III B. Tech II SEM

Regulation: R20

Laboratory Manual

For the course of

NATURAL LANGUAGE PROCESSING

Branch: INFORMATION TECHNOLOGY

VIGNAN'S LARA
INSTITUTE OF TECHNOLOGY & SCIENCE
Approved by AICTE New Delhi & Affiliated to JNTUK Kakinada
Accredited by NAAC 'A+' and NBA | ISO 9001 : 2015
Vadlamudi - 522 213, Guntur District

DEPARTMENT OF INFORMATION TECHNOLOGY

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA
KAKINADA – 533 003, Andhra Pradesh, India

DEPARTMENT OF INFORMATION TECHNOLOGY

III Year – II Semester		L	T	P	C
		0	0	4	2
DATA SCIENCE: NATURAL LANGUAGE PROCESSING (Skill Oriented Course)					

Course Outcomes:

Upon successful completion of the course, the student will be able to:

- Explore natural language processing (NLP) libraries in Python
- Learn various techniques for implementing NLP including parsing & text processing
- Understand how to use NLP for text feature engineering

Python Libraries: nltk, re,word2vec

List of Experiments :

1. Demonstrate Noise Removal for any textual data and remove regular expression pattern such as hash tag from textual data.
2. Perform lemmatization and stemming using python library nltk.
3. Demonstrate object standardization such as replace social media slangs from a text.
4. Perform part of speech tagging on any textual data.
5. Implement topic modeling using Latent Dirichlet Allocation (LDA) in python.
6. Demonstrate Term Frequency – Inverse Document Frequency (TF – IDF) using python
7. Demonstrate word embeddings using word2vec.
8. Implement Text classification using naïve bayes classifier and text blob library.
9. Apply support vector machine for text classification.
10. Convert text to vectors (using term frequency) and apply cosine similarity to provide closeness among two text.
11. Case study 1: Identify the sentiment of tweets

    In this problem, you are provided with tweet data to predict sentiment on electronic products of netizens.
12. Case study 2: Detect hate speech in tweets.

    The objective of this task is to detect hate speech in tweets. For the sake of simplicity, we say a tweet contains hate speech if it has a racist or sexist sentiment associated with it. So, the task is to classify racist or sexist tweets from other tweets.

Web References:

1. https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/
2. https://datahack.analyticsvidhya.com/contest/linguipedia-codefest-natural-language-processing-1/?utm_source=ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python&utm_medium=blog
3. https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/

Experiment 1: Demonstrate Noise Removal for any textual data and remove regular expression pattern such as hash tag from textual data.

Program:

```
import re

text_data = "This is a sample tweet 92 with     a #hashtag and some noise! Visit https://example.com for more info"

# To Remove URLs

text_data = re.sub('http\S+', '', text_data)

print("Cleaned text data:", text_data)

# To Remove hashtags

text_data = re.sub('#\w+', '', text_data)

print("Cleaned text data:", text_data)

# To Remove special characters and numbers

text_data = re.sub('[^A-Za-z\s]', '', text_data)

print("Cleaned text data:", text_data)

# To Remove extra spaces

text_data = re.sub('\s+', ' ', text_data).strip() #strip useful to remove leading and ending spaces

print("Cleaned text data:", text_data)
```

Output:

```
Cleaned text data: This is a sample tweet 92 with     a #hashtag and some noise! Visit  for more info

Cleaned text data: This is a sample tweet 92 with     a  and some noise! Visit  for more info

Cleaned text data: This is a sample tweet  with     a  and some noise Visit  for more info

Cleaned text data: This is a sample tweet with a and some noise Visit for more info
```

Experiment 2: Perform lemmatization and stemming using python library nltk.

Program:

```
from nltk.stem import PorterStemmer, WordNetLemmatizer

text1 = "The striped bats are hanging on their feet for best"

text2= "Building and maintaining a font collection on the computer you use for design work is an important part of life as a designer."

text3= "running runs runner"
```

```
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()
print([stemmer.stem(word) for word in text1.split()])
print([lemmatizer.lemmatize(word, pos='v') for word in text1.split()])
print([stemmer.stem(word) for word in text2.split()])
print([lemmatizer.lemmatize(word, pos='v') for word in text2.split()])
print([stemmer.stem(word) for word in text3.split()])
print([lemmatizer.lemmatize(word, pos='v') for word in text3.split()])
```

Output:

```
['the', 'stripe', 'bat', 'are', 'hang', 'on', 'their', 'feet', 'for', 'best']
['The', 'strip', 'bat', 'be', 'hang', 'on', 'their', 'feet', 'for', 'best']
['build', 'and', 'maintain', 'a', 'font', 'collect', 'on', 'the', 'comput', 'you', 'use', 'for', 'design', 'work', 'is', 'an', 'import', 'part', 'of', 'life', 'as', 'a', 'designer.']
['Building', 'and', 'maintain', 'a', 'font', 'collection', 'on', 'the', 'computer', 'you', 'use', 'for', 'design', 'work', 'be', 'an', 'important', 'part', 'of', 'life', 'as', 'a', 'designer.']
['run', 'run', 'runner']
['run', 'run', 'runner']
```

Experiment 3: Demonstrate object standardization such as replace social media slangs from a text.

Program:

```
import nltk
from nltk.tokenize import word_tokenize
text = "OMG, this is lit! IDK what to say, LOL."
slang_dict = {
   "OMG": "Oh my God",
   "lit": "amazing",
   "IDK": "I don't know",
   "LOL": "laugh out loud"
}
words = word_tokenize(text)
standardized_text = ' '.join([slang_dict.get(word, word) for word in words])
```

```
print("Original Text:", text)

print("Standardized Text:", standardized_text)
```

Output:

Original Text: OMG, this is lit! IDK what to say, LOL.

Standardized Text: Oh my God , this is amazing ! I don't know what to say , laugh out loud .

Experiment 4: Perform part of speech tagging on any textual data.

Program:

```
import nltk

from nltk.tokenize import word_tokenize

from nltk.tag import pos_tag

text1= "The quick brown fox jumps over the lazy dog."

text2= "The striped bats are hanging on their feet for best."

words = word_tokenize(text)

pos_tags = pos_tag(words)

print("Original Text:", text1)

print("POS Tags:", pos_tags)

print("Original Text:", text2)

print("POS Tags:", pos_tags)
```

Output:

Original Text: The quick brown fox jumps over the lazy dog.

POS Tags: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN'), ('.', '.')]

Original Text: The striped bats are hanging on their feet for best.

POS Tags: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN'), ('.', '.')]

Experiment 5: Implement topic modeling using Latent Dirichlet Allocation (LDA) in python.

Program:

```
import gensim

import gensim.corpora as corpora

from gensim.models.ldamodel import LdaModel
```

```
from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

import nltk

documents = [

    "The quick brown fox jumps over the lazy dog.",

    "Never jump over the lazy dog quickly.",

    "A fast brown fox leaps over a sleepy dog."

]

stop_words = set(stopwords.words('english'))

texts = [[word for word in word_tokenize(doc.lower()) if word.isalnum() and word not in stop_words] for doc in documents]

id2word = corpora.Dictionary(texts)

corpus = [id2word.doc2bow(text) for text in texts]

lda_model = LdaModel(corpus=corpus, id2word=id2word, num_topics=2, random_state=42, update_every=1, chunksize=10, passes=10, alpha='auto', per_word_topics=True)

topics = lda_model.print_topics()

for topic in topics:

    print(topic)
```

Output:

```
(0, '0.148*"dog" + 0.147*"lazy" + 0.089*"fox" + 0.088*"brown" + 0.088*"never" + 0.088*"jump" + 0.088*"quickly" + 0.088*"quick" + 0.088*"jumps" + 0.030*"leaps"')

(1, '0.125*"fast" + 0.125*"sleepy" + 0.125*"leaps" + 0.125*"brown" + 0.125*"fox" + 0.123*"dog" + 0.042*"lazy" + 0.042*"jumps" + 0.042*"quick" + 0.042*"quickly"')
```

Experiment 6: Demonstrate Term Frequency – Inverse Document Frequency (TF – IDF) using python.

Program:

```
from sklearn.feature_extraction.text import TfidfVectorizer

documents = [

    "The quick brown fox jumps over the lazy dog.",

    "Never jump over the lazy dog quickly.",

    "A fast brown fox leaps over a sleepy dog."

]
```

```
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(documents)
feature_names = vectorizer.get_feature_names_out()
dense_tfidf = tfidf_matrix.todense()
print("TF-IDF Matrix:")
print(dense_tfidf)
print("\nFeature Names:")
print(feature_names)
```

Output:

```
TF-IDF Matrix:
[[0.29280349 0.22738803 0.         0.29280349 0.         0.38500141
  0.29280349 0.         0.         0.22738803 0.38500141 0.
  0.         0.58560699]
 [0.         0.26806191 0.         0.         0.45386827 0.
  0.34517852 0.         0.45386827 0.26806191 0.         0.45386827
  0.         0.34517852]
 [0.34517852 0.26806191 0.45386827 0.34517852 0.         0.
  0.         0.45386827 0.         0.26806191 0.         0.
  0.45386827 0.        ]]
Feature Names:
['brown' 'dog' 'fast' 'fox' 'jump' 'jumps' 'lazy' 'leaps' 'never' 'over'
 'quick' 'quickly' 'sleepy' 'the']
```

Experiment 7: Demonstrate word embeddings using word2vec.

Program:

```
from gensim.models import Word2Vec
sentences = [
    ['I', 'love', 'machine', 'learning'],
    ['Natural', 'language', 'processing', 'is', 'fun'],
    ['Word2Vec', 'is', 'a', 'great', 'tool'],
    ['I', 'enjoy', 'learning', 'new', 'things']
```

]

```
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)
word_vector = model.wv['learning']
print(word_vector)
```

Output:

```
[-8.6196875e-03  3.6657380e-03  5.1898835e-03  5.7419385e-03
  7.4669183e-03 -6.1676754e-03  1.1056137e-03  6.0472824e-03
 -2.8400505e-03 -6.1735227e-03 -4.1022300e-04 -8.3689485e-03
 -5.6000124e-03  7.1045388e-03  3.3525396e-03  7.2256695e-03
  6.8002474e-03  7.5307419e-03 -3.7891543e-03 -5.6180597e-04
  2.3483764e-03 -4.5190323e-03  8.3887316e-03 -9.8581640e-03
  6.7646410e-03  2.9144168e-03 -4.9328315e-03  4.3981876e-03
 -1.7395747e-03  6.7113843e-03  9.9648498e-03 -4.3624435e-03
 -5.9933780e-04 -5.6956373e-03  3.8508223e-03  2.7866268e-03
  6.8910765e-03  6.1010956e-03  9.5384968e-03  9.2734173e-03
  7.8980681e-03 -6.9895042e-03 -9.1558648e-03 -3.5575271e-04
 -3.0998408e-03  7.8943167e-03  5.9385742e-03 -1.5456629e-03
  1.5109634e-03  1.7900408e-03  7.8175711e-03 -9.5101865e-03
 -2.0553112e-04  3.4691966e-03 -9.3897223e-04  8.3817719e-03
  9.0107834e-03  6.5365066e-03 -7.1162102e-04  7.7104042e-03
 -8.5343346e-03  3.2071066e-03 -4.6379971e-03 -5.0889552e-03
  3.5896183e-03  5.3703394e-03  7.7695143e-03 -5.7665063e-03
  7.4333609e-03  6.6254963e-03 -3.7098003e-03 -8.7456414e-03
  5.4374672e-03  6.5097557e-03 -7.8755023e-04 -6.7098560e-03
 -7.0859254e-03 -2.4970602e-03  5.1432536e-03 -3.6652375e-03
 -9.3700597e-03  3.8267397e-03  4.8844791e-03 -6.4285635e-03
  1.2085581e-03 -2.0748770e-03  2.4403334e-05 -9.8835090e-03
  2.6920044e-03 -4.7501065e-03  1.0876465e-03 -1.5762246e-03
  2.1966731e-03 -7.8815762e-03 -2.7171839e-03  2.6631986e-03
  5.3466819e-03 -2.3915148e-03 -9.5100943e-03  4.5058788e-03]
```

Experiment 8: Implement Text classification using naïve bayes classifier and text blob library.

Program:

```
from textblob import TextBlob
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score, classification_report
data = [
    ('I love this movie', 'positive'),
    ('This film was terrible', 'negative'),
    ('I enjoyed the plot', 'positive'),
    ('The acting was bad', 'negative'),
    ('What a fantastic experience', 'positive'),
    ('I hated the ending', 'negative')
]

texts, labels = zip(*data)
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)
classifier = MultinomialNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

Output:

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

    negative       1.00      1.00      1.00         1
    positive       1.00      1.00      1.00         1
```

accuracy			1.00	2
macro avg	1.00	1.00	1.00	2
weighted avg	1.00	1.00	1.00	2

Experiment 9: Apply support vector machine for text classification.

Program:

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
data = [
    ('I love this movie', 'positive'),
    ('This film was terrible', 'negative'),
    ('I enjoyed the plot', 'positive'),
    ('The acting was bad', 'negative'),
    ('What a fantastic experience', 'positive'),
    ('I hated the ending', 'negative')
]
texts, labels = zip(*data)
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)
classifier = SVC(kernel='linear')
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

Output:

```
Accuracy: 1.0
Classification Report:
                    precision    recall  f1-score   support
```

negative	1.00	1.00	1.00	1
positive	1.00	1.00	1.00	1
accuracy			1.00	2
macro avg	1.00	1.00	1.00	2
weighted avg	1.00	1.00	1.00	2

Experiment 10: Convert text to vectors (using term frequency) and apply cosine similarity to provide closeness among two text.

Program:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
text1 = "I love machine learning"
text2 = "Machine learning is fascinating"
vectorizer = CountVectorizer()
vectors = vectorizer.fit_transform([text1, text2])
cosine_sim = cosine_similarity(vectors)
print(cosine_sim)
```

Output:

```
[[1.         0.57735027]
 [0.57735027 1.        ]]
```

Experiment 11: Case study 1: Identify the sentiment of tweets

In this problem, you are provided with tweet data to predict sentiment on electronic products of netizens.

Program:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
import nltk
from nltk.corpus import stopwords
```

```
from nltk.tokenize import word_tokenize
import string
df = pd.read_csv(r"D:\NLP\tweets.csv")
nltk.download('stopwords')
nltk.download('punkt')
def preprocess_text(text):
    if isinstance(text, float):
        text = ""
    tokens = word_tokenize(text)
    tokens = [word.lower() for word in tokens if word.isalpha()]
    tokens = [word for word in tokens if word not in stopwords.words('english')]
    return ' '.join(tokens)
df['processed_text'] = df['tweet'].apply(preprocess_text)
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['processed_text'])
y = df['sentiment']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
classifier = MultinomialNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

Output:

Accuracy: 0.6483536474440604

Classification Report:

	Precision	recall	f1-score	support
negative	0.69	0.59	0.63	1562
neutral	0.60	0.65	0.62	2230
positive	0.69	0.70	0.70	1705
accuracy			0.65	5497
macro avg	0.66	0.65	0.65	5497

weighted avg	0.65	0.65	0.65	5497

Experiment 12: Case study 2: Detect hate speech in tweets.

The objective of this task is to detect hate speech in tweets. For the sake of simplicity, we say a tweet contains hate speech if it has a racist or sexist sentiment associated with it. So, the task is to classify racist or sexist tweets from other tweets.

Program:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string
df = pd.read_csv(r"D:\NLP\tweets.csv")
nltk.download('stopwords')
nltk.download('punkt')
def preprocess_text(text):
    if isinstance(text, float):
        text = ""
    tokens = word_tokenize(text)
    tokens = [word.lower() for word in tokens if word.isalpha()]
    tokens = [word for word in tokens if word not in stopwords.words('english')]
    return ' '.join(tokens)
df['processed_text'] = df['tweet'].apply(preprocess_text)
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['processed_text'])
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
classifier = MultinomialNB()
```

```
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))

print("Classification Report:\n", classification_report(y_test, y_pred))
```

Output:

Accuracy: 0.6483536474440604

Classification Report:

	Precision	recall	f1-score	support
negative	0.69	0.59	0.63	1562
neutral	0.60	0.65	0.62	2230
positive	0.69	0.70	0.70	1705
accuracy			0.65	5497
macro avg	0.66	0.65	0.65	5497
weighted avg	0.65	0.65	0.65	5497