[Lab 6]

Name: Nguyễn Thành Trung – MSSV: 19522431

Link github: [Data\_Mining/Week6 at main · Shu2301/Data\_Mining (github.com)](https://github.com/Shu2301/Data_Mining/tree/main/Week6)

## 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

**1** 848988730585096192 2017-04-03

20:01:01

**2** 848943072423497728 2017-04-03

16:59:35

2017-04-03

b'And so the robots spared humanity ... https:... b"@ForIn2020 @waltmossberg @mims @defcon\_5

Exa... b'@waltmossberg @mims @defcon\_5 Et tu, Walt?'

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

[idf[key] for key in vocabulary[:5]]

3.3163095197385393,

3.7262581423445837,

3.8171115727956972,

3.8027562798186274]

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]))

# 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. h [ttps://stackoverflow.com/questions/36182502/add-stemming-support-to-countvectorizer-sklearn](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

 

# 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]])

Top-5 documents

query = "tesla nasa"

stemmer = PorterStemmer()

search\_vec(query, 5, vocabulary, stemmer, document\_vectors, original\_documents

* + 1. b'@ashwin7002 @NASA @faa @AFPAA We have not ruled that out.'
    2. b'RT @NASA: Updated @SpaceX #Dragon #ISS rendezvous times: NASA TV coverage begins Sunday at 3:30amET: [h ttp://t.co/qrm0Dz4jPE](http://t.co/qrm0Dz4jPE). Grappl
    3. b"Deeply appreciate @NASA's faith in @SpaceX. We will do whatever it takes to make NASA and the American people proud."
    4. b'Would also like to congratulate @Boeing, fellow winner of the @NASA commercial crew program'
    5. 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

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

 

## 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

* 1. Coldplay
  2. Coldplay

Another's /c/coldplay/anothers+arms\_21079526.html Arms

Bigger /c/coldplay/bigger+stronger\_20032648.html Stronger

Late night watching tv

\nUsed to be you here

...

I want to be bigger stronger drive a faster

ca...

* 1. Coldplay Daylight /c/coldplay/daylight\_20032625.html

To my surprise, and my delight \nI saw

sunris...

* 1. Coldplay Everglow /c/coldplay/everglow\_21104546.html

Oh, they say people come \nThey say

people go...

* 1. 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

1. Artist 120 non-null object
2. Song 120 non-null object
3. Link 120 non-null object
4. 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

Để hủy thao tác xóa ô, hãy sử dụng Ctrl+M Z hoặc tùy chọn Hủy trong trình đơn Chỉnh sửa

Oh

It 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)

Để hủy thao tác xóa ô, hãy sử dụng Ctrl+M Z hoặc tùy chọn Hủy trong trình đơn Chỉnh sửa

['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

# 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)

# 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

## 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

# TODO: Load the dataset in coldplay.csv

data = pd.read\_csv('coldplay.csv')

data.head()

### Artist Song Link Lyrics

**0** Coldplay Another's Arms /c/coldplay/anothers+arms\_21079526.html

Bigger

Late night watching tv \nUsed to be you

here ...

I want to be bigger

1. Coldplay

Stronger /c/coldplay/bigger+stronger\_20032648.html

stronger drive a

faster ca...

1. 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

1. Artist 120 non-null object
2. Song 120 non-null object
3. Link 120 non-null object
4. 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 | | aaaah | | 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 | |  |

|  |  |
| --- | --- |
| 0 | yuletide  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(a

# Sort the dataframe by word count in descending order

Để hủy thao tác xóa ô, hãy sử dụng Ctrl+M Z hoặc tùy chọn Hủy trong trình đơn Chỉnh sửa

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))

xis=0).tolist()[0]})

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 10 | 2000 | 2gether | | 76543 | aaaaaah | | aaaaah | | aaaah | | 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]

|  |  |  |
| --- | --- | --- |
| 1770 | word  you | count  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 |

## Topic Modelling

import pandas as pd

df = pd.read\_csv('random\_headlines.csv')

df.head(10)

### publish\_date headline\_text

1. 20120305 ute driver hurt in intersection crash
2. 20081128 6yo dies in cycling accident
3. 20090325 bumper olive harvest expected
4. 20100201 replica replaces northernmost sign
5. 20080225 woods targets perfect season
6. 20091120 leckie salvages dramatic draw for adelaide

**7** 20130304 anti hunting rally still going ahead

**9** 20130304 thailand signs agreement with muslim rebels

df.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 20000 entries, 0 to 19999 Data columns (total 2 columns):

# Column Non-Null Count Dtype

1. publish\_date 20000 non-null int64
2. 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)

1. df['no\_punctuation'] = df['tokens'].apply(lambda tokens: [token for token in tokens if token not in string.punctuation])
2. stopwords\_set = set(stopwords.words('english'))
3. 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](https://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/lib/nltk\_data'

- '/usr/local/lib/nltk\_data'

- ''

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*](https://pypi.org/simple)

SEARCH STACK OVERFLOW

df['stemmed']

- '/usr/local/share/nltk\_data'

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))

## 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

first ORDINAL

Lovelace PERSON

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

mathematician and writer, chiefly known for her work on Charles Babbage's **ORG** proposed mechanical general-purpose computer, the Analytical Engine **ORG** . She was the first **ORDINAL** to recognise that the machine had applications beyond pure calculation, and published the first **ORDINAL** algorithm

intended to be carried out by such a machine. As a result, she is sometimes regarded as the first **ORDINAL** to recognise the full potential of a "computing machine" and one **CARDINAL** of the first **ORDINAL** computer programmers.

 

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")

contents = file.read()

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)

make\_doc\_GDPR\_compliant(adafile)

## Exercise

jobmarket = "job-market.csv"

jobs\_df = pd.read\_csv("job-market.csv") jobs\_df.fillna(0)

### Id Title Company Date Location Area C

* 1. 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 - ...

**3** 37404330.0 - Belrose Group Pty Ltd 07T00:00:00.000Z Central 0

Coast

Business Commonwealth

Banking Bank - Ryde &

**4** 37404308.0 Contact Business & 2018-10- Sydney Macquarie Centre Private 07T00:00:00.000Z Park

Specialist, Ni... Banking

LB Creative Pty

Ltd

2018-10- Brisbane 07T00:00:00.000Z

CBD &

Inner Suburbs

import pandas as pd

Team member Anaconda 2018-10- Gosford &

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)]

**40784** 0.0 0 0 0 0 0

**40786** 0.0 0 0 0 0 0

**40787** 0.0 0 0 0 0 0

**40788** 0.0 0 0 0 0 0

40789 rows × 13 columns

 

# Step 3: Use scikit-learn to get the top 20 important keywords

vectorizer = TfidfVectorizer(stop\_words='english', max\_features=1000)

favorite\_keyword = 'python'

# Construct a query using the favorite keyword

query = vectorizer.transform([favorite\_keyword])

# 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: 0.0027987719273321284

Job Description: Human Resources & Recruitment

def extract\_ngrams(sequence, n): ngrams = []

Job Description: Information & Communication Technology

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']

['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 ./.



 