Analyse and visualise a dataset

Import Packages

In [2]:

```
import pandas as pd
import gensim
from gensim.utils import simple_preprocess
import nltk
from nltk.tokenize import RegexpTokenizer
import spacy
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score, multilabel_confusion_matrix, ConfusionMatrixDi
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional
from tensorflow.keras.callbacks import EarlyStopping, Callback, ModelCheckpoint
import tensorflow as tf
import tensorflow_addons as tfa
import kerastuner as kt
from joblib import dump, load
import time
import random
import matplotlib
import matplotlib.pyplot as plt
import os.path
import warnings
```

Load Data

In [ ]:

```
data = pd.read_csv('DBP_wiki_data.csv')
train = pd.read_csv('DBPEDIA_train.csv')
val = pd.read_csv('DBPEDIA_val.csv')
test = pd.read_csv('DBPEDIA_test.csv')
```

Visualize and Analyse Data

In [ ]:

```
data.head()
```

Out[ ]:

	text	l1	l2	l3	wiki_name	word_count
0	The 1994 Mindoro earthquake occurred on Novemb...	Event	NaturalEvent	Earthquake	1994_Mindoro_earthquake	59
1	The 1917 Bali earthquake occurred at 06:50 loc...	Event	NaturalEvent	Earthquake	1917_Bali_earthquake	68

	text	l1	l2	l3	wiki_name	word_count
2	The 1941 Colima earthquake occurred on April 1...	Event	NaturalEvent	Earthquake	1941_Colima_earthquake	194
3	The 1983 Coalinga earthquake occurred on May 2...	Event	NaturalEvent	Earthquake	1983_Coalinga_earthquake	98
4	The 2013 Bushehr earthquake occurred with a mo...	Event	NaturalEvent	Earthquake	2013_Bushehr_earthquake	61

Numbers of data in each file are followings.

In [ ]:

```
print("Number of data: ", len(data))
print("Number of train data: ", len(train))
print("Number of validation data: ", len(val))
print("Number of test data: ", len(test))
```

```
Number of data:  342781
Number of train data:  240942
Number of validation data:  36003
Number of test data:  60794
```

There are three levels of classes.

In [ ]:

```
print("Number of classes in level 1:", len(data.l1.unique()))
print("Number of classes in level 2:", len(data.l2.unique()))
print("Number of classes in level 3:", len(data.l3.unique()))
```

```
Number of classes in level 1: 9
Number of classes in level 2: 70
Number of classes in level 3: 219
```

Numbers of data for each category are followings.

In [ ]:

```
data.l1.value_counts()
```

Out[ ]:

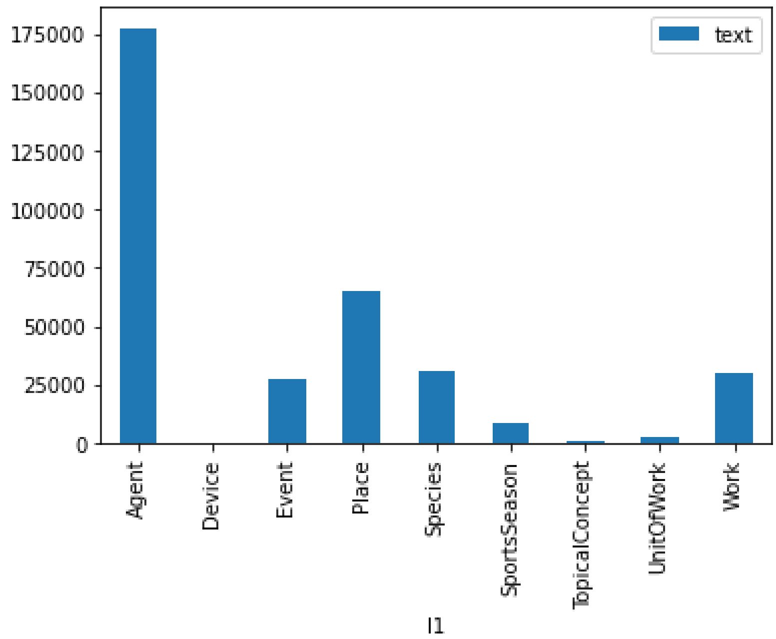
```
Agent             177341
Place              65128
Species            31149
Work               29832
Event              27059
SportsSeason        8307
UnitOfWork          2497
TopicalConcept      1115
Device               353
Name: l1, dtype: int64
```

In [ ]:

```
data[['text','l1']].groupby('l1').count().plot(kind='bar')
```

Out[ ]:

```
<AxesSubplot:xlabel='l1'>
```

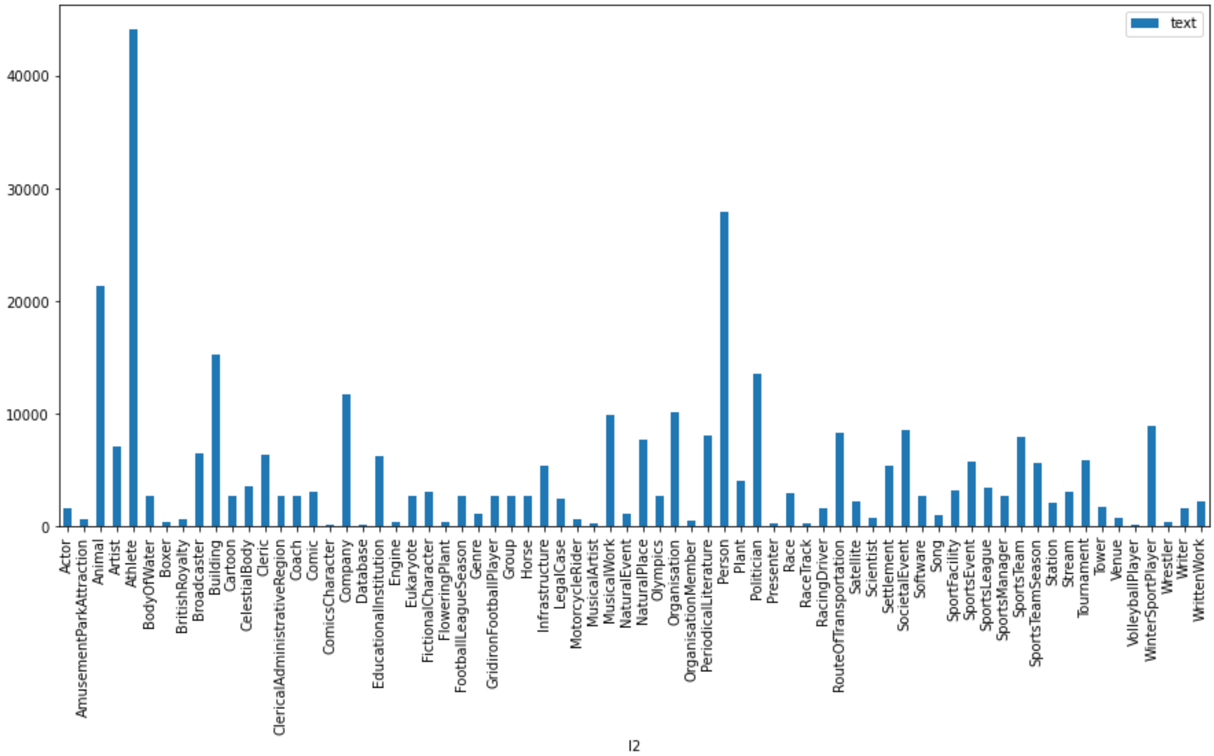



```
In [ ]: data.l2.value_counts()
```

```
Out[ ]: Athlete            44163
        Person             27892
        Animal             21333
        Building           15266
        Politician         13514
                           ...
        MusicalArtist        284
        RaceTrack            242
        ComicsCharacter      203
        VolleyballPlayer     194
        Database             187
        Name: l2, Length: 70, dtype: int64
```

```
In [ ]: data[['text','l2']].groupby('l2').count().plot(kind='bar', figsize=(15,7))
```

```
Out[ ]: <AxesSubplot:xlabel='l2'>
```



In [ ]:

```
data.l3.value_counts()
```

Out[ ]:

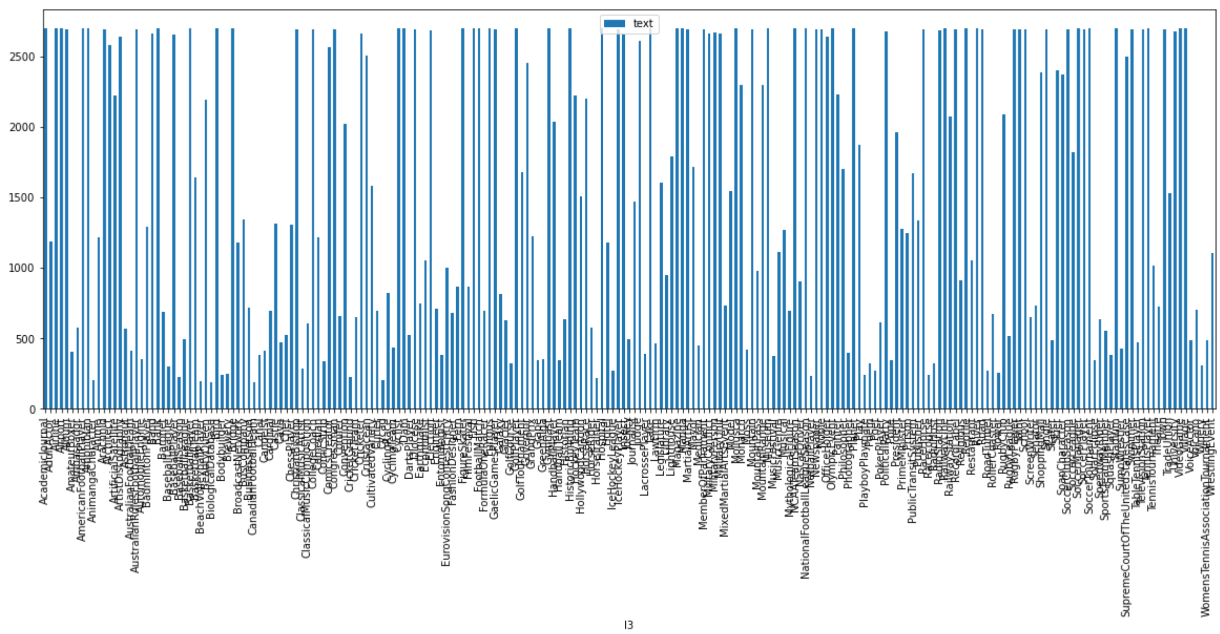
```
GolfPlayer              2700
Planet                  2700
AcademicJournal         2700
FootballMatch           2700
Manga                   2700
                        ...
Cycad                    204
AnimangaCharacter        203
BeachVolleyballPlayer    194
CanadianFootballTeam     190
BiologicalDatabase       187
Name: l3, Length: 219, dtype: int64
```

In [ ]:

```
data[['text','l3']].groupby('l3').count().plot(kind='bar', figsize=(20,7))
```

Out[ ]:

```
<AxesSubplot:xlabel='l3'>
```

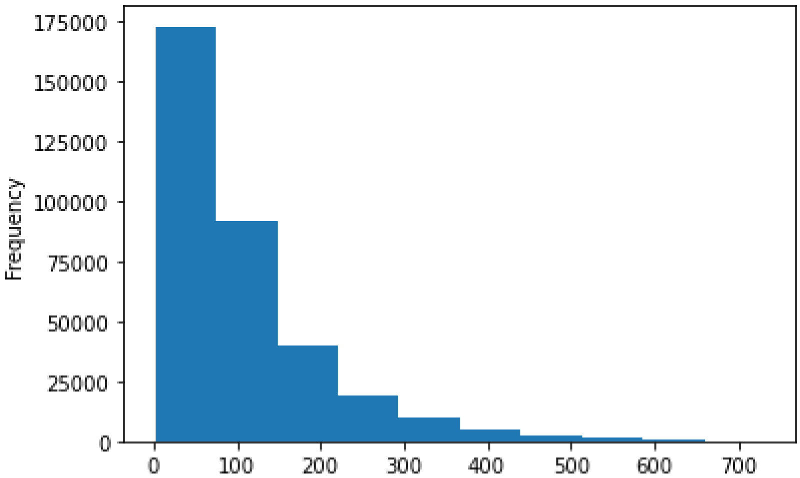


As you can see in the graphs above, the data is imbalanced. The data can be balanced before training as one of the step in the data pre-processing. However, the raw data is going to be used for the experiments this time. F1-score metric will be used to deal with the imbalanced data.

In [ ]:
```
wordcount = data.text.str.split().str.len()
```

In [ ]:
```
wordcount.plot(kind='hist')
```

Out[ ]:
```
<AxesSubplot:ylabel='Frequency'>
```



In [ ]:
```
print("Maximum word count: ", wordcount.max())
print("Minimum word count: ", wordcount.min())
print("Average word count: ", int(wordcount.mean()))
```

```
Maximum word count:  732
Minimum word count:  2
Average word count:  105
```

As you can see above, the text lengths are varied.

Followings are some examples of texts in the dataset.

In [ ]:
```
for n, i in enumerate(random.sample(range(0, len(data)), 5)):
    print('Example {}'.format(n+1))
    print()
    print('Text:')
    print(data.text[i])
    print()
    print('Classes:')
    print(data[['l1','l2','l3']].to_numpy()[i])
    print()
    print()
```

```
Example 1

Text:
Lover's Flat (Japanese: 1Kアパートの恋 Hepburn: Wan-kei Apāto no Koi) is a Japanese m
anga written and illustrated by Hyouta Fujiyama. It is licensed in North America by
Digital Manga Publishing, which released the manga through its imprint, Juné, on 1 A
ugust 2007. It looks at two couples who are neighbours in an apartment complex.

Classes:
['Work' 'Comic' 'Manga']
```

```
Example 2

Text:
Aquila is an educational children's magazine that offers an alternative to mainstrea
m publications. It is for boys and girls of 8-13 and features puzzles, fun facts and
activities - and is advert-free. Each issue revolves mainly around a specific topic,
for example Captain Cook, Science Special, The Equator and Medieval Times - all cove
red in 2013. The \"lively and informative\" magazine is aimed at bright pre-teenager
s interested in hobbies beyond pop music and soaps, who \"need to be able to feel go
od about themselves\" and to realise that \"there are other children out there like
them\" according to D J Taylor's article in the Telegraph in 2003. It was establishe
d in 1993 and is owned and run by New Leaf Publishing Ltd, a small independent publi
shing house situated in the coastal town of Eastbourne in the UK. ATE Superweeks, a
UK summer camp provider, works in association with Aquila magazine to run an annual
summer camp. In 2012 the camp was called The Eco-Venture and had a focus on the envi
ronment.

Classes:
['Work' 'PeriodicalLiterature' 'Magazine']


Example 3

Text:
Karl-Anthony Towns Jr. (born November 15, 1995) is a Dominican-American professional
basketball player for the Minnesota Timberwolves of the National Basketball Associat
ion (NBA). He played college basketball for the University of Kentucky. Towns was na
med to the Dominican Republic national basketball team Olympic squad as a 16-year-ol
d, although the Dominican Republic ultimately did not qualify for the 2012 Olympics.
He was selected with the first overall pick in the 2015 NBA draft by the Minnesota T
imberwolves, and went on to be named NBA Rookie of the Year for the 2015–16 season.

Classes:
['Agent' 'Athlete' 'BasketballPlayer']


Example 4

Text:
Ghosts Upon the Earth is the second album by Christian band Gungor and the seventh a
lbum self-produced by singer Michael Gungor. This album received a nomination at 54t
h Grammy Awards for Best Contemporary Christian Music Album.

Classes:
['Work' 'MusicalWork' 'Album']


Example 5

Text:
Sivali was Queen of Anuradhapura in the 1st century, whose reign lasted the year 35.
She succeeded her brother Chulabhaya as Queen of Anuradhapura and was succeeded by h
er nephew Ilanaga, after an interregnum.

Classes:
['Agent' 'Person' 'Monarch']
```

There are many punctuations in the texts and some articles include non-English words.

Experiment

Four different experiment setups are following:

1. Number of classes
    - Level 1 (9 classes)
    - Level 2 (70 classes)
    - Level 3 (219 classes)
    - Level 1 and 2 (79 classes)
    - All levels (298 classes)

1. Data Preprocessing
    - Unigram
    - Bigram
    - Trigram

1. Classifiers
    - K-Nearest Neighbours (kNN)
    - Recurrent Neural Network (RNN)

1. Hyperparameters
    - kNN
        - Number of neighbours: 1, 3, 5, 10
        - Weight functions: uniform, distance
    - RNN
        - Number of epochs
        - Batch size: 32, 64, 128
        - Embedding dimension: 128, 256, 512
        - Hidden dimension: 128, 256, 512
        - Dropout rate: 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
        - Recurrent dropout rate: 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
        - Single layer/Multi layer
        - Unidirectional/Bidirectional
        - Optimizers: SGD, Adam, RMSprop

Two classifiers, kNN and RNN, will be built, trained, tested, and evaluated separately and the best results of each classifier will be compared at the end.

K-Nearest Neighbours (kNN)

Preprocess Data

Texts

First, experiment different approachs to remove non-english words from the texts. The following text in the train dataset is used for the experiment.

In [ ]:

```
text = train['text'].iloc[155]
print(text)
```

```
Blue Hole (ブルー　ホール Burū Hōru) is a science fiction manga series by Yukinobu Hos
hino involving dinosaurs living in the present. Its title in France is Le Trou Bleu.
It was serialized in Mister Magazine from 1991 to 1992 with two tankōbon published.
```

The text is first tokenized using RegexpTokenizer in the nltk package, which splits a string into substrings using a regular expression.

In [ ]:
```
tokenizer = RegexpTokenizer(r"\w+")
tokens = tokenizer.tokenize(text)
print(tokens)
```

```
['Blue', 'Hole', 'ブルー', 'ホール', 'Burū', 'Hōru', 'is', 'a', 'science', 'fiction',
'manga', 'series', 'by', 'Yukinobu', 'Hoshino', 'involving', 'dinosaurs', 'living',
'in', 'the', 'present', 'Its', 'title', 'in', 'France', 'is', 'Le', 'Trou', 'Bleu',
'It', 'was', 'serialized', 'in', 'Mister', 'Magazine', 'from', '1991', 'to', '1992',
'with', 'two', 'tankōbon', 'published']
```

Then, remove non-English words by removing words that exist in nltk.corpus.words, which contains a list of English words.

In [ ]:
```
english_vocab = set(w.lower() for w in nltk.corpus.words.words())
unusual = set(tokens) - english_vocab
rest = set(tokens) - unusual
print("Unusual vocabularies:")
print(unusual)
print()
print("Remaining vocavularies:")
print(rest)
```

```
Unusual vocabularies:
{'1991', 'Bleu', 'Its', 'Magazine', 'ホール', 'Hole', 'Burū', '1992', 'serialized',
'involving', 'Le', 'dinosaurs', 'ブルー', 'Blue', 'Yukinobu', 'It', 'published', 'Mis
ter', 'France', 'Trou', 'Hōru', 'Hoshino', 'tankōbon'}

Remaining vocavularies:
{'in', 'science', 'the', 'living', 'present', 'fiction', 'is', 'by', 'two', 'a', 'wi
th', 'from', 'was', 'to', 'series', 'manga', 'title'}
```

It removes non-English words but it also removes English words, such as involving and published.

Next approach is to use spacy. First, load spacy English model.

In [ ]:
```
!python -m spacy download en_core_web_sm
```

```
Collecting en_core_web_sm==2.3.1
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_we
b_sm-2.3.1/en_core_web_sm-2.3.1.tar.gz (12.0 MB)
     |████████████████████████████████| 12.0 MB 12.0 MB/s eta 0:00:01
Requirement already satisfied: spacy<2.4.0,>=2.3.0 in /user/HS225/rf00302/.conda/env
s/nlp2021/lib/python3.8/site-packages (from en_core_web_sm==2.3.1) (2.3.5)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /user/HS225/rf00302/.conda/e
nvs/nlp2021/lib/python3.8/site-packages (from spacy<2.4.0,>=2.3.0->en_core_web_sm==
2.3.1) (3.0.2)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /user/HS225/rf00302/.con
da/envs/nlp2021/lib/python3.8/site-packages (from spacy<2.4.0,>=2.3.0->en_core_web_s
m==2.3.1) (1.0.5)
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in /user/HS225/rf00302/.conda/env
s/nlp2021/lib/python3.8/site-packages (from spacy<2.4.0,>=2.3.0->en_core_web_sm==2.
3.1) (1.0.5)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /user/HS225/rf00302/.cond
a/envs/nlp2021/lib/python3.8/site-packages (from spacy<2.4.0,>=2.3.0->en_core_web_sm
==2.3.1) (2.25.1)
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in /user/HS225/rf00302/.cond
a/envs/nlp2021/lib/python3.8/site-packages (from spacy<2.4.0,>=2.3.0->en_core_web_sm
==2.3.1) (1.0.0)
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in /user/HS225/rf00302/.conda/en
```

```
vs/nlp2021/lib/python3.8/site-packages (from spacy<2.4.0,>=2.3.0->en_core_web_sm==2.
3.1) (0.8.2)
Requirement already satisfied: plac<1.2.0,>=0.9.6 in /user/HS225/rf00302/.conda/env
s/nlp2021/lib/python3.8/site-packages (from spacy<2.4.0,>=2.3.0->en_core_web_sm==2.
3.1) (1.1.0)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /user/HS225/rf00302/.conda/env
s/nlp2021/lib/python3.8/site-packages (from spacy<2.4.0,>=2.3.0->en_core_web_sm==2.
3.1) (4.60.0)
Requirement already satisfied: setuptools in /user/HS225/rf00302/.conda/envs/nlp202
1/lib/python3.8/site-packages (from spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (52.
0.0.post20210125)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /user/HS225/rf00302/.conda/env
s/nlp2021/lib/python3.8/site-packages (from spacy<2.4.0,>=2.3.0->en_core_web_sm==2.
3.1) (2.0.5)
Requirement already satisfied: numpy>=1.15.0 in /user/HS225/rf00302/.conda/envs/nlp2
021/lib/python3.8/site-packages (from spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1)
(1.20.2)
Requirement already satisfied: thinc<7.5.0,>=7.4.1 in /user/HS225/rf00302/.conda/env
s/nlp2021/lib/python3.8/site-packages (from spacy<2.4.0,>=2.3.0->en_core_web_sm==2.
3.1) (7.4.5)
Requirement already satisfied: blis<0.8.0,>=0.4.0 in /user/HS225/rf00302/.conda/env
s/nlp2021/lib/python3.8/site-packages (from spacy<2.4.0,>=2.3.0->en_core_web_sm==2.
3.1) (0.7.4)
Requirement already satisfied: chardet<5,>=3.0.2 in /user/HS225/rf00302/.conda/envs/
nlp2021/lib/python3.8/site-packages (from requests<3.0.0,>=2.13.0->spacy<2.4.0,>=2.
3.0->en_core_web_sm==2.3.1) (4.0.0)
Requirement already satisfied: certifi>=2017.4.17 in /user/HS225/rf00302/.conda/env
s/nlp2021/lib/python3.8/site-packages (from requests<3.0.0,>=2.13.0->spacy<2.4.0,>=
2.3.0->en_core_web_sm==2.3.1) (2020.12.5)
Requirement already satisfied: idna<3,>=2.5 in /user/HS225/rf00302/.conda/envs/nlp20
21/lib/python3.8/site-packages (from requests<3.0.0,>=2.13.0->spacy<2.4.0,>=2.3.0->e
n_core_web_sm==2.3.1) (2.10)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /user/HS225/rf00302/.conda/e
nvs/nlp2021/lib/python3.8/site-packages (from requests<3.0.0,>=2.13.0->spacy<2.4.0,>
=2.3.0->en_core_web_sm==2.3.1) (1.26.4)
✓ Download and installation successful
You can now load the model via spacy.load('en_core_web_sm')
```

In [ ]:

```
nlp = spacy.load("en_core_web_sm")
```

Then, process the text and visualize the words by displaying the original word, lemmatized word, part-of-speech and stop words.

In [ ]:

```
doc = nlp(text)
table = []
for token in doc:
    table.append([token.text, token.lemma_, token.pos_, token.is_stop])
pd.set_option('display.max_rows', 1000)
pd.DataFrame(table,columns=['text', 'lemma', 'pos', 'stop'])
```

Out[ ]:

	text	lemma	pos	stop
0	Blue	Blue	PROPN	False
1	Hole	Hole	PROPN	False
2	(	(	PUNCT	False
3	ブルー	ブルー	PROPN	False
4			SPACE	False
5	ホール	ホール	PROPN	False
6	Burū	Burū	PROPN	False

	text	lemma	pos	stop
7	Hōru	Hōru	PROPN	False
8	)	)	PUNCT	False
9	is	be	AUX	True
10	a	a	DET	True
11	science	science	NOUN	False
12	fiction	fiction	NOUN	False
13	manga	manga	NOUN	False
14	series	series	NOUN	False
15	by	by	ADP	True
16	Yukinobu	Yukinobu	PROPN	False
17	Hoshino	Hoshino	PROPN	False
18	involving	involve	VERB	False
19	dinosaurs	dinosaur	NOUN	False
20	living	live	VERB	False
21	in	in	ADP	True
22	the	the	DET	True
23	present	present	NOUN	False
24	.	.	PUNCT	False
25	Its	-PRON-	DET	True
26	title	title	NOUN	False
27	in	in	ADP	True
28	France	France	PROPN	False
29	is	be	AUX	True
30	Le	Le	PROPN	False
31	Trou	Trou	PROPN	False
32	Bleu	Bleu	PROPN	False
33	.	.	PUNCT	False
34	It	-PRON-	PRON	True
35	was	be	AUX	True
36	serialized	serialize	VERB	False
37	in	in	ADP	True
38	Mister	Mister	PROPN	False
39	Magazine	Magazine	PROPN	False
40	from	from	ADP	True
41	1991	1991	NUM	False
42	to	to	ADP	True

	text	lemma	pos	stop
43	1992	1992	NUM	False
44	with	with	ADP	True
45	two	two	NUM	True
46	tankōbon	tankōbon	NOUN	False
47	published	publish	VERB	False
48	.	.	PUNCT	False

As shown in the table above, non-English words are categorized as Proper nouns so removing Proper nouns can remove non-English words.

In [ ]:

```
allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']
tokens = [token.lemma_ for token in doc if token.pos_ in allowed_postags]
print(tokens)
```

```
['science', 'fiction', 'manga', 'series', 'involve', 'dinosaur', 'live', 'present',
'title', 'serialize', 'tankōbon', 'publish']
```

The code above can tokenize a text, lemmatize the tokens and select words that are noun, adjective, verb, or adverb. This approach is going to be used for processing texts to generate input data for kNN classifier.

The train data is first tokenized using gensim function. Punctuations are removed by setting deacc=True.

In [ ]:

```
def sent_to_words(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence), deacc=True))

data = train['text'].tolist()
data_words = list(sent_to_words(data))
```

In [ ]:

```
flat_list = [item for sublist in data_words for item in sublist]
print("Number of vocabularies: ", len(set(flat_list)))
print(flat_list[:30])
```

```
Number of vocabularies:  451169
['william', 'alexander', 'massey', 'october', 'march', 'was', 'united', 'states', 's
enator', 'from', 'nevada', 'born', 'in', 'trumbull', 'county', 'ohio', 'he', 'move
d', 'with', 'his', 'parents', 'to', 'edgar', 'county', 'illinois', 'in', 'he', 'atte
nded', 'the', 'common']
```

Build models to generate input data that contains bigrams and trigrams.

In [ ]:

```
# Build the bigram and trigram models
bigram = gensim.models.Phrases(data_words, min_count=5, threshold=100)
trigram = gensim.models.Phrases(bigram[data_words], threshold=100)

# Faster way to get a sentence clubbed as a trigram/bigram
bigram_mod = gensim.models.phrases.Phraser(bigram)
trigram_mod = gensim.models.phrases.Phraser(trigram)
```

nltk corpus is used to remove stopwords from the texts.

In [ ]:

```
nltk.download('stopwords')
stop_words = nltk.corpus.stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /user/HS225/rf00302/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Load spacy English model and disable parser and ner components for efficiency.

In [ ]:

```
nlp = spacy.load("en_core_web_sm", disable=['parser', 'ner'])
```

Define functions to remove stopwords, make bigrams and trigrams and process lemmatization.

In [ ]:

```
def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not in stop_words]

def make_bigrams(texts):
    return [bigram_mod[doc] for doc in texts]

def make_trigrams(texts):
    return [trigram_mod[bigram_mod[doc]] for doc in texts]

def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    """https://spacy.io/api/annotation"""
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append(" ".join([token.lemma_ for token in doc if token.pos_ in al
    return texts_out
```

Define a function to generate unigrams, bigrams and trigrams.

In [ ]:

```
def get_ngrams(data_words):
    # Remove Stop Words
    data_words_unigrams = remove_stopwords(data_words)

    # Form Bigrams
    data_words_bigrams = make_bigrams(data_words_unigrams)

    # Form trigrams
    data_words_trigrams = make_trigrams(data_words_unigrams)

    # Do lemmatization keeping only noun, adj, vb, adv
    unigrams = lemmatization(data_words_unigrams, allowed_postags=['NOUN', 'ADJ', 'V
    bigrams = lemmatization(data_words_bigrams, allowed_postags=['NOUN', 'ADJ', 'VER
    trigrams = lemmatization(data_words_trigrams, allowed_postags=['NOUN', 'ADJ', 'V
    return [unigrams, bigrams, trigrams]
```

Tokenize texts in the train, validation and test datasets.

In [ ]:

```
train_words = data_words
val_text = val['text'].tolist()
val_words = list(sent_to_words(val_text))
test_text = test['text'].tolist()
test_words = list(sent_to_words(test_text))
```

Generate unigrams, bigrams and trigrams for texts in the train, validation and test datasets.

In [ ]:

```
train_ngrams = get_ngrams(train_words)
val_ngrams = get_ngrams(val_words)
```

```
test_ngrams = get_ngrams(test_words)
```

Followings are some examples of the generated unigrams, bigrams and trigrams.

In [ ]:

```
for n, i in enumerate(random.sample(range(0, len(test)), 3)):
    print('Example {}'.format(n+1))
    print()
    for j in range(len(train_ngrams)):
        print('{} grams'.format(j+1))
        print(train_ngrams[j][i])
    print()
```

```
Example 1

1 grams
locate handle passenger flight also handle metric tonne busy airport airport give up
grade terminal building soon announce expansion project upgrade terminal start opera
tion terminal second large airport terminal sarawak kuche
2 grams
locate airport handled_passenger flight also handle metric tonne airport give upgrad
e terminal building soon announce expansion project upgrade terminal start operation
terminal second large airport terminal sarawak kuche
3 grams
locate state airport_handled_passenger flight also handle metric tonne airport give
upgrade terminal building soon announce expansion project upgrade terminal start ope
ration terminal second large airport terminal sarawak kuche

Example 2

1 grams
heavy metal band consist studio release live album compilation single music video fo
rm group mercyful fate vocalist guitarist follow year band release chart number rele
ase reach number number number follow line change group release peaked number sweden
number number make diamond high chart follow year band release chart number number n
umber line change release eye chart number make eye diamond low chart mercyful remai
n inactive band release bassist este go reach number finland spider lullabye follow
respectively release peak number revenge peaked number number line remain stable day
consist guitarist release reach number release album give soul peak number number fi
nland
2 grams
band consist studio release live album compilation single form group mercyful fate v
ocalist guitarist follow year band release chart number release reach number number
number follow line change group release peaked number sweden number number make foll
ow year band release chart number number number line change release eye chart number
make eye mercyful remain inactive band release bassist este go reach number finland
spider lullabye follow respectively release peak number revenge peaked number number
line remain stable day consist guitarist release reach number release give soul peak
number number finland
3 grams
consist studio release live_album compilation single form group mercyful fate vocali
st guitarist follow year band release chart number release reach number number numbe
r follow line change group release peaked number sweden number number make follow ye
ar band release chart number number number line change release eye chart number make
eye mercyful remain inactive band release bassist este go reach number finland spide
r lullabye follow respectively release peak number revenge peaked number number line
remain stable day consist guitarist release reach number release give soul peak numb
er number finland

Example 3

1 grams
bear partner dance champion edward begin skate together make international place bas
e coach couple move may couple take official website announce retirement competitive
skating edward rank ranking compete first world championship
2 grams
bear partner champion walden edward begin skate together make international memorial
place base coach couple move may couple take official website announce retirement co
```

```
mpetitive_skating walden edward rank ranking compete first world championship
3 grams
bear partner champion walden edward begin skate together make international memorial
place base coach couple move may couple take official website announce retirement co
mpetitive_skating walden edward rank ranking compete first world championship
```

After pre-processing the texts, convert these into Term Frequency – Inverse Document Frequency vectors. The maximum features is set to 25000.

In [ ]:

```
x_train_knn = []
x_val_knn = []
x_test_knn = []
for i in range(len(train_ngrams)):
    vectorizer = TfidfVectorizer(max_features=25000)
    x_train_knn.append(vectorizer.fit_transform(train_ngrams[i]))
    x_val_knn.append(vectorizer.transform(val_ngrams[i]))
    x_test_knn.append(vectorizer.transform(test_ngrams[i]))
    print('{} gram:'.format(i+1))
    print('x_train_knn shape:', x_train_knn[i].shape)
    print('x_val_knn shape:', x_val_knn[i].shape)
    print('x_test_knn shape:', x_test_knn[i].shape)
    print()
    print(x_train_knn[i][0])
    print()

dump(x_train_knn, 'x_train_knn.joblib')
dump(x_val_knn, 'x_val_knn.joblib')
dump(x_test_knn, 'x_test_knn.joblib')
```

```
1 gram:
x_train_knn shape: (240942, 25000)
x_val_knn shape: (36003, 25000)
x_test_knn shape: (60794, 25000)

  (0, 22758)	0.11293036461719066
  (0, 6135)	0.09139289494326952
  (0, 18263)	0.23623180746309863
  (0, 22490)	0.07271699972257549
  (0, 18196)	0.16547934917318857
  (0, 7088)	0.1033077931883648
  (0, 5737)	0.10473330697446871
  (0, 19770)	0.07599581186998143
  (0, 5637)	0.10439699686556363
  (0, 3505)	0.12202884308775892
  (0, 1069)	0.11081368958647135
  (0, 18458)	0.2957758995769945
  (0, 18398)	0.13222162614884553
  (0, 21948)	0.07816970564275608
  (0, 12144)	0.08157364812079615
  (0, 13755)	0.15957658789250104
  (0, 17215)	0.1837943386785453
  (0, 16818)	0.48364154386691516
  (0, 4386)	0.1463689556073038
  (0, 1818)	0.1312830023024098
  (0, 281)	0.14389349550387054
  (0, 12191)	0.46105516398974067
  (0, 21349)	0.10818870232976258
  (0, 4411)	0.1190861571422054
  (0, 1469)	0.1100547791224644
  (0, 15679)	0.1326005451649556
  (0, 14138)	0.2891210932890086
  (0, 1990)	0.05419950060752883

2 gram:
x_train_knn shape: (240942, 25000)
x_val_knn shape: (36003, 25000)
```

```
x_test_knn shape: (60794, 25000)

  (0, 22794)	0.12862665126071768
  (0, 6128)	0.10288587686287913
  (0, 18284)	0.27234977757685913
  (0, 22528)	0.08110113757889596
  (0, 18218)	0.1839346936164663
  (0, 7055)	0.11735948251061132
  (0, 5719)	0.11775244093235264
  (0, 19775)	0.08540302827528658
  (0, 5628)	0.11739724160590016
  (0, 3479)	0.13722505677290742
  (0, 1063)	0.12441445336681323
  (0, 18479)	0.3319451156770565
  (0, 18419)	0.1483876753989993
  (0, 12113)	0.3980318100114374
  (0, 16823)	0.4218371169809202
  (0, 22000)	0.09580089681997973
  (0, 12060)	0.09148323390229698
  (0, 13728)	0.1806080556051851
  (0, 17245)	0.20473180629036167
  (0, 1798)	0.14922966408708288
  (0, 280)	0.1614936732295477
  (0, 4399)	0.13577257993902267
  (0, 1459)	0.12346510397965181
  (0, 15661)	0.15229487112221624
  (0, 14130)	0.3250757825288446
  (0, 1965)	0.06084682550121125

3 gram:
x_train_knn shape: (240942, 25000)
x_val_knn shape: (36003, 25000)
x_test_knn shape: (60794, 25000)

  (0, 22793)	0.12872052589728747
  (0, 6039)	0.10275888527827794
  (0, 18259)	0.271988783502771
  (0, 22521)	0.08082272639754877
  (0, 18191)	0.18357071748472908
  (0, 6964)	0.11781422144128709
  (0, 5653)	0.11752840494708867
  (0, 19765)	0.08531545521722043
  (0, 5565)	0.11740992310363733
  (0, 3421)	0.13767259353883857
  (0, 1051)	0.1245231720450866
  (0, 18447)	0.33179936190568116
  (0, 18390)	0.14819099056245072
  (0, 12071)	0.3984378858749442
  (0, 16789)	0.42259636681517987
  (0, 21995)	0.09719360597822574
  (0, 12025)	0.09134415790322449
  (0, 13652)	0.18047432512109385
  (0, 17231)	0.20391236915092462
  (0, 1775)	0.1490468714274091
  (0, 278)	0.1612796166602445
  (0, 4337)	0.1357947428790675
  (0, 1451)	0.12332284874218312
  (0, 15620)	0.15221203457069116
  (0, 14058)	0.32461862958671145
  (0, 1945)	0.06076617427531916
```

Out[ ]:
```
['x_test_knn.joblib']
```

Classes

There are 3 levels of classes in the dataset. The level 1 and 2 will be combined to form the forth level, and all levels are combined to form the fifth level so classifications with different numbers of classes can be compared.

In [9]:
```
levels = ['l1','l2','l3',['l1','l2'],['l1','l2','l3']]
levels_str = ['Level 1','Level 2','Level 3','Level 1 and 2', 'All']
```

In [ ]:
```
def get_labels(data):
    y = []
    for level in levels:
        labels = data[level].to_numpy()
        if type(labels[0]) == str:
            labels = np.reshape(np.array(labels),(len(labels),1))
        y.append(labels)
    return y
```

In [ ]:
```
train_labels = get_labels(train)
val_labels = get_labels(val)
test_labels = get_labels(test)
```

In [ ]:
```
for i in range(len(levels)):
    print(train_labels[i][0])
```

```
['Agent']
['Politician']
['Senator']
['Agent' 'Politician']
['Agent' 'Politician' 'Senator']
```

Classes are converted into one hot encodings to represent each class as a single 1 in an array of 0s. MultiLabelBinarizer from scikit learn library is used to encode the classes. The binarizers are saved so that it can be used to invert one hot encoding back to the classes later.

In [ ]:
```
y_train = []
y_val = []
y_test = []
mlbs = []
for i in range(len(levels)):
    mlb = MultiLabelBinarizer()
    y_train.append(mlb.fit_transform(train_labels[i]))
    mlbs.append(mlb)
    y_test.append(mlb.transform(test_labels[i]))
    y_val.append(mlb.transform(val_labels[i]))
    print(levels_str[i])
    print('y_train shape:', y_train[i].shape)
    print('y_val shape:', y_val[i].shape)
    print('y_test shape:', y_test[i].shape)
    print()
    print(y_train[i][0])
    print()
dump(y_train, 'y_train.joblib')
dump(y_val, 'y_val.joblib')
dump(y_test, 'y_test.joblib')
dump(mlbs, 'mlbs.joblib')
```

```
Level 1
y_train shape: (240942, 9)
y_val shape: (36003, 9)
y_test shape: (60794, 9)

[1 0 0 0 0 0 0 0 0]
```

```
Level 2
y_train shape: (240942, 70)
y_val shape: (36003, 70)
y_test shape: (60794, 70)

[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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Level 3
y_train shape: (240942, 219)
y_val shape: (36003, 219)
y_test shape: (60794, 219)

[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 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 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]

Level 1 and 2
y_train shape: (240942, 79)
y_val shape: (36003, 79)
y_test shape: (60794, 79)

[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 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]

All
y_train shape: (240942, 298)
y_val shape: (36003, 298)
y_test shape: (60794, 298)

[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 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 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 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]
```

Out[ ]:
```
['mlbs.joblib']
```

Train

The classifiers will be trained with different hyperparameters to find the best ones. GridSearchCV from scikit learn library is used to find the best hyperparameters.

Different metrics were considered to be used to evaluate the performance of the classifier. Although accuracy is popular and commonly used, it is not a good metric for imbalanced data because classifying every data as the most common class achives high accuracy. Using precision can avoid classifying texts to wrong classes and using recall can avoid missing important classes. To balance the precision and recall, f1-score is used. It is a good metric for classifying imbalanced data. Micro is used for average because the number of data for each class varies and every class is equally important.

In [ ]:
```
parameters = {'n_neighbors':[1, 3, 5, 10], 'weights':('uniform', 'distance')}
scoring = "f1_micro"
```

In [ ]:

```
x_train_knn = load('x_train_knn.joblib')
x_val_knn = load('x_val_knn.joblib')
x_test_knn = load('x_test_knn.joblib')
y_train = load('y_train.joblib')
y_val = load('y_val.joblib')
y_test = load('y_test.joblib')
```

In [ ]:

```
count = 1
ngram = []
with warnings.catch_warnings():
    warnings.filterwarnings("ignore")
    for i in range(len(y_train)):
        score = []
        for j in range(len(x_train_knn)):
            print("Experiment ", count)
            print()
            print("Chosen level of classes: ", levels_str[i])
            print()
            print("Preprocessed text: ", j+1, "gram")
            print()
            filename = 'knn{}_{}'.format(i,j+1)
            if (os.path.isfile(filename)):
                clf = load(filename)
            else:
                knn = KNeighborsClassifier()
                clf = GridSearchCV(knn, parameters, scoring)
                clf.fit(x_train_knn[j], y_train[i])
                dump(clf, filename)
            print("Best parameters: ", clf.best_params_)
            print()
            print("Grid scores on development set:")
            print()
            means = clf.cv_results_['mean_test_score']
            stds = clf.cv_results_['std_test_score']
            for mean, std, params in zip(means, stds, clf.cv_results_['params']):
                print("%0.3f (+/-%0.03f) for %r"
                      % (mean, std * 2, params))
            print()
            y_true, y_pred = y_val[i], clf.predict(x_val_knn[j])
            s = f1_score(y_true, y_pred, average='micro') * 100
            print("Validation result: %0.1f%%" % s)
            score.append(s)
            print()
            print()
            count += 1
        ngram.append(score.index(max(score))+1)
```

```
Experiment  1

Chosen level of classes:  Level 1

Preprocessed text:  1 gram

Best parameters:  {'n_neighbors': 1, 'weights': 'uniform'}

Grid scores on development set:

0.659 (+/-0.097) for {'n_neighbors': 1, 'weights': 'uniform'}
0.659 (+/-0.097) for {'n_neighbors': 1, 'weights': 'distance'}
0.624 (+/-0.005) for {'n_neighbors': 3, 'weights': 'uniform'}
0.625 (+/-0.005) for {'n_neighbors': 3, 'weights': 'distance'}
0.645 (+/-0.003) for {'n_neighbors': 5, 'weights': 'uniform'}
0.646 (+/-0.003) for {'n_neighbors': 5, 'weights': 'distance'}
0.603 (+/-0.005) for {'n_neighbors': 10, 'weights': 'uniform'}
```

```
0.634 (+/-0.006) for {'n_neighbors': 10, 'weights': 'distance'}

Validation result: 65.2%


Experiment  2

Chosen level of classes:  Level 1

Preprocessed text:  2 gram

Best parameters:  {'n_neighbors': 1, 'weights': 'uniform'}

Grid scores on development set:

0.709 (+/-0.144) for {'n_neighbors': 1, 'weights': 'uniform'}
0.709 (+/-0.144) for {'n_neighbors': 1, 'weights': 'distance'}
0.701 (+/-0.071) for {'n_neighbors': 3, 'weights': 'uniform'}
0.702 (+/-0.071) for {'n_neighbors': 3, 'weights': 'distance'}
0.630 (+/-0.004) for {'n_neighbors': 5, 'weights': 'uniform'}
0.631 (+/-0.004) for {'n_neighbors': 5, 'weights': 'distance'}
0.584 (+/-0.005) for {'n_neighbors': 10, 'weights': 'uniform'}
0.618 (+/-0.006) for {'n_neighbors': 10, 'weights': 'distance'}

Validation result: 75.3%


Experiment  3

Chosen level of classes:  Level 1

Preprocessed text:  3 gram

Best parameters:  {'n_neighbors': 1, 'weights': 'uniform'}

Grid scores on development set:

0.698 (+/-0.145) for {'n_neighbors': 1, 'weights': 'uniform'}
0.698 (+/-0.145) for {'n_neighbors': 1, 'weights': 'distance'}
0.681 (+/-0.116) for {'n_neighbors': 3, 'weights': 'uniform'}
0.682 (+/-0.116) for {'n_neighbors': 3, 'weights': 'distance'}
0.610 (+/-0.005) for {'n_neighbors': 5, 'weights': 'uniform'}
0.612 (+/-0.005) for {'n_neighbors': 5, 'weights': 'distance'}
0.560 (+/-0.006) for {'n_neighbors': 10, 'weights': 'uniform'}
0.597 (+/-0.007) for {'n_neighbors': 10, 'weights': 'distance'}

Validation result: 74.4%


Experiment  4

Chosen level of classes:  Level 2

Preprocessed text:  1 gram

Best parameters:  {'n_neighbors': 5, 'weights': 'distance'}

Grid scores on development set:

0.493 (+/-0.011) for {'n_neighbors': 1, 'weights': 'uniform'}
0.493 (+/-0.011) for {'n_neighbors': 1, 'weights': 'distance'}
0.478 (+/-0.093) for {'n_neighbors': 3, 'weights': 'uniform'}
0.480 (+/-0.094) for {'n_neighbors': 3, 'weights': 'distance'}
0.537 (+/-0.004) for {'n_neighbors': 5, 'weights': 'uniform'}
0.540 (+/-0.004) for {'n_neighbors': 5, 'weights': 'distance'}
0.482 (+/-0.005) for {'n_neighbors': 10, 'weights': 'uniform'}
0.508 (+/-0.004) for {'n_neighbors': 10, 'weights': 'distance'}

Validation result: 55.8%
```

```
Experiment  5

Chosen level of classes:  Level 2

Preprocessed text:  2 gram

Best parameters:  {'n_neighbors': 3, 'weights': 'distance'}

Grid scores on development set:

0.466 (+/-0.012) for {'n_neighbors': 1, 'weights': 'uniform'}
0.466 (+/-0.012) for {'n_neighbors': 1, 'weights': 'distance'}
0.544 (+/-0.005) for {'n_neighbors': 3, 'weights': 'uniform'}
0.546 (+/-0.004) for {'n_neighbors': 3, 'weights': 'distance'}
0.501 (+/-0.004) for {'n_neighbors': 5, 'weights': 'uniform'}
0.504 (+/-0.004) for {'n_neighbors': 5, 'weights': 'distance'}
0.442 (+/-0.004) for {'n_neighbors': 10, 'weights': 'uniform'}
0.469 (+/-0.004) for {'n_neighbors': 10, 'weights': 'distance'}

Validation result: 56.4%


Experiment  6

Chosen level of classes:  Level 2

Preprocessed text:  3 gram

Best parameters:  {'n_neighbors': 3, 'weights': 'distance'}

Grid scores on development set:

0.450 (+/-0.015) for {'n_neighbors': 1, 'weights': 'uniform'}
0.450 (+/-0.015) for {'n_neighbors': 1, 'weights': 'distance'}
0.525 (+/-0.006) for {'n_neighbors': 3, 'weights': 'uniform'}
0.527 (+/-0.006) for {'n_neighbors': 3, 'weights': 'distance'}
0.481 (+/-0.004) for {'n_neighbors': 5, 'weights': 'uniform'}
0.485 (+/-0.004) for {'n_neighbors': 5, 'weights': 'distance'}
0.418 (+/-0.007) for {'n_neighbors': 10, 'weights': 'uniform'}
0.446 (+/-0.007) for {'n_neighbors': 10, 'weights': 'distance'}

Validation result: 54.8%


Experiment  7

Chosen level of classes:  Level 3

Preprocessed text:  1 gram

Best parameters:  {'n_neighbors': 5, 'weights': 'distance'}

Grid scores on development set:

0.459 (+/-0.005) for {'n_neighbors': 1, 'weights': 'uniform'}
0.459 (+/-0.005) for {'n_neighbors': 1, 'weights': 'distance'}
0.451 (+/-0.089) for {'n_neighbors': 3, 'weights': 'uniform'}
0.452 (+/-0.089) for {'n_neighbors': 3, 'weights': 'distance'}
0.512 (+/-0.004) for {'n_neighbors': 5, 'weights': 'uniform'}
0.515 (+/-0.005) for {'n_neighbors': 5, 'weights': 'distance'}
0.451 (+/-0.005) for {'n_neighbors': 10, 'weights': 'uniform'}
0.476 (+/-0.004) for {'n_neighbors': 10, 'weights': 'distance'}

Validation result: 53.4%


Experiment  8
```

Chosen level of classes: Level 3

Preprocessed text: 2 gram

Best parameters: {'n_neighbors': 3, 'weights': 'distance'}

Grid scores on development set:

```
0.425 (+/-0.002) for {'n_neighbors': 1, 'weights': 'uniform'}
0.425 (+/-0.002) for {'n_neighbors': 1, 'weights': 'distance'}
0.504 (+/-0.005) for {'n_neighbors': 3, 'weights': 'uniform'}
0.506 (+/-0.004) for {'n_neighbors': 3, 'weights': 'distance'}
0.470 (+/-0.004) for {'n_neighbors': 5, 'weights': 'uniform'}
0.474 (+/-0.005) for {'n_neighbors': 5, 'weights': 'distance'}
0.404 (+/-0.004) for {'n_neighbors': 10, 'weights': 'uniform'}
0.431 (+/-0.004) for {'n_neighbors': 10, 'weights': 'distance'}
```

Validation result: 52.6%

Experiment 9

Chosen level of classes: Level 3

Preprocessed text: 3 gram

Best parameters: {'n_neighbors': 3, 'weights': 'distance'}

Grid scores on development set:

```
0.408 (+/-0.003) for {'n_neighbors': 1, 'weights': 'uniform'}
0.408 (+/-0.003) for {'n_neighbors': 1, 'weights': 'distance'}
0.485 (+/-0.006) for {'n_neighbors': 3, 'weights': 'uniform'}
0.487 (+/-0.006) for {'n_neighbors': 3, 'weights': 'distance'}
0.450 (+/-0.004) for {'n_neighbors': 5, 'weights': 'uniform'}
0.454 (+/-0.004) for {'n_neighbors': 5, 'weights': 'distance'}
0.380 (+/-0.006) for {'n_neighbors': 10, 'weights': 'uniform'}
0.408 (+/-0.006) for {'n_neighbors': 10, 'weights': 'distance'}
```

Validation result: 50.8%

Experiment 10

Chosen level of classes: Level 1 and 2

Preprocessed text: 1 gram

Best parameters: {'n_neighbors': 5, 'weights': 'distance'}

Grid scores on development set:

```
0.576 (+/-0.054) for {'n_neighbors': 1, 'weights': 'uniform'}
0.576 (+/-0.054) for {'n_neighbors': 1, 'weights': 'distance'}
0.554 (+/-0.047) for {'n_neighbors': 3, 'weights': 'uniform'}
0.555 (+/-0.047) for {'n_neighbors': 3, 'weights': 'distance'}
0.598 (+/-0.002) for {'n_neighbors': 5, 'weights': 'uniform'}
0.599 (+/-0.002) for {'n_neighbors': 5, 'weights': 'distance'}
0.549 (+/-0.002) for {'n_neighbors': 10, 'weights': 'uniform'}
0.577 (+/-0.003) for {'n_neighbors': 10, 'weights': 'distance'}
```

Validation result: 61.8%

Experiment 11

Chosen level of classes: Level 1 and 2

```
Preprocessed text:  2 gram

Best parameters:  {'n_neighbors': 3, 'weights': 'distance'}

Grid scores on development set:

0.588 (+/-0.066) for {'n_neighbors': 1, 'weights': 'uniform'}
0.588 (+/-0.066) for {'n_neighbors': 1, 'weights': 'distance'}
0.634 (+/-0.047) for {'n_neighbors': 3, 'weights': 'uniform'}
0.635 (+/-0.047) for {'n_neighbors': 3, 'weights': 'distance'}
0.573 (+/-0.004) for {'n_neighbors': 5, 'weights': 'uniform'}
0.575 (+/-0.004) for {'n_neighbors': 5, 'weights': 'distance'}
0.519 (+/-0.005) for {'n_neighbors': 10, 'weights': 'uniform'}
0.551 (+/-0.005) for {'n_neighbors': 10, 'weights': 'distance'}

Validation result: 66.0%


Experiment  12

Chosen level of classes:  Level 1 and 2

Preprocessed text:  3 gram

Best parameters:  {'n_neighbors': 3, 'weights': 'distance'}

Grid scores on development set:

0.574 (+/-0.065) for {'n_neighbors': 1, 'weights': 'uniform'}
0.574 (+/-0.065) for {'n_neighbors': 1, 'weights': 'distance'}
0.615 (+/-0.069) for {'n_neighbors': 3, 'weights': 'uniform'}
0.617 (+/-0.069) for {'n_neighbors': 3, 'weights': 'distance'}
0.553 (+/-0.004) for {'n_neighbors': 5, 'weights': 'uniform'}
0.556 (+/-0.004) for {'n_neighbors': 5, 'weights': 'distance'}
0.495 (+/-0.006) for {'n_neighbors': 10, 'weights': 'uniform'}
0.528 (+/-0.006) for {'n_neighbors': 10, 'weights': 'distance'}

Validation result: 64.8%


Experiment  13

Chosen level of classes:  All

Preprocessed text:  1 gram

Best parameters:  {'n_neighbors': 5, 'weights': 'distance'}

Grid scores on development set:

0.537 (+/-0.036) for {'n_neighbors': 1, 'weights': 'uniform'}
0.537 (+/-0.036) for {'n_neighbors': 1, 'weights': 'distance'}
0.521 (+/-0.062) for {'n_neighbors': 3, 'weights': 'uniform'}
0.522 (+/-0.062) for {'n_neighbors': 3, 'weights': 'distance'}
0.572 (+/-0.003) for {'n_neighbors': 5, 'weights': 'uniform'}
0.574 (+/-0.003) for {'n_neighbors': 5, 'weights': 'distance'}
0.519 (+/-0.002) for {'n_neighbors': 10, 'weights': 'uniform'}
0.546 (+/-0.002) for {'n_neighbors': 10, 'weights': 'distance'}

Validation result: 59.3%


Experiment  14

Chosen level of classes:  All

Preprocessed text:  2 gram

Best parameters:  {'n_neighbors': 3, 'weights': 'distance'}
```

```
Grid scores on development set:

0.533 (+/-0.044) for {'n_neighbors': 1, 'weights': 'uniform'}
0.533 (+/-0.044) for {'n_neighbors': 1, 'weights': 'distance'}
0.595 (+/-0.037) for {'n_neighbors': 3, 'weights': 'uniform'}
0.597 (+/-0.036) for {'n_neighbors': 3, 'weights': 'distance'}
0.542 (+/-0.004) for {'n_neighbors': 5, 'weights': 'uniform'}
0.545 (+/-0.004) for {'n_neighbors': 5, 'weights': 'distance'}
0.484 (+/-0.004) for {'n_neighbors': 10, 'weights': 'uniform'}
0.514 (+/-0.004) for {'n_neighbors': 10, 'weights': 'distance'}

Validation result: 62.0%


Experiment  15

Chosen level of classes:  All

Preprocessed text:  3 gram

Best parameters:  {'n_neighbors': 3, 'weights': 'distance'}

Grid scores on development set:

0.519 (+/-0.043) for {'n_neighbors': 1, 'weights': 'uniform'}
0.519 (+/-0.043) for {'n_neighbors': 1, 'weights': 'distance'}
0.578 (+/-0.051) for {'n_neighbors': 3, 'weights': 'uniform'}
0.579 (+/-0.051) for {'n_neighbors': 3, 'weights': 'distance'}
0.522 (+/-0.003) for {'n_neighbors': 5, 'weights': 'uniform'}
0.525 (+/-0.003) for {'n_neighbors': 5, 'weights': 'distance'}
0.459 (+/-0.006) for {'n_neighbors': 10, 'weights': 'uniform'}
0.491 (+/-0.006) for {'n_neighbors': 10, 'weights': 'distance'}

Validation result: 60.7%
```

According to the results of the grid search provided above, weight function does not have much effects on the results when the number of neighbours is set to 1, 3 or 5. When the number of neighbours is set to 10, distance works better than uniform. The number of neighbours and n-grams do not give a huge effect on the results.

The classifiers with the best hyperparameters for each level of classes are stored.

In [ ]:

```
knns = []
knn_best_params = []
for i in range(len(y_train)):
    filename = 'knn{}_{}'.format(i, ngram[i])
    clf = load(filename)
    knns.append([clf, ngram[i]])
    knn_best_params.append([ngram[i],
                            clf.best_params_['n_neighbors'],
                            clf.best_params_['weights']])
dump(knns, 'knns.joblib')
```

Out[ ]:

```
['knns.joblib']
```

Best Parameters:

In [ ]:

```
pd.DataFrame(knn_best_params, index=levels_str, columns=['ngram', 'n-neighbors', 'we
```

Out[ ]:

	ngram	n-neighbors	weight

	ngram	n-neighbors	weight
Level 1	2	1	uniform
Level 2	2	3	distance
Level 3	1	5	distance
Level 1 and 2	2	3	distance
All	2	3	distance

Bigrams performs the best for every level of classes except for the level 3. The best number of neighbours are varied for different levels. Distance weight function works the best for all except for the level 1.

Test

Test the best classifiers with test data.

In [3]:
```
knns = load('knns.joblib')
```

In [4]:
```
x_test_knn = load('x_test_knn.joblib')
y_test_knn = load('y_test.joblib')
```

In [16]:
```
y_preds = []
for i in range(len(y_test_rnn)):
    clf = knns[i][0]
    ngram = knns[i][1]
    y_pred = clf.predict_proba(x_test_knn[ngram-1])
    y_preds.append(y_pred)
dump(y_preds, 'knn_preds.joblib')
```

Out[16]:
```
['knn_preds.joblib']
```

In [5]:
```
y_pred_proba = load('knn_preds.joblib')
```

In [6]:
```
y_preds = []
for y_pred in y_pred_proba:
    y_preds.append(np.array(y_pred)[...,1].transpose())
```

Find the appropriate threshold for each classifier.

In [7]:
```
thresholds = [0.5, 0.9, 0.99, 0.999, 0.9999]
results = []
best_thres = []
for i in range(len(y_preds)):
    result = []
    for threshold in thresholds:
        y_pred = np.where(y_preds[i] < threshold, 0, 1)
        y_true = y_test_knn[i]
        f1 = f1_score(y_true, y_pred, average='micro')*100
        result.append(f1)
    results.append(result)
    best_thres.append(thresholds[result.index(max(result))])
```

In [10]:
```
pd.DataFrame(results, index=levels_str, columns=thresholds)
```

Out[10]:

	0.5000	0.9000	0.9900	0.9990	0.9999
Level 1	75.305129	75.305129	75.305129	75.305129	75.305129
Level 2	56.528527	49.245239	49.239729	49.239729	49.239729
Level 3	53.726765	44.257014	44.237258	44.237258	44.237258
Level 1 and 2	65.988452	62.157347	62.153849	62.153849	62.153849
All	62.037420	57.293802	57.289682	57.289682	57.289682

It performs the best when the threshold is 0.5 for every classifier. As can be seen in the table above, it performs worse as the threshold increases and there is no difference when the threshold is more than 0.9.

In [11]:
```
print(best_thres)
dump(best_thres, 'knn_thres.joblib')
```

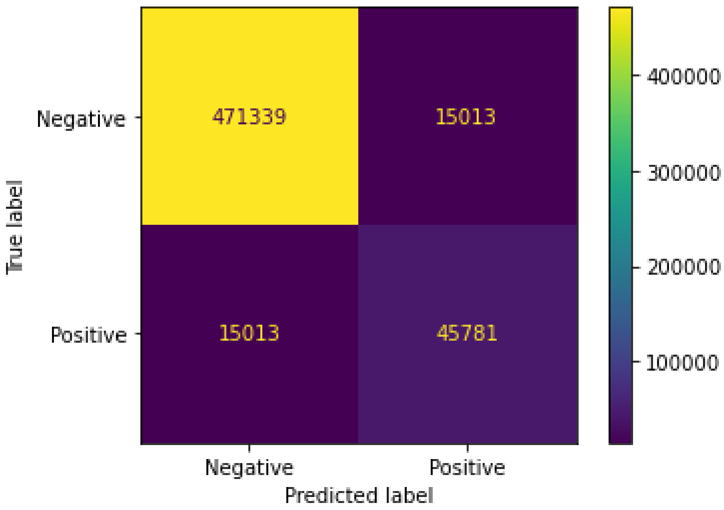
```
[0.5, 0.5, 0.5, 0.5, 0.5]
```

Out[11]:
```
['knn_thres.joblib']
```

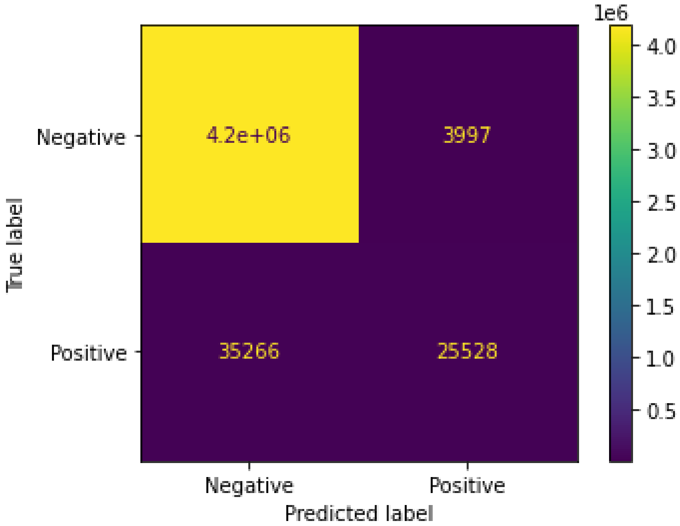
Test Results

In [14]:
```
knn_results = []
with warnings.catch_warnings():
    warnings.filterwarnings("ignore")
    for i in range(len(y_preds)):
        print("Level of classes: ", levels_str[i])
        y_pred = np.where(y_preds[i] < best_thres[i], 0, 1)
        y_true = y_test_knn[i]
        f1 = f1_score(y_true, y_pred, average='micro')*100
        print("Test result: %0.1f%%" % f1)
        clf = knns[i][0]
        training = clf.cv_results_['mean_fit_time'][clf.best_index_]
        print("Training time: %0.1fs" % training)
        roc_auc = roc_auc_score(y_true, y_preds[i], 'micro') * 100
        print("ROC AUC score: %0.1f%%" % roc_auc)
        mcm = multilabel_confusion_matrix(y_true, y_pred)
        tn = sum(mcm[:, 0, 0])
        tp = sum(mcm[:, 1, 1])
        fn = sum(mcm[:, 1, 0])
        fp = sum(mcm[:, 0, 1])
        cm = np.array([[tn, fp],
                       [fn, tp]])
        disp = ConfusionMatrixDisplay(cm,['Negative', 'Positive'])
        disp.plot()
        plt.show()
        print()
        knn_results.append([f1, roc_auc, training])
dump(knn_results, 'knn_results.joblib')
```

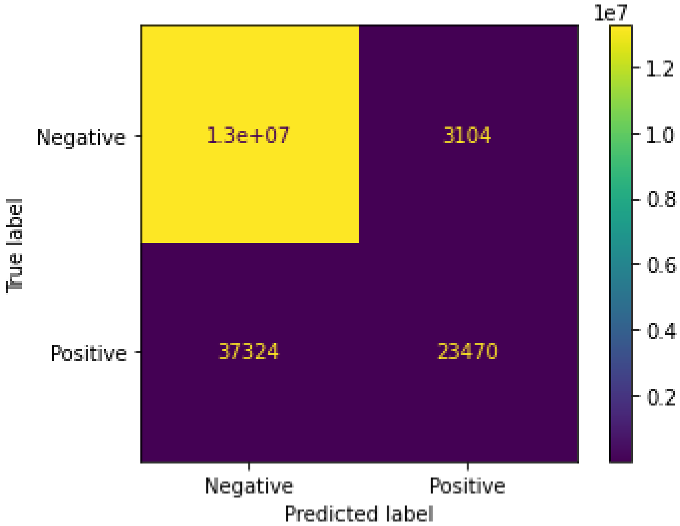
```
Level of classes:  Level 1
Test result: 75.3%
Training time: 0.1s
ROC AUC score: 86.1%
```



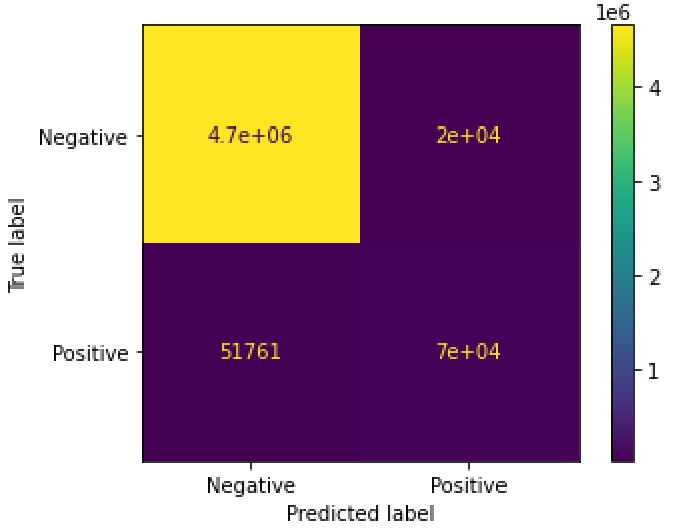

```
Level of classes:  Level 2
Test result: 56.5%
Training time: 0.7s
ROC AUC score: 78.4%
```



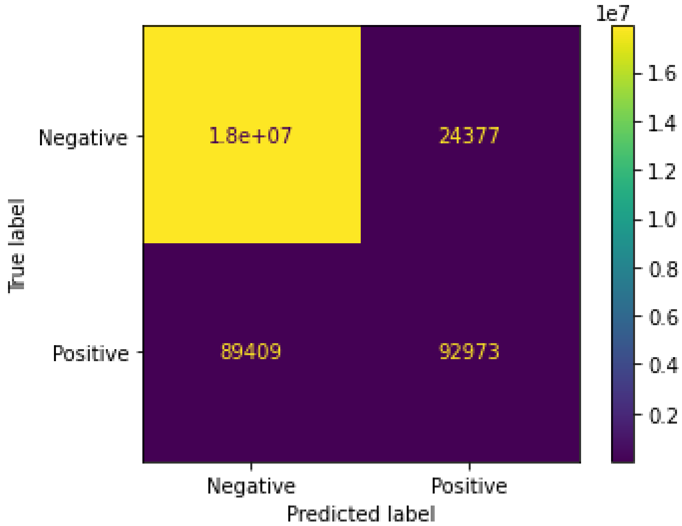

```
Level of classes:  Level 3
Test result: 53.7%
Training time: 2.3s
ROC AUC score: 76.7%
```




```
Level of classes:  Level 1 and 2
Test result: 66.0%
Training time: 0.8s
ROC AUC score: 83.4%
```




```
Level of classes:  All
Test result: 62.0%
Training time: 3.2s
ROC AUC score: 80.5%
```



Out[14]: ['knn_results.joblib']

Summary

In [15]:

```
pd.DataFrame(knn_results, index=levels_str, columns=['F1 score (%)', 'ROC AUC score
```

Out[15]:

	F1 score (%)	ROC AUC score (%)	Training time (s)
Level 1	75.305129	86.109135	0.110348
Level 2	56.528527	78.435288	0.741983
Level 3	53.726765	76.686846	2.315311
Level 1 and 2	65.988452	83.375354	0.841357
All	62.037420	80.473678	3.229993

It can be said that the f1 score is lower when the number of classes increases for data with a single level of classes. F1 score for the data with combined levels with level 1 and 2 and with all levels of classes are higher than the data with level 2 and the data with level 3. The training time increases as the number of supported classes increases.

Recurrent Neural Network (RNN)

Preprocess Data

Training RNN takes a very long time so the data size is reduced to one-tenth.

In [ ]:
```
n_train = int(len(x_train_rnn_pad)/10)
n_val = int(len(x_val_rnn_pad)/10)
n_test = int(len(x_test_rnn_pad)/10)
```

In [ ]:
```
train = train.sample(n_train)
val = val.sample(n_val)
test = test.sample(n_test)
```

In [ ]:
```
print("Number of train data to be used for RNN model: ", len(train))
print("Number of validation data to be used for RNN model: ", len(val))
print("Number of test data to be used for RNN model: ", len(test))
```

```
Number of train data to be used for RNN model:  24094
Number of validation data to be used for RNN model:  3600
Number of test data to be used for RNN model:  6079
```

Text

The texts are processed differently from how these were processed for kNN because the sequence of the text is important for RNN. The maximum number of words is set to 2500 because having many vocablaries makes the training time very long.

In [2]:
```
num_words = 2500
```

The texts are first tokenized using Tokenizer from keras library. Most frequent words will be kept and unknown words will be replaced with 'UNK'. The texts will be transformed to sequences of integers.

In [ ]:
```
tokenizer = Tokenizer(num_words=num_words, oov_token='UNK')
tokenizer.fit_on_texts(train['text'].tolist())
x_train_rnn= tokenizer.texts_to_sequences(train['text'].tolist())
```

In [ ]:
```
x_val_rnn = tokenizer.texts_to_sequences(val['text'].tolist())
x_test_rnn = tokenizer.texts_to_sequences(test['text'].tolist())
```

Max length of the sequence is set to 100 because the average length of the texts in the data is close to 100.

In [ ]:
```
maxlen = 100
```

The sequences are padded to make each sequence to have the same length.

In [ ]:
```
x_train_rnn = sequence.pad_sequences(x_train_rnn, maxlen=maxlen)
x_val_rnn = sequence.pad_sequences(x_val_rnn, maxlen=maxlen)
x_test_rnn = sequence.pad_sequences(x_test_rnn, maxlen=maxlen)
```

In [ ]:

```
x_train_rnn = load('x_train_rnn.joblib')
x_val_rnn = load('x_val_rnn.joblib')
x_test_rnn = load('x_test_rnn.joblib')
```

In [ ]:

```
print(x_train_rnn[:3])
print('x_train_rnn shape:', x_train_rnn.shape)
print('x_val_rnn shape:', x_val_rnn.shape)
print('x_test_rnn shape:', x_test_rnn.shape)
dump(x_train_rnn, 'x_train_rnn.joblib')
dump(x_val_rnn, 'x_val_rnn.joblib')
dump(x_test_rnn, 'x_test_rnn.joblib')
```

```
[[   3    1 1262   13    2  800    4    1   52  299    3  147    9    1
     1    1    2    1  417  856    7    2  209    4    2    1 2036  431
     5  333  215  333 1558    5  333    1  313   16    7   70   13    2
   131  306    4    2    1  800  237    1  252   76    1 1395    5    1
  2137    1   18    2  232  270    2  790   23 1013   34 1018 1655    6
     1 1806 1807    1    1 1738    2    1    1  553    5    2    1 1117
   344  304    2  131  306    4    2  790    7    6  364    1    9    2
     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    0    0    0    0 2356 1739   26  117  275  588    7   19   41
   844 1313  607  179 1501    3    2    1    1  169    6   91    1   99
   147 1739   24   23  556  268   12    2    1   10    8    2 1642    1
     1  328]
 [   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    7
     6  291    4    1  665    9    2  612 1444    5    2  239    5  243
   806 1444]]
x_train_rnn shape: (24094, 100)
x_val_rnn shape: (3600, 100)
x_test_rnn shape: (6079, 100)
```

Out[ ]: ['x_test_rnn.joblib']

Classes

The classes are proccessed as how these were processed for kNN.

In [ ]:

```
train_labels = get_labels(train)
val_labels = get_labels(val)
test_labels = get_labels(test)
```

In [ ]:

```
y_train_rnn = []
y_val_rnn = []
y_test_rnn = []
mlbs_rnn = []
for i in range(len(levels)):
    mlb = MultiLabelBinarizer()
    y_train_rnn.append(mlb.fit_transform(train_labels[i]))
    mlbs_rnn.append(mlb)
    y_test_rnn.append(mlb.transform(test_labels[i]))
    y_val_rnn.append(mlb.transform(val_labels[i]))
    print(levels_str[i])
    print('y_train_rnn shape:', y_train_rnn[i].shape)
    print('y_val_rnn shape:', y_val_rnn[i].shape)
```

```
    print('y_test_rnn shape:', y_test_rnn[i].shape)
    print()
dump(y_train_rnn, 'y_train_rnn.joblib')
dump(y_val_rnn, 'y_val_rnn.joblib')
dump(y_test_rnn, 'y_test_rnn.joblib')
dump(mlbs_rnn, 'mlbs_rnn.joblib')
```

```
Level 1
y_train_rnn shape: (24094, 9)
y_val_rnn shape: (3600, 9)
y_test_rnn shape: (6079, 9)

Level 2
y_train_rnn shape: (24094, 70)
y_val_rnn shape: (3600, 70)
y_test_rnn shape: (6079, 70)

Level 3
y_train_rnn shape: (24094, 219)
y_val_rnn shape: (3600, 219)
y_test_rnn shape: (6079, 219)

Level 1 and 2
y_train_rnn shape: (24094, 79)
y_val_rnn shape: (3600, 79)
y_test_rnn shape: (6079, 79)

All
y_train_rnn shape: (24094, 298)
y_val_rnn shape: (3600, 298)
y_test_rnn shape: (6079, 298)
```

Out[ ]:
```
['mlbs_rnn.joblib']
```

Find the best hyperparameters

Keras tuner is going to be used for finding the best hyperparameters for RNN model.

In [3]:
```
x_train_rnn = load('x_train_rnn.joblib')
x_val_rnn = load('x_val_rnn.joblib')
y_train_rnn = load('y_train_rnn.joblib')
y_val_rnn = load('y_val_rnn.joblib')
```

Define function to build a RNN model. mask_zero=True for the embedding layer to mask out padding value (0). Sigmoid function is used as an activation function on the output layer and categorical cross entropy is used as a loss function for classification.

In [4]:
```
def build_model(hp):
  model = Sequential()
  model.add(Embedding(INPUT_DIM, hp.Choice('embedding_dim', values=[128, 256, 512]),
  dropout_rate = hp.Float('dropout', 0, 0.9, step=0.1)
  recurrent_dropout_rate = hp.Float('recurrent_dropout', 0, 0.9, step=0.1)
  hidden_dim = hp.Choice('hidden_dim', values=[128, 256, 512])
  if hp.Boolean('bidirectional'):
    if hp.Boolean('multilayer'):
        model.add(Bidirectional(LSTM(hidden_dim, dropout=dropout_rate, recurrent_dro
    model.add(Bidirectional(LSTM(hidden_dim, dropout=dropout_rate, recurrent_dropout
  else:
    if hp.Boolean('multilayer'):
        model.add(LSTM(hidden_dim, dropout=dropout_rate, recurrent_dropout=recurrent
    model.add(LSTM(hidden_dim, dropout=dropout_rate, recurrent_dropout=recurrent_dro
  model.add(Dense(OUTPUT_DIM, activation='sigmoid'))
```

```python
    model.compile(loss='categorical_crossentropy', optimizer=hp.Choice('optimizer', va
    return model
```

To tune the batch size, run_trial function is overwritten as follows.

In [5]:

```python
class MyTuner(kt.tuners.Hyperband):
  def run_trial(self, trial, *args, **kwargs):
    # You can add additional HyperParameters for preprocessing and custom training l
    # via overriding `run_trial`
    kwargs['batch_size'] = trial.hyperparameters.Choice('batch_size', values=[32, 64
    super(MyTuner, self).run_trial(trial, *args, **kwargs)
```

Define a callback class to calculate the training time.

In [6]:

```python
class TimeHistory(Callback):
    def on_train_begin(self, logs={}):
        self.times = []

    def on_epoch_begin(self, batch, logs={}):
        self.epoch_time_start = time.time()

    def on_epoch_end(self, batch, logs={}):
        self.times.append(time.time() - self.epoch_time_start)
```

EarlyStopping in the keras library is used to stop training when a f1 score has stopped improving.

In [12]:

```python
stop_early = EarlyStopping(monitor='val_f1_score', patience=3, mode='max')
```

I wanted to tune the model for each level of classes but tuning takes very long time so the model for the level 1 will be tuned and the best hyperparamers of the model is used for the models for other levels.

In [7]:

```python
max_epochs = 10
INPUT_DIM = num_words
OUTPUT_DIM = len(y_train_rnn[0][0])
```

Instantiate the tuner to perform the hypertuning. Objective is set to validation f1 score.

In [13]:

```python
tuner = MyTuner(build_model,
                objective=kt.Objective("val_f1_score", direction="max"),
                max_epochs=max_epochs,
                factor=3,
                directory='RNN/',
                project_name='tuner')
```

Run the hyperparameter search.

In [14]:

```python
tuner.search(x_train_rnn, y_train_rnn[0], epochs=max_epochs, validation_data=(x_val_
```

```
Trial 30 Complete [00h 04m 12s]
val_f1_score: 0.933055579662323

Best val_f1_score So Far: 0.9447222352027893
Total elapsed time: 13h 09m 13s
INFO:tensorflow:Oracle triggered exit
```

Save the best model and best hyperparameters.

In [15]:
```
tuner = MyTuner(build_model,
                objective=kt.Objective("val_f1_score", direction="max"),
                max_epochs=max_epochs,
                factor=3,
                directory='RNN/',
                project_name='tuner')

best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
model = tuner.hypermodel.build(best_hps)
model.save('best_model')
dump(best_hps, 'best_hps.joblib')
```

```
INFO:tensorflow:Reloading Oracle from existing project RNN/tuner/oracle.json
INFO:tensorflow:Reloading Tuner from RNN/tuner/tuner0.json
INFO:tensorflow:Assets written to: best_model/assets
```

Out[15]: ['best_hps.joblib']

Following is the summary of the hyperparameter tuning.

In [17]:
```
num_trials = 30
hyper_parameters = ['batch_size', 'embedding_dim', 'hidden_dim', 'dropout', 'recurre
trials = []
for i in range(num_trials):
    trial = []
    for param in hyper_parameters:
        trial.append(tuner.oracle.get_best_trials(num_trials)[i].hyperparameters[par
    trial.append(tuner.oracle.get_best_trials(num_trials)[i].score*100)
    trials.append(trial)
hyper_parameters.append('validation f1 score')
pd.DataFrame(trials, index=range(1, num_trials+1), columns=hyper_parameters)
```

Out[17]:

	batch_size	embedding_dim	hidden_dim	dropout	recurrent_dropout	multilayer	bidirectional
1	32	512	256	0.1	0.4	True	True
2	64	256	256	0.9	0.5	False	False
3	64	512	512	0.5	0.3	False	False
4	128	512	512	0.7	0.1	True	True
5	32	512	256	0.1	0.4	True	True
6	64	512	128	0.4	0.5	True	True
7	128	128	128	0.2	0.4	False	False
8	64	256	256	0.3	0.8	True	False
9	64	256	512	0.3	0.9	True	False
10	64	512	128	0.4	0.5	True	True
11	128	512	512	0.7	0.1	True	True
12	64	256	512	0.3	0.9	True	False
13	128	256	512	0.2	0.9	False	True
14	64	128	256	0.9	0.8	True	True
15	64	512	128	0.4	0.5	True	True

	batch_size	embedding_dim	hidden_dim	dropout	recurrent_dropout	multilayer	bidirectional
16	32	512	256	0.1	0.4	True	True
17	32	128	256	0.1	0.2	True	True
18	32	128	512	0.6	0.5	False	True
19	128	512	128	0.2	0.5	False	True
20	64	512	256	0.9	0.4	False	True
21	32	128	256	0.1	0.2	True	True
22	32	128	512	0.6	0.5	False	True
23	128	256	128	0.3	0.2	False	False
24	64	512	512	0.7	0.3	False	True
25	128	128	256	0.7	0.3	True	True
26	128	128	512	0.5	0.2	False	False
27	128	128	512	0.1	0.3	True	False
28	32	256	256	0.0	0.1	True	True
29	128	512	512	0.6	0.9	True	False
30	32	512	256	0.5	0.3	False	True

As there are many hyperparameters that are tuned, it is difficult to see which one is better for each hyperparameter. The most obvious one is that SGD performs the worst compared to the other optimizers. It seems like the other parameters do not have much effect on the performance of the classifier.

Next step is to find the number of epochs. ModelCheckPoint from keras library is used to save the weights of the model when it has the best validation f1 score.

In [18]:

```
checkpoint_filepath = 'checkpoint/checkpoint'
model_checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=True,
    monitor='val_f1_score',
    mode='max',
    save_best_only=True)
```

Train the model with the best hyperparameters obtained from the search to find the best number of epochs.

In [20]:

```
model = load_model('best_model')
best_hps = load('best_hps.joblib')
best_batch_size = best_hps['batch_size']

timehistory = TimeHistory()
history = model.fit(x_train_rnn, y_train_rnn[0], epochs=max_epochs, batch_size=best_

val_f1_per_epoch = history.history['val_f1_score']
best_epoch = val_f1_per_epoch.index(max(val_f1_per_epoch)) + 1
```

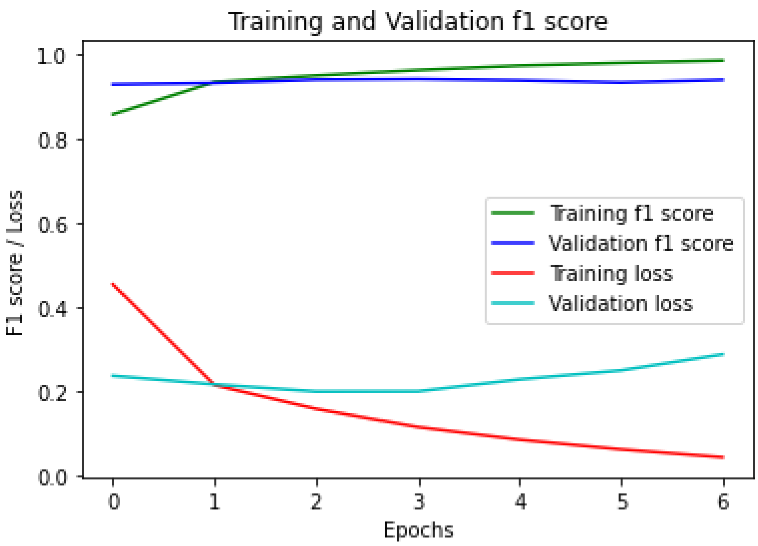
```
params = [best_epoch, best_batch_size, best_hps]
dump(params, 'params.joblib')

total_time = sum(timehistory.times[:best_epoch])
dump(total_time, 'rnn_time0.joblib')
```

```
Epoch 1/10
753/753 [==============================] - 794s 1s/step - loss: 0.4550 - f1_score: 0.8578 - val_loss: 0.2373 - val_f1_score: 0.9292
Epoch 2/10
753/753 [==============================] - 787s 1s/step - loss: 0.2151 - f1_score: 0.9349 - val_loss: 0.2167 - val_f1_score: 0.9325
Epoch 3/10
753/753 [==============================] - 776s 1s/step - loss: 0.1589 - f1_score: 0.9502 - val_loss: 0.2000 - val_f1_score: 0.9406
Epoch 4/10
753/753 [==============================] - 787s 1s/step - loss: 0.1159 - f1_score: 0.9632 - val_loss: 0.2003 - val_f1_score: 0.9419
Epoch 5/10
753/753 [==============================] - 797s 1s/step - loss: 0.0860 - f1_score: 0.9738 - val_loss: 0.2287 - val_f1_score: 0.9389
Epoch 6/10
753/753 [==============================] - 782s 1s/step - loss: 0.0629 - f1_score: 0.9805 - val_loss: 0.2500 - val_f1_score: 0.9336
Epoch 7/10
753/753 [==============================] - 778s 1s/step - loss: 0.0442 - f1_score: 0.9861 - val_loss: 0.2884 - val_f1_score: 0.9400
```

Out[20]: ['rnn_time0.joblib']

In [25]:

```
f1_score = history.history['f1_score']
val_f1_score = history.history['val_f1_score']
loss = history.history['loss']
val_loss = history.history['val_loss']
plt.plot(f1_score, 'g', label='Training f1 score')
plt.plot(val_f1_score, 'b', label='Validation f1 score')
plt.plot(loss, 'r', label='Training loss')
plt.plot(val_loss, 'c', label='Validation loss')
plt.title('Training and Validation f1 score')
plt.xlabel('Epochs')
plt.ylabel('F1 score / Loss')
plt.legend()
plt.show()
```



The training and validation f1 score is over 90% from the first epoch and it didn't improve dramatically after that. Training loss kept decreasing but validation loss started to increase after the third epoch. The fourth epoch gave the best validation f1 score so training will be run four epochs for all RNN models. Following is the best hyperparameters that is going to be used for training RNN models.

In [7]:
```
filename = 'params.joblib'
params = load(filename)
best_epoch = params[0]
best_batch_size = params[1]
best_hps = params[2]

print("Best Hyperparameters: ")
print("Number of epochs: ", best_epoch)
print("Batch size: ", best_batch_size)
print("Embedding dimension: ", best_hps.get('embedding_dim'))
print("Hidden dimension: ", best_hps.get('hidden_dim'))
print("Dropout rate: ", best_hps.get('dropout'))
print("Recurrent dropout rate: ", best_hps.get('recurrent_dropout'))
print("Multi-layer: ", best_hps.get('multilayer'))
print("Bidirectional: ", best_hps.get('bidirectional'))
print("Optimizers: ", best_hps.get('optimizer'))
print()
print()
```

```
Best Hyperparameters:
Number of epochs:  4
Batch size:  32
Embedding dimension:  512
Hidden dimension:  256
Dropout rate:  0.1
Recurrent dropout rate:  0.4
Multi-layer:  True
Bidirectional:  True
Optimizers:  RMSprop
```

Save trained model for level 1.

In [27]:
```
model = load_model('best_model')
model.load_weights(checkpoint_filepath)
model.save('rnn0')
```

```
INFO:tensorflow:Assets written to: rnn0/assets
```

Train

Train RNN models for each level of classes with the best hyperparameters and save the trained models and the training times.

In [9]:
```
for i in range(1,len(y_train_rnn)):
    filename = 'rnn_time{}.joblib'.format(i)
    if not (os.path.isfile(filename)):
        OUTPUT_DIM = len(y_train_rnn[i][0])

        tuner = MyTuner(build_model,
                 objective=kt.Objective("val_f1_score", direction="max"),
                 max_epochs=max_epochs,
                 factor=3,
                 directory='RNN/',
                 project_name='tuner')

        model = tuner.hypermodel.build(best_hps)
```

```
        timehistory = TimeHistory()
        model.fit(x_train_rnn, y_train_rnn[i], epochs=best_epoch, batch_size=best_ba
        total_time = sum(timehistory.times)

        model.save('rnn{}'.format(i))
        dump(total_time, filename)
```

```
Epoch 1/4
753/753 [==============================] - 800s 1s/step - loss: 2.5723 - f1_score:
0.3450 - val_loss: 1.0522 - val_f1_score: 0.7094
Epoch 2/4
753/753 [==============================] - 775s 1s/step - loss: 0.9329 - f1_score:
0.7355 - val_loss: 0.7861 - val_f1_score: 0.7861
Epoch 3/4
753/753 [==============================] - 777s 1s/step - loss: 0.6353 - f1_score:
0.8223 - val_loss: 0.6795 - val_f1_score: 0.8122
Epoch 4/4
753/753 [==============================] - 776s 1s/step - loss: 0.4785 - f1_score:
0.8652 - val_loss: 0.6450 - val_f1_score: 0.8164
INFO:tensorflow:Assets written to: rnn1/assets
Epoch 1/4
753/753 [==============================] - 807s 1s/step - loss: 4.3394 - f1_score:
0.0864 - val_loss: 2.3321 - val_f1_score: 0.4036
Epoch 2/4
753/753 [==============================] - 777s 1s/step - loss: 2.1432 - f1_score:
0.4588 - val_loss: 1.6624 - val_f1_score: 0.5797
Epoch 3/4
753/753 [==============================] - 777s 1s/step - loss: 1.3927 - f1_score:
0.6411 - val_loss: 1.3191 - val_f1_score: 0.6675
Epoch 4/4
753/753 [==============================] - 777s 1s/step - loss: 1.0590 - f1_score:
0.7312 - val_loss: 1.1592 - val_f1_score: 0.7072
INFO:tensorflow:Assets written to: rnn2/assets
Epoch 1/4
753/753 [==============================] - 779s 1s/step - loss: 12.3425 - f1_score:
0.0881 - val_loss: 24.2763 - val_f1_score: 0.0494
Epoch 2/4
753/753 [==============================] - 765s 1s/step - loss: 28.8109 - f1_score:
0.0494 - val_loss: 40.6823 - val_f1_score: 0.0494
Epoch 3/4
753/753 [==============================] - 764s 1s/step - loss: 45.1880 - f1_score:
0.0494 - val_loss: 56.2983 - val_f1_score: 0.0494
Epoch 4/4
753/753 [==============================] - 763s 1s/step - loss: 62.2059 - f1_score:
0.0494 - val_loss: 72.7177 - val_f1_score: 0.0494
INFO:tensorflow:Assets written to: rnn3/assets
Epoch 1/4
753/753 [==============================] - 779s 1s/step - loss: 41.2900 - f1_score:
0.0470 - val_loss: 109.5614 - val_f1_score: 0.0200
Epoch 2/4
753/753 [==============================] - 767s 1s/step - loss: 130.8707 - f1_score:
0.0200 - val_loss: 200.0369 - val_f1_score: 0.0199
Epoch 3/4
753/753 [==============================] - 786s 1s/step - loss: 219.1795 - f1_score:
0.0199 - val_loss: 287.5731 - val_f1_score: 0.0199
Epoch 4/4
753/753 [==============================] - 765s 1s/step - loss: 310.0749 - f1_score:
0.0199 - val_loss: 378.3424 - val_f1_score: 0.0199
INFO:tensorflow:Assets written to: rnn4/assets
```

Test

Test the trained models with the test data.

In [7]:

```
x_test_rnn = load('x_test_rnn.joblib')
y_test_rnn = load('y_test_rnn.joblib')
```

In [10]:

```
y_preds = []
for i in range(len(y_test_rnn)):
    rnn = load_model('rnn{}'.format(i))
    y_pred = rnn.predict(x_test_rnn, best_batch_size)
    y_preds.append(y_pred)
dump(y_preds, 'rnn_preds.joblib')
```

Out[10]:

```
['rnn_preds.joblib']
```

In [93]:

```
y_preds = load('rnn_preds.joblib')
```

Find the appropriate threshold for each model.

In [8]:

```
thresholds = [0.5, 0.9, 0.99, 0.999, 0.9999]
results = []
best_thres = []
for i in range(len(y_preds)):
    result = []
    for threshold in thresholds:
        y_pred = np.where(y_preds[i] < threshold, 0, 1)
        y_true = y_test_rnn[i]
        f1 = f1_score(y_true, y_pred, average='micro')*100
        result.append(f1)
    results.append(result)
    best_thres.append(thresholds[result.index(max(result))])
```

In [11]:

```
pd.DataFrame(results, index=levels_str, columns=thresholds)
```

Out[11]:

	0.5000	0.9000	0.9900	0.9990	0.9999
Level 1	90.834367	89.717958	48.720497	27.444303	12.636770
Level 2	20.377542	51.829555	73.972603	69.683996	45.158015
Level 3	7.310464	21.412160	49.814559	50.936742	33.112404
Level 1 and 2	4.938272	4.938272	4.938272	4.938272	4.938272
All	1.993355	1.993355	1.993355	1.993355	1.993355

The test results varies with the thresholds for the models for a single level of classes. The models for the combined levels performs very bad and there's no difference in the results with different thresholds. The best thresholds for the models are the followings.

In [12]:

```
print(best_thres)
dump(best_thres, 'rnn_thres.joblib')
```

```
[0.5, 0.99, 0.999, 0.5, 0.5]
```

Out[12]:

```
['RNN_thres.joblib']
```

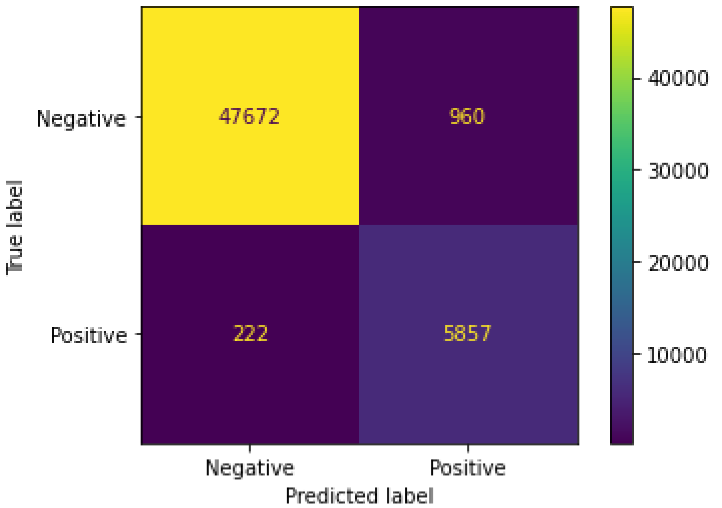
Test Results

In [13]:

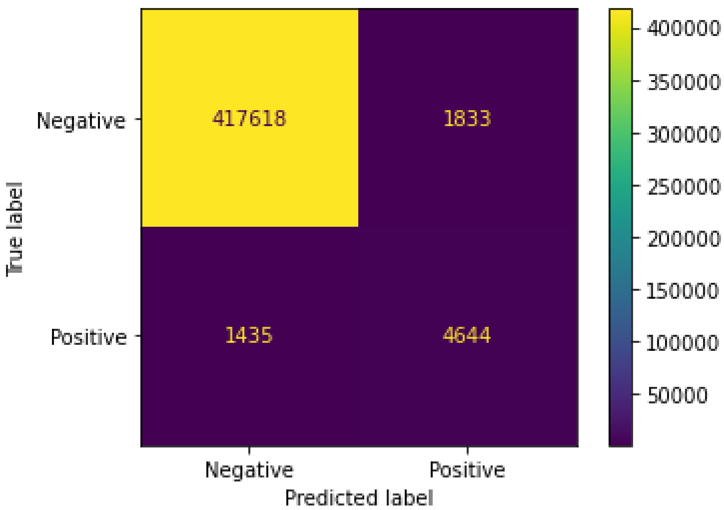
```
rnn_results = []
with warnings.catch_warnings():
    warnings.filterwarnings("ignore")
    for i in range(len(y_preds)):
```

```
        print("Level of classes: ", levels_str[i])
        y_pred = np.where(y_preds[i] < best_thres[i], 0, 1)
        y_true = y_test_rnn[i]
        f1 = f1_score(y_true, y_pred, average='micro')*100
        print("Test result: %0.1f%%" % f1)
        filename = 'rnn_time{}.joblib'.format(i)
        training = load(filename)
        print("Training time: %0.1fs" % training)
        roc_auc = roc_auc_score(y_true, y_preds[i], 'micro') * 100
        print("ROC AUC score: %0.1f%%" % roc_auc)
        mcm = multilabel_confusion_matrix(y_true, y_pred)
        tn = sum(mcm[:, 0, 0])
        tp = sum(mcm[:, 1, 1])
        fn = sum(mcm[:, 1, 0])
        fp = sum(mcm[:, 0, 1])
        cm = np.array([[tn, fp],
                       [fn, tp]])
        disp = ConfusionMatrixDisplay(cm,['Negative', 'Positive'])
        disp.plot()
        plt.show()
        print()
        rnn_results.append([f1, roc_auc, training])
dump(rnn_results, 'rnn_results.joblib')
```

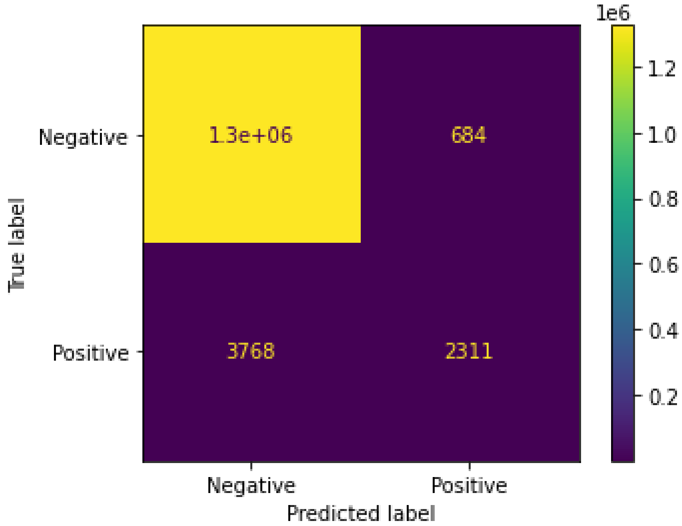
```
Level of classes:  Level 1
Test result: 90.8%
Training time: 3145.8s
ROC AUC score: 99.6%
```



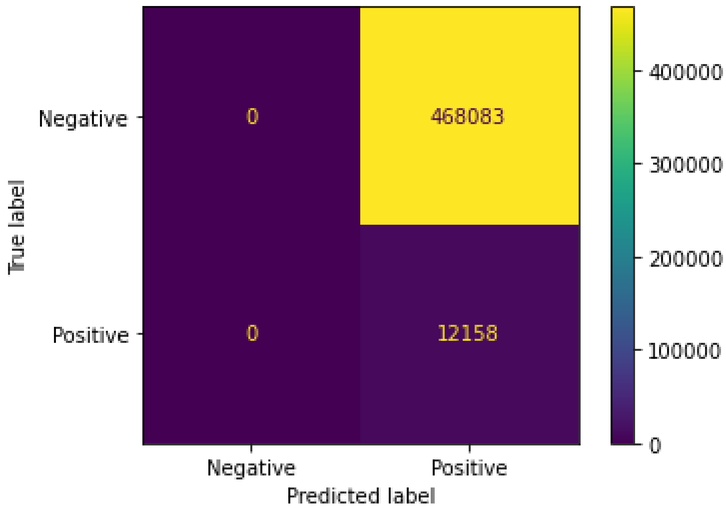

```
Level of classes:  Level 2
Test result: 74.0%
Training time: 3128.3s
ROC AUC score: 98.4%
```



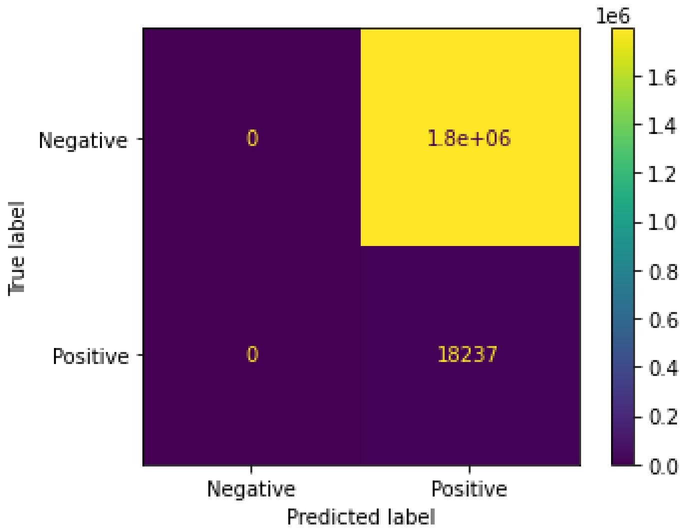
```
Level of classes:  Level 3
Test result: 50.9%
Training time: 3137.5s
ROC AUC score: 94.9%
```




```
Level of classes:  Level 1 and 2
Test result: 4.9%
Training time: 3071.9s
ROC AUC score: 50.0%
```




```
Level of classes:  All
Test result: 2.0%
Training time: 3097.8s
ROC AUC score: 50.0%
```



Out[13]: ['rnn_results.joblib']

RNN test results

Summary

In [18]:

```
pd.DataFrame(rnn_results, index=levels_str, columns=['F1 score (%)', 'ROC AUC score
```

Out[18]:

	F1 score (%)	ROC AUC score (%)	Training time (s)
Level 1	90.834367	99.588824	3145.805464
Level 2	73.972603	98.427639	3128.305139
Level 3	50.936742	94.909046	3137.454882
Level 1 and 2	4.938272	50.000000	3071.896999
All	1.993355	50.000000	3097.795459

The f1 score and ROC AUC score of the models for a single level of classes get lower as the number of classes increases. The models for combined levels of classes did not work very well. It can be seen from the confusion matrix that all classes are applied to every text. Using the best hyperparameters for the model for level 1 classes might caused the results of the other models. There is not much difference in the training times between each model.

Evaluate

In [16]:

```
knn_results = load('knn_results.joblib')
rnn_results = load('rnn_results.joblib')
```

Overall Results

In [19]:

```
pd.DataFrame(np.concatenate((knn_results, rnn_results), axis=1), index=levels_str, c
```

Out[19]:

	kNN f1 score (%)	kNN ROC AUC score (%)	kNN training time (s)	RNN f1 score (%)	RNN ROC AUC score (%)	RNN training time (s)
Level 1	75.305129	86.109135	0.110348	90.834367	99.588824	3145.805464
Level 2	56.528527	78.435288	0.741983	73.972603	98.427639	3128.305139
Level 3	53.726765	76.686846	2.315311	50.936742	94.909046	3137.454882
Level 1 and 2	65.988452	83.375354	0.841357	4.938272	50.000000	3071.896999
All	62.037420	80.473678	3.229993	1.993355	50.000000	3097.795459

As shown in the table above, the RNN models perform better for level 1 and level 2 but worse for level 3 and combined levels of classes in terms of F1 score. However, the ROC AUC score of RNN models for single level of classes are much better than that of kNN models. Training time of RNN models is much longer than training time of kNN models.

Final test

Testing the best kNN models with 3 test data that randomly chosen.

In [119…

```
knns = load('knns.joblib')
x_test_knn = load('x_test_knn.joblib')
y_test_knn = load('y_test.joblib')
x_test_rnn = load('x_test_rnn.joblib')
y_test_rnn = load('y_test_rnn.joblib')
mlbs_knn = load('mlbs.joblib')
mlbs_rnn = load('mlbs_rnn.joblib')
knn_thres = load('knn_thres.joblib')
rnn_thres = load('rnn_thres.joblib')
```

In [121…

```
for i in random.sample(range(0, len(x_test_knn)), 3):
    print("Actual classes: ", mlbs_knn[4].inverse_transform(np.array([y_test_knn[4][
    print()
    for j, knn in enumerate(knns):
        print("Level of classes: ", levels_str[j])
        y_pred_proba = knn[0].predict_proba(x_test_knn[knn[1]][i])
        y_pred = np.array(y_pred_proba)[...,1].flatten()
        print("Output: ", mlbs_knn[j].inverse_transform(np.where(np.array([y_pred])
        print()
    print()
```

```
Actual classes:  [('Dam', 'Infrastructure', 'Place')]

Level of classes:  Level 1
Output:  [('Agent',)]

Level of classes:  Level 2
Output:  [()]

Level of classes:  Level 3
Output:  [()]

Level of classes:  Level 1 and 2
Output:  [('Agent',)]

Level of classes:  All
Output:  [('Agent',)]


Actual classes:  [('Agent', 'Politician', 'PrimeMinister')]

Level of classes:  Level 1
Output:  [('Agent',)]

Level of classes:  Level 2
Output:  [()]

Level of classes:  Level 3
Output:  [()]

Level of classes:  Level 1 and 2
Output:  [('Agent',)]

Level of classes:  All
Output:  [('Agent',)]


Actual classes:  [('Actor', 'AdultActor', 'Agent')]

Level of classes:  Level 1
Output:  [('Agent',)]

Level of classes:  Level 2
Output:  [()]

Level of classes:  Level 3
```

```
Output:  [()]

Level of classes:  Level 1 and 2
Output:  [('Agent',)]

Level of classes:  All
Output:  [('Agent',)]
```

The models didn't provide expected results as shown above. The models for level 2 and level 3 didn't return any labels and other models only returned 'Agent'. kNN did not perform well even though the optimal input data and hyperparameters were used to train the models. It is likely to be resulted by imbalanced data as the models returned 'Agent' which is the most common class. It would provide better results if the data were balanced before the training. It might be a case that kNN does not perform very well for text classification.

Testing the best RNN models with 3 test data that randomly chosen.

In [124…

```
tf.get_logger().setLevel('ERROR')
for i in random.sample(range(0, len(x_test_rnn)), 3):
    print("Actual classes: ", mlbs_rnn[4].inverse_transform(np.array([y_test_rnn[4][
    print()
    for j, level in enumerate(levels):
        rnn = load_model('rnn{}'.format(j))
        print("Level of classes: ", levels_str[j])
        y_pred = rnn.predict(np.array([x_test_rnn[i]]))
        print("Output: ", mlbs_rnn[j].inverse_transform(np.where(y_pred < rnn_thres[
        print()
    print()
```

```
Actual classes:  [('Building', 'Place', 'Prison')]

Level of classes:  Level 1
Output:  [('Place',)]

Level of classes:  Level 2
Output:  [('Building',)]

Level of classes:  Level 3
Output:  [('Prison',)]

Level of classes:  Level 1 and 2
Output:  [('Actor', 'Agent', 'AmusementParkAttraction', 'Animal', 'Artist', 'Athlet
e', 'BodyOfWater', 'Boxer', 'BritishRoyalty', 'Broadcaster', 'Building', 'Cartoon',
'CelestialBody', 'Cleric', 'ClericalAdministrativeRegion', 'Coach', 'Comic', 'Comics
Character', 'Company', 'Database', 'Device', 'EducationalInstitution', 'Engine', 'Eu
karyote', 'Event', 'FictionalCharacter', 'FloweringPlant', 'FootballLeagueSeason',
'Genre', 'GridironFootballPlayer', 'Group', 'Horse', 'Infrastructure', 'LegalCase',
'MotorcycleRider', 'MusicalArtist', 'MusicalWork', 'NaturalEvent', 'NaturalPlace',
'Olympics', 'Organisation', 'OrganisationMember', 'PeriodicalLiterature', 'Person',
'Place', 'Plant', 'Politician', 'Presenter', 'Race', 'RaceTrack', 'RacingDriver', 'R
outeOfTransportation', 'Satellite', 'Scientist', 'Settlement', 'SocietalEvent', 'Sof
tware', 'Song', 'Species', 'SportFacility', 'SportsEvent', 'SportsLeague', 'SportsMa
nager', 'SportsSeason', 'SportsTeam', 'SportsTeamSeason', 'Station', 'Stream', 'Topi
calConcept', 'Tournament', 'Tower', 'UnitOfWork', 'Venue', 'VolleyballPlayer', 'Wint
erSportPlayer', 'Work', 'Wrestler', 'Writer', 'WrittenWork')]

Level of classes:  All
Output:  [('AcademicJournal', 'Actor', 'AdultActor', 'Agent', 'Airline', 'Airport',
'Album', 'AmateurBoxer', 'Ambassador', 'AmericanFootballPlayer', 'Amphibian', 'Amuse
mentParkAttraction', 'Animal', 'AnimangaCharacter', 'Anime', 'Arachnid', 'Architec
t', 'ArtificialSatellite', 'Artist', 'ArtistDiscography', 'Astronaut', 'Athlete', 'A
ustralianFootballTeam', 'AustralianRulesFootballPlayer', 'AutomobileEngine', 'Badmin
tonPlayer', 'Band', 'Bank', 'Baronet', 'BaseballLeague', 'BaseballPlayer', 'Baseball
```

```
Season', 'BasketballLeague', 'BasketballPlayer', 'BasketballTeam', 'BeachVolleyballP
layer', 'BeautyQueen', 'BiologicalDatabase', 'Bird', 'BodyOfWater', 'Bodybuilder',
'Boxer', 'Brewery', 'Bridge', 'BritishRoyalty', 'BroadcastNetwork', 'Broadcaster',
'Building', 'BusCompany', 'BusinessPerson', 'CanadianFootballTeam', 'Canal', 'Canoei
st', 'Cardinal', 'Cartoon', 'Castle', 'Cave', 'CelestialBody', 'Chef', 'ChessPlaye
r', 'ChristianBishop', 'ClassicalMusicArtist', 'ClassicalMusicComposition', 'Cleri
c', 'ClericalAdministrativeRegion', 'Coach', 'CollegeCoach', 'Comedian', 'Comic', 'C
omicStrip', 'ComicsCharacter', 'ComicsCreator', 'Company', 'Congressman', 'Conifer',
'Convention', 'CricketGround', 'CricketTeam', 'Cricketer', 'Crustacean', 'Cultivated
Variety', 'Curler', 'Cycad', 'CyclingRace', 'CyclingTeam', 'Cyclist', 'Dam', 'DartsP
layer', 'Database', 'Device', 'Diocese', 'Earthquake', 'Economist', 'EducationalInst
itution', 'Election', 'Engine', 'Engineer', 'Entomologist', 'Eukaryote', 'Eurovision
SongContestEntry', 'Event', 'FashionDesigner', 'Fern', 'FictionalCharacter', 'Figure
Skater', 'FilmFestival', 'Fish', 'FloweringPlant', 'FootballLeagueSeason', 'Football
Match', 'FormulaOneRacer', 'Fungus', 'GaelicGamesPlayer', 'Galaxy', 'Genre', 'Glacie
r', 'GolfCourse', 'GolfPlayer', 'GolfTournament', 'Governor', 'GrandPrix', 'Grape',
'GreenAlga', 'GridironFootballPlayer', 'Group', 'Gymnast', 'HandballPlayer', 'Handba
llTeam', 'Historian', 'HistoricBuilding', 'HockeyTeam', 'HollywoodCartoon', 'Horse',
'HorseRace', 'HorseRider', 'HorseTrainer', 'Hospital', 'Hotel', 'IceHockeyLeague',
'IceHockeyPlayer', 'Infrastructure', 'Insect', 'Jockey', 'Journalist', 'Judge', 'Lac
rossePlayer', 'Lake', 'LawFirm', 'LegalCase', 'Legislature', 'Library', 'Lighthous
e', 'Magazine', 'Manga', 'MartialArtist', 'Mayor', 'Medician', 'MemberOfParliament',
'MilitaryConflict', 'MilitaryPerson', 'MilitaryUnit', 'MixedMartialArtsEvent', 'Mode
l', 'Mollusca', 'Monarch', 'Moss', 'MotorcycleRider', 'Mountain', 'MountainPass', 'M
ountainRange', 'Museum', 'MusicFestival', 'MusicGenre', 'Musical', 'MusicalArtist',
'MusicalWork', 'MythologicalFigure', 'NCAATeamSeason', 'NascarDriver', 'NationalFoot
ballLeagueSeason', 'NaturalEvent', 'NaturalPlace', 'NetballPlayer', 'Newspaper', 'No
ble', 'OfficeHolder', 'OlympicEvent', 'Olympics', 'Organisation', 'OrganisationMembe
r', 'Painter', 'PeriodicalLiterature', 'Person', 'Philosopher', 'Photographer', 'Pla
ce', 'Planet', 'Plant', 'Play', 'PlayboyPlaymate', 'Poem', 'Poet', 'PokerPlayer', 'P
oliticalParty', 'Politician', 'Pope', 'Presenter', 'President', 'PrimeMinister', 'Pr
ison', 'PublicTransitSystem', 'Publisher', 'Race', 'RaceHorse', 'RaceTrack', 'Raceco
urse', 'RacingDriver', 'RadioHost', 'RadioStation', 'RailwayLine', 'RailwayStation',
'RecordLabel', 'Religious', 'Reptile', 'Restaurant', 'River', 'Road', 'RoadTunnel',
'RollerCoaster', 'RouteOfTransportation', 'Rower', 'RugbyClub', 'RugbyLeague', 'Rugb
yPlayer', 'Saint', 'Satellite', 'School', 'Scientist', 'ScreenWriter', 'Senator', 'S
ettlement', 'ShoppingMall', 'Single', 'Skater', 'Skier', 'SoapCharacter', 'SoccerClu
bSeason', 'SoccerLeague', 'SoccerManager', 'SoccerPlayer', 'SoccerTournament', 'Soci
etalEvent', 'Software', 'SolarEclipse', 'Song', 'Species', 'SpeedwayRider', 'SportFa
cility', 'SportsEvent', 'SportsLeague', 'SportsManager', 'SportsSeason', 'SportsTea
m', 'SportsTeamMember', 'SportsTeamSeason', 'SquashPlayer', 'Stadium', 'Station', 'S
tream', 'SumoWrestler', 'SupremeCourtOfTheUnitedStatesCase', 'Swimmer', 'TableTennis
Player', 'TelevisionStation', 'TennisPlayer', 'TennisTournament', 'Theatre', 'Topica
lConcept', 'Tournament', 'Tower', 'Town', 'TradeUnion', 'UnitOfWork', 'University',
'Venue', 'VideoGame', 'Village', 'VoiceActor', 'Volcano', 'VolleyballPlayer', 'Winer
y', 'WinterSportPlayer', 'WomensTennisAssociationTournament', 'Work', 'Wrestler', 'W
restlingEvent', 'Writer', 'WrittenWork')]


Actual classes:  [('Agent', 'MemberOfParliament', 'Politician')]

Level of classes:  Level 1
Output:  [('Agent',)]

Level of classes:  Level 2
Output:  [('Politician',)]

Level of classes:  Level 3
Output:  [()]

Level of classes:  Level 1 and 2
Output:  [('Actor', 'Agent', 'AmusementParkAttraction', 'Animal', 'Artist', 'Athlet
e', 'BodyOfWater', 'Boxer', 'BritishRoyalty', 'Broadcaster', 'Building', 'Cartoon',
'CelestialBody', 'Cleric', 'ClericalAdministrativeRegion', 'Coach', 'Comic', 'Comics
Character', 'Company', 'Database', 'Device', 'EducationalInstitution', 'Engine', 'Eu
karyote', 'Event', 'FictionalCharacter', 'FloweringPlant', 'FootballLeagueSeason',
'Genre', 'GridironFootballPlayer', 'Group', 'Horse', 'Infrastructure', 'LegalCase',
'MotorcycleRider', 'MusicalArtist', 'MusicalWork', 'NaturalEvent', 'NaturalPlace',
'Olympics', 'Organisation', 'OrganisationMember', 'PeriodicalLiterature', 'Person',
```

```
'Place', 'Plant', 'Politician', 'Presenter', 'Race', 'RaceTrack', 'RacingDriver', 'R
outeOfTransportation', 'Satellite', 'Scientist', 'Settlement', 'SocietalEvent', 'Sof
tware', 'Song', 'Species', 'SportFacility', 'SportsEvent', 'SportsLeague', 'SportsMa
nager', 'SportsSeason', 'SportsTeam', 'SportsTeamSeason', 'Station', 'Stream', 'Topi
calConcept', 'Tournament', 'Tower', 'UnitOfWork', 'Venue', 'VolleyballPlayer', 'Wint
erSportPlayer', 'Work', 'Wrestler', 'Writer', 'WrittenWork')]

Level of classes:  All
Output:  [('AcademicJournal', 'Actor', 'AdultActor', 'Agent', 'Airline', 'Airport',
'Album', 'AmateurBoxer', 'Ambassador', 'AmericanFootballPlayer', 'Amphibian', 'Amuse
mentParkAttraction', 'Animal', 'AnimangaCharacter', 'Anime', 'Arachnid', 'Architec
t', 'ArtificialSatellite', 'Artist', 'ArtistDiscography', 'Astronaut', 'Athlete', 'A
ustralianFootballTeam', 'AustralianRulesFootballPlayer', 'AutomobileEngine', 'Badmin
tonPlayer', 'Band', 'Bank', 'Baronet', 'BaseballLeague', 'BaseballPlayer', 'Baseball
Season', 'BasketballLeague', 'BasketballPlayer', 'BasketballTeam', 'BeachVolleyballP
layer', 'BeautyQueen', 'BiologicalDatabase', 'Bird', 'BodyOfWater', 'Bodybuilder',
'Boxer', 'Brewery', 'Bridge', 'BritishRoyalty', 'BroadcastNetwork', 'Broadcaster',
'Building', 'BusCompany', 'BusinessPerson', 'CanadianFootballTeam', 'Canal', 'Canoei
st', 'Cardinal', 'Cartoon', 'Castle', 'Cave', 'CelestialBody', 'Chef', 'ChessPlaye
r', 'ChristianBishop', 'ClassicalMusicArtist', 'ClassicalMusicComposition', 'Cleri
c', 'ClericalAdministrativeRegion', 'Coach', 'CollegeCoach', 'Comedian', 'Comic', 'C
omicStrip', 'ComicsCharacter', 'ComicsCreator', 'Company', 'Congressman', 'Conifer',
'Convention', 'CricketGround', 'CricketTeam', 'Cricketer', 'Crustacean', 'Cultivated
Variety', 'Curler', 'Cycad', 'CyclingRace', 'CyclingTeam', 'Cyclist', 'Dam', 'DartsP
layer', 'Database', 'Device', 'Diocese', 'Earthquake', 'Economist', 'EducationalInst
itution', 'Election', 'Engine', 'Engineer', 'Entomologist', 'Eukaryote', 'Eurovision
SongContestEntry', 'Event', 'FashionDesigner', 'Fern', 'FictionalCharacter', 'Figure
Skater', 'FilmFestival', 'Fish', 'FloweringPlant', 'FootballLeagueSeason', 'Football
Match', 'FormulaOneRacer', 'Fungus', 'GaelicGamesPlayer', 'Galaxy', 'Genre', 'Glacie
r', 'GolfCourse', 'GolfPlayer', 'GolfTournament', 'Governor', 'GrandPrix', 'Grape',
'GreenAlga', 'GridironFootballPlayer', 'Group', 'Gymnast', 'HandballPlayer', 'Handba
llTeam', 'Historian', 'HistoricBuilding', 'HockeyTeam', 'HollywoodCartoon', 'Horse',
'HorseRace', 'HorseRider', 'HorseTrainer', 'Hospital', 'Hotel', 'IceHockeyLeague',
'IceHockeyPlayer', 'Infrastructure', 'Insect', 'Jockey', 'Journalist', 'Judge', 'Lac
rossePlayer', 'Lake', 'LawFirm', 'LegalCase', 'Legislature', 'Library', 'Lighthous
e', 'Magazine', 'Manga', 'MartialArtist', 'Mayor', 'Medician', 'MemberOfParliament',
'MilitaryConflict', 'MilitaryPerson', 'MilitaryUnit', 'MixedMartialArtsEvent', 'Mode
l', 'Mollusca', 'Monarch', 'Moss', 'MotorcycleRider', 'Mountain', 'MountainPass', 'M
ountainRange', 'Museum', 'MusicFestival', 'MusicGenre', 'Musical', 'MusicalArtist',
'MusicalWork', 'MythologicalFigure', 'NCAATeamSeason', 'NascarDriver', 'NationalFoot
ballLeagueSeason', 'NaturalEvent', 'NaturalPlace', 'NetballPlayer', 'Newspaper', 'No
ble', 'OfficeHolder', 'OlympicEvent', 'Olympics', 'Organisation', 'OrganisationMembe
r', 'Painter', 'PeriodicalLiterature', 'Person', 'Philosopher', 'Photographer', 'Pla
ce', 'Planet', 'Plant', 'Play', 'PlayboyPlaymate', 'Poem', 'Poet', 'PokerPlayer', 'P
oliticalParty', 'Politician', 'Pope', 'Presenter', 'President', 'PrimeMinister', 'Pr
ison', 'PublicTransitSystem', 'Publisher', 'Race', 'RaceHorse', 'RaceTrack', 'Raceco
urse', 'RacingDriver', 'RadioHost', 'RadioStation', 'RailwayLine', 'RailwayStation',
'RecordLabel', 'Religious', 'Reptile', 'Restaurant', 'River', 'Road', 'RoadTunnel',
'RollerCoaster', 'RouteOfTransportation', 'Rower', 'RugbyClub', 'RugbyLeague', 'Rugb
yPlayer', 'Saint', 'Satellite', 'School', 'Scientist', 'ScreenWriter', 'Senator', 'S
ettlement', 'ShoppingMall', 'Single', 'Skater', 'Skier', 'SoapCharacter', 'SoccerClu
bSeason', 'SoccerLeague', 'SoccerManager', 'SoccerPlayer', 'SoccerTournament', 'Soci
etalEvent', 'Software', 'SolarEclipse', 'Song', 'Species', 'SpeedwayRider', 'SportFa
cility', 'SportsEvent', 'SportsLeague', 'SportsManager', 'SportsSeason', 'SportsTea
m', 'SportsTeamMember', 'SportsTeamSeason', 'SquashPlayer', 'Stadium', 'Station', 'S
tream', 'SumoWrestler', 'SupremeCourtOfTheUnitedStatesCase', 'Swimmer', 'TableTennis
Player', 'TelevisionStation', 'TennisPlayer', 'TennisTournament', 'Theatre', 'Topica
lConcept', 'Tournament', 'Tower', 'Town', 'TradeUnion', 'UnitOfWork', 'University',
'Venue', 'VideoGame', 'Village', 'VoiceActor', 'Volcano', 'VolleyballPlayer', 'Winer
y', 'WinterSportPlayer', 'WomensTennisAssociationTournament', 'Work', 'Wrestler', 'W
restlingEvent', 'Writer', 'WrittenWork')]


Actual classes:  [('Agent', 'Mayor', 'Politician')]

Level of classes:  Level 1
Output:  [('Agent',)]

Level of classes:  Level 2
```

```
Output:  [('Politician',)]

Level of classes:  Level 3
Output:  [()]

Level of classes:  Level 1 and 2
Output:  [('Actor', 'Agent', 'AmusementParkAttraction', 'Animal', 'Artist', 'Athlet
e', 'BodyOfWater', 'Boxer', 'BritishRoyalty', 'Broadcaster', 'Building', 'Cartoon',
'CelestialBody', 'Cleric', 'ClericalAdministrativeRegion', 'Coach', 'Comic', 'Comics
Character', 'Company', 'Database', 'Device', 'EducationalInstitution', 'Engine', 'Eu
karyote', 'Event', 'FictionalCharacter', 'FloweringPlant', 'FootballLeagueSeason',
'Genre', 'GridironFootballPlayer', 'Group', 'Horse', 'Infrastructure', 'LegalCase',
'MotorcycleRider', 'MusicalArtist', 'MusicalWork', 'NaturalEvent', 'NaturalPlace',
'Olympics', 'Organisation', 'OrganisationMember', 'PeriodicalLiterature', 'Person',
'Place', 'Plant', 'Politician', 'Presenter', 'Race', 'RaceTrack', 'RacingDriver', 'R
outeOfTransportation', 'Satellite', 'Scientist', 'Settlement', 'SocietalEvent', 'Sof
tware', 'Song', 'Species', 'SportFacility', 'SportsEvent', 'SportsLeague', 'SportsMa
nager', 'SportsSeason', 'SportsTeam', 'SportsTeamSeason', 'Station', 'Stream', 'Topi
calConcept', 'Tournament', 'Tower', 'UnitOfWork', 'Venue', 'VolleyballPlayer', 'Wint
erSportPlayer', 'Work', 'Wrestler', 'Writer', 'WrittenWork')]

Level of classes:  All
Output:  [('AcademicJournal', 'Actor', 'AdultActor', 'Agent', 'Airline', 'Airport',
'Album', 'AmateurBoxer', 'Ambassador', 'AmericanFootballPlayer', 'Amphibian', 'Amuse
mentParkAttraction', 'Animal', 'AnimangaCharacter', 'Anime', 'Arachnid', 'Architec
t', 'ArtificialSatellite', 'Artist', 'ArtistDiscography', 'Astronaut', 'Athlete', 'A
ustralianFootballTeam', 'AustralianRulesFootballPlayer', 'AutomobileEngine', 'Badmin
tonPlayer', 'Band', 'Bank', 'Baronet', 'BaseballLeague', 'BaseballPlayer', 'Baseball
Season', 'BasketballLeague', 'BasketballPlayer', 'BasketballTeam', 'BeachVolleyballP
layer', 'BeautyQueen', 'BiologicalDatabase', 'Bird', 'BodyOfWater', 'Bodybuilder',
'Boxer', 'Brewery', 'Bridge', 'BritishRoyalty', 'BroadcastNetwork', 'Broadcaster',
'Building', 'BusCompany', 'BusinessPerson', 'CanadianFootballTeam', 'Canal', 'Canoei
st', 'Cardinal', 'Cartoon', 'Castle', 'Cave', 'CelestialBody', 'Chef', 'ChessPlaye
r', 'ChristianBishop', 'ClassicalMusicArtist', 'ClassicalMusicComposition', 'Cleri
c', 'ClericalAdministrativeRegion', 'Coach', 'CollegeCoach', 'Comedian', 'Comic', 'C
omicStrip', 'ComicsCharacter', 'ComicsCreator', 'Company', 'Congressman', 'Conifer',
'Convention', 'CricketGround', 'CricketTeam', 'Cricketer', 'Crustacean', 'Cultivated
Variety', 'Curler', 'Cycad', 'CyclingRace', 'CyclingTeam', 'Cyclist', 'Dam', 'DartsP
layer', 'Database', 'Device', 'Diocese', 'Earthquake', 'Economist', 'EducationalInst
itution', 'Election', 'Engine', 'Engineer', 'Entomologist', 'Eukaryote', 'Eurovision
SongContestEntry', 'Event', 'FashionDesigner', 'Fern', 'FictionalCharacter', 'Figure
Skater', 'FilmFestival', 'Fish', 'FloweringPlant', 'FootballLeagueSeason', 'Football
Match', 'FormulaOneRacer', 'Fungus', 'GaelicGamesPlayer', 'Galaxy', 'Genre', 'Glacie
r', 'GolfCourse', 'GolfPlayer', 'GolfTournament', 'Governor', 'GrandPrix', 'Grape',
'GreenAlga', 'GridironFootballPlayer', 'Group', 'Gymnast', 'HandballPlayer', 'Handba
llTeam', 'Historian', 'HistoricBuilding', 'HockeyTeam', 'HollywoodCartoon', 'Horse',
'HorseRace', 'HorseRider', 'HorseTrainer', 'Hospital', 'Hotel', 'IceHockeyLeague',
'IceHockeyPlayer', 'Infrastructure', 'Insect', 'Jockey', 'Journalist', 'Judge', 'Lac
rossePlayer', 'Lake', 'LawFirm', 'LegalCase', 'Legislature', 'Library', 'Lighthous
e', 'Magazine', 'Manga', 'MartialArtist', 'Mayor', 'Medician', 'MemberOfParliament',
'MilitaryConflict', 'MilitaryPerson', 'MilitaryUnit', 'MixedMartialArtsEvent', 'Mode
l', 'Mollusca', 'Monarch', 'Moss', 'MotorcycleRider', 'Mountain', 'MountainPass', 'M
ountainRange', 'Museum', 'MusicFestival', 'MusicGenre', 'Musical', 'MusicalArtist',
'MusicalWork', 'MythologicalFigure', 'NCAATeamSeason', 'NascarDriver', 'NationalFoot
ballLeagueSeason', 'NaturalEvent', 'NaturalPlace', 'NetballPlayer', 'Newspaper', 'No
ble', 'OfficeHolder', 'OlympicEvent', 'Olympics', 'Organisation', 'OrganisationMembe
r', 'Painter', 'PeriodicalLiterature', 'Person', 'Philosopher', 'Photographer', 'Pla
ce', 'Planet', 'Plant', 'Play', 'PlayboyPlaymate', 'Poem', 'Poet', 'PokerPlayer', 'P
oliticalParty', 'Politician', 'Pope', 'Presenter', 'President', 'PrimeMinister', 'Pr
ison', 'PublicTransitSystem', 'Publisher', 'Race', 'RaceHorse', 'RaceTrack', 'Raceco
urse', 'RacingDriver', 'RadioHost', 'RadioStation', 'RailwayLine', 'RailwayStation',
'RecordLabel', 'Religious', 'Reptile', 'Restaurant', 'River', 'Road', 'RoadTunnel',
'RollerCoaster', 'RouteOfTransportation', 'Rower', 'RugbyClub', 'RugbyLeague', 'Rugb
yPlayer', 'Saint', 'Satellite', 'School', 'Scientist', 'ScreenWriter', 'Senator', 'S
ettlement', 'ShoppingMall', 'Single', 'Skater', 'Skier', 'SoapCharacter', 'SoccerClu
bSeason', 'SoccerLeague', 'SoccerManager', 'SoccerPlayer', 'SoccerTournament', 'Soci
etalEvent', 'Software', 'SolarEclipse', 'Song', 'Species', 'SpeedwayRider', 'SportFa
cility', 'SportsEvent', 'SportsLeague', 'SportsManager', 'SportsSeason', 'SportsTea
m', 'SportsTeamMember', 'SportsTeamSeason', 'SquashPlayer', 'Stadium', 'Station', 'S
```

```
tream', 'SumoWrestler', 'SupremeCourtOfTheUnitedStatesCase', 'Swimmer', 'TableTennis
Player', 'TelevisionStation', 'TennisPlayer', 'TennisTournament', 'Theatre', 'Topica
lConcept', 'Tournament', 'Tower', 'Town', 'TradeUnion', 'UnitOfWork', 'University',
'Venue', 'VideoGame', 'Village', 'VoiceActor', 'Volcano', 'VolleyballPlayer', 'Winer
y', 'WinterSportPlayer', 'WomensTennisAssociationTournament', 'Work', 'Wrestler', 'W
restlingEvent', 'Writer', 'WrittenWork')]
```

The RNN models for level 1 and level 2 provide the correct classes for every text. The model for level 3 provides the correct class or no class. It can be because of the hyperparameters used for training models were tuned for the model for level 1. The model might perfome better if it is trained for more epochs as there are more classes. The models for combined levels of classes did not perform well as these return all classes for every text. These models might perform better if these are trained more or hyperparameters are tuned for these models. It might be the case that the architecture of the models need to be changed.

Conclusion

In this experimentation, kNN and RNN models were trained and tested with various hyperparameters for different numbers of classes. N-grams were applied for kNN models but it did not give much difference to the results. The hyperparameters were tuned for each classifier and the best ones were found and used for training the models. Compared to kNN, RNN performs much better for level 1 and level 2 classes although one-tenth of data and much less features were used for training. However, training time for RNN was much longer than that for kNN. The models for level 3 and combined levels did not work well. It can be said that having more classes and multiple classes for each text make the data more difficult to classify. Balancing data, tuning hyperparameters for each RNN model may provide better results.