LDRP Institute of technology and Research

Computer Engineering Department

Subject Name: Natural Language Processing(CT703C-N)

Practical List

Sr. No	Practical	Date	Pg. No.	Sign
1	Basic Text Processing operation on text document.	13/07/2022		
2	Implement N-gram Language model.	20/07/2022		
3	Write a program to extract features from text.	27/07/2022		
4	Implement word embedding using Word2Vec/Glove/ FasText	03/08/2022		
5	Implement LSA and Topic model.	10/08/2022		
6	Implementation text classification using Naive Bayes, SVM.	17/08/2022		
7	Implementation of K-means Clustering algorithm on text.	24/08/2022		
8	Implement PoS Tagging on text.	07/09/2022		
9	Implement text processing with neural network.	14/09/2022		
10	Implement text processing with LSTM.	21/09/2022		
11	Implement HMM/CRF on sequence tagging task.	28/09/2022		

PRACTICAL - 1

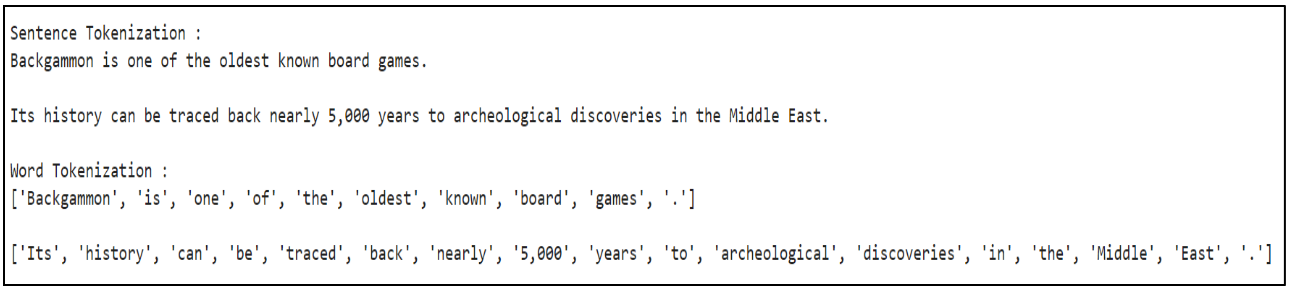
Aim: Basic Text Processing operation on text document.

Code:

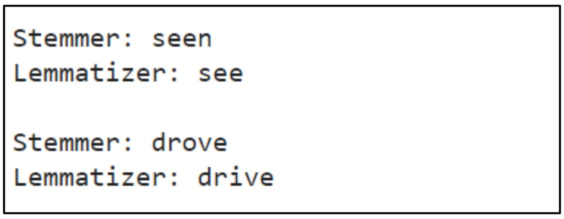
```
import nltk

nltk.download('punkt')

text = "Backgammon is one of the oldest known board games. Its history can be traced back nearly 5,000 years to archeological discoveries in the Middle East. It is a two player game where each player has fifteen checkers which move between twenty-four points according to the roll of two dice."


# Sentence tokenization

sentences = nltk.sent_tokenize(text)

print("Sentence Tokenization :")

for sentence in sentences:

        print(sentence)

        print()


# Word tokenization

print("Word Tokenization :")

for sentence in sentences:

        words = nltk.word_tokenize(sentence)

        print(words)

        print()


# Stemming and Lemmitization

from nltk.stem import PorterStemmer, WordNetLemmatizer

from nltk.corpus import wordnet

nltk.download('wordnet')

nltk.download('omw-1.4')

def compare_stemmer_and_lemmatizer(stemmer, lemmatizer, word, pos):
        print("Stemmer:", stemmer.stem(word))
        print("Lemmatizer:", lemmatizer.lemmatize(word, pos))
        print()
lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()
compare_stemmer_and_lemmatizer(stemmer, lemmatizer, word = "seen", pos = wordnet.VERB)
compare_stemmer_and_lemmatizer(stemmer, lemmatizer, word = "drove", pos = wordnet.VERB)

# Stopword Removal
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words("english"))
sentence = "Backgammon is one of the oldest known board games."
print("Stop word removal")
words = nltk.word_tokenize(sentence)
without_stop_words = [word for word in words if not word in stop_words]
print(without_stop_words)
# Regex
import re
sentence = "The development of snowboarding was inspired by skateboarding, sledding, surfing and skiing."
pattern = r"[^\w]"
print("Regex")
print(re.sub(pattern, " ", sentence))

# Bag of Word
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
documents = ["I like this movie, it's funny.", 'I hate this movie.', 'This was awesome! I like it.', 'Nice one. I love it.']
```

```
count_vectorizer = CountVectorizer()

bag_of_words = count_vectorizer.fit_transform(documents)

feature_name = count_vectorizer.get_feature_names()

print("Bag-of-Words")

pd.DataFrame(bag_of_words.toarray(), columns = feature_name)


# TF-IDF

from sklearn.feature_extraction.text import TfidfVectorizer

import pandas as pd

tfidf_vectorizer = TfidfVectorizer()

values = tfidf_vectorizer.fit_transform(documents)

feature_names = tfidf_vectorizer.get_feature_names()

print("TF-IDF")

pd.DataFrame(values.toarray(), columns = feature_names)
```

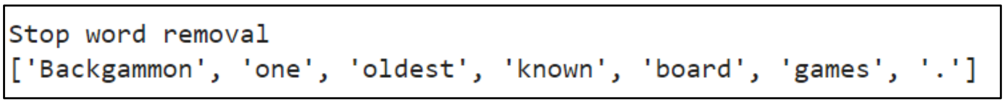
Output:



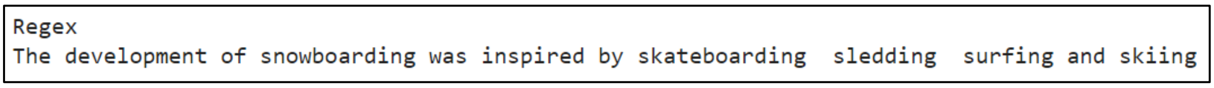
```
Sentence Tokenization :
Backgammon is one of the oldest known board games.

Its history can be traced back nearly 5,000 years to archeological discoveries in the Middle East.

Word Tokenization :
['Backgammon', 'is', 'one', 'of', 'the', 'oldest', 'known', 'board', 'games', '.']

['Its', 'history', 'can', 'be', 'traced', 'back', 'nearly', '5,000', 'years', 'to', 'archeological', 'discoveries', 'in', 'the', 'Middle', 'East', '.']
```



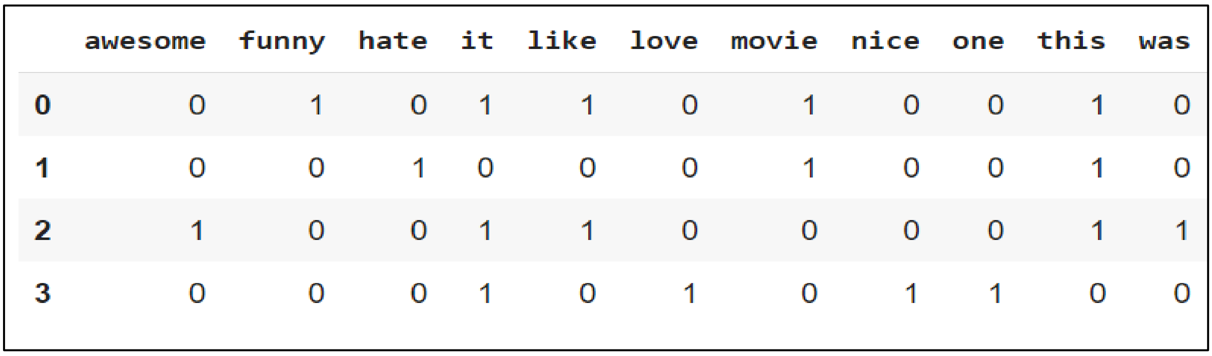
```
Stemmer:  seen
Lemmatizer:  see

Stemmer:  drove
Lemmatizer:  drive
```



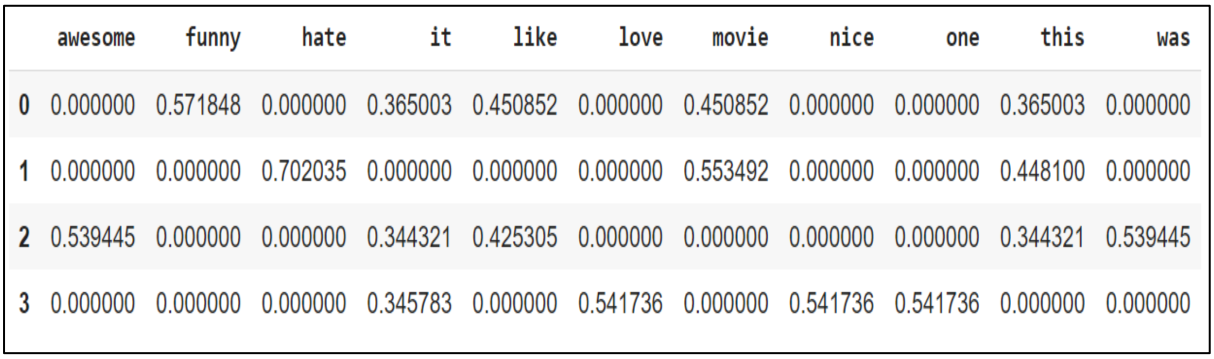
```
Stop word removal
['Backgammon', 'one', 'oldest', 'known', 'board', 'games', '.']
```



```
Regex
The development of snowboarding was inspired by skateboarding  sledding  surfing and skiing
```



	awesome	funny	hate	it	like	love	movie	nice	one	this	was
0	0	1	0	1	1	0	1	0	0	1	0
1	0	0	1	0	0	0	1	0	0	1	0
2	1	0	0	1	1	0	0	0	0	1	1
3	0	0	0	1	0	1	0	1	1	0	0



	awesome	funny	hate	it	like	love	movie	nice	one	this	was
0	0.000000	0.571848	0.000000	0.365003	0.450852	0.000000	0.450852	0.000000	0.000000	0.365003	0.000000
1	0.000000	0.000000	0.702035	0.000000	0.000000	0.000000	0.553492	0.000000	0.000000	0.448100	0.000000
2	0.539445	0.000000	0.000000	0.344321	0.425305	0.000000	0.000000	0.000000	0.000000	0.344321	0.539445
3	0.000000	0.000000	0.000000	0.345783	0.000000	0.541736	0.000000	0.541736	0.541736	0.000000	0.000000

PRACTICAL - 2

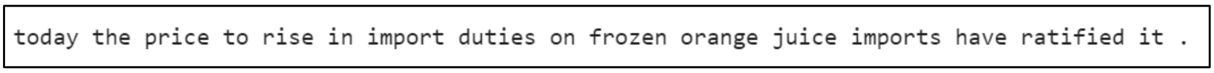
Aim: Implement N-gram Language model.

Code:

```
from nltk.corpus import reuters
from nltk import bigrams, trigrams
from collections import Counter, defaultdict

model = defaultdict(lambda: defaultdict(lambda: 0))

for sentence in reuters.sents():
        for w1, w2, w3 in trigrams(sentence, pad_right=True, pad_left=True):
                model[(w1, w2)][w3] += 1

for w1_w2 in model:
        total_count = float(sum(model[w1_w2].values()))
        for w3 in model[w1_w2]:
                model[w1_w2][w3] /= total_count

import random

text = ["today", "the"]
sentence_finished = False

while not sentence_finished:
        r = random.random()
        accumulator = .0

 for word in model[tuple(text[-2:])].keys():
        accumulator += model[tuple(text[-2:])][word]
```

```
    if accumulator >= r:

        text.append(word)

        break


 if text[-2:] == [None, None]:

    sentence_finished = True


print (' '.join([t for t in text if t]))
```

Output:



```
today the price to rise in import duties on frozen orange juice imports have ratified it .
```

PRACTICAL - 3

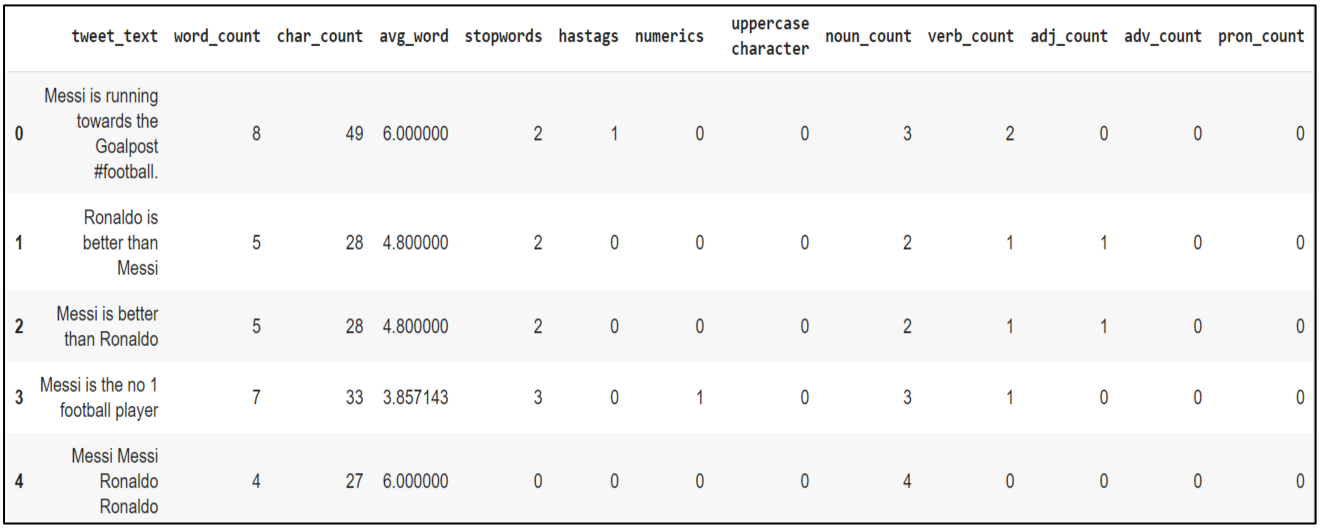
Aim: Write a program to extract features from text.

Code:

```
A = [' Messi is running towards the Goalpost #football.',

'Ronaldo is better than Messi',

'Messi is better than Ronaldo',

'Messi is the no 1 football player',

'Messi Messi Ronaldo Ronaldo',

'mbappe,mbappe,mbappe,mbappe,mbappe,mbappe,mbappe,mbappe,mbappe,']


import pandas as pd

data = pd.DataFrame({'tweet_text':A})


data['word_count'] = data['tweet_text'].apply(lambda x: len(str(x).split(" ")))

data['char_count'] = data['tweet_text'].str.len() ## this also includes spaces


def avg_word(sentence):

        words = sentence.split()

        return (sum(len(word) for word in words)/len(words))

data['avg_word'] = data['tweet_text'].apply(lambda x: avg_word(x))


import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

stop = stopwords.words('english')

data['stopwords'] = data['tweet_text'].apply(lambda x: len([x for x in x.split() if x in stop]))

data['hastags'] = data['tweet_text'].apply(lambda x: len([x for x in x.split() if x.startswith('#')]))

data['numerics'] = data['tweet_text'].apply(lambda x: len([x for x in x.split() if x.isdigit()]))

data['uppercase character'] = data['tweet_text'].apply(lambda x: len([x for x in x.split() if
x.isupper()]))
```

```
pos_family = {
    'noun’: ['NN','NNS','NNP','NNPS'],
    'pron’: ['PRP','PRP$','WP','WP$'],
    'verb’: ['VB','VBD','VBG','VBN','VBP','VBZ'],
    'adj’: ['JJ','JJR','JJS'],
    'adv’: ['RB','RBR','RBS','WRB']
}
from textblob import TextBlob, Word, Blobber
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

def check_pos_tag(x, flag):
    cnt = 0
    try:
        wiki = TextBlob(x)
        for tup in wiki.tags:
            ppo = list(tup)[1]
            if ppo in pos_family[flag]:
                cnt += 1
    except:
        pass
    return cnt

data['noun_count'] = data['tweet_text'].apply(lambda x: check_pos_tag(x, 'noun'))
data['verb_count'] = data['tweet_text'].apply(lambda x: check_pos_tag(x, 'verb'))
data['adj_count'] = data['tweet_text'].apply(lambda x: check_pos_tag(x, 'adj'))
data['adv_count'] = data['tweet_text'].apply(lambda x: check_pos_tag(x, 'adv'))
data['pron_count'] = data['tweet_text'].apply(lambda x: check_pos_tag(x, 'pron'))

data.head()
```

Output:



	tweet_text	word_count	char_count	avg_word	stopwords	hastags	numerics	uppercase character	noun_count	verb_count	adj_count	adv_count	pron_count
0	Messi is running towards the Goalpost #football.	8	49	6.000000	2	1	0	0	3	2	0	0	0
1	Ronaldo is better than Messi	5	28	4.800000	2	0	0	0	2	1	1	0	0
2	Messi is better than Ronaldo	5	28	4.800000	2	0	0	0	2	1	1	0	0
3	Messi is the no 1 football player	7	33	3.857143	3	0	1	0	3	1	0	0	0
4	Messi Messi Ronaldo Ronaldo	4	27	6.000000	0	0	0	0	4	0	0	0	0

PRACTICAL - 4

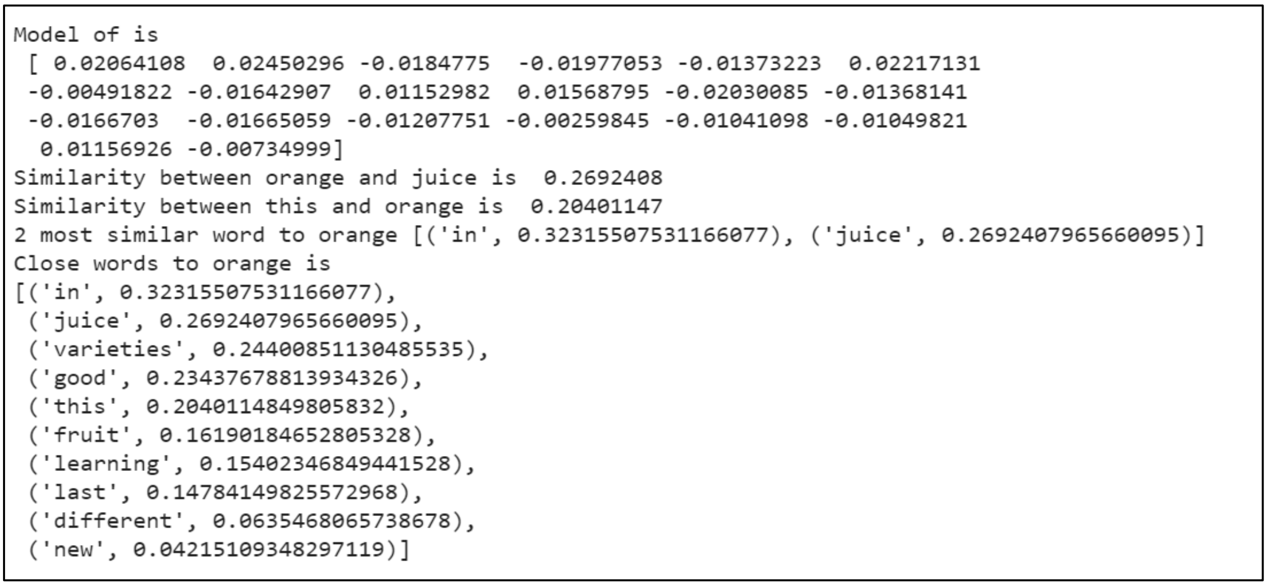
Aim: Implement word embedding using Word2Vec/Glove/fastText.

Code:

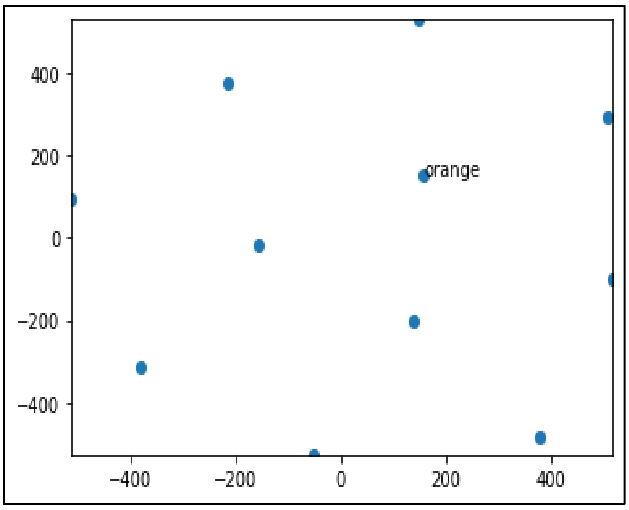
```
from gensim.models import Word2Vec

sentences = [['this', 'is', 'the', 'good', 'machine', 'learning', 'book'],

        ['this', 'is',  'another', 'machine', 'learning', 'book'],

        ['one', 'more', 'new', 'book'],

        ['this', 'is', 'about', 'machine', 'learning', 'post'],

        ['orange', 'juice', 'is', 'the', 'liquid', 'extract', 'of', 'fruit'],

        ['orange', 'juice', 'comes', 'in', 'several', 'different', 'varieties'],

        ['this', 'is', 'the', 'last', 'machine', 'learning', 'book'],

        ['orange', 'juice', 'comes', 'in', 'several', 'different', 'packages'],

        ['orange', 'juice', 'is', 'liquid', 'extract', 'from', 'fruit', 'on', 'orange', 'tree']]


from gensim.models import FastText

model = Word2Vec(sentences, size=20, min_count=1, window=2,sg=0)


is_model = model['is']

print("Model of is \n",is_model)


orange_juice = model.similarity('orange','juice')

print("Similarity between orange and juice is ",orange_juice)


this_orange = model.similarity('this','orange')

print("Similarity between this and orange is ",this_orange)


most_similar_orange = model.most_similar('orange')[:2]

print("2 most similar word to orange",most_similar_orange)
```

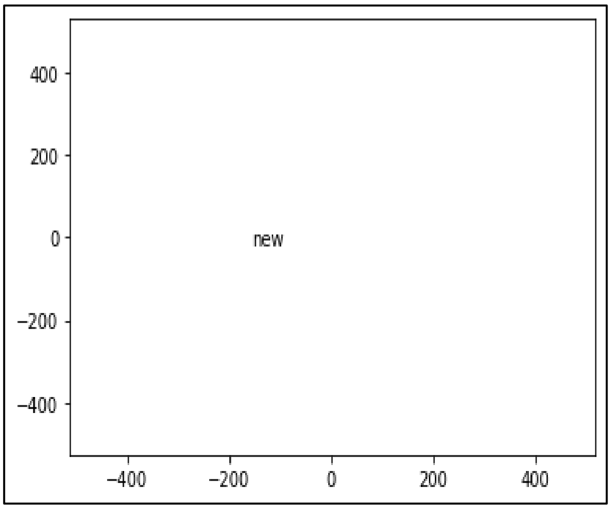
```
close_words = model.similar_by_word('orange')
print("Close words to orange is \n",close_words)
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

def display_closestwords_tsnescatterplot(model, word, size):

    arr = np.empty((0,size), dtype='f')
    word_labels = [word]

    close_words = model.similar_by_word(word)
    arr = np.append(arr, np.array([model[word]]), axis=0)
    for wrd_score in close_words:
        wrd_vector = model[wrd_score[0]]
        word_labels.append(wrd_score[0])
        arr = np.append(arr, np.array([wrd_vector]), axis=0)

    tsne = TSNE(n_components=2, random_state=0)
    np.set_printoptions(suppress=True)
    Y = tsne.fit_transform(arr)
    x_coords = Y[:, 0]
    y_coords = Y[:, 1]
    plt.scatter(x_coords, y_coords)
    for label, x, y in zip(word_labels, x_coords, y_coords):
        plt.annotate(label, xy=(x, y), xytext=(0, 0), textcoords='offset points')
        plt.xlim(x_coords.min()+0.00005, x_coords.max()+0.00005)
        plt.ylim(y_coords.min()+0.00005, y_coords.max()+0.00005)
        plt.show()

display_closestwords_tsnescatterplot(model, 'orange', 4)
```

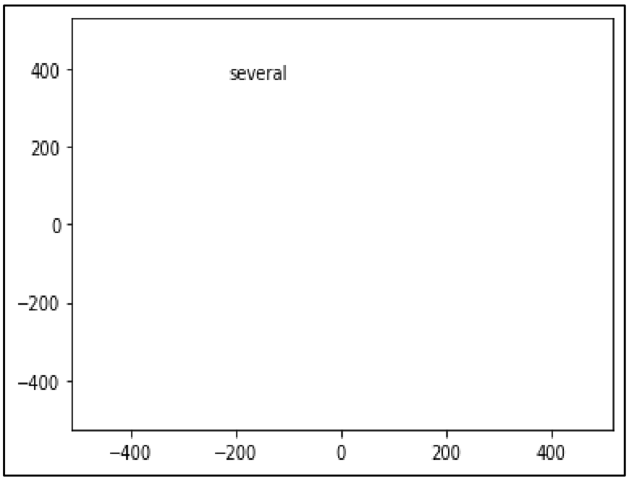
Output:

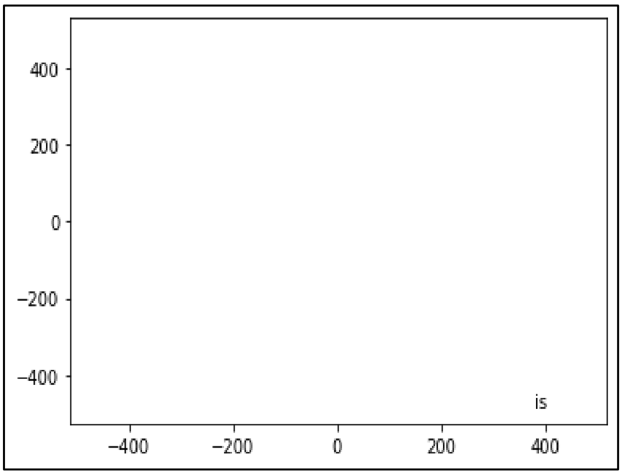


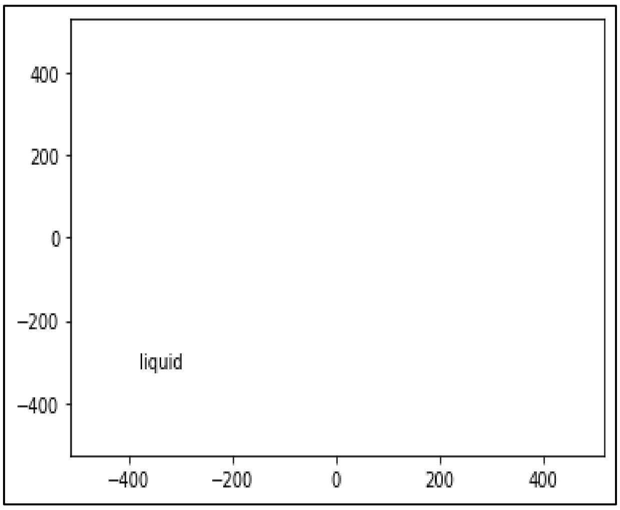
```
Model of is
 [ 0.02064108  0.02450296 -0.0184775  -0.01977053 -0.01373223  0.02217131
 -0.00491822 -0.01642907  0.01152982  0.01568795 -0.02030085 -0.01368141
 -0.0166703  -0.01665059 -0.01207751 -0.00259845 -0.01041098 -0.01049821
  0.01156926 -0.00734999]
Similarity between orange and juice is  0.2692408
Similarity between this and orange is  0.20401147
2 most similar word to orange [('in', 0.32315507531166077), ('juice', 0.2692407965660095)]
Close words to orange is
[('in', 0.32315507531166077),
 ('juice', 0.2692407965660095),
 ('varieties', 0.24400851130485535),
 ('good', 0.23437678813934326),
 ('this', 0.2040114849805832),
 ('fruit', 0.16190184652805328),
 ('learning', 0.15402346849441528),
 ('last', 0.14784149825572968),
 ('different', 0.0635468065738678),
 ('new', 0.04215109348297119)]
```

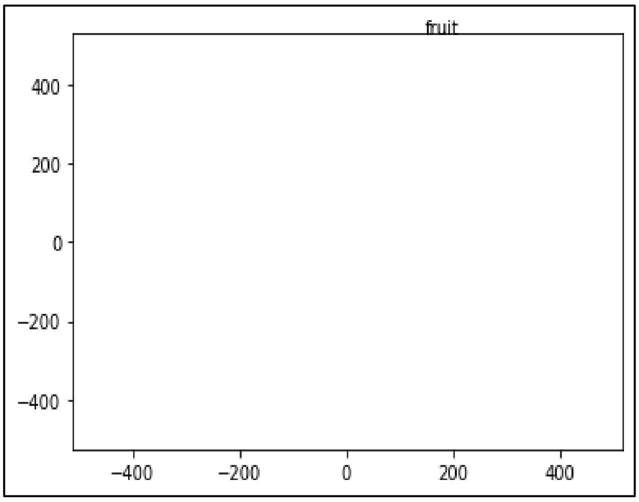


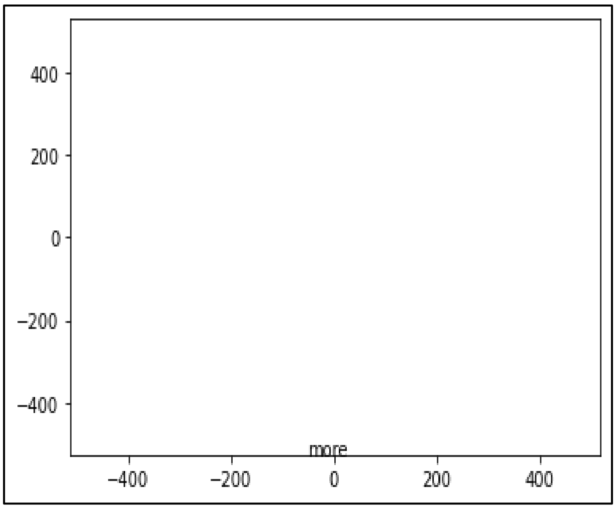


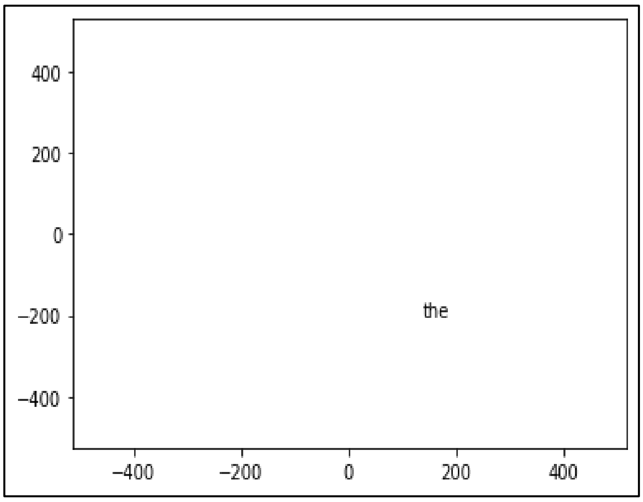

400
200
0
−200
−400
−400
−200
0
200
400
several

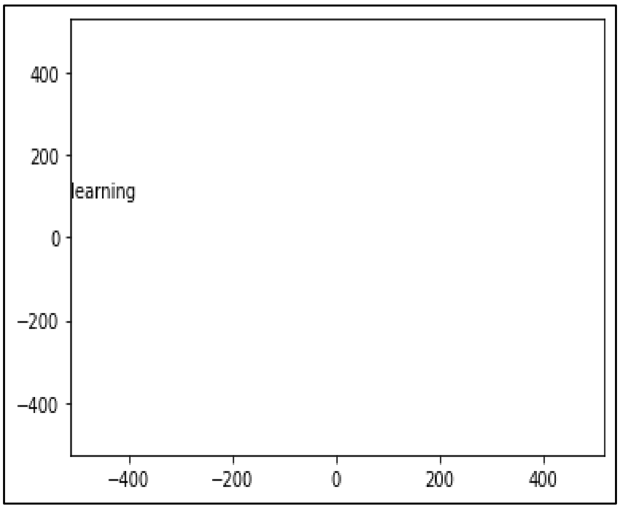

400
200
0
−200
−400
−400
−200
0
200
400
is

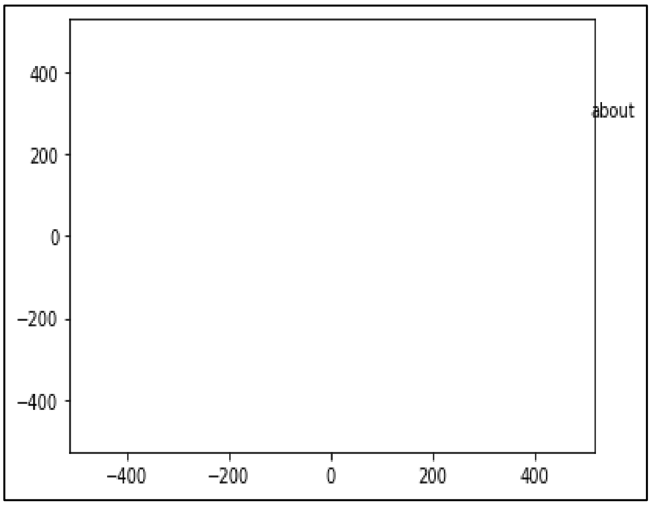

400
200
0
−200
−400
−400
−200
0
200
400
liquid

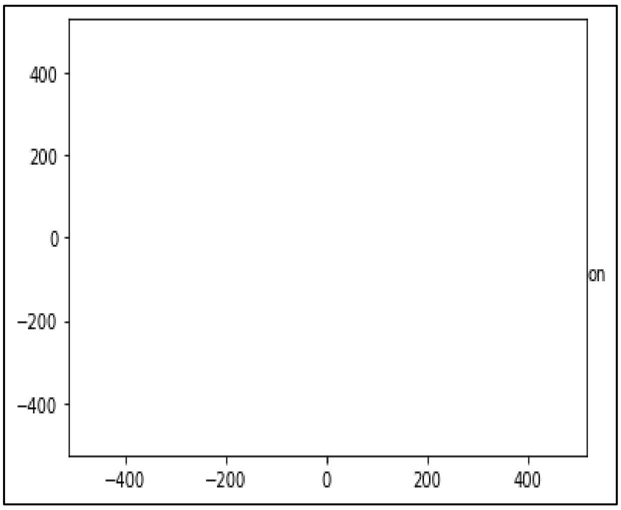

fruit
400
200
0
−200
−400
−400
−200
0
200
400


400
200
0
−200
−400
−400
−200
0
200
400
more


400
200
0
−200
−400
−400
−200
0
200
400
the


learning
400
200
0
−200
−400
−400
−200
0
200
400


about
400
200
0
−200
−400
−400
−200
0
200
400


on
400
200
0
−200
−400
−400
−200
0
200
400

PRACTICAL - 5

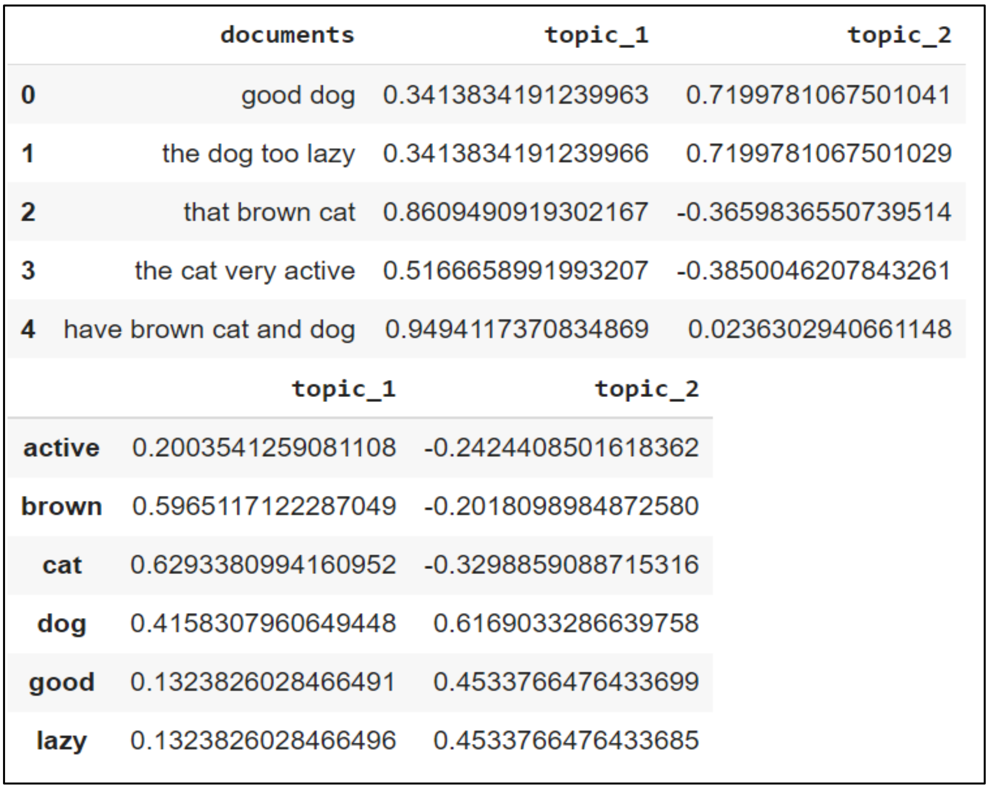
Aim: Implement LSA and Topic model.

Code:

```
import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer

import nltk

from sklearn.decomposition import TruncatedSVD

nltk.download('stopwords')

from nltk.corpus import stopwords


a1 = "He is a good dog."

a2 = "The dog is too lazy."

a3 = "That is a brown cat."

a4 = "The cat is very active."

a5 = "I have brown cat and dog."

df = pd.DataFrame()

df["documents"] = [a1,a2,a3,a4,a5]


df['clean_documents'] = df['documents'].str.replace("[^a-zA-Z#]", " ")

df['clean_documents'] = df['clean_documents'].fillna('').apply(lambda x: ' '.join([w for w in
x.split() if len(w)>2]))

df['clean_documents'] = df['clean_documents'].fillna('').apply(lambda x: x.lower())


tokenized_doc = df['clean_documents'].fillna('').apply(lambda x: x.split())

tokenized_doc = tokenized_doc.apply(lambda x: [itemstop_words =
stopwords.words('english') for item in x if item not in stop_words])

detokenized_doc = []

for i in range(len(df)):

    t = ' '.join(tokenized_doc[i])

    detokenized_doc.append(t)
```

```
df['clean_documents'] = detokenized_doc


vectorizer = TfidfVectorizer(stop_words='english', smooth_idf=True)

X = vectorizer.fit_transform(df['clean_documents'])


svd_model = TruncatedSVD(n_components=2, algorithm='randomized', n_iter=100,
random_state=122)

lsa = svd_model.fit_transform(X)


pd.options.display.float_format = '{:,.16f}'.format

topic_encoded_df = pd.DataFrame(lsa, columns = ["topic_1", "topic_2"])

topic_encoded_df["documents"] = df['clean_documents']

dictionary = vectorizer.get_feature_names()

encoding_matrix = pd.DataFrame(svd_model.components_, index = ["topic_1","topic_2"],
columns = (dictionary)).T

display(topic_encoded_df[["documents", "topic_1", "topic_2"]])

display(encoding_matrix)
```

Output:



	documents	topic_1	topic_2
0	good dog	0.3413834191239963	0.7199781067501041
1	the dog too lazy	0.3413834191239966	0.7199781067501029
2	that brown cat	0.8609490919302167	-0.3659836550739514
3	the cat very active	0.5166658991993207	-0.3850046207843261
4	have brown cat and dog	0.9494117370834869	0.0236302940661148

	topic_1	topic_2
active	0.2003541259081108	-0.2424408501618362
brown	0.5965117122287049	-0.2018098984872580
cat	0.6293380994160952	-0.3298859088715316
dog	0.4158307960649448	0.6169033286639758
good	0.1323826028466491	0.4533766476433699
lazy	0.1323826028466496	0.4533766476433685

PRACTICAL - 6

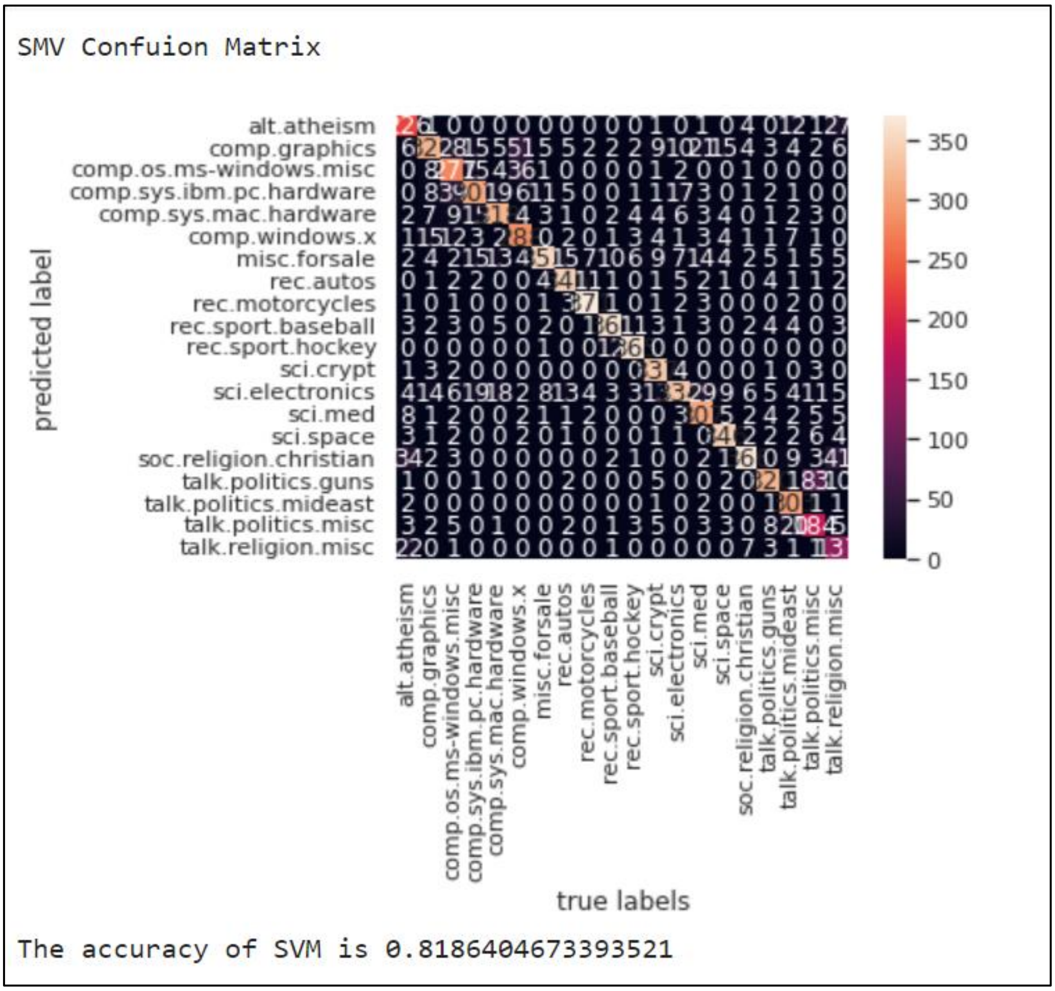
Aim: Implementation text classification using Naïve Bayes, SVM.

Code:

```
import numpy as np, pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import fetch_20newsgroups

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.svm import SVC

from sklearn.pipeline import make_pipeline

from sklearn.metrics import confusion_matrix, accuracy_score

sns.set()


data = fetch_20newsgroups()

text_categories = data.target_names

train_data = fetch_20newsgroups(subset="train", categories=text_categories)

test_data = fetch_20newsgroups(subset="test", categories=text_categories)


model = make_pipeline(TfidfVectorizer(), SVC())

model.fit(train_data.data, train_data.target)

predicted_categories = model.predict(test_data.data)


mat = confusion_matrix(test_data.target, predicted_categories)

sns.heatmap(mat.T, square = True, annot=True, fmt = "d",
xticklabels=train_data.target_names,yticklabels=train_data.target_names)

plt.xlabel("true labels")

plt.ylabel("predicted label")

plt.show()

print("The accuracy of SVM is {}".format(accuracy_score(test_data.target,
predicted_categories)))
```

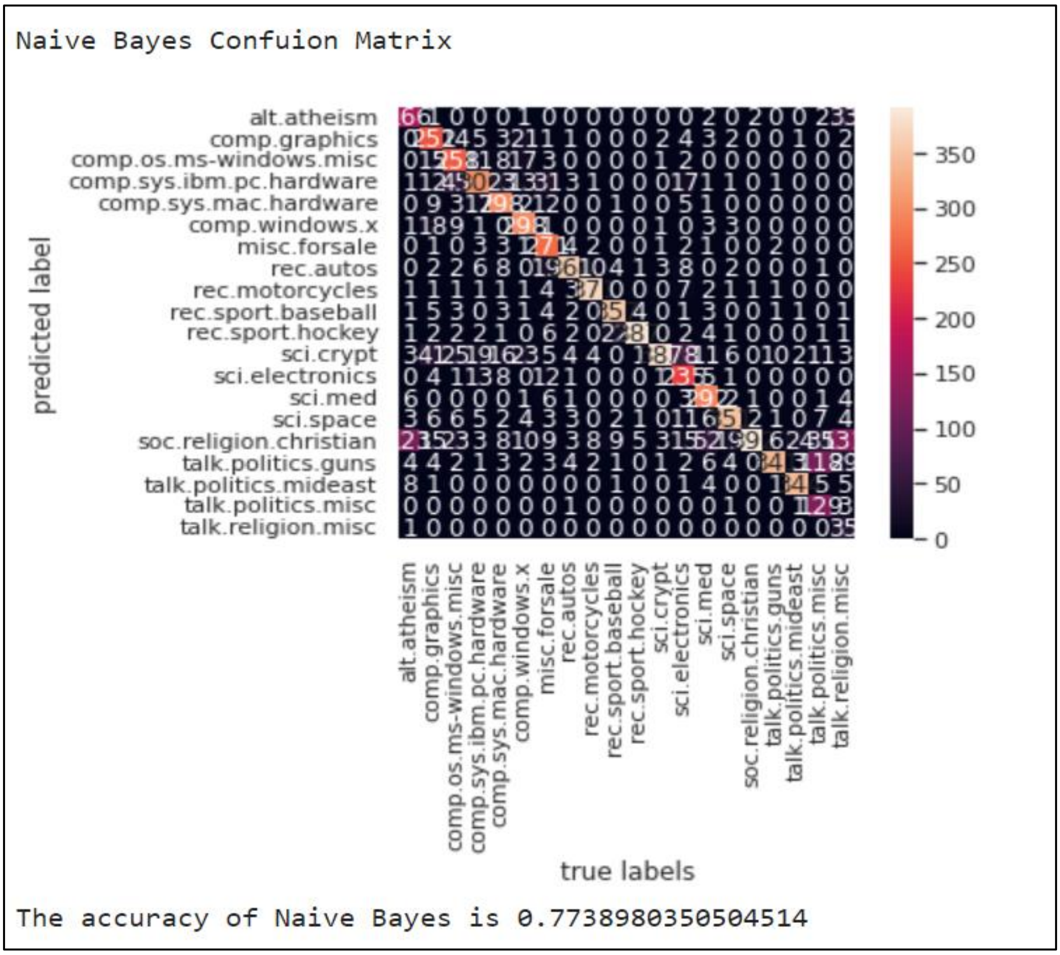
```
model1 = make_pipeline(TfidfVectorizer(), MultinomialNB())

model1.fit(train_data.data, train_data.target)

predicted_categories1 = model1.predict(test_data.data)


print("Naive Bayes Confuion Matrix \n")

mat = confusion_matrix(test_data.target, predicted_categories1)

sns.heatmap(mat.T, square = True, annot=True, fmt = "d",
xticklabels=train_data.target_names,yticklabels=train_data.target_names)

plt.xlabel("true labels")

plt.ylabel("predicted label")

plt.show()

print("The accuracy of Naive Bayes is {}".format(accuracy_score(test_data.target,
predicted_categories1)))
```

Output:



```
SMV Confuion Matrix
```

```
The accuracy of SVM is 0.8186404673393521
```



```
Naive Bayes Confuion Matrix
```

```
The accuracy of Naive Bayes is 0.7738980350504514
```

PRACTICAL-7

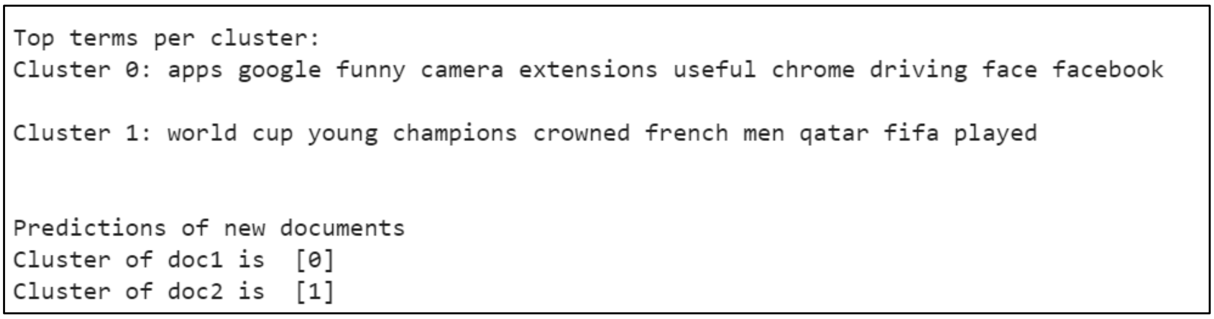
Aim: Implementation of K-means Clustering algorithm on text.

Code:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans

documents = ["the young french men crowned world champions",
        "Google Translate app is getting more intelligent everyday",
        "Facebook face recognition is driving me crazy",
        "who is going to win the Golden Ball title this year",
        "these camera apps are funny",
        "Croacian team made a brilliant world cup campaign reaching the final match",
        "Google Chrome extensions are useful.",
        "Social Media apps leveraging AI incredibly",
        "Qatar 2022 FIFA world cup is played in winter"]

vectorizer = TfidfVectorizer(stop_words = 'english')
data = vectorizer.fit_transform(documents)

clustering_model = KMeans(n_clusters = 2, init = 'k-means++', max_iter = 300, n_init = 10)
clustering_model.fit(data)

print("Top terms per cluster:")
sorted_centroids = clustering_model.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()

for i in range(true_k):
        print("Cluster %d:" % i, end='')
        for ind in sorted_centroids[i, :10]:
```

```
        print(' %s' % terms[ind], end = ' ')
    print()
    print()

print()
print("Predictions of new documents")
new_doc1 = ["how to install Chrome"]
Y = vectorizer.transform(new_doc1)
prediction1 = clustering_model.predict(Y)
print("Cluster of doc1 is ",prediction1)

new_doc2 = ["UCL Final match is played in Madrid this year"]
Y = vectorizer.transform(new_doc2)
prediction2 = clustering_model.predict(Y)
print("Cluster of doc2 is ",prediction2)
```

Output:



```
Top terms per cluster:
Cluster 0: apps google funny camera extensions useful chrome driving face facebook

Cluster 1: world cup young champions crowned french men qatar fifa played


Predictions of new documents
Cluster of doc1 is  [0]
Cluster of doc2 is  [1]
```

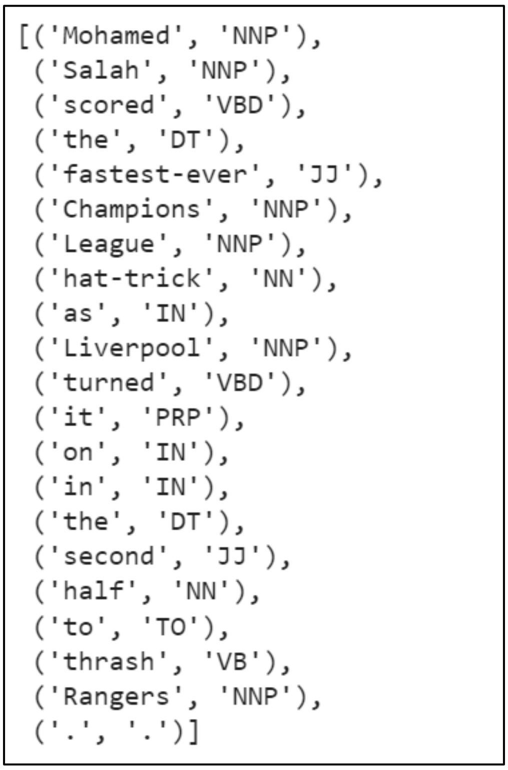
PRACTICAL - 8

Aim: Implement PoS Tagging on text.

Code:

```
import nltk

text = "Mohamed Salah scored the fastest-ever Champions League hat-trick as Liverpool turned it on in the second half to thrash Rangers."


from nltk.tokenize import word_tokenize


token_res = word_tokenize(text)

final_res = nltk.pos_tag(token_res)


print(final_res)
```

Output:



```
[('Mohamed', 'NNP'),
 ('Salah', 'NNP'),
 ('scored', 'VBD'),
 ('the', 'DT'),
 ('fastest-ever', 'JJ'),
 ('Champions', 'NNP'),
 ('League', 'NNP'),
 ('hat-trick', 'NN'),
 ('as', 'IN'),
 ('Liverpool', 'NNP'),
 ('turned', 'VBD'),
 ('it', 'PRP'),
 ('on', 'IN'),
 ('in', 'IN'),
 ('the', 'DT'),
 ('second', 'JJ'),
 ('half', 'NN'),
 ('to', 'TO'),
 ('thrash', 'VB'),
 ('Rangers', 'NNP'),
 ('.', '.')]
```

PRACTICAL - 9

Aim: Implement text processing with neural network

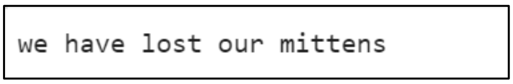
Code:

```
from keras_preprocessing.sequence import pad_sequences
from keras.layers import Embedding, LSTM, Dense, Dropout
from keras.preprocessing.text import Tokenizer
from keras.callbacks import EarlyStopping
from keras.models import Sequential
from tensorflow.keras.utils import to_categorical
import numpy as np
tokenizer = Tokenizer()

def dataset_preparation(data):
    corpus = data.lower().split("\n")
    tokenizer.fit_on_texts(corpus)
    total_words = len(tokenizer.word_index) + 1
    input_sequences = []
    for line in corpus:
        token_list = tokenizer.texts_to_sequences([line])[0]
        for i in range(1, len(token_list)):
            n_gram_sequence = token_list[:i+1]
            input_sequences.append(n_gram_sequence)
    max_sequence_len = max([len(x) for x in input_sequences])
    input_sequences = np.array(pad_sequences(input_sequences ,
    maxlen=max_sequence_len, padding='pre'))
    predictors, label = input_sequences[:,:-1],input_sequences[:,-1]
    label = to_categorical(label, num_classes=total_words)
    return predictors, label, max_sequence_len, total_words
```

```
data = """The cat and her kittens They put on their mittens, To eat a Christmas pie. The poor little kittens. They lost their mittens, And then they began to cry. O mother dear, we sadly fear We cannot go to-day, For we have lost our mittens. " "If it be so, ye shall not go, For ye are naughty kittens."""

def create_model(predictors, label, max_sequence_len, total_words):

    model = Sequential()

    model.add(Embedding(total_words, 10, input_length=max_sequence_len-1))

    model.add(LSTM(150, return_sequences = True))

    model.add(LSTM(100))

    model.add(Dense(total_words, activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    earlystop = EarlyStopping(monitor='val_loss', min_delta=0, patience=5, verbose=0, mode='auto')

    model.fit(predictors, label, epochs=100, verbose=1, callbacks=[earlystop])

    print (model.summary())

    return model


import numpy as np

def generate_text(seed_text, next_words, max_sequence_len):

    for _ in range(next_words):

        token_list = tokenizer.texts_to_sequences([seed_text])[0]

        token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')

        predicted = np.argmax(model.predict(token_list, verbose=0))

        output_word = ""

        for word, index in tokenizer.word_index.items():

            if index == predicted:

                output_word = word

                break

        seed_text += " " + output_word

    return seed_text

predictors, label, max_sequence_len, total_words = dataset_preparation(data)
```

```
model = create_model(predictors, label, max_sequence_len, total_words)

text = generate_text("we have", 3, max_sequence_len)

print (text)
```

Output:



```
we have lost our mittens
```

PRACTICAL - 10

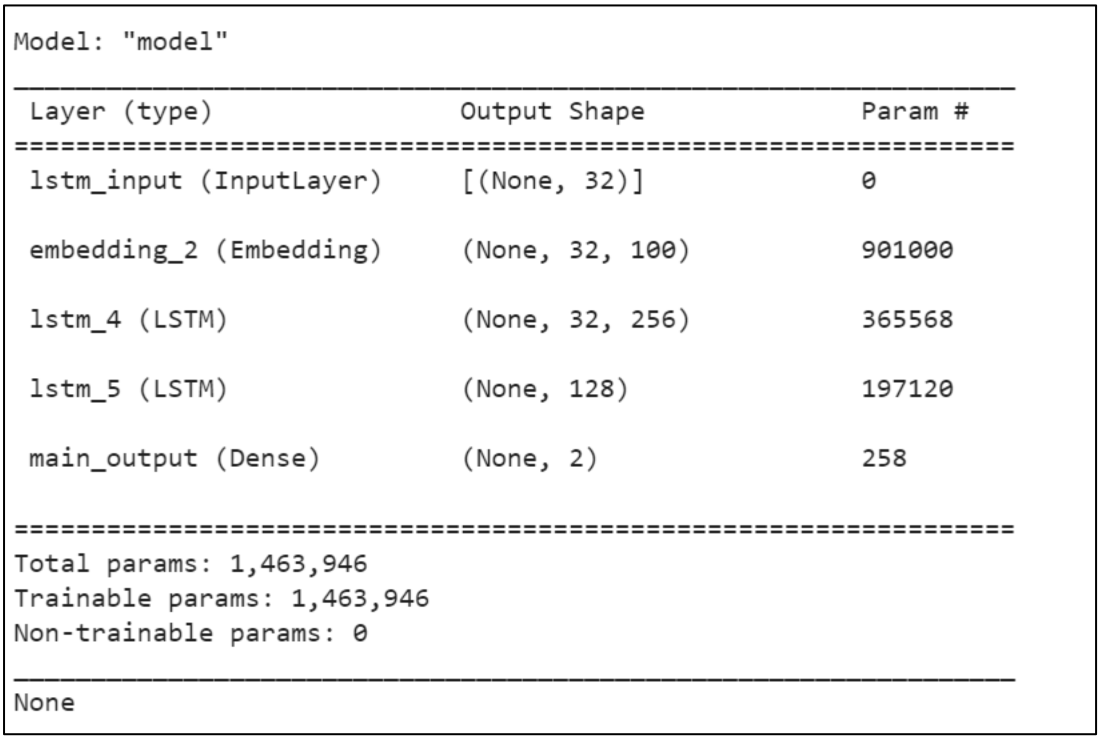
Aim: Implement text processing with LSTM.

Code:

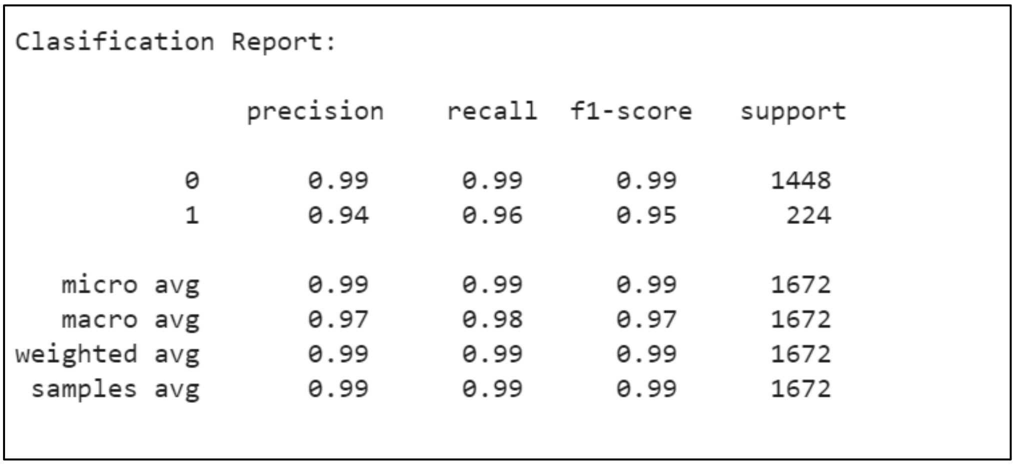
```
from google.colab import drive
drive.mount('/content/drive')


import pandas as pd
import numpy as np
from keras.utils.np_utils import to_categorical
from keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, Input
from keras.layers import Embedding, LSTM, Bidirectional, SimpleRNN, GRU
from keras.models import Sequential, Model


data = pd.read_csv('SMSSpamCollection', sep = '\t', names = ['label','message'])


text = data['message']
class_label = data['label']
classes_list = ["ham","spam"]
label_index = class_label.apply(classes_list.index)
label1 = np.asarray(label_index)
label = to_categorical(np.asarray(label1))


tk=Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',lower=True, split=" ")
tk.fit_on_texts(text)
index=tk.word_index
x = tk.texts_to_sequences(text)
vocab_size = len(index)
```

```
embedding_vecor_length =32
padded_docs = pad_sequences(x, maxlen=embedding_vecor_length, padding='post')

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(padded_docs, label, test_size=0.30, random_state=42)

lstm_input= Input(shape=(embedding_vecor_length,),  dtype='int32', name='lstm_input')
x= Embedding(vocab_size+1, 100, input_length=embedding_vecor_length,trainable=True)(lstm_input)
x1=LSTM(256,return_sequences=True)(x)
lstm_out= LSTM(128,return_sequences=False)(x1)
main_output = Dense(2,activation='softmax', name='main_output')(lstm_out)
model = Model(inputs=lstm_input, outputs=main_output)
model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=['accuracy'])
print(model.summary())

model.fit(X_train, y_train, validation_data = (X_test, y_test),epochs=2,batch_size=100, verbose=2)

predictions_test = model.predict(X_test)
predictions_test1 = np.zeros_like(predictions_test)
predictions_test1[np.arange(len(predictions_test)), predictions_test.argmax(1)] = 1

from sklearn.metrics import classification_report
print(classification_report(y_test,predictions_test1))
```

Output:



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_input (InputLayer)     [(None, 32)]              0

 embedding_2 (Embedding)     (None, 32, 100)           901000

 lstm_4 (LSTM)               (None, 32, 256)           365568

 lstm_5 (LSTM)               (None, 128)               197120

 main_output (Dense)         (None, 2)                 258

=================================================================
Total params: 1,463,946
Trainable params: 1,463,946
Non-trainable params: 0
_________________________________________________________________
None
```



```
Clasification Report:

              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1448
           1       0.94      0.96      0.95       224

   micro avg       0.99      0.99      0.99      1672
   macro avg       0.97      0.98      0.97      1672
weighted avg       0.99      0.99      0.99      1672
 samples avg       0.99      0.99      0.99      1672
```

PRACTICAL - 11

Aim: Implement HMM/CRF on sequence tagging task.

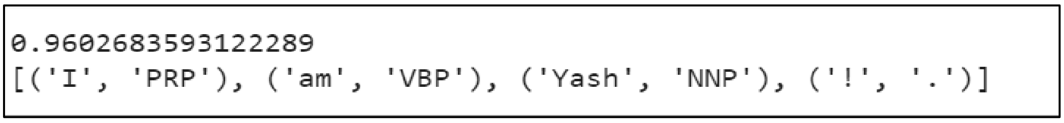
Code:

```
import nltk

nltk.download('treebank')

tagged_sentences = nltk.corpus.treebank.tagged_sents()


def features(sentence, index):

    """ sentence: [w1, w2, ...], index: the index of the word """

    return {

        'word': sentence[index],

        'is_first': index == 0,

        'is_last': index == len(sentence) - 1,

        'is_capitalized': sentence[index][0].upper() == sentence[index][0],

        'is_all_caps': sentence[index].upper() == sentence[index],

        'is_all_lower': sentence[index].lower() == sentence[index],

        'prefix-1': sentence[index][0],

        'prefix-2': sentence[index][:2],

        'prefix-3': sentence[index][:3],

        'suffix-1': sentence[index][-1],

        'suffix-2': sentence[index][-2:],

        'suffix-3': sentence[index][-3:],

        'prev_word': '' if index == 0 else sentence[index - 1],

        'next_word': '' if index == len(sentence) - 1 else sentence[index + 1],

        'has_hyphen': '-' in sentence[index],

        'is_numeric': sentence[index].isdigit(),

        'capitals_inside': sentence[index][1:].lower() != sentence[index][1:]

    }
```

```
from nltk.tag.util import untag

cutoff = int(.75 * len(tagged_sentences))
training_sentences = tagged_sentences[:cutoff]
test_sentences = tagged_sentences[cutoff:]

def transform_to_dataset(tagged_sentences):
    X, y = [], []
    for tagged in tagged_sentences:
        X.append([features(untag(tagged), index) for index in range(len(tagged))])
        y.append([tag for _, tag in tagged])
    return X, y

X_train, y_train = transform_to_dataset(training_sentences)
X_test, y_test = transform_to_dataset(test_sentences)

from sklearn_crfsuite import CRF
model = CRF()
try:
    model.fit(X_train, y_train)
except AttributeError:
    pass
from sklearn_crfsuite import metrics
y_pred = model.predict(X_test)
print(metrics.flat_accuracy_score(y_test, y_pred))

sentence = ['I', 'am', 'Yash','!']

def pos_tag(sentence):
    sentence_features = [features(sentence, index) for index in range(len(sentence))]
    return list(zip(sentence, model.predict([sentence_features])[0]))
```

```
print(pos_tag(sentence))
```

Output:



```
0.9602683593122289
[('I', 'PRP'), ('am', 'VBP'), ('Yash', 'NNP'), ('!', '.')]
```