Prediction Competition 6: Text Analysis, Economic News and ML Research

Anonymized name: KentoNanami

Q1

The assignment aims to classify which of the 15,578 news article text snippets contain words related to “economic”, “economy”, “economics”, or the character sequence “econom”. Prior to importing the dataset, adjustments were made including adding a 'text' header, resolving duplicate issues caused by text starting with "=–", and removing problematic symbols like '<' which affected HTML interpretation. After ensuring data integrity, NLP functions were tested and a cleaning function was developed. The Bag of Words (BoW) method was then used to represent text as word occurrences, followed by TF-IDF to evaluate word importance relative to a corpus. Subsequently, a neural network with batch normalization and two hidden layers was constructed, utilizing 'relu' activation function. Output layer with 'softmax' activation was employed for binary classification. Assessing predictions involved creating a confusion matrix and monitoring loss graphs. To address misclassification issues, a threshold based on neural network output probability was implemented and fine-tuned.

Q2

Figure 1: A wordcloud of most common words scaled by size in the text data with a word containing 'econom'

Figure 2: A wordcloud of most common words scaled by size in the text data without a word containing 'econom'

Q3

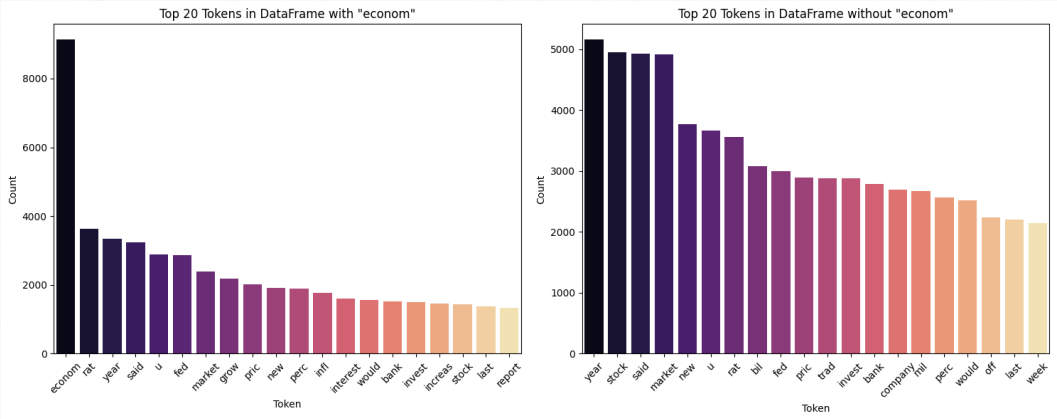
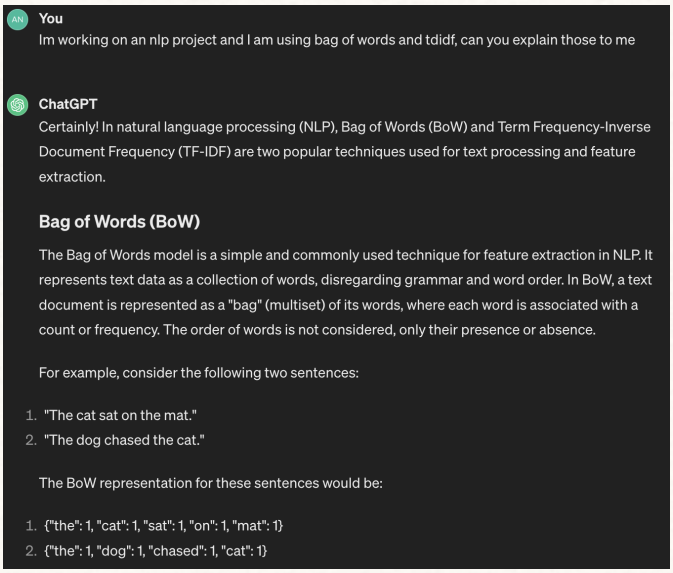


Figure 3: A multigraph plot of the counts of 20 most common words in the text data with a word containing 'econom' in it vs text data without 'econom' in it.

Q4



You

Im working on an nlp project and I am using bag of words and tdidf, can you explain those to me

ChatGPT

Certainly! In natural language processing (NLP), Bag of Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF) are two popular techniques used for text processing and feature extraction.

Bag of Words (BoW)

The Bag of Words model is a simple and commonly used technique for feature extraction in NLP. It represents text data as a collection of words, disregarding grammar and word order. In BoW, a text document is represented as a "bag" (multiset) of its words, where each word is associated with a count or frequency. The order of words is not considered, only their presence or absence.

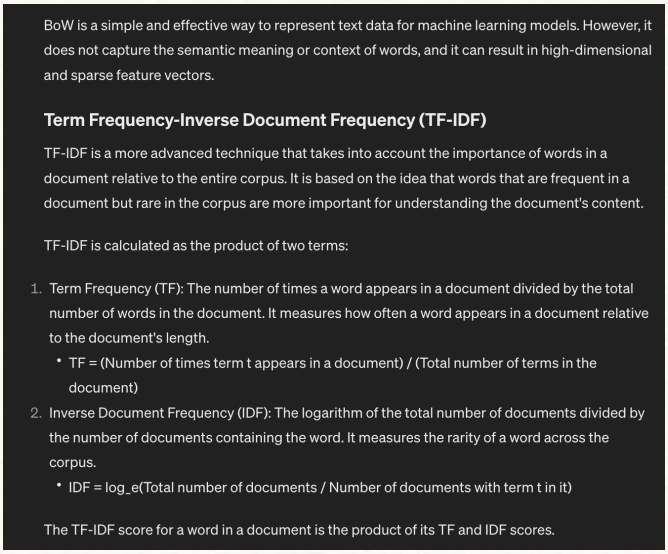
For example, consider the following two sentences:

1. "The cat sat on the mat."
2. "The dog chased the cat."

The BoW representation for these sentences would be:

1. {"the": 1, "cat": 1, "sat": 1, "on": 1, "mat": 1}
2. {"the": 1, "dog": 1, "chased": 1, "cat": 1}

Figure 4: Conversation with Chat GPT



BoW is a simple and effective way to represent text data for machine learning models. However, it does not capture the semantic meaning or context of words, and it can result in high-dimensional and sparse feature vectors.

Term Frequency-Inverse Document Frequency (TF-IDF)

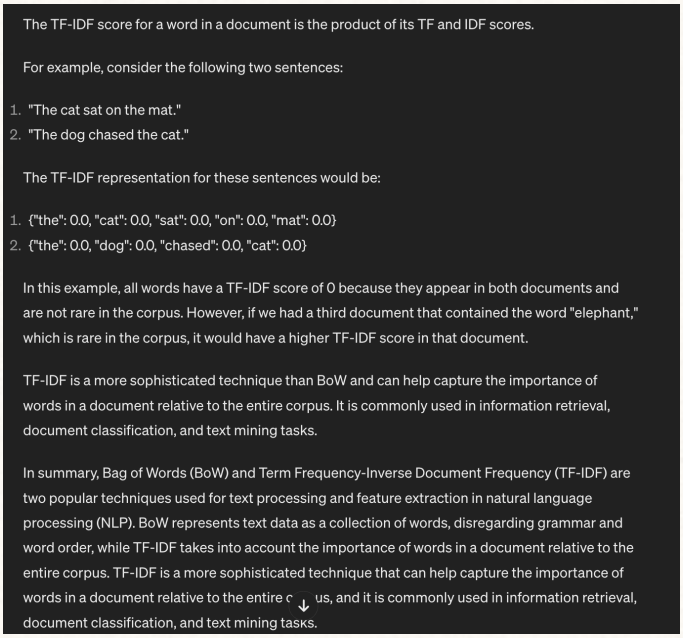
TF-IDF is a more advanced technique that takes into account the importance of words in a document relative to the entire corpus. It is based on the idea that words that are frequent in a document but rare in the corpus are more important for understanding the document's content.

TF-IDF is calculated as the product of two terms:

1. Term Frequency (TF): The number of times a word appears in a document divided by the total number of words in the document. It measures how often a word appears in a document relative to the document's length.
   - TF = (Number of times term t appears in a document) / (Total number of terms in the document)
2. Inverse Document Frequency (IDF): The logarithm of the total number of documents divided by the number of documents containing the word. It measures the rarity of a word across the corpus.
   - IDF = log_e(Total number of documents / Number of documents with term t in it)

The TF-IDF score for a word in a document is the product of its TF and IDF scores.

Figure 5: Conversation with Chat GPT



The TF-IDF score for a word in a document is the product of its TF and IDF scores.

For example, consider the following two sentences:

1. "The cat sat on the mat."
2. "The dog chased the cat."

The TF-IDF representation for these sentences would be:

1. {"the": 0.0, "cat": 0.0, "sat": 0.0, "on": 0.0, "mat": 0.0}
2. {"the": 0.0, "dog": 0.0, "chased": 0.0, "cat": 0.0}

In this example, all words have a TF-IDF score of 0 because they appear in both documents and are not rare in the corpus. However, if we had a third document that contained the word "elephant," which is rare in the corpus, it would have a higher TF-IDF score in that document.

TF-IDF is a more sophisticated technique than BoW and can help capture the importance of words in a document relative to the entire corpus. It is commonly used in information retrieval, document classification, and text mining tasks.

In summary, Bag of Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF) are two popular techniques used for text processing and feature extraction in natural language processing (NLP). BoW represents text data as a collection of words, disregarding grammar and word order, while TF-IDF takes into account the importance of words in a document relative to the entire corpus. TF-IDF is a more sophisticated technique that can help capture the importance of words in a document relative to the entire c ↓ us, and it is commonly used in information retrieval, document classification, and text mining tasks.

Figure 6: Conversation with Chat GPT

ECON 626 Prediction Competition 6 Code

Objective: To predict using classification which of the 15,578 text snippets from news articles include the words “economic”,“economy”, “economics” or the character sequence “econom” (not case-sensitive) in it.

Importing libraries

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")

from sklearn.preprocessing import LabelEncoder

import nltk
from nltk.stem import PorterStemmer
from nltk import LancasterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.metrics import edit_distance
import string
import re
from bs4 import BeautifulSoup

from wordcloud import WordCloud

from collections import Counter

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.feature_extraction.text import TfidfTransformer

from sklearn import preprocessing

from sklearn import svm
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classi

from sklearn.model_selection import GridSearchCV

from scipy.sparse import hstack

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion
```

```
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
```

Importing data

In [ ]:
```
data_path = "/Users/andrew/Downloads/UW courses/ECON 626/Prediction Competit
text_df= pd.read_csv(data_path)
#create a dataframe for our text dataset
```

Note I made two alterations to the dataset pre importing. They are listed below:

- First I added a header to the dataset called 'text'
- After searching for duplicates I found that some of the text started with "=--" which created an error with the csv after saving where some of the values turned to '#NAME'
- I deleted a '<' in line 12575 in the data since the symbol was causing an error where it believed the text inside the symbol was html text and missed an 'econom' sequence. I will look into switch from an 'anything in a <>' regex to a 'specific html commands' regex since this could cause problems in predictions.

In [ ]:
```
text_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15578 entries, 0 to 15577
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    15578 non-null  object
dtypes: object(1)
memory usage: 121.8+ KB
```

In [ ]:
```
text_df.head()
```

Out[ ]:

	text
0	CEA believe that some special factors, such as...
1	approved a $282.6 billion measure to fund the ...
2	(AP) ▒ÛÓ An unexpectedly quick rise in. the pr...
3	$17.28 million would be put into a reserve for...
4	Reserve policy makers voted to keep short-term...

Data Cleaning

In [ ]:
```
num_na = text_df.isna().sum()
num_na
```

Out[ ]:
```
text    0
dtype: int64
```

In [ ]:
```
duplicate_rows = text_df.duplicated(keep=False)

# Check if there are any duplicates
if duplicate_rows.any():
    print(text_df[duplicate_rows])
    # Print the rows that are duplicates
else:
    print("No duplicates found.")
```

```
                                                   text
506    general manager, said Silicon Valley's patent ...
1532   yet to damp innovation in Silicon Valley, at l...
1559   way in what central banks are and what they do...
3469   general manager, said Silicon Valley's patent ...
4255   bond market intensified Thursday, as the gap b...
5837   the 1990s, when government spending and taxes ...
7375   Americans to gamble that the U.S. economy can ...
7539   yet to damp innovation in Silicon Valley, at l...
8196   inflation fears on the rise, investors may wan...
8563   from the recession and the collapsed real esta...
8621   bond market intensified Thursday, as the gap b...
9100   Americans to gamble that the U.S. economy can ...
10721  way in what central banks are and what they do...
11204  on both domestic and foreign policy.</br></br>...
11591  from the recession and the collapsed real esta...
12441  inflation fears on the rise, investors may wan...
14996  on both domestic and foreign policy.</br></br>...
15460  the 1990s, when government spending and taxes ...
```

In [ ]:
```
text_df['length'] = text_df['text'].apply(len)
text_df
```

Out[ ]:

	text	length
0	CEA believe that some special factors, such as...	644
1	approved a $282.6 billion measure to fund the ...	556
2	(AP) ▒ÛÓ An unexpectedly quick rise in. the pr...	453
3	$17.28 million would be put into a reserve for...	469
4	Reserve policy makers voted to keep short-term...	608
...	...	...
15573	Grover Norquist says emphatically and repeated...	423
15574	in die economy, Laura D▒Ûªandrea Tyson, chairm...	580
15575	of living are not rising noticeably. Many of t...	833
15576	economy merely flexes some long-unused muscles...	767
15577	Video Investments, an investment-research and ...	824

15578 rows × 2 columns

Data Preprocessing

In [ ]:
```
sample = text_df['text'].iloc[12573]
sample
```

Out[ ]:
```
'a recesbave been denied their con-sion or is heading t0wardtitutional ri h
ts by such</br></br>Leading indicators -leavå£nmates 1ft ers and the little
doubt as to which way)Uyn,nfåÈ letters addressed the economy is headed intå
i inmates.</br></br>1970\x89ÛÓtoward a recession.\x89Û\x9d Also, it is char
ged that J. Gordon Gifford, editolphotostatic copies of letters of the comm
ission\x89Ûªs monthly inmates have been made report, said all six of th%nd
turned over to the attor-leading indicators were of|,ey general\x89Ûªs offi
ce, in some in January from Decembeijnstances and below the level of Janu-
\t..\t\x89Û_ ary, 1969. He said they had/fhe suit alleges prison of-been de
clining since Septemficials have infringed upon ber.'
```

In [ ]:
```
# Test on a sample text


stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

text = text_df['text'].iloc[12573]

# Remove HTML tags using BeautifulSoup
#clean_text = BeautifulSoup(text).get_text()

#clean_text = ' '.join(re.split(r'<.*?>', clean_text))


clean_text = re.sub("<.*?>",lambda m: " " * len(m.group(0))   , text)
```

```
clean_text = ''.join([char for char in text if char not in string.punctuatic


# # Replace specific emojis with corresponding text
# clean_text = re.sub(r'<3', '<heart>' , clean_text)
# clean_text = re.sub(r"[8:=;]['`\-]?[)d]+" , '<smile>' , clean_text)
# clean_text = re.sub(r"[8:=;]['`\-]?\(+" , '<sadface>' , clean_text)
# clean_text = re.sub(r"[8:=;]['`\-]?[\/|l*]" , '<neutralface>' , clean_text
# clean_text = re.sub(r"[8:=;]['`\-]?p+" , '<lolface>' , clean_text)

# # Remove non-alphabetical symbols
clean_text = re.sub('[^A-Za-z ]+', '', clean_text)

# # Remove stopwords
clean_text = ' '.join([word.lower() for word in clean_text.split() if word.l
clean_text = [stemmer.stem(lemmatizer.lemmatize(word.lower())) for word in w

print(clean_text)
```

```
['recesbav', 'deni', 'consion', 'head', 'twardtitut', 'ri', 'ht', 'suchbrbrl
ead', 'indic', 'leavnmat', 'ft', 'er', 'littl', 'doubt', 'wayuynnf', 'lette
r', 'address', 'economi', 'head', 'int', 'inmatesbrbrtoward', 'recess', 'als
o', 'charg', 'j', 'gordon', 'gifford', 'editolphotostat', 'copi', 'letter',
'commiss', 'monthli', 'inmat', 'made', 'report', 'said', 'six', 'thnd', 'tur
n', 'attorlead', 'indic', 'ofey', 'gener', 'offic', 'januari', 'decembeijns
t', 'level', 'janu', 'ari', 'said', 'hadfh', 'suit', 'alleg', 'prison', 'ofb
een', 'declin', 'sinc', 'septemfici', 'infring', 'upon', 'ber']
```

In [ ]:
```
# Create a function
def text_processing(text:str) -> list:

    clean_text = re.sub("<.*?>",lambda m: " " * len(m.group(0))   , text)
    clean_text = ''.join([char for char in clean_text if char not in string.
    clean_text = re.sub("\s+"," ",clean_text)
    clean_text = re.sub('<.*?>',' ',clean_text)
    clean_text = re.sub(r'[^\w ]+', "", clean_text)

    # Convert the text to lowercase for case-insensitive matching
    clean_text_lower = clean_text.lower()

    # Remove non-alphabetical symbols
    clean_text = re.sub('[^A-Za-z ]+', '', clean_text_lower)

    # Remove stopwords
    clean_text = ' '.join([word for word in clean_text.split() if word not i

    # Stem and lemmatize the tokens
    stemmer = LancasterStemmer()
    lemmatizer = WordNetLemmatizer()
    clean_text_tokens = [lemmatizer.lemmatize(stemmer.stem(word)) for word i

    # Define the words and character sequence to check for
    words_to_check = ["economic", "economy", "economics", "econom", "economi

    # Check if any of the words or character sequence are present in the tok
```

```
    for word in clean_text_tokens:
        for word_to_check in words_to_check:
            if edit_distance(word, word_to_check) <= 2:
                return clean_text_tokens  # Return the corrected tokens if t

    return clean_text_tokens  # Return the original tokens if none of the wo
```

In [ ]:
```
text_processing(text_df['text'].iloc[5500])
```

Out[ ]:
```
['expect',
 'giv',
 'way',
 'renew',
 'adv',
 'short',
 'turnaround',
 'com',
 'liv',
 'cost',
 'resum',
 'upward',
 'march',
 'exceiv',
 'new',
 'car',
 'pric',
 'exampl',
 'lik',
 'edg',
 'model',
 'rol',
 'produc',
 'lin',
 'deal',
 'custom',
 'shav',
 'pric',
 'model',
 'year',
 'progress',
 'year',
 'poor',
 'sal',
 'perform',
 'enco',
 'many',
 'deal',
 'giv',
 'ev',
 'big',
 'discount',
 'u',
 'word',
 'detroit',
 'model',
 'cost',
 'year',
 'car',
 'mean',
 'automobl',
 'pric',
 'jump',
 'isharply',
 'next',
 'fal']
```

In [ ]:
```
text_df['tokens'] = text_df['text'].apply(text_processing)
```

In [ ]:
```
text_df.head()
```

Out[ ]:

	text	length	tokens
0	CEA believe that some special factors, such as...	644	[cea, believ, spec, fact, hug, runup, stock, m...
1	approved a $282.6 billion measure to fund the ...	556	[approv, bil, meas, fund, nat, defens, com, fi...
2	(AP) ≣ÛÓ An unexpectedly quick rise in. the pr...	453	[ap, unexpect, quick, ri, prim, rat, many, nat...
3	$17.28 million would be put into a reserve for...	469	[mil, would, put, reserv, next, year, ev, mone...
4	Reserve policy makers voted to keep short-term...	608	[reserv, policy, mak, vot, keep, shortterm, in...

In [ ]:
```
# Define a function to check if the specified words or character sequence ar
def check_economic_presence(tokens):
    # Define the words and character sequence to check for
    words_to_check = ["economic", "economy", "economics", "econom", "economi

    # Define the regular expression pattern to search for
    pattern = re.compile(r'econom', re.IGNORECASE)

    # Check if any of the words or character sequence are present in the tok
    for word in tokens:
        if word in words_to_check or pattern.search(word):
            return 1  # Return 1 if any of the words or character sequence a

    return 0  # Return 0 if none of the words or character sequence are foun

# Apply the function to the 'tokens' column to create the 'class' column
text_df['class'] = text_df['tokens'].apply(check_economic_presence)
```

In [ ]:
```
text_df.head(25)
```

Out[ ]:

	text	length	tokens	class
0	CEA believe that some special factors, such as...	644	[cea, believ, spec, fact, hug, runup, stock, m...	1
1	approved a $282.6 billion measure to fund the ...	556	[approv, bil, meas, fund, nat, defens, com, fi...	0
2	(AP) ▒ÛÓ An unexpectedly quick rise in. the pr...	453	[ap, unexpect, quick, ri, prim, rat, many, nat...	0
3	$17.28 million would be put into a reserve for...	469	[mil, would, put, reserv, next, year, ev, mone...	0
4	Reserve policy makers voted to keep short-term...	608	[reserv, policy, mak, vot, keep, shortterm, in...	0
5	the lender asked us if we wanted to lock in.</...	824	[lend, ask, u, want, lock, sint, first, hous, ...	0
6	called off its effort to drive down the value ...	752	[cal, effort, driv, valu, doll, cur, return, a...	1
7	striking unions at. the Greenbrier Hotel resor...	421	[striking, un, greenbry, hotel, resort, whit, ...	0
8	short-term political pressure. But an agency t...	596	[shortterm, polit, press, ag, alloc, credit, s...	0
9	typical recession would drop enough to make st...	797	[typ, recess, would, drop, enough, mak, stock,...	0
10	the standard deduction, personal exemption and...	476	[standard, deduc, person, exempt, cutoff, fig,...	0
11	nation's factories continued to fall last mont...	984	[nat, fact, continu, fal, last, mon, provid, s...	1
12	percentage of those entitled to unemployment i...	971	[perc, entitl, unemploy, in, remain, per, cent...	0
13	the aid earlier than planned threatens to dimi...	1038	[aid, ear, plan, threatens, dimin, impact, lar...	0
14	oversaw the conclusion of his final General As...	729	[oversaw, conclud, fin, gen, assembl, sess, la...	1
15	and labor costs rose sharply in the second qua...	740	[lab, cost, ro, sharply, second, quart, spark,...	0
16	But companies still pay, to the tune of $1.6 b...	540	[company, stil, pay, tun, bil, last, year, pla...	0
17	securities were narrowly mixed in quiet tradin...	618	[sec, narrow, mix, quiet, trad, many, biggest,...	0
18	Japan's real gross domestic product would rise...	212	[jap, real, gross, domest, produc, would, ri, ...	1
19	company. Avon said it was ▒ÛÏnot for sale.▒Û▒<...	371	[company, avon, said, sal, eastm, kodak, drop,...	0

	text	length	tokens	class
20	nearly 18 months, the stock market has been be...	1322	[near, month, stock, market, behav, lik, on, v...	0
21	Maryland continue to decline dramatically, the...	392	[maryland, continu, declin, dram, caseload, pr...	0
22	Cornerstone Financial Partners. </br></br>That ...	584	[cornerston, fin, partn, expect, stok, releas,...	1
23	was a direct attack using the mail as a weapon...	711	[direct, attack, u, mail, weapon, mak, everybo...	0
24	The secretary of defense linked the problem di...	1466	[secret, defens, link, problem, direct, nat, s...	1

In [ ]:
```
text_df['tokens'].iloc[1770]
```

Out[ ]:
```
['yearend',
 'predict',
 'econom',
 'forecast',
 'must',
 'struggle',
 'unpleas',
 'real',
 'nobody',
 'thank',
 'correct',
 'predict',
 'bad',
 'new',
 'econom',
 'rosan',
 'cahn',
 'exampl',
 'rememb',
 'od',
 'felt',
 'el',
 'econom',
 'slug',
 'ear',
 'strong',
 'end',
 'happy',
 'predict',
 'slowdown',
 'correct']
```

In [ ]:
```
text_df['tokens'].iloc[2192]
```

Out[ ]:
```
['diff',
 'export',
 'importswa',
 'bil',
 'cur',
 'record',
 'deficit',
 'bil',
 'deficit',
 'continu',
 'decemb',
 'rat',
 'first',
 'month',
 'deficit',
 'would',
 'bil',
 'econom',
 'said',
 'yesterday',
 'surpr',
 'enorm',
 'detery',
 'trad',
 'pict',
 'said',
 'increas',
 'deficit',
 'least',
 'part',
 'due',
 'spec',
 'fact',
 'hoard',
 'import',
 'busy',
 'try',
 'beat',
 'elimin',
 'lucr',
 'tax',
 'break',
 'tax',
 'revid',
 'act',
 'went',
 'effect',
 'today',
 'tim',
 'bil',
 'deficit',
 'horrend',
 'on',
 'year',
 'said',
 'commerc',
```

```
 'undersecret',
 'robert',
 'ortn',
 'get',
 'accustom',
 'think',
 'horrend',
 'on',
 'mon']
```

In [ ]:
```
sum(text_df['class'])
```

Out[ ]:
```
5266
```

In [ ]:
```
df_class_0 = text_df[text_df['class'] == 0]
```

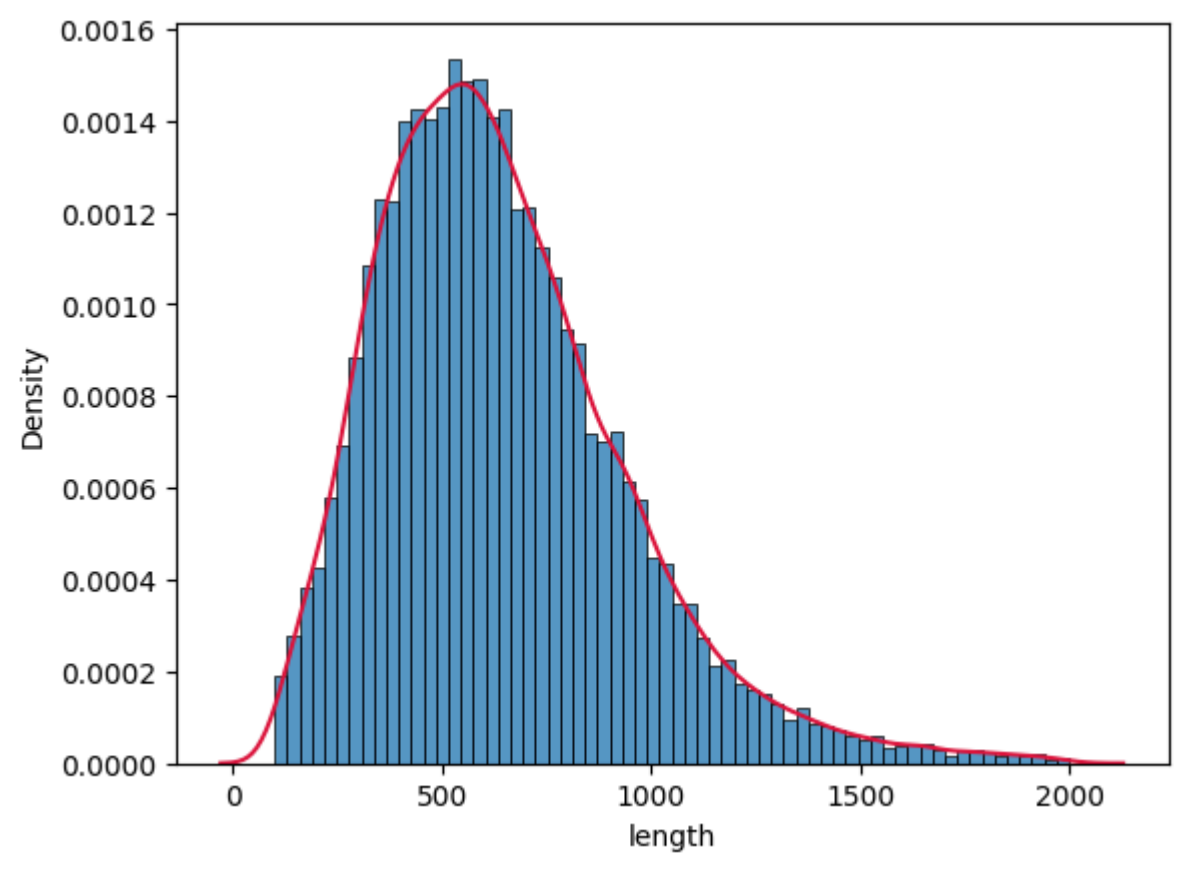
In [ ]:
```
df_class_0['tokens'].head(25)
```

Out[ ]:
```
1     [approv, bil, meas, fund, nat, defens, com, fi...
2     [ap, unexpect, quick, ri, prim, rat, many, nat...
3     [mil, would, put, reserv, next, year, ev, mone...
4     [reserv, policy, mak, vot, keep, shortterm, in...
5     [lend, ask, u, want, lock, sint, first, hous, ...
7     [striking, un, greenbry, hotel, resort, whit, ...
8     [shortterm, polit, press, ag, alloc, credit, s...
9     [typ, recess, would, drop, enough, mak, stock,...
10    [standard, deduc, person, exempt, cutoff, fig,...
12    [perc, entitl, unemploy, in, remain, per, cent...
13    [aid, ear, plan, threatens, dimin, impact, lar...
15    [lab, cost, ro, sharply, second, quart, spark,...
16    [company, stil, pay, tun, bil, last, year, pla...
17    [sec, narrow, mix, quiet, trad, many, biggest,...
19    [company, avon, said, sal, eastm, kodak, drop,...
20    [near, month, stock, market, behav, lik, on, v...
21    [maryland, continu, declin, dram, caseload, pr...
23    [direct, attack, u, mail, weapon, mak, everybo...
25    [mon, fed, most, sharp, increas, grain, pric, ...
26    [recess, already, four, year, spark, gre, soc,...
27    [sharehold, scoreboard, maj, play, industry, s...
29    [int, yesterday, timihc, sel, worthington, dia...
32    [respond, presid, clinton, healthc, propos, bl...
34    [many, expect, increas, risktak, trad, op, cat...
36    [reach, realm, islam, law, just, out, prospect...
Name: tokens, dtype: object
```

Data Visualization

In [ ]:
```
ax = sns.histplot(text_df['length'], kde=False, stat='density' )
sns.kdeplot(text_df['length'], color='crimson', ax=ax)
```

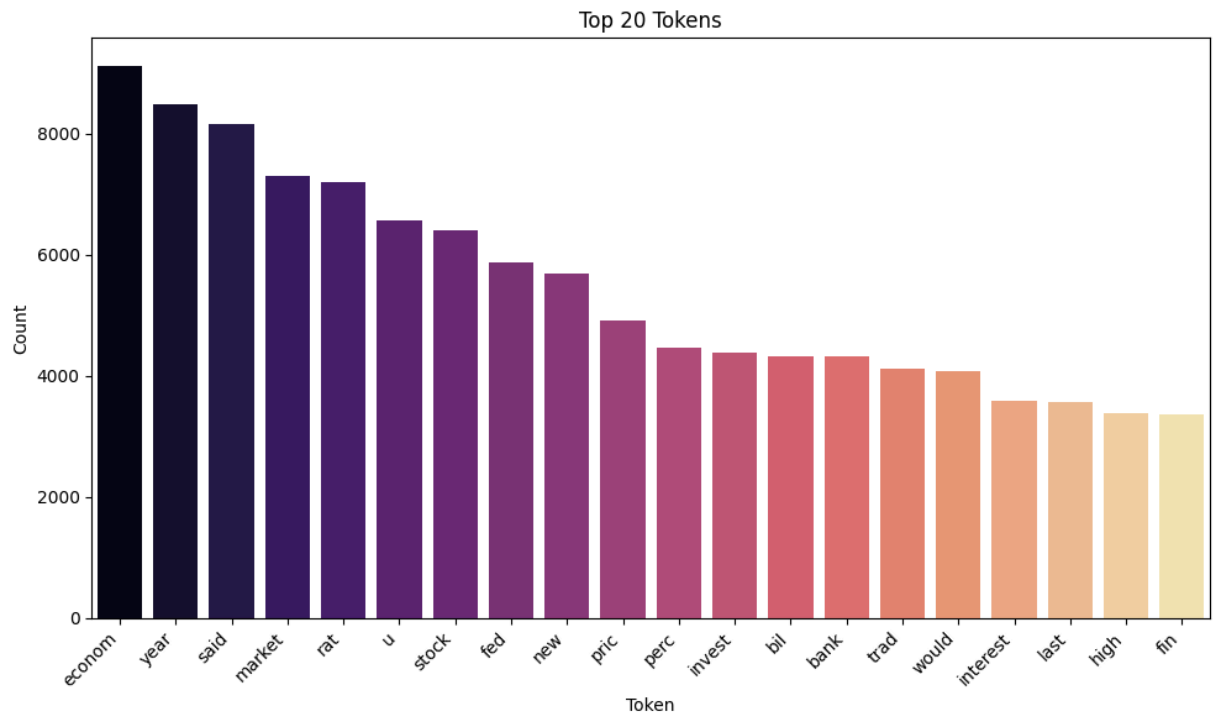
Out[ ]:
```
<Axes: xlabel='length', ylabel='Density'>
```



In [ ]:
```
all_tokens = []

for token_list in text_df['tokens']:
    for word in token_list:
        all_tokens.append(word)

# Count occurrences of each token
token_counts = Counter(all_tokens)
top_words = pd.DataFrame(token_counts.most_common(20), columns = ['word','fr

print(top_words)
```

	word	freq
0	econom	9129
1	year	8497
2	said	8166
3	market	7298
4	rat	7193
5	u	6567
6	stock	6399
7	fed	5866
8	new	5689
9	pric	4922
10	perc	4456
11	invest	4379
12	bil	4322
13	bank	4314
14	trad	4128
15	would	4088
16	interest	3594
17	last	3577
18	high	3388
19	fin	3371

In [ ]:
```
top_tokens = dict(token_counts.most_common(20))
# Plot the top 20 tokens
plt.figure(figsize=(10, 6))
ax = sns.barplot(top_words, x = 'word' , y='freq' , palette='magma')
plt.xlabel('Token')
plt.ylabel('Count')
plt.title('Top 20 Tokens')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



In [ ]:
```
all_econ_tokens = []

econ_df = text_df.loc[text_df['class'] == 1]

for token_list in econ_df['tokens']:
    for word in token_list:
        all_econ_tokens.append(word)

# Count occurrences of each token
econ_token_counts = Counter(all_econ_tokens)

top_econ_tokens_tup = econ_token_counts.most_common(50)

top_econ_tokens_list = [token for token, _ in econ_token_counts.most_common(

# Print the top 50 tokens and their counts
for token, count in econ_token_counts.most_common(50):
    print(f"{token}: {count}")
```

```
econom: 9129
rat: 3630
year: 3338
said: 3234
u: 2896
fed: 2867
market: 2384
grow: 2177
pric: 2024
new: 1917
perc: 1893
infl: 1760
interest: 1601
would: 1562
bank: 1528
invest: 1497
increas: 1450
stock: 1446
last: 1377
report: 1338
high: 1334
trad: 1245
bil: 1240
fin: 1238
nat: 1228
govern: 1201
week: 1157
stat: 1154
presid: 1125
low: 1118
tax: 1103
expect: 1084
off: 1035
produc: 1017
doll: 1007
ev: 1006
ri: 1005
say: 999
recess: 986
reserv: 985
point: 970
consum: 960
month: 960
policy: 958
tim: 950
on: 935
may: 933
could: 933
spend: 927
sint: 918
```

In [ ]:
```
def word_cloud(tokens):
    #Creation of wordcloud
    wordcloud = WordCloud(
        max_font_size=100,
        max_words=50,
```

```
        background_color="white",
        scale=10,
        width=500,
        height=200
    ).generate(" ".join(top_econ_tokens_list))
    #Figure properties
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    plt.show()

word_cloud(top_econ_tokens_list)
```

In [ ]:
```
all_nonecon_tokens = []

nonecon_df = text_df.loc[text_df['class'] == 0]

for token_list in nonecon_df['tokens']:
    for word in token_list:
        all_nonecon_tokens.append(word)

# Count occurrences of each token
nonecon_token_counts = Counter(all_nonecon_tokens)

top_nonecon_tokens_tup = nonecon_token_counts.most_common(50)

top_nonecon_tokens_list = [token for token, _ in nonecon_token_counts.most_c

# Print the top 50 tokens and their counts
for token, count in nonecon_token_counts.most_common(50):
    print(f"{token}: {count}")
```

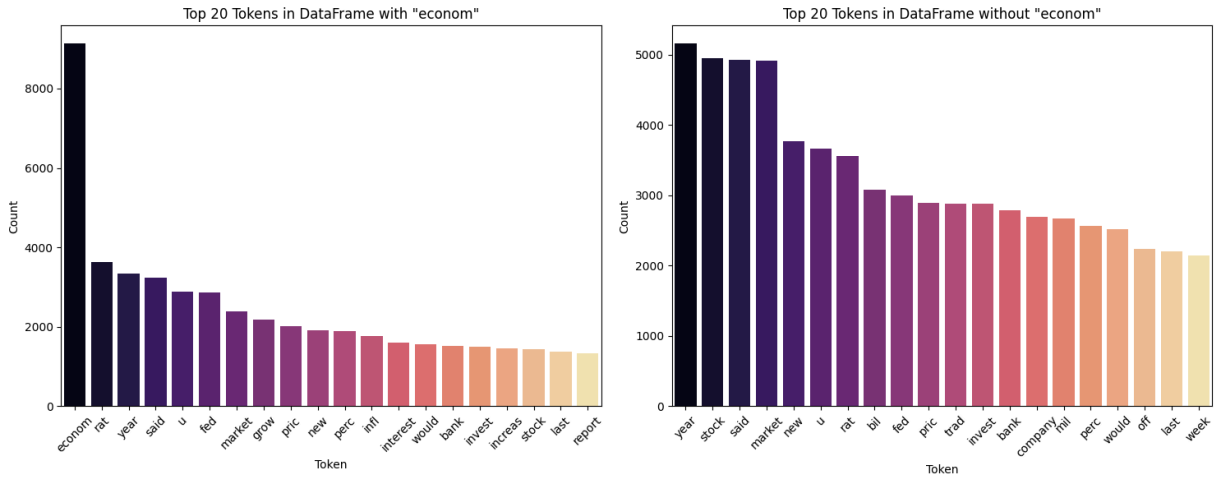
```
year: 5159
stock: 4953
said: 4932
market: 4914
new: 3772
u: 3671
rat: 3563
bil: 3082
fed: 2999
pric: 2898
trad: 2883
invest: 2882
bank: 2786
company: 2694
mil: 2668
perc: 2563
would: 2526
off: 2242
last: 2200
week: 2145
fin: 2133
high: 2054
interest: 1993
point: 1939
increas: 1892
shar: 1865
on: 1834
stat: 1797
cent: 1749
av: 1745
fund: 1700
yesterday: 1575
govern: 1571
report: 1566
tim: 1540
work: 1537
say: 1527
presid: 1480
day: 1467
mak: 1457
nat: 1456
first: 1450
bond: 1444
clos: 1436
low: 1425
index: 1411
doll: 1408
gain: 1403
also: 1397
ev: 1384
```

In [ ]:
```
def word_cloud(tokens):
    #Creation of wordcloud
    wordcloud = WordCloud(
        max_font_size=100,
        max_words=50,
```

```
        background_color="black",
        scale=10,
        width=500,
        height=200
    ).generate(" ".join(top_nonecon_tokens_list))
    #Figure properties
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    plt.show()

word_cloud(top_nonecon_tokens_list)
```

In [ ]:
```
top_econ_words_df = pd.DataFrame(econ_token_counts.most_common(20), columns

top_nonecon_words_df = pd.DataFrame(nonecon_token_counts.most_common(20), cc
```

In [ ]:
```
# Create subplots
fig, axs = plt.subplots(1, 2, figsize=(15, 6))

# Plot top tokens in DataFrame with 'econom'
sns.barplot(data=top_econ_words_df, x='word', y='freq', ax=axs[0], palette='
axs[0].set_title('Top 20 Tokens in DataFrame with "econom"')
axs[0].set_xlabel('Token')
axs[0].set_ylabel('Count')
axs[0].tick_params(axis='x', labelrotation=45)

# Plot top tokens in DataFrame without 'econom'
sns.barplot(data=top_nonecon_words_df, x='word', y='freq', ax=axs[1], palett
axs[1].set_title('Top 20 Tokens in DataFrame without "econom"')
axs[1].set_xlabel('Token')
axs[1].set_ylabel('Count')
axs[1].tick_params(axis='x', labelrotation=45)

# Adjust layout
plt.tight_layout()

# Show plot
plt.show()
```



Neural Network

Splitting and transforming the data

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(text_df[['text']], text_
```

In [ ]:
```
bow_transformer = CountVectorizer(analyzer=text_processing).fit(X_train['tex
```

In [ ]:
```
X_train_bow = bow_transformer.transform(X_train['text'])
X_test_bow = bow_transformer.transform(X_test['text'])
```

In [ ]:
```
print('Shape of Sparse Matrix: ', X_train_bow.shape, X_test_bow.shape)
print('Amount of Non-Zero occurences: ', X_train_bow.nnz,  X_test_bow.nnz)
```

```
Shape of Sparse Matrix:  (10904, 23875) (4674, 23875)
Amount of Non-Zero occurences:  537100 226174
```

In [ ]:
```
max_abs_scaler = preprocessing.MaxAbsScaler()
```

In [ ]:
```
# max_abs_scaler.fit(X_train_bow)
# # transform
# X_train_bow_scaled = max_abs_scaler.transform(X_train_bow)
# X_test_bow_scaled = max_abs_scaler.transform(X_test_bow)
```

In [ ]:
```
X_train_bow_scaled_array = X_train_bow.toarray()
X_test_bow_scaled_array = X_test_bow.toarray()
#X_train_tfidf_array = X_train_tfidf.toarray()
```

In [ ]:
```
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer().fit(X_train_bow)
```

In [ ]:
```
X_train_tfidf = tfidf_transformer.transform(X_train_bow)
X_test_tfidf = tfidf_transformer.transform(X_test_bow)
```

In [ ]:
```
X_train_tfidf_array = X_train_tfidf.toarray()
X_test_tfidf_array = X_test_tfidf.toarray()
```

In [ ]:
```
X_train_bow_scaled_array.shape
```

Out[ ]: (10904, 23875)

In [ ]:
```
y_train.shape
```

Out[ ]: (10904,)

In [ ]:
```
y_train_categorical = to_categorical(y_train, num_classes=2)
y_test_categorical = to_categorical(y_test, num_classes=2)
```

In [ ]:
```
y_train_categorical.shape
```

Out[ ]: (10904, 2)

In [ ]:
```
from tensorflow.keras.layers import BatchNormalization

nrows, nfeat = X_test_tfidf.shape

model = Sequential()
model.add(Input(shape=(nfeat,)))
model.add(BatchNormalization())
model.add(Dense(50, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(2, activation='softmax'))
model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Par
batch_normalization_1 (BatchNormalization)	(None, 23875)	95
dense_8 (Dense)	(None, 50)	1,193
dense_9 (Dense)	(None, 50)	2
dense_10 (Dense)	(None, 2)	

Total params: 1,291,952 (4.93 MB)

Trainable params: 1,244,202 (4.75 MB)

Non-trainable params: 47,750 (186.52 KB)

In [ ]:
```
#optimizer = tf.keras.optimizers.legacy.Adam(learning_rate=0.001)
model.compile(loss="binary_crossentropy",
              optimizer='adam',
              metrics=['accuracy'])
```

In [ ]:
```
model_history = model.fit(X_train_tfidf_array, y_train_categorical, epochs=5
                          validation_split=.3,
```

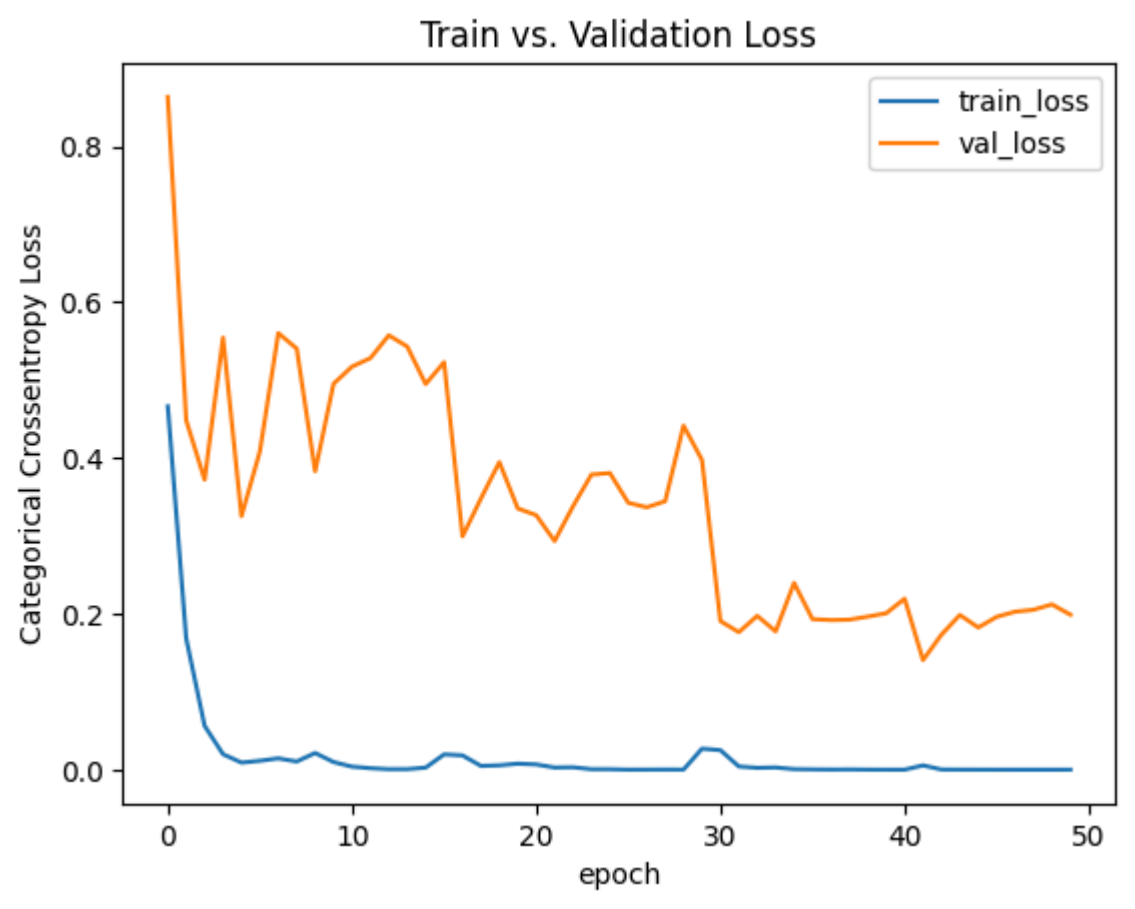
```
                                batch_size=64
                                )
```

```
Epoch 1/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.7152 - loss: 0.5345
- val_accuracy: 0.6537 - val_loss: 0.8633
Epoch 2/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9251 - loss: 0.1832
- val_accuracy: 0.8817 - val_loss: 0.4484
Epoch 3/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9802 - loss: 0.0676
- val_accuracy: 0.8322 - val_loss: 0.3722
Epoch 4/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9946 - loss: 0.0197
- val_accuracy: 0.8295 - val_loss: 0.5545
Epoch 5/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9990 - loss: 0.0069
- val_accuracy: 0.8884 - val_loss: 0.3254
Epoch 6/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9983 - loss: 0.0065
- val_accuracy: 0.8854 - val_loss: 0.4085
Epoch 7/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9957 - loss: 0.0175
- val_accuracy: 0.8848 - val_loss: 0.5603
Epoch 8/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9974 - loss: 0.0078
- val_accuracy: 0.8640 - val_loss: 0.5404
Epoch 9/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9921 - loss: 0.0251
- val_accuracy: 0.8933 - val_loss: 0.3830
Epoch 10/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9958 - loss: 0.0164
- val_accuracy: 0.8927 - val_loss: 0.4953
Epoch 11/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9990 - loss: 0.0036
- val_accuracy: 0.8933 - val_loss: 0.5173
Epoch 12/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9987 - loss: 0.0030
- val_accuracy: 0.8955 - val_loss: 0.5282
Epoch 13/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 1.0000 - loss: 8.8914e
-04 - val_accuracy: 0.8973 - val_loss: 0.5575
Epoch 14/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9997 - loss: 8.1162e
-04 - val_accuracy: 0.9007 - val_loss: 0.5430
Epoch 15/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9997 - loss: 0.0012
- val_accuracy: 0.8961 - val_loss: 0.4950
Epoch 16/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9955 - loss: 0.0136
- val_accuracy: 0.9089 - val_loss: 0.5229
Epoch 17/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9936 - loss: 0.0182
- val_accuracy: 0.9163 - val_loss: 0.2994
Epoch 18/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9979 - loss: 0.0051
- val_accuracy: 0.9224 - val_loss: 0.3480
Epoch 19/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9994 - loss: 0.0020
```

```
- val_accuracy: 0.9215 - val_loss: 0.3947
Epoch 20/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9976 - loss: 0.0071
- val_accuracy: 0.9282 - val_loss: 0.3352
Epoch 21/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9987 - loss: 0.0072
- val_accuracy: 0.9294 - val_loss: 0.3267
Epoch 22/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9991 - loss: 0.0020
- val_accuracy: 0.9325 - val_loss: 0.2934
Epoch 23/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9983 - loss: 0.0051
- val_accuracy: 0.9334 - val_loss: 0.3381
Epoch 24/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9999 - loss: 4.8839e
-04 - val_accuracy: 0.9334 - val_loss: 0.3788
Epoch 25/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9996 - loss: 0.0010
- val_accuracy: 0.9370 - val_loss: 0.3807
Epoch 26/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 1.0000 - loss: 7.9415e
-05 - val_accuracy: 0.9383 - val_loss: 0.3425
Epoch 27/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 1.0000 - loss: 1.1598e
-04 - val_accuracy: 0.9389 - val_loss: 0.3367
Epoch 28/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9999 - loss: 1.4333e
-04 - val_accuracy: 0.9401 - val_loss: 0.3445
Epoch 29/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 1.0000 - loss: 3.4319e
-05 - val_accuracy: 0.9325 - val_loss: 0.4415
Epoch 30/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9903 - loss: 0.0282
- val_accuracy: 0.9322 - val_loss: 0.3975
Epoch 31/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9856 - loss: 0.0457
- val_accuracy: 0.9535 - val_loss: 0.1908
Epoch 32/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9992 - loss: 0.0024
- val_accuracy: 0.9557 - val_loss: 0.1765
Epoch 33/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9991 - loss: 0.0029
- val_accuracy: 0.9575 - val_loss: 0.1974
Epoch 34/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9984 - loss: 0.0041
- val_accuracy: 0.9578 - val_loss: 0.1775
Epoch 35/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 1.0000 - loss: 7.9538e
-04 - val_accuracy: 0.9545 - val_loss: 0.2396
Epoch 36/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9999 - loss: 3.7480e
-04 - val_accuracy: 0.9578 - val_loss: 0.1932
Epoch 37/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 1.0000 - loss: 1.8499e
-04 - val_accuracy: 0.9584 - val_loss: 0.1921
Epoch 38/50
```

```
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9999 - loss: 2.4668e
-04 - val_accuracy: 0.9584 - val_loss: 0.1925
Epoch 39/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 1.0000 - loss: 1.0866e
-04 - val_accuracy: 0.9587 - val_loss: 0.1965
Epoch 40/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 1.0000 - loss: 3.5893e
-05 - val_accuracy: 0.9587 - val_loss: 0.2007
Epoch 41/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 1.0000 - loss: 2.7653e
-05 - val_accuracy: 0.9578 - val_loss: 0.2194
Epoch 42/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9987 - loss: 0.0034
- val_accuracy: 0.9572 - val_loss: 0.1407
Epoch 43/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 1.0000 - loss: 3.5677e
-04 - val_accuracy: 0.9575 - val_loss: 0.1733
Epoch 44/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 1.0000 - loss: 1.0776e
-04 - val_accuracy: 0.9575 - val_loss: 0.1986
Epoch 45/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 1.0000 - loss: 4.1310e
-05 - val_accuracy: 0.9584 - val_loss: 0.1825
Epoch 46/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 1.0000 - loss: 5.3241e
-05 - val_accuracy: 0.9578 - val_loss: 0.1962
Epoch 47/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 1.0000 - loss: 2.3470e
-05 - val_accuracy: 0.9587 - val_loss: 0.2028
Epoch 48/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 1.0000 - loss: 1.2902e
-05 - val_accuracy: 0.9587 - val_loss: 0.2052
Epoch 49/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 1.0000 - loss: 3.0316e
-05 - val_accuracy: 0.9590 - val_loss: 0.2121
Epoch 50/50
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 1.0000 - loss: 4.9945e
-05 - val_accuracy: 0.9597 - val_loss: 0.1989
```

In [ ]:
```
plt.plot(model_history.history['loss'], label='train_loss')
plt.plot(model_history.history['val_loss'], label='val_loss')
plt.legend()
plt.title('Train vs. Validation Loss')
plt.xlabel('epoch')
plt.ylabel('Categorical Crossentropy Loss')
plt.show()
```



In [ ]:
```
# Make predictions using your trained neural network
nn_y_pred = model.predict(X_test_tfidf_array)

# Define an initial threshold
threshold = 0.5

# Calculate the proportion of predicted 1s
proportion_1s = np.mean(nn_y_pred[:, 1])

# Adjust the threshold based on the proportion of 1s
if proportion_1s > 0.5:
    threshold -= 0.05  # Decrease the threshold if there are too many 1s
else:
    threshold += 0.05  # Increase the threshold if there are too few 1s

# Threshold the probabilities to get the predicted classes
y_pred_int = (nn_y_pred[:, 1] > threshold).astype(int)

# Now you can evaluate the predictions as usual
```

```
147/147 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

In [ ]:
```
len(y_pred_int)
```

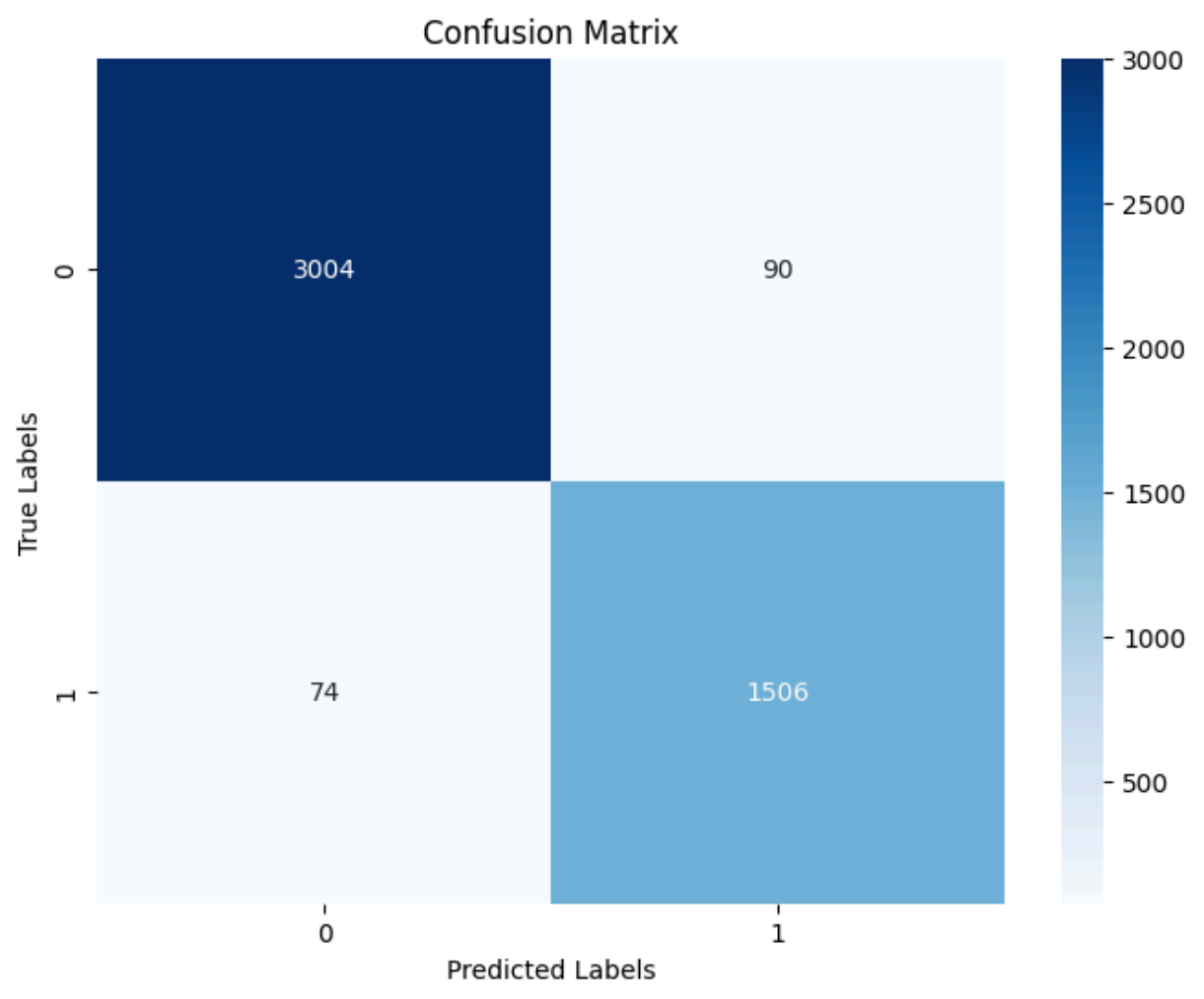
Out[ ]: 4674

In [ ]:
```
# y_pred_int = np.argmax(nn_y_pred, axis=1)
```

In [ ]:
```
report = classification_report(y_test, y_pred_int)
print("Classification Report:\n", report)
```

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

           0       0.98      0.97      0.97      3094
           1       0.94      0.95      0.95      1580

    accuracy                           0.96      4674
   macro avg       0.96      0.96      0.96      4674
weighted avg       0.97      0.96      0.96      4674
```

In [ ]:
```
cm = confusion_matrix(y_test, y_pred_int)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```



In [ ]:
```
y_pred_int
```

Out[ ]:
```
array([0, 0, 1, ..., 1, 0, 1])
```

In [ ]:
```
text_bow_transformer = CountVectorizer(analyzer=text_processing).fit(text_df
```

In [ ]:
```
text_bow = bow_transformer.transform(text_df['text'])
```

In [ ]:
```
max_abs_scaler.fit(text_bow)

# text_bow_scaled = max_abs_scaler.transform(text_bow)
```

Out[ ]:
```
▾ MaxAbsScaler
MaxAbsScaler()
```

In [ ]:
```
text_tfidf_transformer = TfidfTransformer().fit(text_bow)
```

In [ ]:
```
text_tfidf = tfidf_transformer.transform(text_bow)
```

In [ ]:
```
text_tfidf_array = text_tfidf.toarray()
```

In [ ]:
```
nn_class_pred = model.predict(text_tfidf_array)
```

```
487/487 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

In [ ]:
```
len(nn_class_pred)
```

Out[ ]:
```
15578
```

In [ ]:
```
# Calculate the predictions based on the threshold directly from nn_class_pr
prediction_unseen_classes = [1 if prob[1] > 0.00000000000672 else 0 for prob

# No need to use np.ravel() here
```

In [ ]:
```
len(prediction_unseen_classes)
```

Out[ ]:
```
15578
```

In [ ]:
```
class_pred_int = np.argmax(nn_class_pred, axis=1)
```

In [ ]:
```
text_df['predictions'] = prediction_unseen_classes
```

In [ ]:
```
sum(text_df['predictions'])
```

Out[ ]:
```
7765
```

In [ ]:
```
import pandas as pd

# Assuming text_df is your DataFrame
filtered_df = text_df[(text_df['class'] == 1) & (text_df['predictions'] == 0
```

```
# Display the filtered DataFrame
print(filtered_df)
```

```
                                                   text  length  \
569    stocks rose slightly today and the ; broader m...     501
1072   BEACH, Fin., Feb. 15□ÛÓIf there□Ûªs a recessio...     722
2617   to offset the reported deficit, but said the m...     571
2795   are completely burned. Only odorless vapors es...    1251
3438   of companies doing business in South Africa do...     819
3642   Columd high level positions in local bia Commi...     970
4332   offering low prices. The legislation's support...    1799
5838   funds, index funds received about 25 percent o...     575
9214   at the three leading U.S. auto executives yest...     734
9929   Ways and Means Committee will introduce its ec...     723
11141  to lending to low- and moderate-income groups....     678
12362  from secretaries' salaries to research -- what...     826
14348  falling to C$141 million from C$323 million.</...     331
14630  have been introduced in the Senate. Here's a c...     394

                                                  tokens  class  predictions
569    [stock, ro, slight, today, broad, market, mix,...      1            0
1072   [beach, fin, feb, ther, recess, yet, com, litt...      1            0
2617   [offset, report, deficit, said, money, repres,...      1            0
2795   [complet, burn, odorless, vap, escap, flu, aft...      1            0
3438   [company, busy, sou, afric, doesnt, seem, driv...      1            0
3642   [columd, high, level, posit, loc, bia, commit,...      1            0
4332   [off, low, pric, legisl, support, said, new, l...      1            0
5838   [fund, index, fund, receiv, perc, inflow, deva...      1            0
9214   [three, lead, u, auto, execut, yesterday, figh...      1            0
9929   [way, mean, commit, introduc, economicstimul, ...      1            0
11141  [lend, low, moderateincom, group, stil, acquis...      1            0
12362  [secret, sal, research, known, budget, jargon,...      1            0
14348  [fal, c, mil, c, mil, stil, bmo, remain, optim...      1            0
14630  [introduc, sen, her, comparison, provid, econo...      1            0
```

In [ ]:
```
# Assuming text_df is your DataFrame
text_df.loc[(text_df['class'] == 1) & (text_df['predictions'] == 0), 'predic
```

In [ ]:
```
sum(text_df['predictions'])
```

Out[ ]:
```
7779
```

In [ ]:
```
preds = text_df['predictions']

predictions_df = pd.DataFrame({'predictions': preds})

header = pd.DataFrame({
    'predictions': [21108082, 'KentoNanami']
})

header

output_df = pd.concat([header, predictions_df], axis=0)

# Specify the path where you want to save the CSV file
output_csv_path = "/Users/andrew/Downloads/UW courses/ECON 626/Prediction Co
```

```
# Write the combined DataFrame to a CSV file
output_df.to_csv(output_csv_path, index=False, header=False)
```

In [ ]:
```
preds
```

Out[ ]:
```
0        1
1        0
2        1
3        0
4        1
        ..
15573    1
15574    1
15575    1
15576    1
15577    1
Name: predictions, Length: 15578, dtype: int64
```

In [ ]:
```

```