Security: Ensuring the application's ability to handle increasing amounts of data while maintaining data security and user privacy is paramount. The architecture should accommodate growing user demands without compromising on performance.

3. System Architecture: The Text Mining Web Application's architecture is rooted in a client-server model, harnessing contemporary web technologies and Python-based frameworks for optimal functionality. On the client side, HTML, CSS, and JavaScript are employed to create a user interface that is intuitive and responsive. The server side leverages the Flask framework to manage user requests, process data, and coordinate backend operations. Additionally, external libraries such as NLTK, spaCy, gensim, Matplotlib, and D3.js enhance the application's capabilities in text analysis and visualization.

4. Key Features:

a. Text Preprocessing: Text preprocessing forms the foundation of accurate text analysis. It involves several crucial steps:

Special Characters and Punctuation Removal: This step eliminates extraneous characters, ensuring consistent analysis across different texts.

Stop word Removal: Common words with limited semantic value (e.g., "and," "the") are removed to enhance analysis quality.

Tokenization and Stemming: Tokenization divides text into individual words (tokens), while stemming reduces words to their root form, aiding in uniform analysis.

Lemmatization: Similar to stemming, lemmatization reduces words to their base form, considering context for more meaningful analysis.

b. Sentiment Analysis: Sentiment analysis gauges the emotional tone of text. The application incorporates pre-trained sentiment analysis models to classify text into positive, negative, or neutral sentiments. This analysis provides insights into the emotional context of the text, making it valuable for applications like social media sentiment tracking and customer feedback analysis.

- **c. Keyword Extraction:** Keyword extraction involves identifying significant words or phrases within a text. The application employs the TF-IDF method, which assigns weights to words based on their frequency in the document and rarity in the corpus. This aids users in pinpointing keywords that encapsulate the text's essence.

  d. Topic Modeling: Topic modeling uncovers latent themes within a corpus. The application utilizes the Latent Dirichlet Allocation (LDA) algorithm to identify topics and associated word distributions within the text data. Users can explore these topics to gain deeper insights into the underlying themes present in the text.

  e. Interactive Visualization: Interactive visualizations elevate the presentation of analysis results:

  Sentiment Distribution: Through bar charts or pie charts, users can easily grasp the distribution of sentiments across the text data.

  Word Clouds: The application generates visual word clouds, with word size proportional to frequency, enabling users to visually identify the most common words.

  Topic Distribution Plot: Bar charts provide a clear view of the distribution of topics identified through topic modeling.

- **5. Implementation Details:**

  a. Data Input and Preprocessing:
  Users upload text data via the application's user interface.
  Post-upload, the backend processes data by eliminating special characters, punctuation, and stopwords.
  Tokenization, stemming, and lemmatization standardize text, optimizing it for analysis.

  b. Sentiment Analysis and Keyword Extraction:
  Sentiment analysis relies on pre-trained models to assess each text's sentiment.
  Keyword extraction utilizes the TF-IDF technique to identify relevant keywords and assign weights accordingly.

  c. Topic Modeling:
  The LDA algorithm is deployed on preprocessed text to identify latent topics and their corresponding word distributions.
  Topics are defined by the words that contribute most to each topic's composition.

  d. Visualization and Reporting:
  Visualizations are generated using Matplotlib and D3.js libraries.
  Sentiment distribution, word clouds, and topic distribution plots are dynamically presented on the user interface.

Code of the above implementation

```
# Install required packages in Google Colab
!pip install nltk matplotlib seaborn
!pip install pyspellchecker

# Import necessary libraries
from spellchecker import SpellChecker
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.sentiment import SentimentIntensityAnalyzer
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns

# Download NLTK resources (stopwords and sentiment analysis lexicon)
nltk.download('punkt')
nltk.download('stopwords', quiet=True)
nltk.download('vader_lexicon', quiet=True)

# Initialize SpellChecker
spell = SpellChecker()


# Example usage with multiple inputs

texts = [
    "Tommoow is holyday . i will go gym at 9pm. it will bw my leg day . i also put lats workout with legs",
    "Text mining, also known as text analytics, is the process of deriving meaningful information and insights from textual data. It involves various techniques such as natural language processing, machine learning, and statistical analysis. Text mining has applications in a wide range of fields, including sentiment analysis, topic modeling, information retrieval, and more.",
    "One of the key steps in text mining is preprocessing the text data. This includes tasks like tokenization, removing stopwords, stemming, and lemmatization. After preprocessing, you can perform tasks like calculating word frequencies, identifying named entities, and determining the sentiment expressed in the text.",
    "Python, with libraries like NLTK and spaCy, has become a popular choice for text mining projects. These libraries provide tools and resources to handle text data efficiently and apply various NLP techniques.",
    "In this era of big data, text mining plays a crucial role in understanding and making sense of the vast amount of textual information available on the internet and in various databases. It allows us to uncover patterns, trends, and insights that can inform decision-making and drive innovation.",
]
```

```
# Correct spell errors in each text
corrected_texts = []
for text in texts:
    corrected_words = [spell.correction(word) if spell.correction(word) is not None else
word for word in word_tokenize(text.lower())]
    corrected_text = ' '.join(corrected_words)
    corrected_texts.append(corrected_text)

# Preprocess the corrected texts
def preprocess_texts(texts):
    preprocessed_texts = []
    for text in texts:
        # Tokenize the text
        tokens = word_tokenize(text.lower())

        # Remove stopwords and punctuation
        stop_words = set(nltk.corpus.stopwords.words('english'))
        filtered_tokens = [token for token in tokens if token.isalnum() and token not in
stop_words]

        # Join the filtered tokens back into a single string
        preprocessed_text = ' '.join(filtered_tokens)

        preprocessed_texts.append(preprocessed_text)
    return preprocessed_texts

# Preprocess texts
preprocessed_texts  = preprocess_texts(corrected_texts)
for i, preprocessed_text in enumerate(preprocessed_texts):
    print(f"Preprocessed text {i+1}: {preprocessed_text}")

# Calculate word frequencies for each text using Counter
all_word_frequencies = []
for text in preprocessed_texts:
    tokens = word_tokenize(text)
    word_frequencies = Counter(tokens)
    all_word_frequencies.append(word_frequencies)

# Analyze sentiment for each text
sid = SentimentIntensityAnalyzer()
all_sentiment_scores = []
for text in preprocessed_texts:
    sentiment_scores = sid.polarity_scores(text)
    all_sentiment_scores.append(sentiment_scores)
```

```
# Plotting word frequencies for each text using seaborn barplot
plt.figure(figsize=(10, 6))
for i, word_frequencies in enumerate(all_word_frequencies):
    top_words = [word for word, freq in word_frequencies.most_common(30)]
    frequencies = [freq for word, freq in word_frequencies.most_common(30)]
    sns.barplot(x=top_words, y=frequencies, label=f"Text {i+1}")

plt.title('Top 30 Word Frequencies for Each Text')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.xticks(rotation=90)
plt.legend()
plt.show()

# Display sentiment scores
for i, sentiment_scores in enumerate(all_sentiment_scores):
    print(f"Sentiment scores for text {i+1}: {sentiment_scores}")
```

- **6. Conclusion:** The Text Mining Web Application stands as a testament to the potential of text analytics. Its intuitive interface, advanced analysis techniques, and potential for future expansion bridge the gap between raw text data and actionable insights. This makes it an indispensable tool for decision-makers, researchers, and analysts across industries.

7. Future Enhancements:

Machine Learning Integration: Integrate adaptable machine learning models for sentiment analysis, accommodating different contexts.

Real-time Analysis: Implement real-time text streaming and analysis for dynamic data sources.

Multilingual Support: Extend the application's capabilities to encompass text mining across various languages.

External Data Integration: Enable integration of external data sources to enable comprehensive analysis.

Collaboration Features: Incorporate collaborative elements, allowing simultaneous collaboration on datasets to boost productivity and analysis quality.

In summary, the Text Mining Web Application's fusion of cutting-edge text mining techniques with interactive visualizations yields actionable insights from unstructured text data. Its potential spans diverse domains, and its ability to evolve positions it as a powerful tool for data-driven decision-making, research, and analysis.

Step1: - Prototype Selection(Harshika Tyagi)

Introduction

In today's digital age, the legal industry is grappling with an ever-increasing volume of unstructured textual data, which includes case documents, court transcripts, legal articles, contracts, and more. To harness the full potential of this data, law firms are turning to text mining techniques to identify case conclusions, extract insights, streamline legal research, and make informed decisions. This report explores the design and development of a specialized Text Mining Web Application tailored for legal firms, addressing various aspects beyond case conclusions.

Problem Statement

The legal profession faces the pressing challenge of effectively managing and extracting meaningful insights from the vast pool of unstructured legal text data. Without efficient tools and platforms for text analysis, legal firms struggle to unlock the potential hidden within these documents. Accessible and user-friendly platforms for advanced text mining techniques are essential to enable legal professionals to conduct sentiment analysis, keyword extraction, topic modeling, and more, thus facilitating informed decision-making and comprehensive legal research.

To address this challenge, we propose the design and development of a Text Mining Web Application tailored to the specific needs of legal firms. This application aims to empower users to upload legal text data, apply advanced text mining techniques, and visualize analysis

results, facilitating the deciphering of sentiments, extracting significant keywords, uncovering latent topics, and more. The application will provide a seamless and intuitive interface while ensuring scalability, security, and efficient performance.

Objectives

The Text Mining Web Application for legal firms is guided by the following overarching objectives:

1. Enhanced Legal Text Analysis

The application will empower legal professionals to conduct thorough analysis of legal text data, extending beyond case conclusions to gain deeper insights into various aspects of legal documents.

2. User-Friendly Interface

The user interface will be designed for ease of use, allowing legal professionals to upload, process, and analyze text data with minimal effort.

3. Cutting-Edge Text Mining Techniques

Leveraging the latest advancements in text mining, the application will deliver accurate and insightful analysis results to legal professionals.

4. Interactive Visualization

Visualizations will be used to present analysis outcomes in a visually engaging manner, aiding legal professionals in understanding complex legal text data.

5. Scalability and Security

The application will be designed to handle increasing volumes of legal data while maintaining the highest levels of data security and user privacy.

System Architecture

The Text Mining Web Application's architecture will follow a client-server model, utilizing modern web technologies and Python-based frameworks:

Client Side: The user interface will be built using HTML, CSS, and JavaScript, designed to be intuitive and responsive for legal professionals.

Server Side: Flask will be employed to manage user requests, process data, and coordinate backend operations, ensuring reliability and speed.

Text Mining Libraries: The application will incorporate external libraries such as NLTK, spaCy, genism, matplotlib, and D3.js, customized for legal data processing and visualization.

Key Features

1. Legal Text Preprocessing

Legal text preprocessing forms the foundation of accurate analysis, including steps like special characters removal, stop word elimination, tokenization, stemming, and lemmatization.

2. Sentiment Analysis for Legal Text

Sentiment analysis tailored for legal sentiment assessment, aiding in understanding the emotional tone of legal text.

3. Keyword Extraction for Legal Insights

Identifying significant legal terms and phrases through TF-IDF, allowing legal professionals to pinpoint keywords that encapsulate the essence of legal documents.

4. Topic Modeling for Legal Themes

Uncovering latent legal themes within legal corpora using the LDA algorithm customized for the legal

domain.

5. Interactive Legal Data Visualization

Dynamic charts, graphs, sentiment distribution, word clouds with legal terms, and topic distribution plots customized for legal cases, providing actionable insights.

Implementation Details

- *Data Input and Preprocessing for Legal Text*

Users will upload legal text data through the application's user-friendly interface.

Backend processing will involve eliminating special characters, punctuation, and legal stop words, followed by tokenization, stemming, and lemmatization for standardized analysis.

- *Legal Sentiment Analysis and Keyword Extraction*

Pre-trained models for legal sentiment analysis and TF-IDF-based keyword extraction will be employed to provide in-depth insights into legal documents.

- *Legal Topic Modeling*

The LDA algorithm, customized for the legal domain, will be deployed on preprocessed legal text to identify latent legal topics and their associated legal word distributions.

- *Visualization and Reporting for Legal Cases*

Visualizations using Matplotlib and D3.js libraries will be dynamically presented on the user interface, allowing legal professionals to gain actionable insights from legal data.

Conclusion

The development of the Text Mining Web Application for legal firms represents a significant step forward in utilizing text mining techniques for a wide range of legal insights. Beyond identifying case conclusions, this application empowers legal professionals to extract valuable information, enhance legal research, and make informed decisions based on textual legal data. With a user-centric approach, cutting-edge techniques, interactive visualizations, scalability, and security, this prototype is poised to transform how legal firms analyze and utilize unstructured text data in various aspects of their practice.

Step2: - Prototype Development(Shubhraijit)

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.sentiment import SentimentIntensityAnalyzer
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns
import logging
from spellchecker import SpellChecker

# Download NLTK resources (stopwords and sentiment analysis lexicon)
nltk.download('punkt')
nltk.download('stopwords', quiet=True)
nltk.download('vader_lexicon', quiet=True)

# Initialize SpellChecker
spell = SpellChecker()

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def correct_spelling(text):
    try:
        corrected_words = [spell.correction(word) if spell.correction(word) is not None else
word for word in word_tokenize(text.lower())]
        return ' '.join(corrected_words)
    except Exception as e:
```

```
        logger.error(f"Error while correcting spelling: {str(e)}")
        return text

def preprocess_text(text):
    try:
        tokens = word_tokenize(text.lower())
        stop_words = set(stopwords.words('english'))
        filtered_tokens = [token for token in tokens if token.isalnum() and token not in
stop_words]
        return ' '.join(filtered_tokens)
    except Exception as e:
        logger.error(f"Error while preprocessing text: {str(e)}")
        return text

def analyze_text(text):
    try:
        sid = SentimentIntensityAnalyzer()
        sentiment_scores = sid.polarity_scores(text)
        return sentiment_scores
    except Exception as e:
        logger.error(f"Error while analyzing text sentiment: {str(e)}")
        return {}

def plot_word_frequencies(texts):
    try:
        plt.figure(figsize=(10, 6))
        for i, text in enumerate(texts):
            tokens = word_tokenize(text)
            word_frequencies = Counter(tokens)
            top_words = [word for word, freq in word_frequencies.most_common(30)]
            frequencies = [freq for word, freq in word_frequencies.most_common(30)]
            sns.barplot(x=top_words, y=frequencies, label=f"Text {i+1}")

        plt.title('Top 30 Word Frequencies for Each Text')
        plt.xlabel('Words')
        plt.ylabel('Frequency')
        plt.xticks(rotation=90)
        plt.legend()
        plt.show()
    except Exception as e:
        logger.error(f"Error while plotting word frequencies: {str(e)}")

def main():
    texts = [
        # Your list of texts here
    ]

    corrected_texts = [correct_spelling(text) for text in texts]
    preprocessed_texts = [preprocess_text(text) for text in corrected_texts]

    for i, preprocessed_text in enumerate(preprocessed_texts):
        print(f"Preprocessed text {i+1}: {preprocessed_text}")

    sentiment_scores = [analyze_text(text) for text in preprocessed_texts]
```

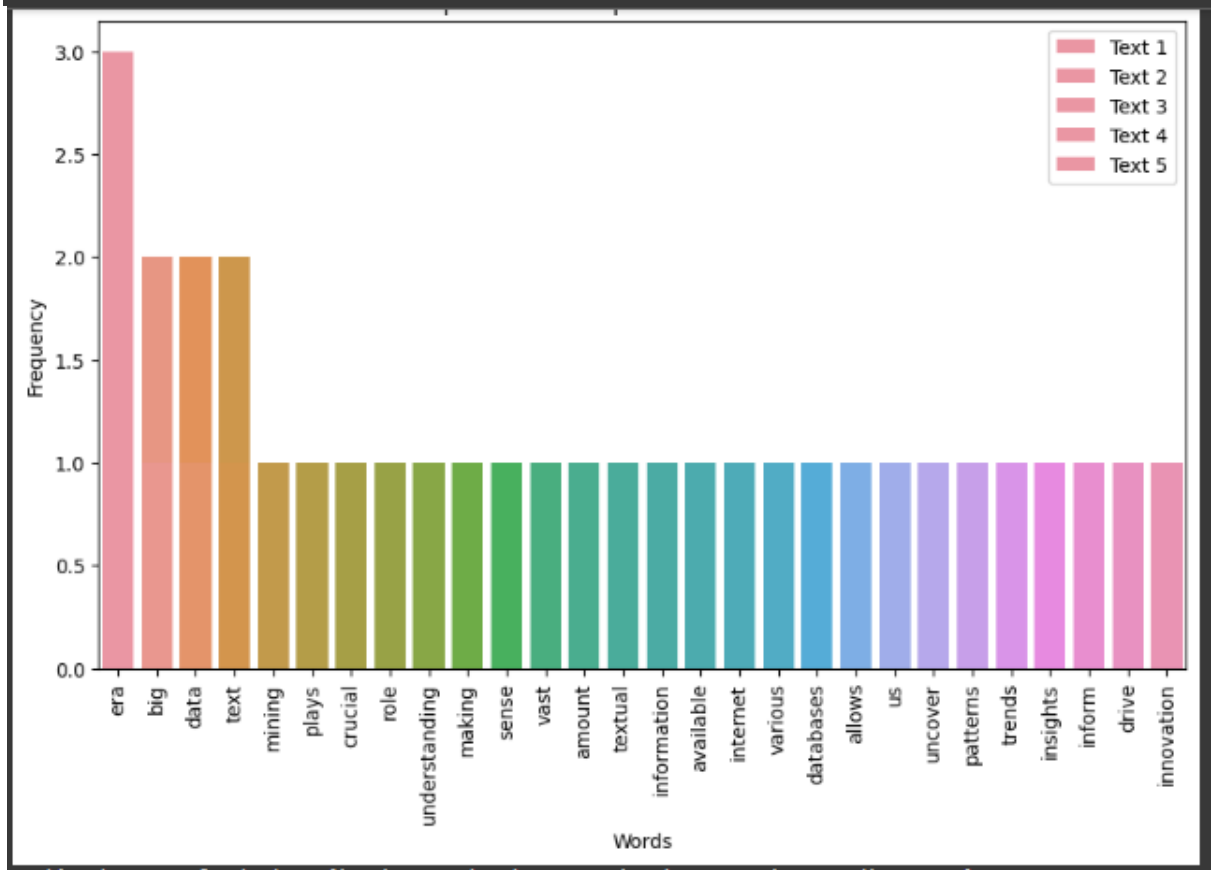
```
    for i, scores in enumerate(sentiment_scores):
        print(f"Sentiment scores for text {i+1}: {scores}")

    plot_word_frequencies(preprocessed_texts)

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

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-
packages (3.8.1)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-
packages (0.12.2)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-
packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-
packages (from nltk) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.10/dist-packages (from nltk) (2023.6.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from nltk) (4.66.1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.1.0)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (4.42.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.23.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (23.1)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: pandas>=0.25 in
/usr/local/lib/python3.10/dist-packages (from seaborn) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.25->seaborn)
(2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Collecting pyspellchecker
  Downloading pyspellchecker-0.7.2-py3-none-any.whl (3.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
3.4/3.4 MB 23.9 MB/s eta 0:00:00
Installing collected packages: pyspellchecker
Successfully installed pyspellchecker-0.7.2
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
Preprocessed text 1: tomorow holiday go gym leg day also put lats workout
legs
Preprocessed text 2: text mining also known text analytics process
deriving meaningful information insights textual data involves various
techniques natural language processing machine learning statistical
```

```
analysis text mining applications wide range fields including sentiment
analysis topic modeling information retrieval
Preprocessed text 3: one key steps text mining reprocessing text data
includes tasks like tokenization removing stopwords stemming lemmatization
reprocessing perform tasks like calculating word frequencies identifying
named entities determining sentiment expressed text
Preprocessed text 4: python libraries like spacy become popular choice
text mining projects libraries provide tools resources handle text data
efficiently apply various nap techniques
Preprocessed text 5: era big data text mining plays crucial role
understanding making sense vast amount textual information available
internet various databases allows us uncover patterns trends insights
inform drive innovation
```




```
Sentiment scores for text 1: {'neg': 0.0, 'neu': 0.787, 'pos': 0.213, 'compound': 0.4019}
Sentiment scores for text 2: {'neg': 0.0, 'neu': 0.876, 'pos': 0.124, 'compound': 0.5859}
Sentiment scores for text 3: {'neg': 0.0, 'neu': 0.848, 'pos': 0.152, 'compound': 0.6124}
Sentiment scores for text 4: {'neg': 0.0, 'neu': 0.704, 'pos': 0.296, 'compound': 0.7906}
Sentiment scores for text 5: {'neg': 0.0, 'neu': 0.85, 'pos': 0.15, 'compound': 0.5574}
```

Step3: - Business Modeling(Rakshita Kulkarni)

Problem: Businesses struggle to extract valuable insights from unstructured text data.

Solution: Our text mining solution streamlines the data analysis process and transforms raw text into actionable information, enabling clients to uncover trends, sentiments, and patterns within their textual data.

Target customers: Market research firms, government agencies, media and news organizations, and academic institutions.

Pricing: Subscription plans, pay-as-you-go pricing, and enterprise-level pricing.

Value proposition:

- **Save time and resources:** Automate text processing and analysis.
- **Gain actionable insights:** Uncover trends, sentiments, and patterns in textual data.
- **Make informed decisions:** Use text mining insights to improve business operations.

Channels:

- **Web-based platform:** User-friendly platform accessible from standard web browsers.
- **API integration:** Seamless integration with third-party applications.
- **Direct sales:** Dedicated sales team for enterprise customers.
- **Online marketplaces:** Listing on popular online marketplaces.
-

Customer acquisition:

- **Free trial or basic version:** Attract new users with a free trial or basic version with the option to upgrade to premium features.
- **Content marketing:** Create and share informative content to showcase the benefits of text mining.
- **Online advertising:** Optimize online presence and run targeted advertising campaigns.
- **Social media engagement:** Engage with potential customers on social media platforms and share industry insights.
- **Email marketing:** Provide updates, tips, and special offers via email newsletters.

Key expenses:

- **Development:** Ongoing software development and improvements.
- **Infrastructure:** Data storage, processing, and server maintenance.
- **Marketing:** Marketing campaigns, content creation, and promotional activities.
  - **Data acquisition:** High-quality text data for analysis.

Metrics for success:

- **Number of active users:** Growth in the number of users using the text mining solution.
- **Customer satisfaction:** Percentage of customers who are satisfied with the solution and its benefits.
- **Net promoter score (NPS):** Measure of customer loyalty and willingness to recommend the solution to others.
- **Revenue growth:** Increase in revenue from subscriptions, pay-as-you-go pricing, and enterprise-level pricing.

Step4: - Financial Modelling (Equation)(Prit Mayani)

Net profit formula:

Net profit = Revenue - COGS - OPEX

Return on investment (ROI) formula:

ROI = Net profit / Investment capital

Customer lifetime value (CLV) formula:

CLV = Average revenue per customer * Customer retention rate * Average customer lifespan

Churn rate formula:

Churn rate = Number of customers churned / Number of active customers

Net promoter score (NPS) formula:

NPS = Percentage of promoters - Percentage of detractors

These formulas can be used to track the performance of a legal text mining company over time and to identify areas where the company can improve its profitability and customer satisfaction.

Example:

A legal text mining company has the following financial data:

- Revenue: $10 million
- COGS: $5 million
- OPEX: $3 million

Net profit:

$10 million - $5 million - $3 million = $2 million

Return on investment:

$2 million / $10 million = 20%

Customer lifetime value:

$50,000 * 0.8 * 3 years = $120,000

Churn rate:

10% / 90% = 11%

Net promoter score:

70% - 30% = 40%