Prerna Ladkani

Roll:37 Div:D11AD

Experiment 7

Aim:Perform the steps involved in Text Analytics in Python & R

Lab Outcomes (LO): Design Text Analytics Application on a given data set. (LO4)

Task to be performed :

1.Explore Top-5 Text Analytics Libraries in Python (w.r.t Features & Applications)

2.Explore Top-5 Text Analytics Libraries in R (w.r.t Features & Applications)

3.Perform the following experiments using Python & R

4.Tokenization (Sentence & Word)

5.Frequency Distribution

6.Remove stopwords & punctuations

7.Lexicon Normalization (Stemming, Lemmatization)

8.Part of Speech tagging

9.Named Entity Recognization

Scrape data from a website

4.Prepare a document with the Aim, Tasks performed, Program, Output, and Conclusion.

```
#scattertext
# Install and load necessary libraries
import scattertext as st
import pandas as pd
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string

# Create a simple DataFrame
data = {'category': ['Document1', 'Document2'],
        'text': ['This is a simple document.', 'Another document for testing.']}

df = pd.DataFrame(data)

# Tokenization (Sentence & Word)
df['tokenized_sent'] = df['text'].apply(lambda x: [sent for sent in st.whitespace_nlp_with_sentences(x).sents])
df['tokenized_word'] = df['text'].apply(lambda x: [token.lower() for token in word_tokenize(x) if token.isalpha()])

# Frequency Distribution
word_freq = pd.Series([item for sublist in df['tokenized_word'] for item in sublist]).value_counts()

# Remove stopwords & punctuations
stop_words = set(stopwords.words('english'))
df['filtered_words'] = df['tokenized_word'].apply(lambda x: [word for word in x if (word not in stop_words and word not in string.punctuation

# Lexicon Normalization (No Stemming/Lemmatization in this example)
df['normalized_text'] = df['text'].apply(lambda x: x.lower())

# Part of Speech tagging and Named Entity Recognition (Not directly applicable in scattertext)
# Scrape data from a website (Not applicable in scattertext)

# Display results
print("Tokenization (Sentence):")
print(df['tokenized_sent'])
print("\nTokenization (Word):")
print(df['tokenized_word'])
print("\nFrequency Distribution:")
print(word_freq)
print("\nRemove stopwords & punctuations:")
print(df['filtered_words'])
```

```
print("\nLexicon Normalization:")
print(df['normalized_text'])
```

```
Tokenization (Sentence):
0      [[this, is, a, simple, document, .]]
1    [[another, document, for, testing, .]]
Name: tokenized_sent, dtype: object

Tokenization (Word):
0      [this, is, a, simple, document]
1    [another, document, for, testing]
Name: tokenized_word, dtype: object

Frequency Distribution:
document    2
this        1
is          1
a           1
simple      1
another     1
for         1
testing     1
dtype: int64

Remove stopwords & punctuations:
0              [simple, document]
1    [another, document, testing]
Name: filtered_words, dtype: object

Lexicon Normalization:
0       this is a simple document.
1    another document for testing.
Name: normalized_text, dtype: object
```

```
#spacy
import spacy

# Load the English NLP model
nlp = spacy.load('en_core_web_sm')

# Example text
text = "SpaCy is a powerful NLP library."

# Lexicon Normalization (Lemmatization)
doc = nlp(text)
lemmatized_text = [token.lemma_ for token in doc]

# Part of Speech tagging
pos_tags = [(token.text, token.pos_) for token in doc]

# Named Entity Recognition
entities = [(ent.text, ent.label_) for ent in doc.ents]

# Display results
print("Lexicon Normalization (Lemmatization):", lemmatized_text)
print("\nPart of Speech tagging:", pos_tags)
print("\nNamed Entity Recognition:", entities)
```

```
Lexicon Normalization (Lemmatization): ['SpaCy', 'be', 'a', 'powerful', 'NLP', 'library', '.']

Part of Speech tagging: [('SpaCy', 'PROPN'), ('is', 'AUX'), ('a', 'DET'), ('powerful', 'ADJ'), ('NLP', 'PROPN'), ('library', 'NOUN'), ('

Named Entity Recognition: [('NLP', 'ORG')]
```

```
#textblob
import nltk

# Download the stopwords resource
nltk.download('stopwords')

# Now you can use TextBlob without encountering the LookupError
from textblob import TextBlob
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Example text
text = "TextBlob is easy to use and helpful for text processing."

# Tokenization
blob = TextBlob(text)
tokenized_text = blob.words

# Remove stopwords
stop_words = set(stopwords.words('english'))
filtered_text = [word for word in tokenized_text if word.lower() not in stop_words]

# Display results
print("Tokenization:", tokenized_text)
print("\nRemove stopwords:", filtered_text)
```

```
Tokenization: ['TextBlob', 'is', 'easy', 'to', 'use', 'and', 'helpful', 'for', 'text', 'processing']

Remove stopwords: ['TextBlob', 'easy', 'use', 'helpful', 'text', 'processing']
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
#sklearn
import nltk

# Download the stopwords resource
nltk.download('stopwords')

# Now you can use scikit-learn without encountering the LookupError
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Example data
text_data = ["This is a positive example.", "This is a negative example.", "Another positive text."]

# Tokenization
tokenized_text = [word_tokenize(text.lower()) for text in text_data]

# Remove stopwords
stop_words = set(stopwords.words('english'))
filtered_text = [[word for word in tokens if word.isalpha() and word not in stop_words] for tokens in tokenized_text]

# Display results
print("Tokenization:", tokenized_text)
print("\nRemove stopwords:", filtered_text)
```

```
Tokenization: [['this', 'is', 'a', 'positive', 'example', '.'], ['this', 'is', 'a', 'negative', 'example', '.'], ['another', 'positive',

Remove stopwords: [['positive', 'example'], ['negative', 'example'], ['another', 'positive', 'text']]
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
#numpy
import numpy as np

# Example array
arr = np.array([[1, 2, 3], [4, 5, 6]])

# Frequency Distribution for NumPy array
fdist = np.unique(arr, return_counts=True)

# Display results
print("Frequency Distribution for NumPy array:")
print(dict(zip(fdist[0], fdist[1])))
```

```
Frequency Distribution for NumPy array:
{1: 1, 2: 1, 3: 1, 4: 1, 5: 1, 6: 1}
```

R

```
#shiny
# Tokenization (Word)
text <- "This is a sample text for tokenization in Shiny."
tokens <- unlist(strsplit(tolower(text), "\\s+"))

# Display results separately
print("Tokenization (Word):")
print(tokens)

# Frequency Distribution
word_freq <- table(tokens)
print("\nFrequency Distribution:")
print(word_freq)

# Remove stopwords & punctuations
stopwords <- c("this", "is", "a", "for", "in")
filtered_tokens <- tokens[!(tokens %in% stopwords) & grepl("[a-zA-Z]", tokens)]
print("\nRemove stopwords & punctuations:")
print(filtered_tokens)

# Lexicon Normalization (No Stemming/Lemmatization in this example)
normalized_text <- tolower(text)
print("\nLexicon Normalization:")
print(normalized_text)

# Part of Speech tagging (Not applicable in Shiny, typically done in text analysis packages)
# Named Entity Recognition (Not applicable in Shiny, typically done in text analysis packages)
# Scrape data from a website (Not applicable in Shiny, typically done outside of Shiny)
```

```
[1] "Tokenization (Word):"
[1] "this"         "is"           "a"            "sample"       "text"
[6] "for"          "tokenization" "in"           "shiny."
[1] "\nFrequency Distribution:"
tokens
           a          for           in           is       sample       shiny.
           1            1            1            1            1            1
        text         this tokenization
           1            1            1
[1] "\nRemove stopwords & punctuations:"
[1] "sample"       "text"         "tokenization" "shiny."
[1] "\nLexicon Normalization:"
[1] "this is a sample text for tokenization in shiny."
```

```r
# Install and load tm package
if (!require("tm")) install.packages("tm")
library(tm)

# Create a basic corpus
corpus <- Corpus(VectorSource(c("This is a sample document.", "Another document for testing.")))

# Perform basic text preprocessing
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, removeWords, stopwords("english"))

# Display the preprocessed text
inspect(corpus)
```

```
Loading required package: tm

Warning message in library(package, lib.loc = lib.loc, character.only = TRUE, logical.return = TRUE, :
“there is no package called ‘tm’”
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

also installing the dependencies ‘NLP’, ‘Rcpp’, ‘slam’, ‘BH’


Loading required package: NLP

Warning message in tm_map.SimpleCorpus(corpus, content_transformer(tolower)):
“transformation drops documents”
Warning message in tm_map.SimpleCorpus(corpus, removePunctuation):
“transformation drops documents”
Warning message in tm_map.SimpleCorpus(corpus, removeNumbers):
“transformation drops documents”
Warning message in tm_map.SimpleCorpus(corpus, removeWords, stopwords("english")):
“transformation drops documents”
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 2

[1]    sample document        another document  testing
```

```r
# Install and load quanteda package
if (!require("quanteda")) install.packages("quanteda")
library(quanteda)

# Create a simple corpus
texts <- c("This is a simple document.", "Another document for testing.")
corpus <- corpus(texts)

# Tokenization (Sentence & Word)
tokens_sent <- tokens(corpus, what = "sentence")
tokens_word <- tokens(corpus, remove_punct = TRUE, remove_numbers = TRUE)

# Frequency Distribution
word_freq <- table(unlist(tokens_word))

# Remove stopwords & punctuations
tokens_filtered <- tokens_remove(tokens_word, stopwords("en"))

# Lexicon Normalization (Stemming)
tokens_stem <- tokens_wordstem(tokens_filtered)

# Display results
print("Tokenization (Sentence):")
print(tokens_sent)
print("\nTokenization (Word):")
print(tokens_word)
print("\nFrequency Distribution:")
print(word_freq)
print("\nRemove stopwords & punctuations:")
print(tokens_filtered)
print("\nLexicon Normalization (Stemming):")
print(tokens_stem)
```

```
[1] "Tokenization (Sentence):"
Tokens consisting of 2 documents.
```

```
text1 :
[1] "This is a simple document."

text2 :
[1] "Another document for testing."

[1] "\nTokenization (Word):"
Tokens consisting of 2 documents.
text1 :
[1] "This"     "is"       "a"        "simple"   "document"

text2 :
[1] "Another"  "document" "for"      "testing"

[1] "\nFrequency Distribution:"

       a  Another document      for       is   simple  testing     This
       1        1        2        1        1        1        1        1
[1] "\nRemove stopwords & punctuations:"
Tokens consisting of 2 documents.
text1 :
```