In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-pyt
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all fil

import os
train=pd.read_csv('training.csv')
# You can write up to 20GB to the current directory (/kaggle/working/) that gets preser
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside
```

Displaying the 5 rows of the dataset

In [2]:

```
train.head()
```

Out[2]:

	id	name	document_text	cat_name
0	22474	Information Regarding the Merger of Navios Mar...	At a special meeting held on March 24, 2021 sh...	Corporate Communications
1	27460	Announcement on Approving the Change of Member...	On April 2, 2021, the China Financial Futures ...	Securities Settlement
2	6926	SFC Suspends Shiu Yau Wah for Five Months	The Securities and Futures Commission (SFC) ha...	Antitrust
3	6982	Renminbi RMB Haircut - February 4, 2020	Pursuant to Section 2.6.2 of the Clearing Hous...	Securities Settlement
4	5022	Anti-Money Laundering, Countering Financing of...	Money laundering and terrorism financing (ML/T...	Financial Crime

Displaying unique size of train attributes

In [3]:

```
len(np.unique(list(train['document_text'])))
```

Out[3]: 9151

In [4]:

```
len(np.unique(list(train['id'])))
```

Out[4]: 9859

In [5]:

```
len(np.unique(list(train['name'])))
```

Out[5]: 8594

In [6]:

```
len(np.unique(list((train['cat_name']))))
```

Out[6]: 50

In [7]:

```
newresult = train.groupby(['id','name'])['document_text'].agg(list).to_dict()
# for k,v in newresult:
#     print(k,v)
```

Cat_name groupped into one array

In [8]:

```
train2=train.groupby('id', sort=False).agg(lambda x: list(set(x))).reset_index()
train2.head()

train2 = train2.explode('name')


train2 = train2.explode('document_text')

train2.head()
```

Out[8]:

	id	name	document_text	cat_name
0	22474	Information Regarding the Merger of Navios Mar...	At a special meeting held on March 24, 2021 sh...	[Corporate Communications]
1	27460	Announcement on Approving the Change of Member...	On April 2, 2021, the China Financial Futures ...	[Trade Settlement, Securities Settlement]
2	6926	SFC Suspends Shiu Yau Wah for Five Months	The Securities and Futures Commission (SFC) ha...	[Compliance Management, Licensure and certific...
3	6982	Renminbi RMB Haircut - February 4, 2020	Pursuant to Section 2.6.2 of the Clearing Hous...	[Payments and Settlements, Securities Settleme...
4	5022	Anti-Money Laundering, Countering Financing of...	Money laundering and terrorism financing (ML/T...	[Regulatory Reporting, Money-Laundering and Te...

In [9]:

```
# for k,v in newresult.items():
#     print(k,v)
```

In [10]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# initialize the TfidfVectorizer without any parameters
tfidf_vect = TfidfVectorizer()

# with stop words removed
tfidf_vect = TfidfVectorizer(stop_words="english")

# generate tfidf matrix
dtm= tfidf_vect.fit_transform(train["document_text"])

print("type of dtm:", type(dtm))
print("size of tfidf matrix:", dtm.shape)
```

```
type of dtm: <class 'scipy.sparse.csr.csr_matrix'>
size of tfidf matrix: (47102, 33459)
```

In [11]:

```
# 1. Check vocabulary

# Vocabulary is a dictionary mapping a word to an index

# the number of words in the vocabulary
print("total number of words:", len(tfidf_vect.vocabulary_))

print("type of vocabulary:", \
      type(tfidf_vect.vocabulary_))
```

```
total number of words: 33459
type of vocabulary: <class 'dict'>
```

Displaying some of the text and most freq words

In [12]:

```
# 3.4 check words with top tf-idf wights in a document,
# e.g. 1st document

# get mapping from word index to word
# i.e. reversal mapping of tfidf_vect.vocabulary_
voc_lookup={tfidf_vect.vocabulary_[word]:word \
            for word in tfidf_vect.vocabulary_}

print("\nOriginal text: \n"+train["document_text"][0])

print("\ntfidf weights: \n")

# first, covert the sparse matrix row to a dense array
doc0=dtm[0].toarray()[0]
print("Vectorized document shape: ", doc0.shape, "\n")

# get index of top 20 words
print("top words:")
top_words=(doc0.argsort())[::-1][0:20]
for i in top_words:
    print("{0}:\t{1:.3f}".format(voc_lookup[i], doc0[i]))
#[(voc_lookup[i], '%.3f'%doc0[i]) for i in top_words]
```

```
Original text:
At a special meeting held on March 24, 2021 shareholders of Navios Maritime Containers
L.P. (NMCI) approved the proposed merger with Navios Maritime Partners L.P. The merger i
s anticipated to become effective on March 31, 2021. The details are as follows: Company
Issue: Navios Maritime Containers L.P. Common Units CUSIP#: Y62151108 Symbol: NMCI Antic
ipated Last Trading Date: March 31, 2021 Anticipated Marketplace Effective Date for Susp
ension: April 1, 2021 Merger Consideration: 0.39 of a common unit representing limited p
artner interests in Navios Partners for each share held.

tfidf weights:

Vectorized document shape:  (33459,)

top words:
navios: 0.593
maritime:       0.389
nmci:   0.296
containers:     0.271
anticipated:    0.240
merger: 0.200
y62151108:      0.165
partners:       0.160
```

```
2021:   0.155
march:  0.140
held:   0.110
common: 0.108
31:     0.095
partner:        0.093
39:     0.083
representing:   0.081
date:   0.081
effective:      0.080
units:  0.072
cusip:  0.072
```

Using Multilabel binarizer to convert array of names into array of class variables

In [13]:

```python
from sklearn.preprocessing import MultiLabelBinarizer
import numpy as np

mlb = MultiLabelBinarizer()
# labels=list(np.unique(train2['cat_name']))
# labels=[list(labels)]
# labels
# mlb.fit(labels)
Y=mlb.fit_transform(train2['cat_name'])
# check size of indicator matrix
# print some rows
print(Y[0:5])
print(Y.shape)
# check classes
print(mlb.classes_)

# check # of samples in each class
np.sum(Y, axis=0)
```

```
[[0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 1 0 1]
 [0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
  0 0 0 0 0 0 0 1 0 0 0 1 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
  0 0 1 1 0 0 1 0 0 0 0 0 0 0 0]]
(9859, 50)
['Accounting and Finance' 'Antitrust' 'Banking' 'Broker Dealer'
 'Commodities Trading' 'Compliance Management' 'Consumer protection'
 'Contract Provisions' 'Corporate Communications' 'Corporate Governance'
 'Definitions' 'Delivery' 'Examinations' 'Exemptions' 'Fees and Charges'
 'Financial Accounting' 'Financial Crime' 'Forms' 'Fraud' 'IT Risk'
 'Information Filing' 'Insurance' 'Legal' 'Legal Proceedings' 'Licensing'
 'Licensure and certification' 'Liquidity Risk' 'Listing' 'Market Abuse'
 'Market Risk' 'Monetary and Economic Policy' 'Money Services'
 'Money-Laundering and Terrorist Financing' 'Natural Disasters'
 'Payments and Settlements' 'Powers and Duties' 'Quotation'
 'Records Maintenance' 'Regulatory Actions' 'Regulatory Reporting'
 'Required Disclosures' 'Research' 'Risk Management' 'Securities Clearing'
 'Securities Issuing' 'Securities Management' 'Securities Sales'
 'Securities Settlement' 'Trade Pricing' 'Trade Settlement']
```

Out[13]:

```
array([ 935,  880, 1078,  670,  682, 1391,  969, 1153,  518,  958,  570,
        821, 1742, 1190, 1301,  535, 1178,  508,  906,  435, 1387,  737,
        907, 1343,  999,  982,  534, 1124,  722, 1633,  802,  869,  505,
```

```
        1079, 1099,  797,  611,  630, 1621, 1042,  627,  554,  982, 1141,
        1107,  664, 1737,  852,  872,  723])
```

In [28]:

```
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import LinearSVC
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import classification_report

# split dataset into train (70%) and test sets (30%)
X_train, X_test, Y_train, Y_test = train_test_split(\
                train2['document_text'], Y, test_size=0.30, random_state=0)

classifier = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words="english",\
                              min_df=5)),
    ('clf', OneVsRestClassifier(LinearSVC(C=3.0)))])

classifier.fit(X_train, Y_train)
```

Out[28]:

```
Pipeline(steps=[('tfidf', TfidfVectorizer(min_df=5, stop_words='english')),
                ('clf', OneVsRestClassifier(estimator=LinearSVC(C=3.0)))])
```

Classification using SVM

In [29]:

```
from sklearn.metrics import classification_report

predicted = classifier.predict(X_test)

print(predicted.shape)
print("predicted:")
print(predicted[0:2])
print("actual:")
print(Y_test[0:2])

print(classification_report\
      (Y_test, predicted, target_names=mlb.classes_))
```

```
(2958, 50)
predicted:
[[1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
actual:
[[1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
                                    precision    recall  f1-score   support

          Accounting and Finance       0.86      0.57      0.69       294
                       Antitrust       0.87      0.70      0.78       265
                         Banking       0.93      0.77      0.84       344
                  Broker Dealer        0.95      0.71      0.81       220
              Commodities Trading      0.93      0.75      0.83       195
           Compliance Management       0.88      0.74      0.80       420
```

```
                     Consumer protection       0.87      0.73      0.79       286
                     Contract Provisions       0.87      0.71      0.78       326
                Corporate Communications       0.76      0.53      0.62       177
                    Corporate Governance       0.73      0.52      0.61       297
                             Definitions       0.82      0.57      0.68       176
                                Delivery       0.97      0.73      0.83       241
                            Examinations       0.79      0.70      0.74       518
                              Exemptions       0.83      0.62      0.71       359
                        Fees and Charges       0.96      0.80      0.87       404
                    Financial Accounting       0.89      0.58      0.70       154
                         Financial Crime       0.88      0.71      0.79       349
                                   Forms       0.90      0.65      0.75       156
                                   Fraud       0.97      0.74      0.84       278
                                 IT Risk       0.84      0.55      0.66       139
                      Information Filing       0.77      0.70      0.73       412
                               Insurance       0.88      0.76      0.82       228
                                   Legal       0.68      0.51      0.58       275
                       Legal Proceedings       0.82      0.71      0.76       414
                               Licensing       0.86      0.63      0.73       328
              Licensure and certification       0.83      0.64      0.73       307
                          Liquidity Risk       0.83      0.68      0.75       154
                                 Listing       0.92      0.80      0.86       339
                            Market Abuse       0.94      0.79      0.86       220
                             Market Risk       0.74      0.69      0.72       465
             Monetary and Economic Policy       0.82      0.64      0.72       251
                          Money Services       0.77      0.63      0.69       268
Money-Laundering and Terrorist Financing       0.98      0.71      0.82       149
                       Natural Disasters       0.94      0.80      0.86       310
                 Payments and Settlements       0.89      0.79      0.84       327
                       Powers and Duties       0.70      0.47      0.56       242
                               Quotation       0.90      0.72      0.80       190
                     Records Maintenance       0.83      0.58      0.68       187
                      Regulatory Actions       0.84      0.73      0.78       492
                    Regulatory Reporting       0.73      0.56      0.64       303
                    Required Disclosures       0.80      0.45      0.58       184
                                Research       0.81      0.64      0.71       135
                         Risk Management       0.82      0.69      0.75       294
                     Securities Clearing       0.96      0.89      0.92       352
                      Securities Issuing       0.85      0.71      0.77       345
                   Securities Management       0.93      0.73      0.82       209
                        Securities Sales       0.81      0.74      0.77       531
                   Securities Settlement       0.88      0.79      0.83       252
                           Trade Pricing       0.88      0.78      0.83       259
                        Trade Settlement       0.89      0.72      0.80       214

                               micro avg       0.85      0.69      0.76     14234
                               macro avg       0.86      0.68      0.76     14234
                            weighted avg       0.85      0.69      0.76     14234
                             samples avg       0.78      0.67      0.70     14234

/opt/conda/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1318: Undefine
dMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples wi
th no predicted labels. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [30]:

```python
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import LinearSVC
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import classification_report
```

```
from sklearn.naive_bayes import MultinomialNB

# split dataset into train (70%) and test sets (10%)
X_train, X_test, Y_train, Y_test = train_test_split(\
                train2['document_text'], Y, test_size=0.30, random_state=0)

classifier = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words="english",\
                              min_df=5)),
    ('clf', OneVsRestClassifier(MultinomialNB()))])

classifier.fit(X_train, Y_train)
```

Out[30]:

```
Pipeline(steps=[('tfidf', TfidfVectorizer(min_df=5, stop_words='english')),
                ('clf', OneVsRestClassifier(estimator=MultinomialNB()))])
```

Classification using MultinomialNB

In [31]:

```
from sklearn.metrics import classification_report

predicted = classifier.predict(X_test)

print(predicted.shape)
print("predicted:")
print(predicted[0:2])
print("actual:")
print(Y_test[0:2])

print(classification_report\
      (Y_test, predicted, target_names=mlb.classes_))
```

```
(2958, 50)
predicted:
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
actual:
[[1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
                               precision    recall  f1-score   support

        Accounting and Finance       0.86      0.02      0.04       294
                     Antitrust       0.87      0.37      0.52       265
                       Banking       0.91      0.18      0.30       344
                 Broker Dealer       0.92      0.05      0.10       220
           Commodities Trading       0.96      0.34      0.51       195
         Compliance Management       0.97      0.21      0.35       420
           Consumer protection       1.00      0.09      0.16       286
           Contract Provisions       0.83      0.44      0.57       326
     Corporate Communications       0.91      0.12      0.21       177
          Corporate Governance       1.00      0.02      0.04       297
                   Definitions       1.00      0.02      0.04       176
                      Delivery       0.94      0.47      0.63       241
                  Examinations       0.84      0.32      0.47       518
                    Exemptions       0.98      0.11      0.20       359
              Fees and Charges       0.97      0.32      0.49       404
          Financial Accounting       1.00      0.05      0.09       154
               Financial Crime       0.87      0.35      0.50       349
                         Forms       1.00      0.21      0.35       156
```

```
                                Fraud       0.87      0.37      0.52       278
                              IT Risk       1.00      0.06      0.11       139
                   Information Filing       0.89      0.16      0.27       412
                            Insurance       0.83      0.26      0.40       228
                                Legal       0.50      0.01      0.01       275
                    Legal Proceedings       0.83      0.46      0.59       414
                            Licensing       0.97      0.18      0.31       328
          Licensure and certification       0.82      0.19      0.31       307
                       Liquidity Risk       0.90      0.06      0.11       154
                              Listing       0.92      0.30      0.46       339
                         Market Abuse       1.00      0.27      0.43       220
                          Market Risk       0.84      0.29      0.43       465
           Monetary and Economic Policy       0.86      0.19      0.31       251
                       Money Services       0.89      0.09      0.17       268
Money-Laundering and Terrorist Financing       1.00      0.15      0.27       149
                    Natural Disasters       0.76      0.35      0.48       310
             Payments and Settlements       0.99      0.44      0.61       327
                    Powers and Duties       1.00      0.00      0.01       242
                            Quotation       1.00      0.18      0.30       190
                  Records Maintenance       1.00      0.09      0.17       187
                   Regulatory Actions       0.79      0.50      0.61       492
                 Regulatory Reporting       0.88      0.07      0.13       303
                 Required Disclosures       0.00      0.00      0.00       184
                             Research       0.57      0.03      0.06       135
                      Risk Management       0.94      0.10      0.18       294
                  Securities Clearing       0.99      0.54      0.70       352
                   Securities Issuing       0.98      0.19      0.31       345
                Securities Management       1.00      0.51      0.68       209
                     Securities Sales       0.86      0.40      0.54       531
                Securities Settlement       0.91      0.35      0.51       252
                        Trade Pricing       0.76      0.27      0.40       259
                     Trade Settlement       0.92      0.36      0.52       214

                            micro avg       0.89      0.25      0.39     14234
                            macro avg       0.89      0.22      0.33     14234
                         weighted avg       0.89      0.25      0.36     14234
                          samples avg       0.40      0.22      0.26     14234

/opt/conda/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1318: Undefine
dMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels wit
h no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1318: Undefine
dMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples wi
th no predicted labels. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [ ]:

```python
# Exercise 3.3.1 Grid search

# import pipeline class
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
# import GridSearch
from sklearn.model_selection import GridSearchCV

# build a pipeline which does two steps all together:
# (1) generate tfidf, and (2) train classifier
# each step is named, i.e. "tfidf", "clf"
C=[0.5,1,5,10,50,100,1000]
for i in C:
    print('C=',i)
    text_clf = Pipeline([('tfidf', TfidfVectorizer()),
                         ('clf', OneVsRestClassifier(LinearSVC(C=i)))
```

```
                    ])

    # set the range of parameters to be tuned
    # each parameter is defined as
    # <step name>__<parameter name in step>
    # e.g. min_df is a parameter of TfidfVectorizer()
    # "tfidf" is the name for TfidfVectorizer()
    # therefore, 'tfidf__min_df' is the parameter in grid search

    parameters = {'tfidf__min_df':[1,2,4,3,5,10,6,7,8,9,15,20],
                  'tfidf__stop_words':[None,"english"]
    #             'clf__C': [0.5,1.0,2.0,3,4,5,6,7,8,9,10],
    }

    # the metric used to select the best parameters
    metric =  "f1_macro"

    # GridSearch also uses cross validation
    gs_clf = GridSearchCV\
    (text_clf, param_grid=parameters, \
     scoring=metric, cv=10)

    # due to data volume and large parameter combinations
    # it may take long time to search for optimal parameter combination
    # you can use a subset of data to test
    gs_clf = gs_clf.fit(X_train, Y_train)
    for param_name in gs_clf.best_params_:
        print("{0}:\t{1}".format(param_name,\
                                     gs_clf.best_params_[param_name]))

    print("best f1 score: {:.3f}".format(gs_clf.best_score_))
```

```
C= 0.5

tfidf__min_df:  1

tfidf__stop_words:      english

best f1 score: 0.710

C= 1
```

In [57]:

```
for param_name in gs_clf.best_params_:
    print("{0}:\t{1}".format(param_name,\
                                 gs_clf.best_params_[param_name]))

print("best f1 score: {:.3f}".format(gs_clf.best_score_))
```

```
tfidf__min_df:  20

tfidf__stop_words:      english

best f1 score: 0.450
```

In [ ]:

Installing essential packages

In [1]:
```
pip install transformers
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publ
ic/simple/
Collecting transformers
  Downloading transformers-4.25.1-py3-none-any.whl (5.8 MB)
     |████████████████████████████████| 5.8 MB 6.8 MB/s
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.8/dist-packag
es (from transformers) (2022.6.2)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.8/dist-packages (fr
om transformers) (1.21.6)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.8/dist-packages (fro
m transformers) (4.64.1)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from
transformers) (2.23.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/dist-packages
(from transformers) (21.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.8/dist-packages (from
transformers) (3.8.2)
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1
  Downloading tokenizers-0.13.2-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(7.6 MB)
     |████████████████████████████████| 7.6 MB 77.7 MB/s
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.8/dist-packages (fr
om transformers) (6.0)
Collecting huggingface-hub<1.0,>=0.10.0
  Downloading huggingface_hub-0.11.1-py3-none-any.whl (182 kB)
     |████████████████████████████████| 182 kB 91.4 MB/s
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.8/di
st-packages (from huggingface-hub<1.0,>=0.10.0->transformers) (4.4.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.8/dist
-packages (from packaging>=20.0->transformers) (3.0.9)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/li
b/python3.8/dist-packages (from requests->transformers) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (f
rom requests->transformers) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packa
ges (from requests->transformers) (2022.12.7)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.8/dist-packag
es (from requests->transformers) (3.0.4)
Installing collected packages: tokenizers, huggingface-hub, transformers
Successfully installed huggingface-hub-0.11.1 tokenizers-0.13.2 transformers-4.25.1
```

In [2]:
```
pip install torchmetrics
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publ
ic/simple/
Collecting torchmetrics
  Downloading torchmetrics-0.11.0-py3-none-any.whl (512 kB)
     |████████████████████████████████| 512 kB 8.5 MB/s
Requirement already satisfied: numpy>=1.17.2 in /usr/local/lib/python3.8/dist-packages
(from torchmetrics) (1.21.6)
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-packages (from
torchmetrics) (21.3)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.8/dist-packag
es (from torchmetrics) (4.4.0)
Requirement already satisfied: torch>=1.8.1 in /usr/local/lib/python3.8/dist-packages (f
rom torchmetrics) (1.13.0+cu116)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.8/dist
-packages (from packaging->torchmetrics) (3.0.9)
```

```
Installing collected packages: torchmetrics
Successfully installed torchmetrics-0.11.0
```

In [3]:
```
pip install torchvision
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publ
ic/simple/
Requirement already satisfied: torchvision in /usr/local/lib/python3.8/dist-packages (0.
14.0+cu116)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.8/dist-pa
ckages (from torchvision) (7.1.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.8/dist-packag
es (from torchvision) (4.4.0)
Requirement already satisfied: torch==1.13.0 in /usr/local/lib/python3.8/dist-packages
(from torchvision) (1.13.0+cu116)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from tor
chvision) (1.21.6)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from
torchvision) (2.23.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (f
rom requests->torchvision) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/li
b/python3.8/dist-packages (from requests->torchvision) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packa
ges (from requests->torchvision) (2022.12.7)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.8/dist-packag
es (from requests->torchvision) (3.0.4)
```

In [4]:
```
import random, pickle
import numpy as np
from torch.nn import BCEWithLogitsLoss, BCELoss
from sklearn.metrics import classification_report, confusion_matrix, multilabel_confusi
import tensorflow as tf
import torch
import pandas as pd
from torchmetrics.classification import MultilabelF1Score

from transformers import AutoConfig, AutoModel, AutoTokenizer, AutoModelForSequenceClas

import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import TensorDataset, random_split
from torch.utils.data import DataLoader, RandomSampler, SequentialSampler

import copy
from sklearn.utils import shuffle
import glob

import time
import datetime

import pandas as pd
import numpy as np
import io
```

In [5]:
```
# If there's a GPU available...
if torch.cuda.is_available():

    # Tell PyTorch to use the GPU.
```

```
    device = torch.device("cuda")

    print('There are %d GPU(s) available.' % torch.cuda.device_count())

    print('We will use the GPU:', torch.cuda.get_device_name(0))

# If not...
else:
    print('No GPU available, using the CPU instead.')
    device = torch.device("cpu")
```

```
There are 1 GPU(s) available.
We will use the GPU: A100-SXM4-40GB
```

In [6]:

```
from google.colab import drive

drive.mount('/content/gdrive/', force_remount=True)
```

```
Mounted at /content/gdrive/
```

In [7]:

```
path = "/content/gdrive/MyDrive/BIA667_FinalProject/train.csv"
```

In [8]:

```
# from google.colab import files


# uploaded = files.upload()
```

In [9]:

```
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer

from sklearn.model_selection import train_test_split
```

In [10]:

```
train_df = pd.read_csv(path)
# train_df = pd.read_csv('C:/Users/Admin/Documents/Stevens Docs/BIA 656/Project/train.c
```

Preprocessing data i.e is removing spec characters from the ttext so we have only words in the document

In [13]:

```
train_df['document_text']= train_df['document_text'].str.lower()
train_df['name']= train_df['name'].str.lower()
```

In [14]:

```
spec_chars = ["!",'"',"#","%","&","'","(",")",
              "*","+",",","-",".","/",":",";","<",
              "=",">","?","@","[","\\","]","^","_",
              "`","{","|","}","~","–"]
for char in spec_chars:
    train_df['document_text'] = train_df['document_text'].str.replace(char, ' ')
    train_df['name'] = train_df['name'].str.replace(char, ' ')
```

```
<ipython-input-14-4b1c388cc09c>:6: FutureWarning: The default value of regex will change
from True to False in a future version. In addition, single character regular expression
```

```
s will *not* be treated as literal strings when regex=True.
  train_df['document_text'] = train_df['document_text'].str.replace(char, ' ')
<ipython-input-14-4b1c388cc09c>:7: FutureWarning: The default value of regex will change
from True to False in a future version. In addition, single character regular expression
s will *not* be treated as literal strings when regex=True.
  train_df['name'] = train_df['name'].str.replace(char, ' ')
```

In [15]:
```
df2 = train_df.groupby(['name','document_text'])['cat_name'].apply(list)
df3=df2.to_frame().reset_index()
```

In [16]:
```
df3['comb_name_doctext'] = df3['name'] + df3['document_text'].apply(lambda x: '. ' + x)
```

In [17]:
```
df3.head()
```

Out[17]:

	name	document_text	cat_name	comb_name_doctext
0	correction to symbol information regarding t...	qutoutiao inc qtt will effect a one for ten...	[Broker Dealer]	correction to symbol information regarding t...
1	digital health 2020 eu on the move wojci...	european data protection supervisor published ...	[Research, Natural Disasters, Powers and Dutie...	digital health 2020 eu on the move wojci...
2	updated correction to merger consideration ...	the business combination of quidel corp qdel ...	[Broker Dealer, Corporate Communications]	updated correction to merger consideration ...
3	updated information regarding the business c...	the business combination of twc tech holdings ...	[Forms, Listing, Broker Dealer, Securities Set...	updated information regarding the business c...
4	updated closed information regarding the bus...	the business combination of mountain crest acq...	[Broker Dealer, Corporate Communications]	updated closed information regarding the bus...

In [18]:
```
len(df3)
```

Out[18]: 9263

Converting cat_name to array of binary class variables

In [19]:
```
mlb = MultiLabelBinarizer()
labels = df3['cat_name'].values
label_onehot = mlb.fit_transform(labels)
```

BERT for pretrained word vectors

In [20]:
```
# Load the BERT tokenizer.
print('Loading BERT tokenizer...')
#tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)


tokenizer = AutoTokenizer.from_pretrained(
```

```
        "nlpaueb/legal-bert-base-uncased"

    )
```

```
Loading BERT tokenizer...
```

In [21]:

```
bert_model = AutoModel.from_pretrained(
   # "bert-base-uncased",
   "nlpaueb/legal-bert-base-uncased",
    output_attentions = False, # Whether the model returns attentions weights.
    output_hidden_states = True, # Whether the model returns all hidden-states.

    )
bert_model.cuda()
```

```
Some weights of the model checkpoint at nlpaueb/legal-bert-base-uncased were not used wh
en initializing BertModel: ['cls.predictions.transform.dense.weight', 'cls.predictions.t
ransform.LayerNorm.bias', 'cls.predictions.decoder.bias', 'cls.predictions.bias', 'cls.s
eq_relationship.weight', 'cls.seq_relationship.bias', 'cls.predictions.transform.dense.b
ias', 'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.decoder.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a model trai
ned on another task or with another architecture (e.g. initializing a BertForSequenceCla
ssification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model
that you expect to be exactly identical (initializing a BertForSequenceClassification mo
del from a BertForSequenceClassification model).
```

Out[21]:

```
BertModel(
  (embeddings): BertEmbeddings(
    (word_embeddings): Embedding(30522, 768, padding_idx=0)
    (position_embeddings): Embedding(512, 768)
    (token_type_embeddings): Embedding(2, 768)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (encoder): BertEncoder(
    (layer): ModuleList(
      (0): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
          (intermediate_act_fn): GELUActivation()
        )
        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
```

```
        )
      )
      (1): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
          (intermediate_act_fn): GELUActivation()
        )
        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (2): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
          (intermediate_act_fn): GELUActivation()
        )
        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (3): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
```

```
        (dense): Linear(in_features=768, out_features=3072, bias=True)
        (intermediate_act_fn): GELUActivation()
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (4): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
        (intermediate_act_fn): GELUActivation()
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (5): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
        (intermediate_act_fn): GELUActivation()
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (6): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
```

```
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
          (intermediate_act_fn): GELUActivation()
        )
        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (7): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
          (intermediate_act_fn): GELUActivation()
        )
        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (8): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
          (intermediate_act_fn): GELUActivation()
        )
        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (9): BertLayer(
```

```
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (10): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (11): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
```

```
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
```

In [23]:

```python
import torch
from torch.utils.data import TensorDataset, DataLoader

def get_pretrained_word_vectors(sentences, tokenizer, bert_model):
    """
    Obtain the pretrained word vectors for the given sentences using a BERT model.

    Parameters:
        - sentences (list): List of sentences to process
        - tokenizer (object): Tokenizer object to use for encoding the sentences
        - bert_model (object): BERT model to use for generating the word vectors

    Returns:
        - token_embeddings (list): List of token embeddings for the input sentences
        - attention_masks (torch.Tensor): Tensor of attention masks for the input sente
        - input_ids (torch.Tensor): Tensor of input IDs for the input sentences
    """
    input_ids = []
    attention_masks = []

    # Tokenize each sentence and create input IDs and attention masks
    for sent in sentences:
        # Tokenize and create special tokens, pad or truncate, and create attention mas
        encoded_dict = tokenizer.encode_plus(
            sent,  # Sentence to encode
            add_special_tokens=True,  # Add '[CLS]' and '[SEP]'
            max_length=400,  # Pad & truncate all sentences
            truncation=True,
            padding='max_length',
            return_attention_mask=True,  # Construct attention masks
            return_tensors='pt',  # Return pytorch tensors
        )

        # Append the encoded sentence and attention mask to the lists
        input_ids.append(encoded_dict['input_ids'])
        attention_masks.append(encoded_dict['attention_mask'])

    # Convert lists to tensors
    input_ids = torch.cat(input_ids, dim=0)
    attention_masks = torch.cat(attention_masks, dim=0)

    # Create a TensorDataset and DataLoader for the input IDs and attention masks
    dataset = TensorDataset(input_ids, attention_masks)
    data_loader = DataLoader(dataset, batch_size=8, num_workers=2)

    token_embeddings = []
```

```
    # Set the model to evaluation mode
    bert_model.eval()
    with torch.no_grad():
        # Iterate over the data in the data loader
        for input_id, attention_mask in data_loader:
            # Get the hidden states of the model for the input IDs and attention masks
            outputs = bert_model(input_id.to(device), attention_mask.to(device))
            hidden_states = outputs[2]

            # Stack the last four hidden states and permute the dimensions
            emb = torch.stack(hidden_states[-4:], dim=0)
            emb = emb.permute(1, 2, 0, 3)

            # Take the mean of the last four hidden states along the third dimension
            emb = emb.mean(axis=2)

            # Append the token embeddings to the list
            # Append the token embeddings to the list
            token_embeddings.append(emb)

    return token_embeddings, attention_masks, input_ids
```

In [24]:

```
token_embeddings, masks, input_ids = get_pretrained_word_vectors(df3['comb_name_doctext
```

In [25]:

```
token_embeddings_splice = np.array([j.cpu().numpy() for i in token_embeddings for j in
token_embeddings_masked = (masks.view(-1, 400, 1) * token_embeddings_splice)
```

Dividing the data into train and test set with Y labels

In [27]:

```
X_train, X_test, Y_train, Y_test = train_test_split(token_embeddings_masked, label_oneh
```

In [28]:

```
from torch.utils.data import TensorDataset, random_split, Dataset
from torch.utils.data import DataLoader, RandomSampler, SequentialSampler
```

Creating Custom Dataset for it to be used in Neural Network

In [29]:

```
class Text_dataset(Dataset):
    def __init__(self, features, labels):
        self.length = len(labels)
        self.features = torch.Tensor(features)
        self.labels = torch.Tensor(labels)

    def __getitem__(self, index):
        return self.features[index], self.labels[index]

    def __len__(self):
        return self.length
```

In [30]:

```
# datasets
train_dataset = Text_dataset(X_train, Y_train)
test_dataset = Text_dataset(X_test, Y_test)
```

In [31]:
```
train_dataset.features.size()
```

Out[31]: torch.Size([7410, 400, 768])

Defining the RNN Model class which has unigram,bigram,trigram and classifier layers

In [32]:
```
class TextRNN(nn.Module):
    def __init__(self, DOC_LEN, embedding_dim, dropout_ratio):
        super(TextRNN, self).__init__()
        self.dropout_ratio = dropout_ratio
        in_channels = 400
        out_channels = 600
        bias = False

        # reduce the length
        self.reduce = nn.Sequential(
            nn.Linear(in_features=embedding_dim, out_features=in_channels),
            # nn.Dropout(dropout_ratio),
            nn.ReLU()
        )

        # unigram RNN
        self.unigram = nn.GRU(input_size=in_channels, hidden_size=out_channels, num_lay

        # bigram RNN
        self.bigram = nn.GRU(input_size=in_channels, hidden_size=out_channels, num_laye

        # trigram RNN
        self.trigram = nn.GRU(input_size=in_channels, hidden_size=out_channels, num_lay

        # simple classifier
        self.classifier = nn.Sequential(
            nn.Dropout(dropout_ratio),
            nn.Linear(in_features=out_channels*3, out_features=50)
        )

    def forward(self, x):
        # reduce length
        x = self.reduce(x)
        # print(x.shape)
        x = torch.transpose(x, dim0=1, dim1=2)  # (-1, DOC_LEN, embedding_dim): embeddi
        # print(x.shape)
        # unigram RNN output
        uni_gram_output, _ = self.unigram(x)
        uni_gram_output = uni_gram_output[:, -1, :]

        # bigram RNN output
        bi_gram_output, _ = self.bigram(x)
        bi_gram_output = bi_gram_output[:, -1, :]

        # trigram RNN output
        tri_gram_output, _ = self.trigram(x)
        tri_gram_output = tri_gram_output[:, -1, :]

        # concatenate
        x = torch.cat((uni_gram_output,
                       bi_gram_output,
```

```
                    tri_gram_output
                    ), dim=1)
        # classifier
        x = self.classifier(x)

        return x
```

In [33]:

```
model_RNN=TextRNN(400,768,0.5)
```

In [34]:

```
# Compute weights to handle imbalanced classes
class_counts = Y_train.sum(axis=0)
weights = [1/count for count in class_counts]
weights = [weight/sum(weights) for weight in weights]
weights = torch.Tensor(weights).to(device)
```

Defining train function for the model

In [35]:

```
def train_model(model, train_dataset, test_dataset, device, lr=0.0001, epochs=20, batch
    # construct dataloader
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=True)
    test_pred = DataLoader(test_dataset, batch_size=len(test_dataset))

    # move model to device
    model = model.to(device)

    # history
    history = {'train_loss': [],
               'train_f1': [],
               'test_loss': [],
               'test_f1': []
               }

    # setup loss function and optimizer
    criterion = nn.BCEWithLogitsLoss(reduction='none')
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    metric = MultilabelF1Score(num_labels=50).to(device)

    # training loop
    print('Training Start')
    for epoch in range(epochs):
        model.train()
        train_loss = 0
        train_f1 = 0
        test_loss = 0
        test_f1 = 0
        for x, y in train_loader:
            # move data to device
            x = x.to(device)
            y = y.to(device)
            # forward
            outputs = model(x.float())
            pred = torch.round(torch.sigmoid(outputs))
            cur_train_loss = criterion(outputs, y)
            cur_train_loss = (cur_train_loss * weights).mean()*100
            cur_train_f1 = metric(pred, y)
            # backward
```

```
                cur_train_loss.backward()
                optimizer.step()
                optimizer.zero_grad()
                # loss and acc
                train_loss += cur_train_loss
                train_f1 += cur_train_f1

            # test start
            model.eval()
            with torch.no_grad():
                for x, y in test_loader:
                    # move
                    x = x.to(device)
                    y = y.to(device)
                    # predict
                    outputs = model(x.float())
                    pred = torch.round(torch.sigmoid(outputs))
                    cur_test_loss = criterion(outputs, y).mean()
                    cur_test_loss = (cur_train_loss * weights).mean()*100
                    cur_test_f1 = metric(pred, y)
                    # loss and acc
                    test_loss += cur_test_loss
                    test_f1 += cur_test_f1

            # epoch output
            train_loss = (train_loss/len(train_loader)).item()
            train_f1 = (train_f1/len(train_loader)).item()
            val_loss = (test_loss/len(test_loader)).item()
            val_f1 = (test_f1/len(test_loader)).item()
            history['train_loss'].append(train_loss)
            history['train_f1'].append(train_f1)
            history['test_loss'].append(val_loss)
            history['test_f1'].append(val_f1)
            print(f"Epoch:{epoch + 1} / {epochs}, train loss:{train_loss:.4f} train f1:{tra


        with torch.no_grad():
            for x, y in test_pred:
                x = x.to(device)
                y = y.to(device)

                outputs = model(x.float())
                pred = torch.round(torch.sigmoid(outputs))
                preds = pred.cpu().detach().numpy()

        return history, preds
```

In [36]:

```
history, preds = train_model(model=model_RNN,
                             train_dataset=train_dataset,
                             test_dataset=test_dataset,
                             device=device,
                             lr=0.0005,
                             epochs=40,
                             batch_size=128)
```

```
Training Start
Epoch:1 / 40, train loss:0.6253 train f1:0.0038, valid loss:1.1084 valid f1:0.0000
Epoch:2 / 40, train loss:0.5372 train f1:0.0049, valid loss:0.9878 valid f1:0.0096
Epoch:3 / 40, train loss:0.4938 train f1:0.0622, valid loss:0.9451 valid f1:0.1249
```

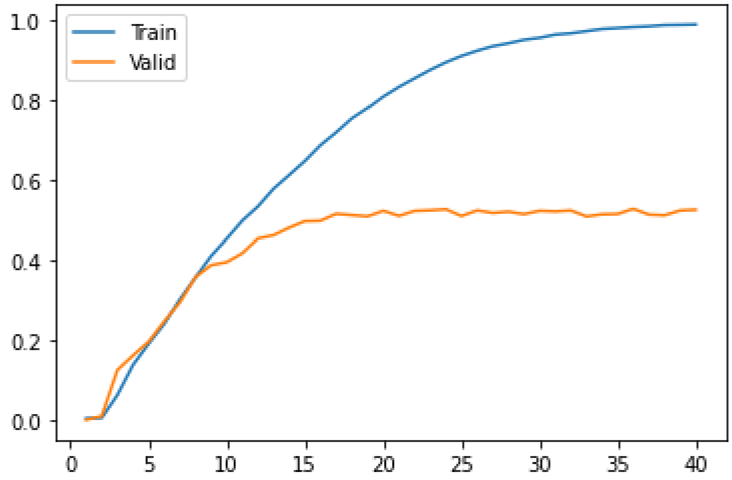
```
Epoch:4 / 40, train loss:0.4658 train f1:0.1375, valid loss:0.8998 valid f1:0.1615
Epoch:5 / 40, train loss:0.4440 train f1:0.1906, valid loss:0.9001 valid f1:0.1959
Epoch:6 / 40, train loss:0.4225 train f1:0.2403, valid loss:0.8345 valid f1:0.2467
Epoch:7 / 40, train loss:0.4011 train f1:0.3019, valid loss:0.7973 valid f1:0.2943
Epoch:8 / 40, train loss:0.3804 train f1:0.3566, valid loss:0.7716 valid f1:0.3578
Epoch:9 / 40, train loss:0.3592 train f1:0.4102, valid loss:0.6918 valid f1:0.3873
Epoch:10 / 40, train loss:0.3423 train f1:0.4531, valid loss:0.7030 valid f1:0.3948
Epoch:11 / 40, train loss:0.3221 train f1:0.4983, valid loss:0.6898 valid f1:0.4171
Epoch:12 / 40, train loss:0.3038 train f1:0.5337, valid loss:0.6222 valid f1:0.4537
Epoch:13 / 40, train loss:0.2822 train f1:0.5780, valid loss:0.5735 valid f1:0.4620
Epoch:14 / 40, train loss:0.2624 train f1:0.6123, valid loss:0.5143 valid f1:0.4806
Epoch:15 / 40, train loss:0.2450 train f1:0.6471, valid loss:0.4902 valid f1:0.4968
Epoch:16 / 40, train loss:0.2229 train f1:0.6869, valid loss:0.4142 valid f1:0.4978
Epoch:17 / 40, train loss:0.2048 train f1:0.7187, valid loss:0.3707 valid f1:0.5149
Epoch:18 / 40, train loss:0.1858 train f1:0.7538, valid loss:0.3747 valid f1:0.5120
Epoch:19 / 40, train loss:0.1693 train f1:0.7797, valid loss:0.3213 valid f1:0.5088
Epoch:20 / 40, train loss:0.1511 train f1:0.8087, valid loss:0.3093 valid f1:0.5226
Epoch:21 / 40, train loss:0.1365 train f1:0.8329, valid loss:0.2539 valid f1:0.5099
Epoch:22 / 40, train loss:0.1212 train f1:0.8544, valid loss:0.2619 valid f1:0.5223
Epoch:23 / 40, train loss:0.1070 train f1:0.8755, valid loss:0.1923 valid f1:0.5237
Epoch:24 / 40, train loss:0.0943 train f1:0.8944, valid loss:0.1921 valid f1:0.5263
Epoch:25 / 40, train loss:0.0839 train f1:0.9104, valid loss:0.1546 valid f1:0.5095
Epoch:26 / 40, train loss:0.0735 train f1:0.9234, valid loss:0.1646 valid f1:0.5235
Epoch:27 / 40, train loss:0.0656 train f1:0.9347, valid loss:0.1426 valid f1:0.5173
Epoch:28 / 40, train loss:0.0592 train f1:0.9421, valid loss:0.1163 valid f1:0.5203
Epoch:29 / 40, train loss:0.0523 train f1:0.9508, valid loss:0.1046 valid f1:0.5142
Epoch:30 / 40, train loss:0.0475 train f1:0.9558, valid loss:0.0949 valid f1:0.5225
Epoch:31 / 40, train loss:0.0417 train f1:0.9638, valid loss:0.0809 valid f1:0.5210
Epoch:32 / 40, train loss:0.0382 train f1:0.9671, valid loss:0.0759 valid f1:0.5234
Epoch:33 / 40, train loss:0.0335 train f1:0.9726, valid loss:0.0631 valid f1:0.5085
Epoch:34 / 40, train loss:0.0295 train f1:0.9780, valid loss:0.0642 valid f1:0.5137
Epoch:35 / 40, train loss:0.0266 train f1:0.9803, valid loss:0.0500 valid f1:0.5147
Epoch:36 / 40, train loss:0.0239 train f1:0.9829, valid loss:0.0435 valid f1:0.5275
Epoch:37 / 40, train loss:0.0218 train f1:0.9847, valid loss:0.0489 valid f1:0.5130
Epoch:38 / 40, train loss:0.0192 train f1:0.9878, valid loss:0.0369 valid f1:0.5113
Epoch:39 / 40, train loss:0.0183 train f1:0.9884, valid loss:0.0437 valid f1:0.5234
Epoch:40 / 40, train loss:0.0167 train f1:0.9893, valid loss:0.0405 valid f1:0.5252
```

Plot for F1 score for RNN model

In [37]:

```python
import matplotlib.pyplot as plt
plt.plot(range(1, 41), history['train_f1'], label='Train')
plt.plot(range(1, 41), history['test_f1'], label='Valid')
plt.legend()
plt.plot()
```

Out[37]: []



In [38]:

```
#ModelRNN
from sklearn.metrics import classification_report

print(classification_report(Y_test, preds))
```

```
              precision    recall  f1-score   support

           0       0.60      0.41      0.49       191
           1       0.79      0.64      0.71       171
           2       0.55      0.49      0.52       194
           3       0.67      0.48      0.56       134
           4       0.74      0.67      0.70       104
           5       0.67      0.46      0.54       264
           6       0.69      0.39      0.49       187
           7       0.78      0.58      0.67       219
           8       0.70      0.29      0.41       112
           9       0.56      0.26      0.36       177
          10       0.34      0.20      0.25       102
          11       0.81      0.62      0.70       147
          12       0.66      0.46      0.54       344
          13       0.53      0.37      0.43       229
          14       0.55      0.51      0.53       223
          15       0.62      0.43      0.51       109
          16       0.74      0.60      0.66       218
          17       0.76      0.32      0.45       101
          18       0.71      0.65      0.68       165
          19       0.43      0.29      0.34        73
          20       0.56      0.43      0.49       261
          21       0.71      0.56      0.62       149
          22       0.45      0.17      0.24       175
          23       0.71      0.47      0.57       259
          24       0.55      0.40      0.46       184
          25       0.59      0.38      0.46       178
          26       0.63      0.39      0.48       101
          27       0.74      0.55      0.63       203
          28       0.87      0.65      0.75       138
          29       0.70      0.53      0.60       319
          30       0.59      0.35      0.44       152
          31       0.51      0.38      0.43       169
          32       0.81      0.54      0.65        89
          33       0.68      0.61      0.64       199
          34       0.85      0.59      0.70       200
          35       0.32      0.08      0.13       143
          36       0.67      0.58      0.62       111
          37       0.59      0.34      0.43       112
          38       0.73      0.53      0.61       304
```

```
          39       0.61      0.41      0.49       179
          40       0.46      0.24      0.32       134
          41       0.44      0.18      0.26       121
          42       0.61      0.55      0.58       172
          43       0.88      0.76      0.82       204
          44       0.66      0.47      0.55       192
          45       0.82      0.62      0.70       123
          46       0.70      0.66      0.68       308
          47       0.77      0.60      0.67       152
          48       0.68      0.54      0.60       157
          49       0.84      0.60      0.70       144

   micro avg       0.67      0.48      0.56      8796
   macro avg       0.65      0.47      0.54      8796
weighted avg       0.66      0.48      0.55      8796
 samples avg       0.58      0.45      0.48      8796

/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: Undefine
dMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples wi
th no predicted labels. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Defining advanced model with CNN with unigram,bigram,trigram,quadgram,pentagram and one convolution layer with max pooling

In [39]:

```
class TextCNN(nn.Module):
    def __init__(self, DOC_LEN, embedding_dim, dropout_ratio):
        super(TextCNN, self).__init__()
        self.DOC_LEN = DOC_LEN
        self.in_channels = 400
        self.out_channels = 600
        self.bias = False

        # reduce the length
        self.reduce_length = nn.Sequential(
            nn.Linear(in_features=embedding_dim, out_features=self.in_channels),
            nn.ReLU()
        )

        # 1D CNNs for unigrams, bigrams, trigrams, quadgrams, and pentagrams
        self.unigram_cnn = self._build_1d_cnn(kernel_size=1)
        self.bigram_cnn = self._build_1d_cnn(kernel_size=2)
        self.trigram_cnn = self._build_1d_cnn(kernel_size=3)
        self.quadgram_cnn = self._build_1d_cnn(kernel_size=4)
        self.pentagram_cnn = self._build_1d_cnn(kernel_size=5)

        # simple classifier
        self.classifier = nn.Sequential(
            nn.Dropout(dropout_ratio),
            nn.Linear(in_features=self.out_channels*5, out_features=50)
        )

    def _build_1d_cnn(self, kernel_size):
        """Helper function to build 1D CNN layers for a given kernel size."""
        return nn.Sequential(
            nn.Conv1d(in_channels=self.in_channels, out_channels=self.out_channels, ker
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=self.DOC_LEN - kernel_size + 1),
            nn.Flatten()
        )
```

```
    def forward(self, x):
        # reduce length
        x = self.reduce_length(x)
        x = torch.transpose(x, dim0=1, dim1=2)  # (-1, DOC_LEN, embedding_dim)

        # 1D CNN outputs
        unigram_output = self.unigram_cnn(x)
        bigram_output = self.bigram_cnn(x)
        trigram_output = self.trigram_cnn(x)
        quadgram_output = self.quadgram_cnn(x)
        pentagram_output = self.pentagram_cnn(x)

        # concatenate 1D CNN outputs
        x = torch.cat((unigram_output, bigram_output, trigram_output, quadgram_output,

        # classifier

        x = self.classifier(x)

        return x
```

In [40]:

```
model_CNN=TextCNN(400,768,0.5)
```

In [41]:

```
history, preds = train_model(model=model_CNN,
                             train_dataset=train_dataset,
                             test_dataset=test_dataset,
                             device=device,
                             lr=0.0005,
                             epochs=40,
                             batch_size=128)
```

```
Training Start
Epoch:1 / 40, train loss:0.5925 train f1:0.0026, valid loss:1.0962 valid f1:0.0000
Epoch:2 / 40, train loss:0.5010 train f1:0.0495, valid loss:0.8874 valid f1:0.1518
Epoch:3 / 40, train loss:0.4129 train f1:0.2651, valid loss:0.8092 valid f1:0.3788
Epoch:4 / 40, train loss:0.3421 train f1:0.4463, valid loss:0.6782 valid f1:0.5184
Epoch:5 / 40, train loss:0.2919 train f1:0.5603, valid loss:0.5135 valid f1:0.6041
Epoch:6 / 40, train loss:0.2566 train f1:0.6342, valid loss:0.4853 valid f1:0.6382
Epoch:7 / 40, train loss:0.2315 train f1:0.6801, valid loss:0.4338 valid f1:0.6796
Epoch:8 / 40, train loss:0.2086 train f1:0.7119, valid loss:0.3959 valid f1:0.6945
Epoch:9 / 40, train loss:0.1907 train f1:0.7413, valid loss:0.3563 valid f1:0.7316
Epoch:10 / 40, train loss:0.1748 train f1:0.7681, valid loss:0.3440 valid f1:0.7205
Epoch:11 / 40, train loss:0.1610 train f1:0.7883, valid loss:0.3082 valid f1:0.7485
Epoch:12 / 40, train loss:0.1480 train f1:0.8048, valid loss:0.3221 valid f1:0.7592
Epoch:13 / 40, train loss:0.1361 train f1:0.8246, valid loss:0.2428 valid f1:0.7642
Epoch:14 / 40, train loss:0.1249 train f1:0.8395, valid loss:0.2354 valid f1:0.7580
Epoch:15 / 40, train loss:0.1149 train f1:0.8553, valid loss:0.2244 valid f1:0.7655
Epoch:16 / 40, train loss:0.1059 train f1:0.8679, valid loss:0.1923 valid f1:0.7702
Epoch:17 / 40, train loss:0.0949 train f1:0.8812, valid loss:0.2187 valid f1:0.7773
Epoch:18 / 40, train loss:0.0884 train f1:0.8930, valid loss:0.1699 valid f1:0.7858
Epoch:19 / 40, train loss:0.0787 train f1:0.9066, valid loss:0.1741 valid f1:0.7783
Epoch:20 / 40, train loss:0.0724 train f1:0.9145, valid loss:0.1458 valid f1:0.7902
Epoch:21 / 40, train loss:0.0674 train f1:0.9209, valid loss:0.1417 valid f1:0.7780
Epoch:22 / 40, train loss:0.0630 train f1:0.9289, valid loss:0.1141 valid f1:0.7927
Epoch:23 / 40, train loss:0.0580 train f1:0.9320, valid loss:0.1295 valid f1:0.7958
Epoch:24 / 40, train loss:0.0535 train f1:0.9392, valid loss:0.1321 valid f1:0.7924
Epoch:25 / 40, train loss:0.0490 train f1:0.9452, valid loss:0.0996 valid f1:0.7954
Epoch:26 / 40, train loss:0.0450 train f1:0.9501, valid loss:0.0774 valid f1:0.7967
Epoch:27 / 40, train loss:0.0417 train f1:0.9549, valid loss:0.0924 valid f1:0.8037
```

```
Epoch:28 / 40, train loss:0.0379 train f1:0.9601, valid loss:0.0903 valid f1:0.7953
Epoch:29 / 40, train loss:0.0367 train f1:0.9612, valid loss:0.0712 valid f1:0.7898
Epoch:30 / 40, train loss:0.0342 train f1:0.9641, valid loss:0.0571 valid f1:0.8014
Epoch:31 / 40, train loss:0.0320 train f1:0.9673, valid loss:0.0751 valid f1:0.8034
Epoch:32 / 40, train loss:0.0298 train f1:0.9705, valid loss:0.0548 valid f1:0.7976
Epoch:33 / 40, train loss:0.0281 train f1:0.9718, valid loss:0.0609 valid f1:0.8036
Epoch:34 / 40, train loss:0.0270 train f1:0.9725, valid loss:0.0461 valid f1:0.8048
Epoch:35 / 40, train loss:0.0259 train f1:0.9739, valid loss:0.0540 valid f1:0.7946
Epoch:36 / 40, train loss:0.0235 train f1:0.9768, valid loss:0.0429 valid f1:0.8055
Epoch:37 / 40, train loss:0.0230 train f1:0.9777, valid loss:0.0411 valid f1:0.7991
Epoch:38 / 40, train loss:0.0223 train f1:0.9772, valid loss:0.0360 valid f1:0.7951
Epoch:39 / 40, train loss:0.0217 train f1:0.9785, valid loss:0.0426 valid f1:0.7927
Epoch:40 / 40, train loss:0.0194 train f1:0.9813, valid loss:0.0341 valid f1:0.8020
```

Classification Report for CNN with accuracy of 81%

In [42]:

```python
#ModelCNN
from sklearn.metrics import classification_report

print(classification_report(Y_test, preds))
```

```
              precision    recall  f1-score   support

           0       0.95      0.70      0.80       191
           1       0.97      0.78      0.87       171
           2       0.94      0.83      0.88       194
           3       0.97      0.85      0.90       134
           4       0.93      0.88      0.91       104
           5       0.92      0.88      0.90       264
           6       0.92      0.78      0.85       187
           7       0.91      0.83      0.86       219
           8       0.87      0.64      0.74       112
           9       0.85      0.54      0.66       177
          10       0.91      0.69      0.78       102
          11       0.92      0.79      0.85       147
          12       0.83      0.65      0.73       344
          13       0.94      0.86      0.89       229
          14       0.98      0.94      0.96       223
          15       0.96      0.85      0.90       109
          16       0.97      0.78      0.87       218
          17       0.86      0.73      0.79       101
          18       0.95      0.85      0.89       165
          19       0.84      0.59      0.69        73
          20       0.86      0.64      0.73       261
          21       0.94      0.86      0.90       149
          22       0.80      0.50      0.62       175
          23       0.83      0.75      0.79       259
          24       0.83      0.74      0.78       184
          25       0.88      0.82      0.85       178
          26       0.93      0.70      0.80       101
          27       0.93      0.83      0.88       203
          28       0.96      0.87      0.91       138
          29       0.86      0.65      0.74       319
          30       0.93      0.68      0.79       152
          31       0.85      0.60      0.70       169
          32       0.97      0.75      0.85        89
          33       0.93      0.91      0.92       199
          34       0.92      0.80      0.86       200
          35       0.85      0.24      0.37       143
          36       0.95      0.91      0.93       111
          37       0.88      0.60      0.71       112
          38       0.85      0.73      0.79       304
          39       0.83      0.72      0.77       179
          40       0.83      0.60      0.70       134
```

```
          41       0.94      0.50      0.66       121
          42       0.81      0.80      0.80       172
          43       0.94      0.94      0.94       204
          44       0.88      0.73      0.80       192
          45       0.95      0.76      0.85       123
          46       0.85      0.80      0.82       308
          47       0.90      0.88      0.89       152
          48       0.97      0.79      0.87       157
          49       0.90      0.86      0.88       144

   micro avg       0.90      0.75      0.82      8796
   macro avg       0.90      0.75      0.81      8796
weighted avg       0.90      0.75      0.81      8796
 samples avg       0.86      0.74      0.78      8796

/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: Undefine
dMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples wi
th no predicted labels. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [ ]: