```
1 !wget https://github.com/hjysam/text_classification/raw/main/data/reuters21578.tar
2 !tar -xvf reuters21578.tar
```

```
--2023-12-18 19:47:06--  https://github.com/hjysam/text_classification/raw/main/data/reuters21578.tar
Resolving github.com (github.com)... 140.82.112.4
Connecting to github.com (github.com)|140.82.112.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/hjysam/text_classification/main/data/reuters21578.tar [following]
--2023-12-18 19:47:06--  https://raw.githubusercontent.com/hjysam/text_classification/main/data/reuters21578.tar
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 28016640 (27M) [application/octet-stream]
Saving to: ‘reuters21578.tar’

reuters21578.tar    100%[===================>]  26.72M  --.-KB/s    in 0.1s

2023-12-18 19:47:08 (217 MB/s) - ‘reuters21578.tar’ saved [28016640/28016640]

README.txt
all-exchanges-strings.lc.txt
all-orgs-strings.lc.txt
all-people-strings.lc.txt
all-places-strings.lc.txt
all-topics-strings.lc.txt
cat-descriptions_120396.txt
feldman-cia-worldfactbook-data.txt
lewis.dtd
reut2-000.sgm
reut2-001.sgm
reut2-002.sgm
reut2-003.sgm
reut2-004.sgm
reut2-005.sgm
reut2-006.sgm
reut2-007.sgm
reut2-008.sgm
reut2-009.sgm
reut2-010.sgm
```

```
reut2-011.sgm
reut2-012.sgm
reut2-013.sgm
reut2-014.sgm
reut2-015.sgm
reut2-016.sgm
reut2-017.sgm
reut2-018.sgm
reut2-019.sgm
reut2-020.sgm
reut2-021.sgm
```

```
1 !ls
```

```
all-exchanges-strings.lc.txt        lewis.dtd      reut2-005.sgm  reut2-012.sgm  reut2-019.sgm
all-orgs-strings.lc.txt             README.txt     reut2-006.sgm  reut2-013.sgm  reut2-020.sgm
all-people-strings.lc.txt           reut2-000.sgm  reut2-007.sgm  reut2-014.sgm  reut2-021.sgm
all-places-strings.lc.txt           reut2-001.sgm  reut2-008.sgm  reut2-015.sgm  reuters21578.tar
all-topics-strings.lc.txt           reut2-002.sgm  reut2-009.sgm  reut2-016.sgm  sample_data
cat-descriptions_120396.txt         reut2-003.sgm  reut2-010.sgm  reut2-017.sgm
feldman-cia-worldfactbook-data.txt  reut2-004.sgm  reut2-011.sgm  reut2-018.sgm
```

```
# Data manipulation and analysis
!pip install numpy
!pip install pandas
!pip install datasets

# Web scraping
!pip install beautifulsoup4

# Text processing and NLP
!pip install nltk
!pip install gensim

# Visualization
!pip install matplotlib
!pip install seaborn

# Machine learning and scikit-learn
!pip install scikit-learn
!pip install transformers

# Model persistence
!pip install torch
```

Step 1: Python Module Library

```
# Standard libraries
import os
import re
from collections import Counter
from itertools import chain

# Data manipulation and analysis
import numpy as np
import pandas as pd

# Web scraping
from bs4 import BeautifulSoup

# Text processing and NLP
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from gensim.models import Word2Vec

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Machine learning and scikit-learn
import sklearn
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer, TfidfVectorizer
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay, multilabel_confusion_matrix
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.neural_network import MLPClassifier
```

```
from sklearn.base import BaseEstimator, TransformerMixin
from transformers import (
    AutoModelForSequenceClassification,
    AutoTokenizer,
    EvalPrediction,
    Trainer,
    TrainingArguments,
)

# Model persistence
import pickle
import torch

nltk.download('punkt')
nltk.download('stopwords')
p = print
d = display
```

```
[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.
```

Step 2: Data Extraction

Downloaded From: https://archive.ics.uci.edu/dataset/137/reuters+21578+text+categorization+collection

```
def minor_preprocess(file):

  with open(file, 'rb') as f:
    lines = f.readlines()
    utf8_safe_lines = [line.decode('utf-8', 'ignore') for line in lines]
    xml_safe_lines = [re.sub(r'\d*;', '', line) for line in utf8_safe_lines] # Get rid of problematic strings
    no_newlines = [line.replace('\n', ' ') for line in xml_safe_lines]
  f.close()

  return ''.join(no_newlines)

def compile_data(datapath='/content/'):
  dataset = []
  for file in os.listdir(datapath):
    if file.endswith('.sgm'):
      preprocessed_data = minor_preprocess(datapath + '/' + file)
      records = [record + '</REUTERS>' for record in preprocessed_data.split('</REUTERS>') if record] # Retain all original form

      dataset.extend(records)

  return dataset


def compile_dictionary(data):
    data_dict = {
        'new_id': [],
        'people': [],
        'places': [],
        'orgs': [],
        'exchanges': [],
        'title':[],
        'body': [],
        'r_topics': [],
        'lewissplit': [],
        'cgisplit': [],
        'topic': [],
    }
```

```
38
39      soup = BeautifulSoup(data, 'xml')
40
41      reuters_tag = soup.find('REUTERS')
42      if reuters_tag:
43          r_topics_value = reuters_tag.get('TOPICS')
44          if r_topics_value:
45              data_dict['r_topics'] = r_topics_value
46
47          new_id_value = reuters_tag.get('NEWID')
48          if new_id_value:
49              data_dict['new_id'] = new_id_value
50
51          lewissplit_value = reuters_tag.get('LEWISSPLIT')
52          if lewissplit_value:
53              data_dict['lewissplit'] = lewissplit_value
54
55          cgisplit_value = reuters_tag.get('CGISPLIT')
56          if cgisplit_value:
57              data_dict['cgisplit'] = cgisplit_value
58
59      people_tag = soup.find('PEOPLE')
60      if people_tag and people_tag.contents:
61          cleaned_people = [tag.text for tag in people_tag.find_all('D')]
62          data_dict['people'] = cleaned_people
63
64      places_tag = soup.find('PLACES')
65      if places_tag and places_tag.contents:
66          cleaned_places = [tag.text for tag in places_tag.find_all('D')]
67          data_dict['places'] = cleaned_places
68
69      orgs_tag = soup.find('ORGS')
70      if orgs_tag and orgs_tag.contents:
71          cleaned_orgs = [tag.text for tag in orgs_tag.find_all('D')]
72          data_dict['orgs'] = cleaned_orgs
73
74      exchanges_tag = soup.find('EXCHANGES')
```

```
    if exchanges_tag and exchanges_tag.contents:
        cleaned_exc = [tag.text for tag in exchanges_tag.find_all('D')]
        data_dict['exchanges'] = cleaned_exc

    body_tag = soup.find('BODY')
    if body_tag and body_tag.contents:
        data_dict['body'] = body_tag.contents[0]

    title_tag = soup.find('TITLE')
    if title_tag and title_tag.contents:
        data_dict['title'] = title_tag.contents[0]

    topics_tag = soup.find('TOPICS')
    if topics_tag and topics_tag.contents:
        cleaned_topics = [tag.text for tag in topics_tag.find_all('D')]
        data_dict['topic'] = cleaned_topics

    return data_dict

```

```
# Convert the list into a dictionary with fields of interest
reuters_dicts = [compile_dictionary(data) for data in compile_data()]

```

```
df = pd.DataFrame(reuters_dicts)
d(df.head())
df.to_csv('raw_data.csv')
```

	new_id	people	places	orgs	exchanges	title	body	r_topics	lewissplit	cgisplit	topic
0	2001	[]	[uk]	[]	[]	JAGUAR SEES STRONG GROWTH IN NEW MODEL SALES	Jaguar Plc > is about to sell its new XJ-6 mod...	YES	TRAIN	TRAINING-SET	[earn]
1	2002	[]	[]	[]	[]	OCCIDENTAL PETROLEUM COMMON STOCK OFFERING RAI...	[]	NO	TRAIN	TRAINING-SET	[]
2	2003	[]	[usa, iraq]	[]	[]	CCC ACCEPTS BONUS BID ON WHEAT FLOUR TO IRAQ	The Commodity Credit Corporation, CCC, has acc...	YES	TRAIN	TRAINING-SET	[grain, wheat]
3	2004	[]	[]	[]	[]	DIAMOND SHAMROCK RAISES CRUDE POSTED PRICES ON...	[]	YES	TRAIN	TRAINING-SET	[crude]
4	2005	[]	[usa]	[]	[]	NORD RESOURCES CORP > 4TH QTR NET	Shr 19 cts vs 13 cts Net 2,656,000 vs 1,71...	YES	TRAIN	TRAINING-SET	[earn]

Dataset Overview

```
p('Statistical Table')
d(df.describe())

p('\nData Type for each Column')
d(df.info())

# Check for empty lists in each column
p('\nNumber of empty list in each column')
d(df.applymap(lambda x: isinstance(x, list) and not x).sum())

```

Statistical Table

	new_id	people	places	orgs	exchanges	title	body	r_topics	lewissplit	cgisplit	topic
count	21600	21600	21600	21600	21600	21600	21600	21600	21600	21600	21600
unique	21579	218	1097	68	58	20006	18780	4	4	3	656
top	[]	[]	[usa]	[]	[]	[]	[]	YES	TRAIN	TRAINING-SET	[]
freq	22	20444	10878	20719	21118	759	2557	13476	14668	20856	10233

```
Data Type for each Column
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21600 entries, 0 to 21599
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   new_id      21600 non-null  object
 1   people      21600 non-null  object
```

Since there is a significant number of empty lists for people, organizations, and exchanges, we have decided to drop these columns. Additionally, places are not expected to contribute significantly to the topic classification task, therefore, this column will also be removed.

As a result:

- the selected features are 'title' and 'body'
- the label is 'topic'.

```
dtypes: object(11)
```

Next, check if the data size is extracted correctly based on different split methods.

```
p('The Modified Apte ("ModApte") Split')
p('train size: ' + str(len(df[(df.lewissplit=='TRAIN') & (df.r_topics=='YES')])))
p('test size: ' + str(len(df[(df.lewissplit=='TEST') & (df.r_topics=='YES')])))
p('not-used size: ' + str(len(df[df.lewissplit=='NOT-USED'])))
p()

p('The Modified Hayes ("ModHayes") Split')
p('train size: ' + str(len(df[df.cgisplit=='TRAINING-SET'])))
p('test size: ' + str(len(df[df.cgisplit=='PUBLISHED-TESTSET'])))
p('not-used size: ' + str(len(df[df.cgisplit=='NOT-USED'])))
```

```
The Modified Apte ("ModApte") Split
train size: 9603
test size: 3299
not-used size: 722

The Modified Hayes ("ModHayes") Split
train size: 20856
test size: 722
not-used size: 0
```

The size mentioned above aligns with the information provided in the dataset readme file, confirming that there are no mistakes in the data extraction. Please refer to:

- Line 459: VIII.B. The Modified Apte ('ModApte') Split
- Line 510: VIII.C. The Modified Hayes ('ModHayes') Split

For simplicity, we will choose ModApte over ModHayes, as it offers a more balanced distribution of training (9603) and testing (3299) data sizes, equivalent to a ratio of 65% versus 35%.

Step 3: Data Preprocessing

Combine title and body together as text column

```
p("Combine title and body together")
df['text'] = df['title'].apply(lambda x: ''.join(x)) + ' '  + df['body'].apply(lambda x: ''.join(x))
df.insert(df.columns.get_loc('body') + 1, 'text', df.pop('text'))
df = df.drop(['body', 'title'], axis=1)
```

```
    Combine title and body together
```

Remove special characters, newlines and number

```
df['text'] = df['text'].replace('[^a-zA-Z0-9\s]', '', regex=True)       # Special Characters
df['text'] = df['text'].replace('\n', ' ', regex=True)                  # Remove newlines
df['text'] = df['text'].replace('\d', '', regex=True)                   # Remove numbers

```

Removing stopwords through tokenization

```
# Define function to remove stopwords
def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    words = word_tokenize(text)
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)

# Apply the function to the DataFrame
df['text'] = df['text'].apply(remove_stopwords)

```

Stemming through tokenizing

```
# Initialize Porter Stemmer
ps = PorterStemmer()

# Define function to apply stemming
def apply_stemming(text):
    words = word_tokenize(text)
    stemmed_words = [ps.stem(word) for word in words]
    return ' '.join(stemmed_words)

# Apply the function to the DataFrame
df['text'] = df['text'].apply(apply_stemming)

```

Result of the text columns after preprocessing

```
df.text.head()
```

```
0    jaguar see strong growth new model sale jaguar...
1    occident petroleum common stock offer rais mln...
2    ccc accept bonu bid wheat flour iraq commod cr...
3    diamond shamrock rais crude post price one dlr...
4    nord resourc corp th qtr net shr ct vs ct net ...
Name: text, dtype: object
```

Recreate a new data dictionery based on ModApte split; for more information, refer to step 2 for the reason.

- Train
- Test
- Unused

```
#Create a new function to cater for ModApte split for data dict
def create_data_dict(df, lewissplit, r_topics):
    data_dict = {
        'text': df[(df.lewissplit == lewissplit) & (df.r_topics == r_topics)].text.to_list(),
        'topic': df[(df.lewissplit == lewissplit) & (df.r_topics == r_topics)].topic.to_list(),
    }
    return data_dict

# 'test' dict
test_dict = create_data_dict(df, 'TEST', 'YES')

# 'train' dict
train_dict = create_data_dict(df, 'TRAIN', 'YES')

# 'unused' dict (Will not be ignored)
unused_dict = create_data_dict(df, 'NOT-USED', '')
```

Step 4: Data Exploration / Cleaning

Word Frequency Analysis for Text

- Identify the most frequent words.
- Identify the words that appear for one time.
- Count the number of unique words

```
from nltk.tokenize import word_tokenize
from collections import Counter

# List of sentences
sentences = train_dict['text']

# Tokenize the text
all_tokens = [word_tokenize(sentence) for sentence in sentences]
```

```
9
10 # Flatten the list of tokens
11 all_tokens_flat = [token for sublist in all_tokens for token in sublist]
12
13 # Create a Counter to count word frequencies
14 word_freq = Counter(all_tokens_flat)
15
16 # Identify the most frequent words (top N)
17 most_frequent_words = word_freq.most_common(10)
18
19 # Find words that appear only once
20 words_appear_one_time = [word for word, freq in word_freq.items() if freq == 1]
21
22 # Count the number of unique words
23 num_unique_words = len(word_freq)
24
25 # Display the results
26 p("Most Frequent Words:")
27 p(most_frequent_words)
28 p("\nNumber of Words Appear for One Time:")
29 p(len(words_appear_one_time ))
30 p('\nExample of Words Appear for One Time')
31 p(words_appear_one_time)
32 p("\nNumber of Unique Words:", num_unique_words)
33
```

```
Most Frequent Words:
[('said', 23727), ('mln', 15324), ('dlr', 12669), ('pct', 9805), ('reuter', 9193), ('vs', 9187), ('year', 7138), ('bank', 6485)

Number of Words Appear for One Time:
7469

Example of Words Appear for One Time
['init', 'barcelo', 'gloeilampenfabriek', 'kph', 'anatolia', 'ataturk', 'seeker', 'tb', 'withdrfaw', 'taiwain', 'resourec', 'in

Number of Unique Words: 20133
```

- As stopwords have applied on previous step, so there isn't any more words like the, will and is.
- There are 7231 words that appear for one time as compared to a total of 19737 unique words

Label Analysis

- Topic is considered as the label for this text classification

```
# Topic list
my_list = train_dict['topic']

p('Average token for each element')
average_token_per_element = sum([len(element) for element in my_list])/len(my_list)
p(average_token_per_element)

# Flatten the list of lists
flattened_list = [item for sublist in my_list for item in sublist]

# Use Counter to count occurrences
element_counts = Counter(flattened_list)
p('\nNumber of unique topic')
p(len(element_counts))

p('\nDensity')
p(average_token_per_element/len(element_counts))

# Sort the items by frequency in descending order
sorted_counts = sorted(element_counts.items(), key=lambda x: x[1], reverse=True)

p('\nTopic that Occurence for only one time')
filtered_counts = [item for item in sorted_counts if item[1] == 1]
p(len([item[0] for item in filtered_counts ]))

p("\ntopic occurences for one time")
# Filter tuples where the value is equal to 1
occurance_only_one = [element[0] for element in filtered_counts]
p(occurance_only_one)

# Extract & plot top 10
top_10 = sorted_counts[:10]
plt.figure(figsize=(10, 5))
bars = plt.barh(range(len(top_10)), [count for _, count in top_10], align='center')

# Add labels to the bars
for bar, (item, count) in zip(bars, top_10):
```

```
    plt.text(bar.get_width(), bar.get_y() + bar.get_height() / 2, f'{count}', ha='left', va='center')

plt.yticks(range(len(top_10)), [item for item, _ in top_10])
plt.title('Top 10 Topic Occurrences')
plt.xlabel('Frequency')
plt.ylabel('Element')
plt.show()

```

```
Average token for each element
1.0047901697386235

Number of unique topic
115

Density
0.008737305823814117
```

Check if we can remove some of the one-time occurance by comparing to test set

```
20
```

```
# Initialize a list to store samples with one time occurance labels
topic_occurence_one_test = []

for sample in test_dict['topic']:
    if any(label in occurance_only_one for label in sample):
        topic_occurence_one_test.append(sample)

p('Number of topic occurence one time in test')
p(len(topic_occurence_one_test),'out of', len(test_dict['text']))
p('\nShare in percentage')
p(str(round(len(topic_occurence_one_test)/len(test_dict['text'])*100,2)) + '%')
```

```
Number of topic occurence one time in test
7 out of 3299

Share in percentage
0.21%
```

Safe to drop these one-time occurrance for train and test to reduce the label size.

```
def remove_unwanted_topic(dict):
    df1 = pd.DataFrame(dict)
    df1 = df1[~df1['topic'].apply(lambda x: any(topic in x for topic in occurance_only_one))]
    return df1.to_dict(orient='list')

train_dict = remove_unwanted_topic(train_dict)
test_dict = remove_unwanted_topic(test_dict)

p('new number of train data', len(train_dict['text']) )
p('new number of test data',len(test_dict['text']))

```

```
new number of train data 9588
new number of test data 3292
```

Step 5: Prediction based on Classifier Model

This code prepares training data by extracting text (x_train) and corresponding topics (y_train). It then transforms the text into numerical vectors using TF-IDF, and the topics into a binary matrix, facilitating the training of machine learning models.

```
# Custom transformer to convert text to Word2Vec embeddings
class Word2VecTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, model):
        self.model = model

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return [self.model.wv[x] if x in self.model.wv else [0] * self.model.vector_size for x in X]
```

```
x_train, y_train = train_dict['text'], train_dict['topic']
x_test, y_test = test_dict['text'], test_dict['topic']

# TF-IDF Vectorization
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000, ngram_range=(1,2))
vectorizer.fit(x_train)

# Word2Vec Model
word2vec_model = Word2Vec(sentences=x_train, vector_size=100, window=5, min_count=1, workers=4)

# Create a pipeline with both transformations
combined_features = FeatureUnion([
    ('tfidf', vectorizer),
    ('word2vec', Word2VecTransformer(word2vec_model))
])

# Combined_features as pickle
with open('combined_features.sav', 'wb') as combined_features_file:
    pickle.dump(combined_features, combined_features_file)

# Transform the data
X_train_combined = combined_features.fit_transform(x_train)
X_test_combined = combined_features.transform(x_test)


# Transform Labels to Binary Matrix
mlb = MultiLabelBinarizer()
y_train_bin = mlb.fit_transform(y_train)
y_test_bin = mlb.transform(y_test)
```

```
WARNING:gensim.models.word2vec:Each 'sentences' item should be a list of words (usually unicode strings). First item here is in
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:895: UserWarning: unknown class(es) ['cottonseed', 'f-c
  warnings.warn(
```

Transforms test text data (test_dict['text']) into numerical vectors using the pre-trained TF-IDF vectorizer. It also converts test topics (test_dict['topic']) into a binary matrix using the MultiLabelBinarizer. Finally, it prints the classes (unique topics) present in the binary matrix,

providing insight into the labeled topics used in testing.

```
1 p(mlb.classes_)
```

```
['acq' 'alum' 'austdlr' 'barley' 'bop' 'can' 'carcass' 'cocoa' 'coconut'
 'coconut-oil' 'coffee' 'copper' 'copra-cake' 'corn' 'cornglutenfeed'
 'cotton' 'cpi' 'cpu' 'crude' 'dfl' 'dlr' 'dmk' 'earn' 'fishmeal' 'fuel'
 'gas' 'gnp' 'gold' 'grain' 'groundnut' 'heat' 'hog' 'housing' 'income'
 'instal-debt' 'interest' 'inventories' 'ipi' 'iron-steel' 'jet' 'jobs'
 'l-cattle' 'lead' 'lei' 'linseed' 'livestock' 'lumber' 'meal-feed'
 'money-fx' 'money-supply' 'naphtha' 'nat-gas' 'nickel' 'nzdlr' 'oat'
 'oilseed' 'orange' 'palladium' 'palm-oil' 'palmkernel' 'pet-chem'
 'platinum' 'plywood' 'pork-belly' 'potato' 'propane' 'rand' 'rape-oil'
 'rapeseed' 'reserves' 'retail' 'rice' 'rubber' 'saudriyal' 'ship'
 'silver' 'sorghum' 'soy-meal' 'soy-oil' 'soybean' 'stg' 'strategic-metal'
 'sugar' 'sun-oil' 'sunseed' 'tapioca' 'tea' 'tin' 'trade' 'veg-oil'
 'wheat' 'wool' 'wpi' 'yen' 'zinc']
```

Training multiple classifiers is advantageous for diverse insights and enhanced performance which provides robustness, balances bias and variance trade-offs, and allows for exploratory analysis, ultimately contributing to a more versatile and reliable machine learning system. Due to limited time, only 3 classifers are run to give the quick result.

```
#Wait for more than 1min to run

classifiers = [
    OneVsRestClassifier(LogisticRegression(random_state=0)),
    OneVsRestClassifier(SGDClassifier(alpha=0.0001, random_state=0)),
    OneVsRestClassifier(MultinomialNB(alpha=0.1)),
    #OneVsRestClassifier(RandomForestClassifier(n_estimators=50, max_depth=3, random_state=0)),
    #OneVsRestClassifier(SVC(kernel='linear', C=1.0, random_state=0)),
    #OneVsRestClassifier(MLPClassifier(hidden_layer_sizes=(50,), max_iter=200, alpha=0.0001, solver='adam', random_state=0))

]

entries = []
cv_df = pd.DataFrame()
for clf in classifiers:
    print(str(clf))
    scores = cross_val_score(clf, X_train_combined, y_train_bin, cv=5, scoring='accuracy')
    for fold_idx, accuracy in enumerate(scores):
        entries.append((clf.estimator, fold_idx, accuracy))

cv_df = pd.DataFrame(entries, columns=['classifier', 'fold_idx', 'accuracy'])
```

```
      warnings.warn(
    /usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 34 is present in all training exampl
      warnings.warn(
    /usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 57 is present in all training exampl
      warnings.warn(
    /usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 17 is present in all training exampl
      warnings.warn(
    /usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 44 is present in all training exampl
      warnings.warn(
    /usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 63 is present in all training exampl
      warnings.warn(
```

```
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 14 is present in all training exampl
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 23 is present in all training exampl
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 34 is present in all training exampl
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 57 is present in all training exampl
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 17 is present in all training exampl
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 44 is present in all training exampl
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 63 is present in all training exampl
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 12 is present in all training exampl
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 53 is present in all training exampl
  warnings.warn(
OneVsRestClassifier(estimator=MultinomialNB(alpha=0.1))
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 5 is present in all training example
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 14 is present in all training exampl
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 23 is present in all training exampl
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 34 is present in all training exampl
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 57 is present in all training exampl
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 17 is present in all training exampl
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 44 is present in all training exampl
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 63 is present in all training exampl
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 12 is present in all training exampl
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/multiclass.py:77: UserWarning: Label not 53 is present in all training exampl
  warnings.warn(
```

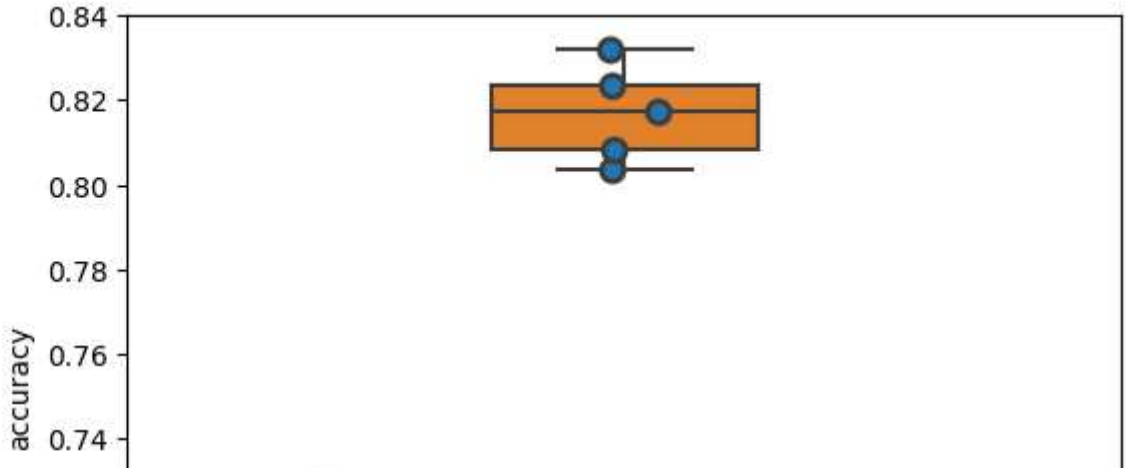
Accuracy table for different classifiers

Best classifier is SGD

```
# Your existing code for creating the boxplot
cv_df['classifier'] = cv_df['classifier'].astype(str)
sns.boxplot(x='classifier', y='accuracy', data=cv_df)
sns.stripplot(x='classifier', y='accuracy', data=cv_df,
              size=8, jitter=True, edgecolor="gray", linewidth=2)

# Shorten x-axis labels
plt.xticks(rotation=45, ha='right')  # Adjust the rotation angle as needed

plt.show()
```




```
# Train Classifier
clf = OneVsRestClassifier(SGDClassifier(alpha=0.0001, random_state=0))
clf.fit(X_train_combined, y_train_bin)
# Predictions and Evaluation
y_pred = clf.predict(X_test_combined)
```

```
print("Classification Report:")
print(classification_report(y_test_bin, y_pred, target_names=mlb.classes_, zero_division=1))
```

Classification Report Analysis

The classification report highlights the model's proficiency in specific topics, as evident from high precision, recall, and F1-scores. For example, the 'acq' topic boasts a precision of 0.98, recall of 0.93, and an F1-score of 0.95. However, the model's performance is unevenly distributed across various topics.

The macro average for recall and F1-score, standing at 0.49 and 0.55 respectively, suggests an opportunity to enhance the model's performance on minority classes.

Conversely, the weighted average, signaling high precision (0.95), recall (0.80), and F1-score (0.84), indicates effective performance on common topics. Yet, this may not accurately represent the model's efficacy on less frequent topics.

The samples average, portraying elevated precision (0.97), recall (0.87), and F1-score (0.86), demonstrates commendable performance on individual instances. However, it might not fully encapsulate the model's effectiveness for each specific topic.

In summary, while the model excels in specific areas, there exists room for improvement, particularly in achieving a more balanced performance across all topics.

Save the classifier model as pickle file

```
# Save the model to a file
with open('classifier.sav', 'wb') as file:
    pickle.dump(clf, file)
```

Classifer Prediction

```
#When handling the list for new text input
def preprocess_text_advanced(text_input):
    # If text_input is a list, process each element separately
    if isinstance(text_input, list):
        processed_texts = []
        for single_text in text_input:
            # Process each element using the single-text processing function
            processed_text = preprocess_text_advanced_single(single_text)
            processed_texts.append(processed_text)
        return processed_texts
    else:
        # If text_input is a single string, process it directly
        return preprocess_text_advanced_single(text_input)

# Download NLTK resources
nltk.download('stopwords')
nltk.download('punkt')

#When handling the single for new text input
def preprocess_text_advanced_single(text_input):
    # Lowercasing
    text_input = text_input.lower()

    # Removing HTML tags (if any)
    text_input = re.sub(r'<.*?>', '', text_input)

    # Removing special characters, numbers, and newlines
    text_input = re.sub(r'[^a-zA-Z\s]', '', text_input)
    text_input = re.sub(r'\d+', '', text_input)
    text_input = text_input.replace('\n', ' ')

    # Tokenization
    tokens = word_tokenize(text_input)

    # Removing stop words
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]
```

```
38
39      # Stemming
40      stemmer = PorterStemmer()
41      stemmed_words = [stemmer.stem(word) for word in tokens]
42
43      # Joining tokens back into a string
44      preprocessed_text = ' '.join(stemmed_words)
45
46      return preprocessed_text
47
48  def classifier_topic_prediction(text_input):
49
50      # Preprocess the new text
51      processed_text = preprocess_text_advanced(text_input)
52
53      # Load the TF-IDF vectorizer (already fitted on your training data)
54      with open('combined_features.sav', 'rb') as combined_features_file:
55          combined_features = pickle.load(combined_features_file)
56
57      # Transform the data
58      X_test_combined = combined_features.transform(processed_text)
59
60      #Load the classifier model
61      with open('classifier.sav', 'rb') as classifer_model:
62          classifier = pickle.load(classifer_model)
63
64      # Predict the topic
65      one_hot_array = classifier.predict(X_test_combined)
66      position = np.argmax(one_hot_array)
67      p('Predicted Topic: ', mlb.classes_[position])
68
69
70  def classifier_topic_prediction_array(text_input):
71
72      if len(text_input) > 1:
73
74          for i in text_input:
```

```
            classifier_topic_prediction([i])

    else:
        classifier_topic_prediction(text_input)

```

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

Prepared 3 kinds of Texts:

- train_text (Extracted from train data)
- new_text (Copy from internet)
- test_text_array (Extracted from test data)

```
train_text = """The U.S. Agriculture Department
reported the farmer-owned reserve national five-day average
price through February 25 as follows (Dlrs/Bu-Sorghum Cwt) -
         Natl   Loan           Release   Call
         Avge   Rate-X  Level    Price  Price
 Wheat   2.55   2.40       IV     4.65     --
                            V     4.65     --
                           VI     4.45     --
 Corn    1.35   1.92       IV     3.15   3.15
                            V     3.25     --
 X - 1986 Rates.

          Natl   Loan          Release   Call
          Avge   Rate-X  Level   Price  Price
 Oats     1.24   0.99        V    1.65    --
 Barley   n.a.   1.56       IV    2.55   2.55
                            V    2.65    --
 Sorghum  2.34   3.25-Y     IV    5.36   5.36
                            V    5.54    --
    Reserves I, II and III have matured. Level IV reflects
grain entered after Oct 6, 1981 for feedgrain and after July
23, 1981 for wheat. Level V wheat/barley after 5/14/82,
corn/sorghum after 7/1/82. Level VI covers wheat entered after
January 19, 1984.  X-1986 rates. Y-dlrs per CWT (100 lbs).
n.a.-not available."""]

new_text = """TBEIJING: China is set to showcase the strides made towards self-reliance at an inaugural global trade fair this

The participation of Western businesses at the China International Supply Chain Expo will also be in focus as Sino-US ties warm

China has promoted the five-day event - which begins in Beijing on Nov 28 - as the world’s first supply chain expo at the nation

It will be the latest in a series of global trade shows China has held in the second half of the year. At least two have already

Chinese state media have framed these events as a sign of Beijing’s contributions to the global economy and its commitment to in

A quarter of the 515 companies and institutions participating in the supply chain expo are foreign, hailing from 55 countries an
```

```
US firms make up around 20 per cent of the foreign registrations, with the likes of Amazon, Apple, Tesla, Intel, HP and Qualcomm

The number of US exhibitors is “much higher than expected”, according to the expo organiser.

We hope that the participating US companies actively engage in the expo, yielding fruitful outcomes," said CCPIT Vice-Chairman M

We will continue to vigorously promote high-level opening up and better protect the rights and interests of foreign investors pe

test_text_array =  ['indonesian commod exchang may expand indonesian commod exchang like start trade least one new commod possib
```

```
p('Classifier')
p()
classifier_topic_prediction_array(train_text)
p()
classifier_topic_prediction_array(new_text)
p()
classifier_topic_prediction_array(test_text_array)
```

```
Classifier

Predicted Topic:  barley

Predicted Topic:  trade

Predicted Topic:  rubber
Predicted Topic:  grain
Predicted Topic:  gold
Predicted Topic:  acq
```

Step 6: Prediction based on BERT Transformer

To execute Step 6, make sure to open the IPython notebook in Google Colab since a GPU is necessary. Afterward, click on "Runtime," navigate to "Change Runtime Type," and choose "T4 GPU" from the options.

This series of functions and operations help in creating and preprocessing data dictionaries for the ModApte split with the removal of unwanted topics

```
#Create a new function to cater for ModApte split for data dict
def create_data_dict(df, lewissplit, r_topics):
    data_dict = {
        'text': df[(df.lewissplit == lewissplit) & (df.r_topics == r_topics)].text.to_list(),
        'topic': df[(df.lewissplit == lewissplit) & (df.r_topics == r_topics)].topic.to_list(),
    }
    return data_dict

# 'test' dict
test_dict = create_data_dict(df, 'TEST', 'YES')

# 'train' dict
train_dict = create_data_dict(df, 'TRAIN', 'YES')

# 'unused' dict (Will not be ignored)
unused_dict = create_data_dict(df, 'NOT-USED', '')

def remove_unwanted_topic(dict):
    df1 = pd.DataFrame(dict)
    df1 = df1[~df1['topic'].apply(lambda x: any(topic in x for topic in occurance_only_one))]
    return df1.to_dict(orient='list')

train_dict = remove_unwanted_topic(train_dict)
test_dict = remove_unwanted_topic(test_dict)
```

New dataset Dictionary

```
from datasets import DatasetDict, Dataset

train_dataset = Dataset.from_dict(train_dict)
test_dataset = Dataset.from_dict(test_dict)
unused_dataset = Dataset.from_dict(unused_dict)

dataset = DatasetDict({
    'train': train_dataset,
    'test': test_dataset,
    'unused': unused_dataset,
    })


```

```
dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'topic'],
        num_rows: 9588
    })
    test: Dataset({
        features: ['text', 'topic'],
        num_rows: 3292
    })
    unused: Dataset({
        features: ['text', 'topic'],
        num_rows: 0
    })
})
```

Prepares the labels for a multi-label classification task by transforming them into a multi-hot encoding format and creating dictionaries for label mappings. The assertion check ensures the correctness of the label processing

```python
# Check number of unique labels
unique_labels = set(chain.from_iterable(dataset['train']["topic"]))
print(f"We have {len(unique_labels)} unique labels:\n{unique_labels}")

# Transform topics into multi-hot encoding format
mlb = MultiLabelBinarizer()
mlb.fit(dataset['train']['topic'])
dataset = dataset.map(
    lambda x: {"labels": torch.from_numpy(mlb.transform(x["topic"])).float()}, batched=True)

labels = mlb.classes_
id2label = {idx:label for idx, label in enumerate(labels)}
label2id = {label:idx for idx, label in enumerate(labels)}
num_labels = len(id2label)

assert num_labels == len(unique_labels)
```

```
We have 95 unique labels:
{'yen', 'lumber', 'oilseed', 'propane', 'zinc', 'dfl', 'livestock', 'copra-cake', 'inventories', 'tea', 'sugar', 'pet-chem', 'c
```

```
Map: 100%                                                    9588/9588 [00:00<00:00, 46533.43 examples/s]
```

```
Map: 100%                                                    3292/3292 [00:00<00:00, 38430.69 examples/s]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:895: UserWarning: unknown class(es) ['cottonseed', 'f-c
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:895: UserWarning: unknown class(es) ['sfr'] will be ign
  warnings.warn(
```

Useful to quickly inspect a subset of label mappings and ensure that the indices and labels are correctly mapped in the id2label dictionary. It can help catch any unexpected issues with the label processing

```
# sanity check:
for idx, label in id2label.items():
    if idx>=10:
        break

    print(f"{idx}: {label}")
```

```
0: acq
1: alum
2: austdlr
3: barley
4: bop
5: can
6: carcass
7: cocoa
8: coconut
9: coconut-oil
```

Prepares the training dataset for model training by tokenizing the text and removing unnecessary columns

```
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-cased")

# Tokenize and remove unwanted columns
def tokenize_function(example):
    return tokenizer(example["text"], padding="max_length", truncation=True, max_length=512)

columns = dataset["train"].column_names
columns.remove("text")
columns.remove("labels")

#The dataset.map function is used to apply the tokenize_function to each example in the training dataset.
#The batched=True parameter indicates that tokenization should be applied in batches for efficiency,
#and the specified columns are removed from the resulting tokenized dataset
tokenized_dataset = dataset.map(tokenize_function, batched=True, remove_columns=columns)
```

```
tokenizer_config.json: 100%  29.0/29.0 [00:00<00:00, 2.31kB/s]
config.json: 100%  465/465 [00:00<00:00, 33.0kB/s]
vocab.txt: 100%  213k/213k [00:00<00:00, 1.61MB/s]
tokenizer.json: 100%  436k/436k [00:00<00:00, 6.12MB/s]
Map: 100%  9588/9588 [00:04<00:00, 2438.35 examples/s]
```

```
example = tokenized_dataset['train'][0]
print(example.keys())
```

```
dict_keys(['text', 'labels', 'input_ids', 'attention_mask'])
```

```
[id2label[idx] for idx, label in enumerate(example['labels']) if label == 1.0]
```

```
['earn']
```

To convert the tokenized dataset to the PyTorch format.

```
tokenized_dataset.set_format("torch")
```

This code initializes a sequence classification model using the `AutoModelForSequenceClassification` class from the transformers library. Let's break down the code:

```
# Import the necessary library
from transformers import AutoModelForSequenceClassification

# Initialize the sequence classification model
model = AutoModelForSequenceClassification.from_pretrained(
    "distilbert-base-cased",              # Pre-trained model name or path
    num_labels=num_labels,                 # Number of classification labels
```

```
    problem_type="multi_label_classification",  # Problem type (multi-label classification in this case)
    id2label=id2label,                      # Mapping from label indices to label names
    label2id=label2id                       # Mapping from label names to label indices
)
```

Explanation:

1. **Model Initialization:** The `AutoModelForSequenceClassification.from_pretrained` function initializes a pre-trained sequence classification model. In this case, it uses the "distilbert-base-cased" model.
2. **Number of Labels:** The `num_labels` parameter is set to the number of unique labels in the dataset. This is necessary for initializing the classification head of the model.
3. **Problem Type:** The `problem_type` parameter is set to "multi_label_classification" to indicate that it's a multi-label classification problem.
4. **Label Mapping:** The `id2label` and `label2id` parameters are used to provide mappings between label indices and label names. These mappings are helpful during model evaluation to convert between indices and human-readable labels.

Now, the `model` variable holds an instance of a pre-trained sequence classification model ready for fine-tuning on the specific multi-label classification task.

```
model = AutoModelForSequenceClassification.from_pretrained(
    "distilbert-base-cased",
    num_labels=num_labels,
    problem_type="multi_label_classification",
    id2label=id2label,
    label2id=label2id
    )
```

```
model.safetensors: 100%                                                263M/263M [00:01<00:00, 241MB/s]
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-cased and
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
# test forward pass
tokenized_dataset['train'][0]['labels'].type()
```

```
'torch.FloatTensor'
```

Generate classification report using pipeline class