[Lab 6]

Name: Nguyễn Thành Trung – MSSV: 19522431

Link github: [Data_Mining/Week6 at main · Shu2301/Data_Mining (github.com)]

IN CLASS

1. Feature Engineering
   - Text Normalization

```
import numpy as np
import pandas as pd
data = pd.read_csv('elonmusk_tweets.csv')
print(len(data))
data.head()
```

2819

	id	created_at	text
0	849636868052275200	2017-04-05 14:56:29	b'And so the robots spared humanity ... https:...
1	848988730585096192	2017-04-03 20:01:01	b"@ForIn2020 @waltmossberg @mims @defcon_5 Exa...
2	848943072423497728	2017-04-03 16:59:35	b'@waltmossberg @mims @defcon_5 Et tu, Walt?'
		2017-04-03	

```
from __future__ import print_function, division
from nltk.stem import PorterStemmer, WordNetLemmatizer
import nltk
nltk.download('punkt')
import string
from nltk.corpus import stopwords
import math
from collections import Counter
nltk.download('stopwords')
import pprint
pp = pprint.PrettyPrinter(indent=4)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
def normalize(document):
    # TODO: remove punctuation
    text = "".join([ch for ch in document if ch not in string.punctuation])

    # TODO: tokenize text
    tokens = nltk.word_tokenize(text)

    # TODO: Stemming
    stemmer = PorterStemmer()
    ret = " ".join([stemmer.stem(word.lower()) for word in tokens])
    return ret

original_documents = [x.strip() for x in data['text']]
documents = [normalize(d).split() for d in original_documents]
documents[0]
```

```
['band', 'so', 'the', 'robot', 'spare', 'human', 'httpstcov7jujqwfcv']
```

- **Implement TF-IDF**

```
# Flatten all the documents
flat_list = [word for doc in documents for word in doc]
```

```
# TODO: remove stop words from the vocabulary
words = [word for word in flat_list if word not in stopwords.words('english')]

# TODO: we take the 500 most common words only
counts = Counter(words)
vocabulary = counts.most_common(500)
print([x for x in vocabulary if x[0] == 'tesla'])
```

```
vocabulary = [x[0] for x in vocabulary] assert
len(vocabulary) == 500

# vocabulary.sort()
vocabulary[:5]
```

```
[('tesla', 287)]
['brt', 'tesla', 'spacex', 'model', 'thi']
```

```
def tf(vocabulary, documents):
    matrix = [0] * len(documents)
    for i, document in enumerate(documents):
        counts = Counter(document)
        matrix[i] = [0] * len(vocabulary)
        for j, term in enumerate(vocabulary):
            matrix[i][j] = counts[term]
    return matrix

tf = tf(vocabulary, documents)
np.array(vocabulary)[np.where(np.array(tf[1]) > 0)], np.array(tf[1])[np.where(np.array(tf[1]) > 0)] (array(['tesla',

    'exactli'], dtype='<U17'), array([1, 1]))
```

```
def idf(vocabulary, documents):
    """TODO: compute IDF, storing values in a dictionary""" idf =
    {}
    num_documents = len(documents)
    for i, term in enumerate(vocabulary):
      idf[term] = math.log(num_documents / sum(term in document for document in documents), 2) return
    idf

      3.3163095197385393,
      3.726258142344583 7,
      3.8171115727956972,
      3.8027562798186274]
[idf[key] for key in vocabulary[:5]]
```

```
def vectorize(document, vocabulary, idf): vector
    = [0]*len(vocabulary)
    counts = Counter(document)
    for i,term in enumerate(vocabulary):
        vector[i] = idf[term] * counts[term]
    return vector

document_vectors = [vectorize(s, vocabulary, idf) for s in documents]
np.array(vocabulary)[np.where(np.array(document_vectors[1]) > 0)], np.array(document_vectors[1])[np.where(np.array(document_vectors[1]) > 0)]
```

```
(array(['tesla', 'exactli'], dtype='<U17'), array([3.31630952, 6.65361284]))
```

1.3. Compare the results with the reference implementation of scikit-learn library.

Now we use the scikit-learn library. As you can see that, the way we do text normalization affects the result. Feel free to further improve upon (OPTIONAL), e.g. https://stackoverflow.com/questions/36182502/add-stemming-support-to-countvectorizer-sklearn

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer from
sklearn.metrics.pairwise import linear_kernel

tfidf = TfidfVectorizer(analyzer='word', ngram_range=(1,1), min_df = 1, stop_words = 'english', max_features=500) features

= tfidf.fit(original_documents)
corpus_tf_idf = tfidf.transform(original_documents)

sum_words = corpus_tf_idf.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in tfidf.vocabulary_.items()]
print(sorted(words_freq, key = lambda x: x[1], reverse=True)[:5])
print('testla', corpus_tf_idf[1, features.vocabulary_['tesla']])
```

```
[('http', 163.54366542841234), ('https', 151.85039944652075), ('rt', 112.61998731390989), ('tesla', 95.96401470715628), ('xe2', 88.2094
testla 0.3495243100660956
```

1.4. Apply TF-IDF for information retrieval

We can use the vector representation of documents to implement an information retrieval system. We test with the query Q = "tesla nasa"

```
def cosine_similarity(v1,v2):
    """TODO: compute cosine similarity""" sumxx,
    sumxy, sumyy = 0, 0, 0
    for i in range(len(v1)):
      x = v1[i]; y = v2[i]
      sumxx += x*x
      sumyy += y*y
      sumxy += x*y
    if sumxy == 0:
        result = 0
    else:
        result = sumxy/math.sqrt(sumxx*sumyy) return
    result

def search_vec(query, k, vocabulary, stemmer, document_vectors, original_documents): q =
    query.split()
    q = [stemmer.stem(w) for w in q]
    query_vector = vectorize(q, vocabulary, idf)

    # TODO: rank the documents by cosine similarity
    scores = [[cosine_similarity(query_vector, document_vectors[d]), d] for d in range(len(document_vectors))]
    scores.sort(key=lambda x: -x[0])

    print('Top-{0} documents'.format(k))
    for i in range(k):
        print(i, original_documents[scores[i][1]])

query = "tesla nasa"
stemmer = PorterStemmer()
search_vec(query, 5, vocabulary, stemmer, document_vectors, original_documents
```

```
Top-5 documents
0 b'@ashwin7002 @NASA @faa @AFPAA We have not ruled that out.'
1 b'RT @NASA: Updated @SpaceX #Dragon #ISS rendezvous times: NASA TV coverage begins Sunday at 3:30amET: http://t.co/qrm0Dz4jPE. Grappl
2 b"Deeply appreciate @NASA's faith in @SpaceX. We will do whatever it takes to make NASA and the American people proud."
3 b'Would also like to congratulate @Boeing, fellow winner of the @NASA commercial crew program'
4 b"@astrostephenson We're aiming for late 2015, but NASA needs to have overlapping capability to be safe. Would do the same"
```

```
new_features = tfidf.transform([query])

cosine_similarities = linear_kernel(new_features, corpus_tf_idf).flatten()
related_docs_indices = cosine_similarities.argsort()[::-1]

topk = 5
print('Top-{0} documents'.format(topk))
for i in range(topk):
    print(i, original_documents[related_docs_indices[i]])
```

```
Top-5 documents
0 b'@ashwin7002 @NASA @faa @AFPAA We have not ruled that out.'
1 b"SpaceX could not do this without NASA. Can't express enough appreciation. https://t.co/uQpI60zAV7"
2 b'@NASA launched a rocket into the northern lights http://t.co/tR2cSeMV'
3 b'Whatever happens today, we could not have done it without @NASA, but errors are ours alone and me most of all.'
4 b'RT @NASA: Updated @SpaceX #Dragon #ISS rendezvous times: NASA TV coverage begins Sunday at 3:30amET: http://t.co/qrm0Dz4jPE. Grappl
```

2. Text Processing

- Preprocessing

```
# Import NLTK and all the needed libraries import
nltk
nltk.download('punkt') #Run this line one time to get the resource
nltk.download('stopwords') #Run this line one time to get the resource
nltk.download('wordnet') #Run this line one time to get the resource
nltk.download('averaged_perceptron_tagger') #Run this line one time to get the resource import
numpy as np
import pandas as pd
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data] /root/nltk_data...
[nltk_data] Unzipping taggers/averaged_perceptron_tagger.zip.
```

```
# TODO: Load the dataset in coldplay.csv data =
pd.read_csv('coldplay.csv')
data.head(10)
```

	Artist	Song	Link	Lyrics
0	Coldplay	Another's Arms	/c/coldplay/anothers+arms_21079526.html	Late night watching tv \nUsed to be you here ...
1	Coldplay	Bigger Stronger	/c/coldplay/bigger+stronger_20032648.html	I want to be bigger stronger drive a faster ca...
2	Coldplay	Daylight	/c/coldplay/daylight_20032625.html	To my surprise, and my delight \nI saw sunris...
3	Coldplay	Everglow	/c/coldplay/everglow_21104546.html	Oh, they say people come \nThey say people go...
4	Coldplay	Every Teardrop Is A Waterfall	/c/coldplay/every+teardrop+is+a+waterfall_2091...	I turn the music up, I got my records on \nI ...

```
# TODO: Explore the data import

pandas as pd

# Create a DataFrame
df = pd.read_csv('coldplay.csv')

# Print the summary
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120 entries, 0 to 119
Data columns (total 4 columns):
 #   Column Non-Null Count Dtype
---  ------ -------------- -----
 0   Artist  120 non-null   object
 1   Song    120 non-null   object
 2   Link    120 non-null   object
 3   Lyrics  120 non-null   object
dtypes: object(4)
memory usage: 3.9+ KB
None
```

```
# TODO: Select the song 'Every Teardrop Is A Waterfall'
song_row = df[df['Song'] == 'Every Teardrop Is A Waterfall']

# Get the lyrics text for the selected song and save it into a variable lyrics
= song_row['Lyrics'].values[0]

# Print the lyrics
print(lyrics)
```

I turn the music up, I got my records on
I shut the world outside until the lights come on
Maybe the streets alight, maybe the trees are gone I feel
my heart start beating to my favourite song

And all the kids they dance, all the kids all night Until
Monday morning feels another life
I turn the music up
I'm on a roll this time
And heaven is in sight

I turn the music up, I got my records on
From underneath the rubble sing a rebel song Don't
want to see another generation drop
I'd rather be a comma than a full stop

Maybe I'm in the black, maybe I'm on my knees Maybe
I'm in the gap between the two trapezes But my heart
is beating and my pulses start
Cathedrals in my heart

As we saw oh this light I swear you, emerge blinking into To
tell me it's alright
As we soar walls, every siren is a symphony And
every tear's a waterfall
Is a waterfall
Oh
Is a waterfall Oh
oh oh
Is a is a waterfall Every
tear
Is a waterfall
Oh oh oh

So you can hurt, hurt me bad But
still I'll raise the flag
Oh was a wa wa wa wa wa-aterfall
A wa wa wa wa wa-aterfall

Every tear
Every tear
Every teardrop is a waterfall

Every tear
Every tear
Every teardrop is a waterfall

Every tear
Every tear
Every teardrop is a waterfall

TODO: Tokenize the lyrics of the song and save the tokens into a variable and print it # Select

the row for the song 'Every Teardrop Is A Waterfall'
song_row = df[df['Song'] == 'Every Teardrop Is A Waterfall']

Get the lyrics text for the selected song and save it into a variable lyrics
= song_row['Lyrics'].values[0]

# Tokenize the lyrics and save the tokens into a variable
tokens = nltk.word_tokenize(lyrics)

# Print the tokens
print(tokens)

```
['I', 'turn', 'the', 'music', 'up', ',', 'I', 'got', 'my', 'records', 'on', 'I', 'shut', 'the', 'world', 'outside', 'until', 'the', 'li
```

TODO: Remove the punctuation, then save the result into a variable and print it import

string
Select the row for the song 'Every Teardrop Is A Waterfall' song_row =
df[df['Song'] == 'Every Teardrop Is A Waterfall']

Get the lyrics text for the selected song and save it into a variable lyrics
= song_row['Lyrics'].values[0]

```
# Remove the punctuation from the lyrics
lyrics_no_punct = lyrics.translate(str.maketrans("", "", string.punctuation))

# Tokenize the lyrics without punctuation and save the tokens into a variable tokens =
nltk.word_tokenize(lyrics_no_punct)

# Print the tokens without punctuation print(tokens)
```

```
['I', 'turn', 'the', 'music', 'up', 'I', 'got', 'my', 'records', 'on', 'I', 'shut', 'the', 'world', 'outside', 'until', 'the', 'lights'
```

```
# TODO: remove the stop words using NLTK. Then put the result into a variable and print it #

Select the row for the song 'Every Teardrop Is A Waterfall'
song_row = df[df['Song'] == 'Every Teardrop Is A Waterfall']

# Get the lyrics text for the selected song and save it into a variable lyrics
= song_row['Lyrics'].values[0]

# Remove the punctuation from the lyrics
lyrics_no_punct = lyrics.translate(str.maketrans("", "", string.punctuation))

# Tokenize the lyrics without punctuation
tokens = nltk.word_tokenize(lyrics_no_punct)

# Remove the stop words from the tokens
stopwords = nltk.corpus.stopwords.words("english")
tokens_no_stopwords = [token for token in tokens if token.lower() not in stopwords]

# Print the tokens without punctuation and stop words print(tokens_no_stopwords)
```



```
['turn', 'music', 'got', 'records', 'shut', 'world', 'outside', 'lights', 'come', 'Maybe', 'streets', 'alight', 'maybe', 'trees', 'gone
```

```
# TODO: Perform lemmatization using WordNetLemmatizer on our tokens #

Select the row for the song 'Every Teardrop Is A Waterfall'
song_row = df[df['Song'] == 'Every Teardrop Is A Waterfall']

# Get the lyrics text for the selected song and save it into a variable lyrics
= song_row['Lyrics'].values[0]

# Remove the punctuation from the lyrics
lyrics_no_punct = lyrics.translate(str.maketrans("", "", string.punctuation))

# Tokenize the lyrics without punctuation and stop words tokens
= nltk.word_tokenize(lyrics_no_punct)
stopwords = nltk.corpus.stopwords.words("english")
tokens_no_stopwords = [token for token in tokens if token.lower() not in stopwords]

# Perform lemmatization on the tokens lemmatizer
= nltk.WordNetLemmatizer()
tokens_lemmatized = [lemmatizer.lemmatize(token) for token in tokens_no_stopwords]

# Print the lemmatized tokens
print(tokens_lemmatized)
```

```
['turn', 'music', 'got', 'record', 'shut', 'world', 'outside', 'light', 'come', 'Maybe', 'street', 'alight', 'maybe', 'tree', 'gone', '
```

```
# TODO: use the function pos_tag of NLTK to perform POS-tagging and print the result #

Select the row for the song 'Every Teardrop Is A Waterfall'
song_row = df[df['Song'] == 'Every Teardrop Is A Waterfall']

# Get the lyrics text for the selected song and save it into a variable lyrics
= song_row['Lyrics'].values[0]

# Remove the punctuation from the lyrics
lyrics_no_punct = lyrics.translate(str.maketrans("", "", string.punctuation)) #

Tokenize the lyrics without punctuation and stop words
```

```
tokens = nltk.word_tokenize(lyrics_no_punct)
stopwords = nltk.corpus.stopwords.words("english")
tokens_no_stopwords = [token for token in tokens if token.lower() not in stopwords]

# Perform POS-tagging on the tokens
pos_tags = nltk.pos_tag(tokens_no_stopwords)

# Print the POS-tags
print(pos_tags)
```

```
[('turn', 'NN'), ('music', 'NN'), ('got', 'VBD'), ('records', 'NNS'), ('shut', 'VBN'), ('world', 'NN'), ('outside', 'IN'), ('lights', '
```

```
from nltk.corpus import wordnet

def get_wordnet_pos(pos_tag):
    output = np.asarray(pos_tag) for
    i in range(len(pos_tag)):
        if pos_tag[i][1].startswith('J'):
            output[i][1] = wordnet.ADJ
        elif pos_tag[i][1].startswith('V'):
            output[i][1] = wordnet.VERB
        elif pos_tag[i][1].startswith('R'):
            output[i][1] = wordnet.ADV
        else:
            output[i][1] = wordnet.NOUN
    return output


# TODO: Perform the lemmatization properly


# Get the lyrics text for the selected song and save it into a variable lyrics
# Select the row for the song 'Every Teardrop Is A Waterfall'
song_row = df[df['Song'] == 'Every Teardrop Is A Waterfall']
# Remove the punctuation from the lyrics
lyrics_no_punct = lyrics.translate(str.maketrans("", "", string.punctuation))

# Tokenize the lyrics without punctuation and stop words tokens
= nltk.word_tokenize(lyrics_no_punct)
stopwords = nltk.corpus.stopwords.words("english")
tokens_no_stopwords = [token for token in tokens if token.lower() not in stopwords]

# Perform POS-tagging on the tokens
pos_tags = nltk.pos_tag(tokens_no_stopwords)

# Create a WordNetLemmatizer object lemmatizer
= WordNetLemmatizer()

# Perform lemmatization on the tokens with proper POS tagging
lemmatized_tokens = []
for token, pos in pos_tags: if
    pos.startswith('J'):
        # Adjective
        lemma = lemmatizer.lemmatize(token, pos='a') elif
    pos.startswith('V'):
        # Verb
        lemma = lemmatizer.lemmatize(token, pos='v') elif
    pos.startswith('N'):
        # Noun
        lemma = lemmatizer.lemmatize(token, pos='n') elif
    pos.startswith('R'):
        # Adverb
        lemma = lemmatizer.lemmatize(token, pos='r')
    else:
        # Default to noun
        lemma = lemmatizer.lemmatize(token)
    lemmatized_tokens.append(lemma)

# Print the lemmatized tokens
print(lemmatized_tokens)
```

```
['turn', 'music', 'get', 'record', 'shut', 'world', 'outside', 'light', 'come', 'Maybe', 'street', 'alight', 'maybe', 'tree', 'go', 'fe
```

```
# TODO: Perform stemming

from nltk.stem import PorterStemmer

# Select the row for the song 'Every Teardrop Is A Waterfall'
song_row = df[df['Song'] == 'Every Teardrop Is A Waterfall']

# Get the lyrics text for the selected song and save it into a variable
lyrics = song_row['Lyrics'].values[0]

# Remove the punctuation from the lyrics
lyrics_no_punct = lyrics.translate(str.maketrans("", "", string.punctuation))

# Tokenize the lyrics without punctuation and stop words
tokens = nltk.word_tokenize(lyrics_no_punct)
stopwords = nltk.corpus.stopwords.words("english")
tokens_no_stopwords = [token for token in tokens if token.lower() not in stopwords]

# Perform stemming on the tokens
stemmer = PorterStemmer()
stemmed_tokens = [stemmer.stem(token) for token in tokens_no_stopwords]

# Print the stemmed tokens
print(stemmed_tokens)
```

```
['turn', 'music', 'got', 'record', 'shut', 'world', 'outsid', 'light', 'come', 'mayb', 'street', 'alight', 'mayb', 'tree', 'gone', 'fee
```

```
import nltk
import numpy as np
import pandas as pd


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

```
Mounted at /content/drive
```

```
A = "Outside the classroom, Stallman pursued his studies with even more diligence, rushing off to fulfill his laboratory-assistant duties at B = "To facilitate the process, AI Lab hackers had built a system that displayed both the source and display modes on a split screen. Despite C = "With no dorm and no dancing, Stallman's social universe imploded. Like an astronaut experiencing the aftereffects of zero-gravity, Stall


# TODO: compute the Jaccard similarities #
Split the sentences

    # Compute the intersection and union

# Compute and print the Jaccard Similarity #

Define the three sets of words


# Compute the Jaccard Similarity between sets AB, BC, and AC
jaccard_similarity_AB = len(A.string(B)) / len(A.union(B))
jaccard_similarity_BC = len(B.string(C)) / len(B.union(C))
jaccard_similarity_AC = len(A.string(C)) / len(A.union(C))

# Print the Jaccard Similarity between sets AB, BC, and AC
print("Jaccard Similarity AB: ", jaccard_similarity_AB)
print("Jaccard Similarity BC: ", jaccard_similarity_BC)
print("Jaccard Similarity AC: ", jaccard_similarity_AC)


# TODO: compute the TF-IDF of A, B and C and the cosine similarities of all possibilities from

sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

```
# Create a TfidfVectorizer object
vectorizer = TfidfVectorizer()

# Compute the TF-IDF matrix
tfidf_matrix = vectorizer.fit_transform([A, B, C])

# Compute the cosine similarities between all pairs
cosine_sim_AB = cosine_similarity(tfidf_matrix[0], tfidf_matrix[1])[0][0]
cosine_sim_BC = cosine_similarity(tfidf_matrix[1], tfidf_matrix[2])[0][0]
cosine_sim_AC = cosine_similarity(tfidf_matrix[0], tfidf_matrix[2])[0][0]

# Print the TF-IDF matrix and cosine similarities #
print("TF-IDF matrix:")
# print(tfidf_matrix.toarray())
print("cos(A, B):", cosine_sim_AB)
print("cos(B, C):", cosine_sim_BC)
print("cos(A, C):", cosine_sim_AC)
```

```
cos(A, B): 0.16793269576264072
cos(B, C): 0.13618963113796592
cos(A, C): 0.2850296032333907
```

3. Text Classification

```
# Import NLTK and all the needed libraries import
nltk
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

data.head()
# TODO: Load the dataset in coldplay.csv
data = pd.read_csv('coldplay.csv')
```

	Artist	Song	Link	Lyrics
0	Coldplay	Another's Arms	/c/coldplay/anothers+arms_21079526.html	Late night watching tv \nUsed to be you here ...
1	Coldplay	Bigger Stronger	/c/coldplay/bigger+stronger_20032648.html	I want to be bigger stronger drive a faster ca...
2	Coldplay	Daylight	/c/coldplay/daylight 20032625 html	To my surprise, and my delight \nI saw

```
# TODO: Explore the data #
Create a DataFrame
data = pd.read_csv('coldplay.csv')

# Print the summary
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120 entries, 0 to 119
Data columns (total 4 columns):
 #   Column Non-Null Count Dtype
---  ------ -------------- -----
 0   Artist 120 non-null   object
 1   Song   120 non-null   object
 2   Link   120 non-null   object
 3   Lyrics 120 non-null   object
dtypes: object(4)
memory usage: 3.9+ KB
None
```

```
# TODO: Compute a BOW
from sklearn.feature_extraction.text import CountVectorizer

# Select all the rows for Coldplay songs
coldplay_rows = df[df['Artist'] == 'Coldplay']

# Get the lyrics text for all the Coldplay songs and save them into a list lyrics_list =
coldplay_rows['Lyrics'].tolist()
```

```
# Create a CountVectorizer object vectorizer =
CountVectorizer()

# Fit and transform the lyrics into a bag-of-words matrix bow_matrix
= vectorizer.fit_transform(lyrics_list)

# Print the shape of the bag-of-words matrix
print(bow_matrix.shape)
```

(120, 1776)

```
# TODO: Create a new dataframe containing the BOW outputs and the corresponding words as columns. And print it from

sklearn.feature_extraction.text import CountVectorizer

# Select all the rows for Coldplay songs
coldplay_rows = df[df['Artist'] == 'Coldplay']

# Get the lyrics text for all the Coldplay songs and save them into a list lyrics_list =
coldplay_rows['Lyrics'].tolist()

# Create a CountVectorizer object vectorizer =
CountVectorizer()

# Fit and transform the lyrics into a bag-of-words matrix
bow_matrix = vectorizer.fit_transform(lyrics_list)

# Get the feature names (i.e., the vocabulary) of the bag-of-words matrix feature_names =
vectorizer.get_feature_names_out()

# Convert the bag-of-words matrix to a dense matrix and create a new dataframe
bow_df = pd.DataFrame(bow_matrix.toarray(), columns=feature_names)

# Print the new dataframe
print(bow_df)
```

	10	2000	2gether	76543	aaaaaah	aaaaah	aaah	about	above	achin	\
0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	
..	..	...	...	...	...	...	...	...	...	...	
115	0	0	0	0	0	0	0	1	2	0	
116	0	0	0	0	0	0	0	0	0	0	
117	0	0	1	0	0	0	0	0	0	0	
118	0	0	0	0	0	0	0	0	0	0	
119	0	0	0	0	0	0	0	0	0	0	

	...	yellow	yes	yesterday	yet	you	young	your	yours	yourself	\
0	...	0	0	0	0	4	0	4	0	2	
1	...	0	0	0	0	0	0	0	0	0	
2	...	0	0	0	0	0	0	0	0	0	
3	...	0	0	0	0	16	0	0	0	0	
4	...	0	0	0	0	2	0	0	0	0	
..	...	...	...	...	...	...	...	...	...	...	
115	...	0	0	0	0	5	0	3	0	0	
116	...	0	0	0	0	9	0	0	0	0	
117	...	0	0	0	0	7	0	4	0	0	
118	...	0	0	0	0	16	0	1	0	0	
119	...	0	0	0	0	5	0	0	0	0	

	yuletide
0	0
1	0
2	0
3	0
4	0
..	...
115	0
116	0

117	0
118	0
119	0

[120 rows x 1776 columns]

```
sum_bow = bow_df.sum()
sum_bow.idxmax()
```

```
'you'
```

```
# TODO: print the 10 most used word by Coldplay

import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

# Select all the rows for Coldplay songs
coldplay_rows = df[df['Artist'] == 'Coldplay']

# Get the lyrics text for all the Coldplay songs and save them into a list lyrics_list =
coldplay_rows['Lyrics'].tolist()

# Create a CountVectorizer object vectorizer =
CountVectorizer()

# Fit and transform the lyrics into a bag-of-words matrix
bow_matrix = vectorizer.fit_transform(lyrics_list)

# Get the feature names (i.e., the vocabulary) of the bag-of-words matrix feature_names =
vectorizer.get_feature_names_out()

# Convert the bag-of-words matrix to a dense matrix and create a new dataframe bow_df =
pd.DataFrame(bow_matrix.toarray(), columns=feature_names)

# Print the new dataframe
print(bow_df)

# Create a new dataframe with the word counts and feature names
word_counts_df = pd.DataFrame({'word': feature_names, 'count': bow_matrix.sum(axis=0).tolist()[0]})
```



```
# Sort the dataframe by word count in descending order
sorted_word_counts_df = word_counts_df.sort_values('count', ascending=False)

# Print the top 10 most used words by Coldplay print(sorted_word_counts_df.head(10))
```

	10	2000	2gether	76543	aaaaaah	aaaaah	aaah	about	above	achin	\
0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	
..	..	...	...	...	...	...	...	...	...	...	
115	0	0	0	0	0	0	0	1	2	0	
116	0	0	0	0	0	0	0	0	0	0	
117	0	0	1	0	0	0	0	0	0	0	
118	0	0	0	0	0	0	0	0	0	0	
119	0	0	0	0	0	0	0	0	0	0	

	...	yellow	yes	yesterday	yet	you	young	your	yours	yourself	\
0	...	0	0	0	0	4	0	4	0	2	
1	...	0	0	0	0	0	0	0	0	0	
2	...	0	0	0	0	0	0	0	0	0	
3	...	0	0	0	0	16	0	0	0	0	
4	...	0	0	0	0	2	0	0	0	0	
..	...	...	...	...	...	...	...	...	...	...	
115	...	0	0	0	0	5	0	3	0	0	
116	...	0	0	0	0	9	0	0	0	0	
117	...	0	0	0	0	7	0	4	0	0	
118	...	0	0	0	0	16	0	1	0	0	
119	...	0	0	0	0	5	0	0	0	0	

	yuletide
0	0
1	0
2	0
3	0
4	0
..	...

115	0
116	0
117	0
118	0
119	0

[120 rows x 1776 columns]

	word	count
1770	you	994
1523	the	777
39	and	650
1571	to	481
746	it	458
991	oh	334
730	in	318
892	me	314
948	my	288
996	on	285

4. Topic Modelling

```
import pandas as pd
df = pd.read_csv('random_headlines.csv')
```

```
df.head(10)
```

	publish_date	headline_text
0	20120305	ute driver hurt in intersection crash
1	20081128	6yo dies in cycling accident
2	20090325	bumper olive harvest expected
3	20100201	replica replaces northernmost sign
4	20080225	woods targets perfect season
5	20091120	leckie salvages dramatic draw for adelaide
9	20130304	thailand signs agreement with muslim rebels
7	20130304	anti hunting rally still going ahead

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   publish_date   20000 non-null int64
 1   headline_text  20000 non-null object
dtypes: int64(1), object(1)
memory usage: 312.6+ KB
```

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
import string
```

```
df['lowercase'] = df['headline_text'].str.lower()
df['tokens'] = df['lowercase'].apply(word_tokenize)
df['no_punctuation'] = df['tokens'].apply(lambda tokens: [token for token in tokens if token not in string.punctuation]) stopwords_set =
set(stopwords.words('english'))
df['no_stopwords'] = df['no_punctuation'].apply(lambda tokens: [token for token in tokens if token not in stopwords_set]) stemmer
= PorterStemmer()
df['stemmed'] = df['no_stopwords'].apply(lambda tokens: [stemmer.stem(token) for token in tokens])
```

```
---------------------------------------------------------------------------
LookupError                               Traceback (most recent call last)
<ipython-input-10-797e3db7910d> in <cell line: 2>()
      1 df['lowercase'] = df['headline_text'].str.lower()
----> 2 df['tokens'] = df['lowercase'].apply(word_tokenize)
      3 df['no_punctuation'] = df['tokens'].apply(lambda tokens: [token for token in
tokens if token not in string.punctuation])
      4 stopwords_set = set(stopwords.words('english'))
      5 df['no_stopwords'] = df['no_punctuation'].apply(lambda tokens: [token for token in
tokens if token not in stopwords_set])

                                  8 frames
/usr/local/lib/python3.10/dist-packages/nltk/data.py in find(resource_name, paths) 581
        sep = "*" * 70
    582     resource_not_found = f"\n{sep}\n{msg}\n{sep}\n"
--> 583     raise LookupError(resource_not_found)
    584
    585

LookupError:
**********************************************************
  *************** Resource punkt not found.
  Please use the NLTK Downloader to obtain the resource:

  >>> import nltk
  >>> nltk.download('punkt')

  For more information see: h ttps://www.nltk.org/data.html

  Attempted to load tokenizers/punkt/PY3/english.pickle

  Searched in:
    - '/root/nltk_data'
    - '/usr/nltk_data'
    - '/usr/share/nltk_data'
    - '/usr/lib/nltk_data'
    - '/usr/share/nltk_data'
    - '/usr/local/share/nltk_data'
    - '/usr/lib/nltk_data'
    - '/usr/local/lib/nltk_data'
**********************************************************
```

SEARCH STACK OVERFLOW

```
df['stemmed'] =
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
import string
from gensim.corpora import Dictionary



lemmatizer = WordNetLemmatizer()
df['lemmatized'] = df['no_stopwords'].apply(lambda tokens: [lemmatizer.lemmatize(token) for token in tokens])


# Create a dictionary of the tokens
dictionary = Dictionary(df['stemmed'])

# Filter out rare and common tokens
dictionary.filter_extremes(no_below=5, no_above=0.5)

# Convert each headline to its BOW representation
df['bow'] = df['stemmed'].apply(lambda tokens: dictionary.doc2bow(tokens))
```

5. Named Entity Recognition

```
adafile = "ada_lovelace.txt"


def clean_file(filename):
    with open(filename, 'r') as file: contents
        = file.read()
```

```
    redacted_contents = contents.replace("Ada Lovelace", "[REDACTED]")

    with open(filename, 'w') as file:
        file.write(redacted_contents)

clean_file(adafile)


import spacy

def identify_entities(filename):
    nlp = spacy.load("en_core_web_sm")

    with open(filename, 'r') as file: contents
        = file.read()

    doc = nlp(contents)

    for entity in doc.ents:
        print(entity.text, entity.label_)

identify_entities(adafile)
```

```
Augusta Ada King PERSON
Countess PERSON
Lovelace PERSON
Byron ORG
10 December 1815 DATE
27 November 1852 DATE
English LANGUAGE
Charles Babbage's ORG
the Analytical Engine ORG
first ORDINAL
first ORDINAL
first ORDINAL
one CARDINAL
Lovelace PERSON
first ORDINAL
Mary Somerville PERSON
Charles Babbage PERSON
1833 DATE
Somerville GPE many
years DATE
Andrew Crosse PERSON
David Brewster PERSON
Charles Wheatstone PERSON Michael
Faraday PERSON
Charles Dickens PERSON
```

```
import spacy
from spacy import displacy
from IPython.display import display def

visualize_entities(filename):

    nlp = spacy.load("en_core_web_sm")

    with open(filename, 'r') as file: contents
        = file.read()

    doc = nlp(contents)

    displacy.render(doc, style="ent", jupyter=True)


visualize_entities(adafile)
```

Augusta Ada King PERSON , Countess PERSON of Lovelace PERSON (née Byron ORG ; 10 December 1815 DATE – 27 November 1852 DATE) was an English LANGUAGE

```
import spacy

def replace_name_by_redacted(filename): nlp =
    spacy.load("en_core_web_sm")

    with open(filename, 'r') as file: contents
        = file.read()

    doc = nlp(contents)

    redacted_contents = contents for
    entity in doc.ents:
        if entity.label_ == "PERSON":
            redacted_contents = redacted_contents.replace(entity.text, "[REDACTED]")

    with open(filename, 'w') as file:
        file.write(redacted_contents)

replace_name_by_redacted(adafile)
```

```
import spacy

def make_doc_GDPR_compliant(filename):

    nlp = spacy.load("en_core_web_sm")

    redacted_contents = file.contents( for
    entity in doc.ents:
        if entity.label_ == "PERSON":
            redacted_contents = redacted_contents.replace(entity.text, "[REDACTED]")

    with open(filename, 'w') as file:
        file.write(redacted_contents)

make_doc_GDPR_compliant(adafile)
```

contents = file.read()

6. Exercise

```
jobmarket = "job-market.csv"


jobs_df = pd.read_csv("job-market.csv")
jobs_df.fillna(0)
```

	Id	Title	Company	Date	Location	Area C
0	37404348.0	Casual Stock Replenisher	Aldi Stores	2018-10-07T00:00:00.000Z	Sydney	North West & Hills District
1	37404337.0	Casual Stock Replenisher	Aldi Stores	2018-10-07T00:00:00.000Z	Richmond & Hawkesbury	0
2	37404356.0	RETAIL SALES SUPERSTARS and STYLISTS Wanted - ...	LB Creative Pty Ltd	2018-10-07T00:00:00.000Z	Brisbane	CBD & Inner Suburbs
		Team member	Anaconda	2018-10-	Gosford &	

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer from
sklearn.linear_model import LogisticRegression

# Step 1: Filter the jobs for the IT sector only
jobs_df['Title'] = jobs_df['Title'].fillna('')
it_jobs_df = jobs_df[jobs_df['Title'].str.contains('IT', case=False)]


favorite_keyword = 'python'
```

40784	0.0	0	0	0	0	0

```
# Construct a query using the favorite keyword
query = vectorizer.transform([favorite_keyword])
```

40786	0.0	0	0	0	0	0

```
# Step 3: Use scikit-learn to get the top 20 important keywords
vectorizer = TfidfVectorizer(stop_words='english', max_features=1000)
```

40787	0.0	0	0	0	0	0
40788	0.0	0	0	0	0	0

40789 rows × 13 columns

```
# Compute similarity scores between the query and job descriptions
similarity_scores = model.predict_proba(query)[0]

# Rank the job descriptions based on similarity scores
ranked_jobs = sorted(zip(similarity_scores, it_jobs_df['Title'], job_descriptions), reverse=True)

# Print the top 5 job descriptions most similar to the query
print("\nTop 5 job descriptions most similar to the query:") for
score, job_title, job_description in ranked_jobs[:5]:
    print("Job Title:", job_title)
    print("Similarity Score:", score)
    print("Job Description:", job_description)
    print()
```

```
Top 5 job descriptions most similar to the query: Job
Title: Credit Controller - Temporary Position Similarity
Score: 0.004172072785443803
Job Description: Accounting

Job Title: Wait Staff
Similarity Score: 0.0033997001440222003 Job
Description: Hospitality & Tourism

Job Title: Solution Architect (IAM)
Similarity Score: 0.003043823447949755
Job Description: Information & Communication Technology

Job Title: Recruitment Consultant
Similarity Score: 0.0027987719279321284
Job Description: Human Resources & Recruitment
```

Job Description: Information & Communication Technology

```
def extract_ngrams(sequence, n):
    ngrams = []
    sequence_length = len(sequence)
    for i in range(sequence_length - n + 1):
        ngram = sequence[i:i+n]
        ngrams.append(ngram)
    return ngrams


sentence = "I like deadline and want to immerse myself in deadline."  #

Extract word tri-grams
words = sentence.split()
word_trigrams = extract_ngrams(words, 3)

print("Word Tri-grams:")
for trigram in word_trigrams: print(trigram)

# Extract letter tri-grams
letters = list(sentence.replace(" ", ""))
letter_trigrams = extract_ngrams(letters, 3)

print("\nLetter Tri-grams:")
for trigram in letter_trigrams:
    print(trigram)
```

```
Word Tri-grams:
['I', 'like', 'deadline']
['like', 'deadline', 'and']
['deadline', 'and', 'want']
['and', 'want', 'to']
['want', 'to', 'immerse']
['to', 'immerse', 'myself']
['immerse', 'myself', 'in']
['myself', 'in', 'deadline.']

Letter Tri-grams:
['I', 'l', 'i']
['l', 'i', 'k']
['i', 'k', 'e']
['k', 'e', 'd']
```

['e', 'd', 'e']
['d', 'e', 'a']
['e', 'a', 'd']
['a', 'd', 'l']
['d', 'l', 'i']
['l', 'i', 'n']
['i', 'n', 'e']
['n', 'e', 'a']
['e', 'a', 'n']
['a', 'n', 'd']
['n', 'd', 'w']
['d', 'w', 'a']
['w', 'a', 'n']
['a', 'n', 't']
['n', 't', 't']
['t', 't', 'o']
['t', 'o', 'i']
['o', 'i', 'm']
['i', 'm', 'm']
['m', 'm', 'e']
['m', 'e', 'r']
['e', 'r', 's']
['r', 's', 'e']
['s', 'e', 'm']
['e', 'm', 'y']
['m', 'y', 's']
['y', 's', 'e']
['s', 'e', 'l']
['e', 'l', 'f']
['l', 'f', 'i']
['f', 'i', 'n']
['i', 'n', 'd']
['n', 'd', 'e']
['d', 'e', 'a']
['e', 'a', 'd']
['a', 'd', 'l']
['d', 'l', 'i']
['l', 'i', 'n']
['i', 'n', 'e']

```
import random

def modify_phrase(phrase):
    words = phrase.split()
    modified_words = []

    for word in words:
        if len(word) <= 4:
            modified_words.append(word)
        else:
            first_letter = word[0]
            last_letter = word[-1]
            middle_letters = list(word[1:-1])
            random.shuffle(middle_letters)
            modified_word = first_letter + ''.join(middle_letters) + last_letter modified_words.append(modified_word)

    modified_phrase = ' '.join(modified_words) return
    modified_phrase

# Example phrase
phrase = "I couldn't believe that I could completely understand what I was reading: the astounding power of the human mind" modified_phrase =

modify_phrase(phrase)

print("Original phrase:")
print(phrase)
print()
print("Modified phrase:") print(modified_phrase)
```

```
Original phrase:
I couldn't believe that I could completely understand what I was reading: the astounding power of the human mind

Modified phrase:
I clu'ndot blievee that I cloud clpmeotley unasdnterd what I was rniaegd: the aidnosuntg pweor of the haumn mind
```

```
alice = "alice.txt"
```

```
import nltk

nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
import nltk

# Read the input file
with open(alice, 'r') as file:
    text = file.read()

# Tokenize the text into sentences
sentences = nltk.sent_tokenize(text)

# Perform POS tagging on each sentence tagged_sentences =
[]
for sentence in sentences:
    tagged_sentence = nltk.pos_tag(nltk.word_tokenize(sentence))
    tagged_sentences.append(tagged_sentence)

# Save the POS tagged output to a separate file
output_file = 'alice_pos_tagged.txt'
with open(output_file, 'w') as file:
    for tagged_sentence in tagged_sentences:
        tagged_text = ' '.join([f"{word}/{tag}" for word, tag in tagged_senten
        file.write(tagged_text + '\n')

print(f"POS tagged output saved to '{output_file}'.")
```

```
POS tagged output saved to 'alice_pos_tagged.txt'.
```

```
# Open the POS tagged file for reading
with open('alice_pos_tagged.txt', 'r') as file:
    pos_tagged_text = file.read()

# Print the contents of the POS tagged file
print("POS tagged text:")
print(pos_tagged_text)
```

The/DT first/JJ question/NN of/IN course/NN was/VBD ,/, how/WRB to/TO get/VB dry/JJ again/RB :/: they/PRP had/VBD a/DT consultation/N
Indeed/RB ,/, she/PRP had/VBD quite/RB a/DT long/JJ argument/NN with/IN the/DT Lory/NNP ,/, who/WP at/IN last/JJ turned/JJ sulky/NN , At/IN last/JJ
the/DT Mouse/NNP ,/, who/WP seemed/VBD to/TO be/VB a/DT person/NN of/IN authority/NN among/IN them/PRP ,/, called/VBN o I/PRP 'LL/VBP soon/RB
make/VBP you/PRP dry/JJ enough/RB !/. ''/''
They/PRP all/DT sat/VBD down/RP at/IN once/RB ,/, in/IN a/DT large/JJ ring/NN ,/, with/IN the/DT Mouse/NNP in/IN the/DT middle/NN ./. Alice/NNP
kept/VBD her/PRP$ eyes/NNS anxiously/RB fixed/VBN on/IN it/PRP ,/, for/IN she/PRP felt/VBD sure/JJ she/PRP would/MD catch/V 'Ahem/RB !/. ''/''
said/VBD the/DT Mouse/NNP with/IN an/DT important/JJ air/NN ,/, 'are/'' you/PRP all/DT ready/JJ ?/. This/DT
is/VBZ the/DT driest/JJ thing/NN I/PRP know/VBP ./.
Silence/NNP all/DT round/NN ,/, if/IN you/PRP please/VBP !/.
``/`` William/NNP the/DT Conqueror/NNP ,/, whose/WP$ cause/NN was/VBD favoured/VBN by/IN the/DT pope/NN ,/, was/VBD soon/RB submitted
Edwin/NNP and/CC Morcar/NNP ,/, the/DT earls/NN of/IN Mercia/NNP and/CC Northumbria/NNP --/: ''/'' '/POS 'Ugh/POS !/. ''/''
said/VBD the/DT Lory/NNP ,/, with/IN a/DT shiver/NN ./. '/POS
I/PRP beg/VBP your/PRP$ pardon/NN !/. ''/''
said/VBD the/DT Mouse/NNP ,/, frowning/NN ,/, but/CC very/RB politely/RB :/: 'Did/CD you/PRP speak/VB ?/. ''/''
'Not/CD I/PRP !/. ''/''
said/VBD the/DT Lory/NNP hastily/RB ./.
'/POS I/PRP thought/VBD you/PRP did/VBD ,/, ''/'' said/VBD the/DT Mouse/NNP ./. ''/''
--/: I/PRP proceed/VBP ./.
``/`` Edwin/NNP and/CC Morcar/NNP ,/, the/DT earls/NN of/IN Mercia/NNP and/CC Northumbria/NNP ,/, declared/VBD for/IN him/PRP :/: and said/VBD
the/DT Duck/NNP ./.
'Found/IN IT/NNP ,/, ''/'' the/DT Mouse/NNP replied/VBD rather/RB crossly/RB :/: 'of/JJ course/NN you/PRP know/VBP what/WP ''/'' it/PR '/POS
I/PRP know/VBP what/WP ``/`` it/PRP ''/'' means/VBZ well/RB enough/RB ,/, when/WRB I/PRP find/VBP a/DT thing/NN ,/, ''/'' said/V The/DT
question/NN is/VBZ ,/, what/WP did/VBD the/DT archbishop/NN find/VB ?/. ''/''
The/DT Mouse/NNP did/VBD not/RB notice/VB this/DT question/NN ,/, but/CC hurriedly/RB went/VBD on/IN ,/, ''/'' ''/'' --/: found/VBD it
William/NNP 's/POS conduct/NN at/IN first/JJ was/VBD moderate/JJ ./.
But/CC the/DT insolence/NN of/IN his/PRP$ Normans/NNPS --/: ''/'' How/WRB are/VBP you/PRP getting/VBG on/IN now/RB ,/, my/PRP$ dear/J it/PRP
continued/VBD ,/, turning/VBG to/TO Alice/NNP as/IN it/PRP spoke/VBD ./.