In [ ]:
```
#Pretrained word2vec embeddings
```

In [1]:
```
#from torchtext import data
#from torchtext import datasets
import torch
import spacy
import random
import numpy as np
import pandas as pd
import time
import re # for regular expressions
import torch.nn as nn
import torch.nn.functional as F
from sklearn.model_selection import train_test_split
import spacy
nlp = spacy.load('en')

data =  pd.read_csv('transferlearning-dl-spring2020/train.csv')

data.head()

questions = data['text']
labels = data['target']

train_data, valid_data, ytrain, yvalid = train_test_split(questions, labels,
                                                          random_state=42,
                                                          test_size=0.2)


train_data.head()
```

Out[1]:
```
251      @USER All you need to know is he is empty inside
3720                           @USER I. AM. READ. E! URL
8376    @USER Go roger I quit watching anyway nfl is o...
4471                @USER Yoo our dogs should totally fuck
2835               @USER He is a troll.  Not open to facts
Name: text, dtype: object
```

In [2]:
```
data.head()
```

Out[2]:

	id	text	target
0	86426	@USER She should ask a few native Americans wh...	1
1	16820	Amazon is investigating Chinese employees who ...	0
2	62688	@USER Someone should'veTaken" this piece of sh...	1
3	43605	@USER @USER Obama wanted liberals & illega...	0
4	97670	@USER Liberals are all Kookoo !!!	1

In [3]:
```
def remove_pattern(input_txt, pattern):
    r = re.findall(pattern, input_txt)
    for i in r:
        input_txt = re.sub(i, '', input_txt)

    return input_txt


data['text'] = np.vectorize(remove_pattern)(data['text'], "@[\w]*")


data['text'] = data['text'].str.replace("[^a-zA-Z#]", " ")

data['text'] = data['text'].str.replace('#','')


data['text'] = data['text'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>2]))



tokenized_tweet = data['text'].apply(lambda x: x.split()) # tokenizing

tokenized_tweet.head()

len(tokenized_tweet)

#from nltk.stem.porter import *
#stemmer = PorterStemmer()

#tokenized_tweet = tokenized_tweet.apply(lambda x: [stemmer.stem(i) for i in x]) # stemming

from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
tokenized_tweet.apply(lambda x: [lemmatizer.lemmatize(i) for i in x]) # stemming
#lemmatized_output = ' '.join([lemmatizer.lemmatize(token) for token in tokens])
```

Out[3]:
```
0       [She, should, ask, few, native, Americans, wha...
1       [Amazon, investigating, Chinese, employee, who...
2       [Someone, should, veTaken, this, piece, shit, ...
3       [Obama, wanted, liberal, amp, illegals, move, ...
4                            [Liberals, are, all, Kookoo]
5       [wa, literally, just, talking, about, this, lo...
6                                  [Buy, more, icecream]
7                 [not, fault, you, support, gun, control]
8       [What, the, difference, between, Kavanaugh, an...
9       [you, are, lying, corrupt, traitor, Nobody, wa...
10      [like, soda, like, like, boarder, with, lot, ICE]
11                   [you, are, also, the, king, taste]
12      [MAGA, Sing, like, one, listening, Love, like,...
13      [The, time, right, for, this, House, respond, ...
14      [Besides, Jax, mom, and, maybe, Ope, hand, dow...
15      [gun, control, That, all, these, kid, are, ask...
16      [fuck, going, people, There, the, men, room, a...
17      [Been, Willie, fan, since, before, most, you, ...
18      [Tbh, these, day, just, don, like, people, gen...
19      [South, Korean, Official, Leaders, will, discu...
20                        [You, can, tell, hooper, too]
21              [feel, like, better, chasing, the, title]
22      [She, whom, are, you, referring, Hillary, You,...
23      [Glad, see, your, friend, are, supporting, Met...
24      [Yes, you, are, but, wa, asking, what, about, ...
25      [wonder, being, apologetic, and, more, social,...
26      [Any, update, ending, your, blatant, racism, W...
27                                         [Blow, hard]
28      [That, mean, you, are, max, lvl, Twitter, user...
29      [Please, explain, what, controlled, opposition...
                              ...
9316       [Berkeley, Antifa, not, agree, with, you, URL]
9317    [Advocate, for, gun, control, while, breaking,...
9318    [More, press, should, talk, about, this, remar...
9319    [MohammadHassanKhalid, traitor, the, USA, amp,...
9320                       [nah, coz, you, cunt, blocked]
9321                      [Cats, are, just, special, dog]
9322    [Best, news, ever, for, GOP, Ready, the, strai...
9323    [Strength, letting, the, universe, You, are, n...
9324                                [Wow, You, are, good]
9325    [still, lie, just, like, Obama, talk, right, a...
9326                                           [precious]
9327                                          [Booooring]
9328    [Everything, else, wa, ten, year, ago, YOU, AR...
9329                              [Right, Dang, She, the]
9330    [McRaven, engaged, publicity, stunt, never, sp...
9331    [man, You, are, going, trigger, the, fanboys, ...
9332    [Chelsea, never, end, You, are, always, the, b...
9333                                 [head, fuckin, egg]
9334    [Alt, Right, amp, Antifa, are, for, coward, Fa...
9335    [Did, you, serve, You, rate, bottom, the, barr...
9336    [just, trying, make, good, with, his, libnut, ...
9337    [have, the, conservative, accepted, the, antis...
9338    [Can, all, agree, that, Tomlins, seat, heating...
9339    [involved, because, wa, there, Now, need, man,...
9340    [How, much, lonely, she, and, how, much, she, ...
9341                                  [BUT, GUN, CONTROL]
9342    [say, you, are, mad, now, you, will, say, tire...
9343     [Retweet, complete, amp, followed, all, patriot]
9344    [And, why, report, this, garbage, don, give, c...
9345                                             [Pussy]
Name: text, Length: 9346, dtype: object
```

In [4]:
```
tokenized_tweet[1:10]
```

Out[4]:
```
1    [Amazon, investigating, Chinese, employees, wh...
2    [Someone, should, veTaken, this, piece, shit, ...
3    [Obama, wanted, liberals, amp, illegals, move,...
4                         [Liberals, are, all, Kookoo]
5    [was, literally, just, talking, about, this, l...
6                                [Buy, more, icecream]
7             [not, fault, you, support, gun, control]
8    [What, the, difference, between, Kavanaugh, an...
9    [you, are, lying, corrupt, traitor, Nobody, wa...
Name: text, dtype: object
```

In [8]:
```
# Using pre-trained word2vec embeddings created by Google

import gensim

model = gensim.models.KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=Tr
ue)

#model.train(tokenized_tweet, total_examples= len(combi['description']), epochs=50)

def word_vector_pretrained(tokens, size):
    vec = np.zeros(size).reshape((1, size))
    count = 0.
    for word in tokens:
        try:
            vec += model[word].reshape((1, size))
            count += 1.
        except KeyError: # handling the case where the token is not in vocabulary

            continue
    if count != 0:
        vec /= count
    return vec

wordvec_arrays = np.zeros((len(tokenized_tweet), 300))

for i in range(len(tokenized_tweet)):
    wordvec_arrays[i,:] = word_vector_pretrained(tokenized_tweet[i], 300)

wordvec_df = pd.DataFrame(wordvec_arrays)
wordvec_df.shape

X_train, X_test , y_train, y_test = train_test_split(wordvec_df, labels, test_size=0.2, random_state=1)
```

In [9]:
```
np.unique(y_train)
```

Out[9]:
```
array([0, 1])
```

In [10]:
```
tokenized_tweet[1:5]
```

Out[10]:
```
1    [Amazon, investigating, Chinese, employees, wh...
2    [Someone, should, veTaken, this, piece, shit, ...
3    [Obama, wanted, liberals, amp, illegals, move,...
4                         [Liberals, are, all, Kookoo]
Name: text, dtype: object
```

In [11]:
```
X_train[1:10]
```

Out[11]:

	0	1	2	3	4	5	6	7	8	9	...	290	291
5633	-0.006725	0.010405	-0.052097	0.091202	-0.059473	0.020924	0.077883	-0.019114	0.072357	0.102734	...	-0.086390	0.052601
4507	0.029175	0.037109	0.033051	0.113220	-0.112213	0.008484	0.028618	0.043793	0.001225	-0.042809	...	0.010437	0.056587
3891	0.033418	0.003791	0.030201	0.066354	-0.040874	0.010851	0.012351	-0.074940	0.095403	0.074498	...	-0.059332	0.024525
5429	0.017432	0.039795	-0.022937	-0.007690	0.017090	-0.051917	0.034424	-0.123288	0.058185	0.031116	...	-0.018947	-0.060611
1104	0.023802	0.033569	0.142127	0.134559	-0.098013	-0.016130	0.057371	-0.051627	0.050809	0.048138	...	-0.015437	0.078177
6560	0.061476	0.052678	0.018585	0.084633	-0.116528	-0.019473	-0.007350	-0.011179	0.071787	0.027387	...	-0.034809	0.024897
3332	0.066352	0.058706	0.082484	0.095835	-0.066877	0.002801	0.020893	0.003180	0.003379	0.077595	...	-0.090912	0.021102
6211	0.007294	-0.013245	-0.009155	-0.039429	0.000488	0.053101	-0.025269	0.000488	0.074219	0.051147	...	-0.229492	-0.023682
2364	0.061823	-0.040497	0.053802	0.194775	-0.059338	0.013220	0.075983	-0.038326	0.097266	0.041626	...	-0.051514	-0.032813

9 rows × 300 columns

In [12]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
```

In [13]:
```
lreg = LogisticRegression(random_state = 10)
lreg.fit(X_train, y_train) # training the model

prediction = lreg.predict(X_test) # predicting on the validation set

prediction_int = prediction.astype(np.int)

f1_score(y_test, prediction,average = 'macro') # calculating f1 score
```

Out[13]:
```
0.6691278341911253
```

In [14]:
```
data_test =  pd.read_csv('transferlearning-dl-spring2020/test.csv')
```

In [15]:
```
data_test['text'] = np.vectorize(remove_pattern)(data_test['text'], "@[\w]*")


data_test['text'] = data_test['text'].str.replace("[^a-zA-Z#]", " ")

data_test['text'] = data_test['text'].str.replace('#','')


data_test['text'] = data_test['text'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>2]))



tokenized_tweet_test = data_test['text'].apply(lambda x: x.split()) # tokenizing

tokenized_tweet_test.head()

len(tokenized_tweet_test)

#from nltk.stem.porter import *
#stemmer = PorterStemmer()

#tokenized_tweet_test = tokenized_tweet_test.apply(lambda x: [stemmer.stem(i) for i in x]) # stemming

from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
tokenized_tweet_test.apply(lambda x: [lemmatizer.lemmatize(i) for i in x]) # stemming
```

Out[15]:
```
0                      [home, you, drunk, MAGA, Trump, URL]
1                                        [no, Tough, shit]
2       [Canada, doesn, need, another, CUCK, already, ...
3       [should, scare, every, American, She, playing,...
4       [LOL, Throwing, the, BULLSHIT, Flag, such, non...
5                                    [You, are, correct]
6       [Kind, like, when, conservative, wanna, associ...
7       [The, only, thing, the, Democrats, have, lying...
8       [you, know, what, going, happen, now, going, h...
9       [You, are, not, very, smart, are, you, Why, yo...
10      [That, expected, you, placate, the, violent, l...
11      [Wow, you, liberal, really, don, have, sense, ...
12                                      [FUCKING, READY]
13      [too, Her, wedding, wa, the, best, She, very, ...
14      [URL, any, your, announcement, every, come, fr...
15      [out, British, people, are, basically, full, r...
16      [GUNCONTROL, advocate, must, STOP, falling, al...
17                                          [Fuck, off]
18      [dumb, and, dumber, all, one, president, two, ...
19      [did, Twitter, silence, alex, jones, retaliati...
20                    [Thats, because, you, are, old, man]
21      [She, beautiful, person, teach, value, her, ki...
22                         [tear, made, water, and, feeling]
23      [babysitting, kid, people, how, old, you, thin...
24      [WTC, This, threat, Maga, QAnon, WakeUpAmerica...
25      [Unfortunately, America, system, like, that, w...
26        [Guy, you, are, always, present, bet, why, Lol]
27      [MAGA, YOU, ARE, ALL, FOR, TRUMP, FOLLOW, AND,...
28      [For, the, record, know, Doug, Jones, personal...
29      [There, are, many, dumb, argument, for, gun, c...
                              ...
3864    [Probably, while, screaming, about, gun, control]
3865    [Thinking, about, the, Student, Loan, Crisis, ...
3866    [the, tying, increase, the, cost, drug, Canada...
3867    [Damn, straight, Nothing, but, good, vibe, com...
3868    [THIS, SHOULD, REMIND, ALL, Patriots, maga, WH...
3869    [sorry, what, BAG, listening, that, amp, after...
3870    [Americans, make, great, client, But, many, ap...
3871    [You, should, know, better, use, common, sense...
3872                    [Keep, lying, bud, one, listening]
3873          [You, are, great, model, for, inspiration]
3874    [Chris, Chris, Chris, Are, you, forgetting, th...
3875        [Where, will, Antifa, get, their, cloth, now]
3876              [She, role, model, Adam, you, are, not]
3877    [The, entire, way, the, dems, have, handled, t...
3878    [Beto, Rourke, Ted, Cruz, Latest, Polls, Democ...
3879    [Project, much, you, Every, campaign, flier, g...
3880    [Link, and, knuckle, then, you, going, dodge, ...
3881    [Omg, not, even, interested, his, age, but, da...
3882                    [shit, you, weren, joking, wtf]
3883    [Bibi, look, like, Stalin, when, Stalin, wa, y...
3884    [right, say, that, housing, association, shoul...
3885    [Boise, State, fan, can, tell, you, two, thing...
3886    [advocating, for, conduct, within, bound, Huma...
3887                          [when, you, coming, ohio]
3888    [Liars, like, the, Antifa, twin, you, vigorous...
3889    [Billy, you, have, short, memory, Obama, tried...
3890          [She, not, the, brightest, light, the, tree]
3891    [Sometimes, get, strong, vibe, from, people, a...
3892    [Benidorm, Creamfields, Maga, Not, too, shabby...
3893    [Spanishrevenge, justice, HumanRights, and, Fr...
Name: text, Length: 3894, dtype: object
```

In [16]:
```
wordvec_arrays_test = np.zeros((len(tokenized_tweet_test), 300))

for i in range(len(tokenized_tweet_test)):
    wordvec_arrays_test[i,:] = word_vector_pretrained(tokenized_tweet_test[i], 300)

wordvec_df_test = pd.DataFrame(wordvec_arrays_test)
wordvec_df_test.shape
```

Out[16]:
```
(3894, 300)
```

In [17]:
```
X_test1 = np.array(wordvec_df_test)

prediction1 = lreg.predict(X_test1) # predicting on the validation set

prediction_int1 = prediction1.astype(np.int)
```

In [18]:
```
prediction_int1
```

Out[18]:
```
array([0, 1, 0, ..., 0, 0, 0])
```

In [19]:
```
data_test['Target'] = prediction_int1
submission = data_test[['id','Target']]
submission.to_csv('pretrained_word2vec.csv', index=False) # writing data to a CSV file
submission
```

Out[19]:

	id	Target
0	90194	0
1	77444	1
2	13384	0
3	54920	1
4	56117	1
5	67757	0
6	12681	0
7	12609	0
8	70380	0
9	12108	0
10	14726	1
11	74477	0
12	49845	0
13	47311	0
14	75689	0
15	84102	0
16	10607	0
17	98992	1
18	53264	1
19	54842	0
20	48995	0
21	72353	0
22	50759	1
23	14574	0
24	93119	0
25	43133	1
26	59537	0
27	32317	0
28	76680	0
29	80561	0
...	...	...
3864	54190	0
3865	28037	0
3866	77430	0
3867	75815	0
3868	87290	0
3869	99475	0
3870	43323	0
3871	37666	0
3872	77905	0
3873	83400	0
3874	84081	0
3875	71649	0
3876	20841	0
3877	90959	0
3878	32598	0
3879	43964	0
3880	97745	0
3881	86716	0
3882	21033	1
3883	66832	0
3884	28996	0
3885	64713	0
3886	63482	0
3887	11132	0
3888	87416	0
3889	90041	0
3890	98824	0
3891	95338	0
3892	67210	0
3893	46552	0

3894 rows × 2 columns

In [ ]:
```
# Pretrained word2vec embeddings 2
```

In [20]:
```
# Using pre-trained word2vec embeddings created by Google

import gensim.downloader as api

model = api.load("glove-twitter-200")

#model = gensim.models.KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=T
rue)

#model.train(tokenized_tweet, total_examples= len(combi['description']), epochs=50)

def word_vector_pretrained(tokens, size):
    vec = np.zeros(size).reshape((1, size))
    count = 0.
    for word in tokens:
        try:
            vec += model[word].reshape((1, size))
            count += 1.
        except KeyError: # handling the case where the token is not in vocabulary

            continue
    if count != 0:
        vec /= count
    return vec

wordvec_arrays = np.zeros((len(tokenized_tweet), 200))

for i in range(len(tokenized_tweet)):
    wordvec_arrays[i,:] = word_vector_pretrained(tokenized_tweet[i], 200)

wordvec_df = pd.DataFrame(wordvec_arrays)
wordvec_df.shape

X_train, X_test , y_train, y_test = train_test_split(wordvec_df, labels, test_size=0.2, random_state=1)
```

In [21]:
```
lreg = LogisticRegression(random_state = 10)
lreg.fit(X_train, y_train) # training the model

prediction = lreg.predict(X_test) # predicting on the validation set

prediction_int = prediction.astype(np.int)

f1_score(y_test, prediction,average = 'macro') # calculating f1 score
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[21]:
```
0.6552467092841944
```

In [23]:
```
data_test =  pd.read_csv('transferlearning-dl-spring2020/test.csv')

data_test['text'] = np.vectorize(remove_pattern)(data_test['text'], "@[\w]*")


data_test['text'] = data_test['text'].str.replace("[^a-zA-Z#]", " ")

data_test['text'] = data_test['text'].str.replace('#','')


data_test['text'] = data_test['text'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>2]))



tokenized_tweet_test = data_test['text'].apply(lambda x: x.split()) # tokenizing

tokenized_tweet_test.head()

len(tokenized_tweet_test)

#from nltk.stem.porter import *
#stemmer = PorterStemmer()

#tokenized_tweet_test = tokenized_tweet_test.apply(lambda x: [stemmer.stem(i) for i in x]) # stemming

from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
tokenized_tweet_test.apply(lambda x: [lemmatizer.lemmatize(i) for i in x]) # stemming




wordvec_arrays_test = np.zeros((len(tokenized_tweet_test), 200))

for i in range(len(tokenized_tweet_test)):
    wordvec_arrays_test[i,:] = word_vector_pretrained(tokenized_tweet_test[i], 200)

wordvec_df_test = pd.DataFrame(wordvec_arrays_test)
wordvec_df_test.shape



X_test1 = np.array(wordvec_df_test)

prediction1 = lreg.predict(X_test1) # predicting on the validation set

prediction_int1 = prediction1.astype(np.int)

prediction_int1

data_test['Target'] = prediction_int1
submission = data_test[['id','Target']]
submission.to_csv('pretrained_word2vec1.csv', index=False) # writing data to a CSV file
submission
```

Out[23]:

	id	Target
0	90194	0
1	77444	1
2	13384	0
3	54920	0
4	56117	1
5	67757	0
6	12681	0
7	12609	0
8	70380	0
9	12108	0
10	14726	1
11	74477	0
12	49845	0
13	47311	0
14	75689	0
15	84102	0
16	10607	0
17	98992	1
18	53264	1
19	54842	0
20	48995	0
21	72353	0
22	50759	0
23	14574	0
24	93119	0
25	43133	1
26	59537	0
27	32317	0
28	76680	0
29	80561	1
...	...	...
3864	54190	0
3865	28037	0
3866	77430	0
3867	75815	0
3868	87290	0
3869	99475	0
3870	43323	0
3871	37666	0
3872	77905	0
3873	83400	0
3874	84081	0
3875	71649	1
3876	20841	0
3877	90959	0
3878	32598	0
3879	43964	0
3880	97745	1
3881	86716	0
3882	21033	1
3883	66832	0
3884	28996	0
3885	64713	0
3886	63482	1
3887	11132	0
3888	87416	0
3889	90041	0
3890	98824	0
3891	95338	1
3892	67210	0
3893	46552	0

3894 rows × 2 columns

In [ ]:
```
# Fine tuned word2vec
```

In [56]:
```
model_w2v = gensim.models.Word2Vec(
            tokenized_tweet,
            size=50, # desired no. of features/independent variables
            window=5, # context window size
            min_count=2,
            sg = 1, # 1 for skip-gram model
            hs = 0,
            negative = 10, # for negative sampling
            workers= 2, # no.of cores
            seed = 34)

model_w2v.train(tokenized_tweet, total_examples= len(tokenized_tweet), epochs=200)

def word_vector(tokens, size):
    vec = np.zeros(size).reshape((1, size))
    count = 0.
    for word in tokens:
        try:
            vec += model_w2v[word].reshape((1, size))
            count += 1.
        except KeyError: # handling the case where the token is not in vocabulary

            continue
    if count != 0:
        vec /= count
    return vec

wordvec_arrays = np.zeros((len(tokenized_tweet), 50))

for i in range(len(tokenized_tweet)):
    wordvec_arrays[i,:] = word_vector(tokenized_tweet[i], 50)

wordvec_df = pd.DataFrame(wordvec_arrays)
wordvec_df.shape
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:19: DeprecationWarning: Call to deprecat
ed `__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__() instead).
```

Out[56]:
```
(9346, 50)
```

In [58]:
```
X_train, X_test , y_train, y_test = train_test_split(wordvec_df, labels, test_size=0.2, random_state=1)

lreg = LogisticRegression(random_state = 20)
lreg.fit(X_train, y_train) # training the model

prediction = lreg.predict(X_test) # predicting on the validation set

prediction_int = prediction.astype(np.int)

f1_score(y_test, prediction,average = 'macro') # calculating f1 score
```

Out[58]:
```
0.5291078614322713
```

In [ ]:
```
# CNN for text classification
```

In [60]:
```
top_data_df_small = data
```

In [61]:
```
from gensim.utils import simple_preprocess
# Tokenize the text column to get the new column 'tokenized_text'
top_data_df_small['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in top_data_df_smal
l['text']]
print(top_data_df_small['tokenized_text'].head(10))
```

```
0    [she, should, ask, few, native, americans, wha...
1    [amazon, investigating, chinese, employees, wh...
2    [someone, should, vetaken, this, piece, shit, ...
3    [obama, wanted, liberals, amp, illegals, move,...
4                         [liberals, are, all, kookoo]
5    [was, literally, just, talking, about, this, l...
6                                [buy, more, icecream]
7             [not, fault, you, support, gun, control]
8    [what, the, difference, between, kavanaugh, an...
9    [you, are, lying, corrupt, traitor, nobody, wa...
Name: tokenized_text, dtype: object
```

In [62]:
```
from gensim.parsing.porter import PorterStemmer
porter_stemmer = PorterStemmer()
# Get the stemmed_tokens
top_data_df_small['stemmed_tokens'] = [[porter_stemmer.stem(word) for word in tokens] for tokens in top
_data_df_small['tokenized_text'] ]
top_data_df_small['stemmed_tokens'].head(10)
```

Out[62]:
```
0    [she, should, ask, few, nativ, american, what,...
1    [amazon, investig, chines, employe, who, ar, s...
2    [someon, should, vetaken, thi, piec, shit, vol...
3    [obama, want, liber, amp, illeg, move, into, r...
4                          [liber, ar, all, kookoo]
5    [wa, liter, just, talk, about, thi, lol, all, ...
6                                [bui, more, icecream]
7             [not, fault, you, support, gun, control]
8    [what, the, differ, between, kavanaugh, and, o...
9    [you, ar, ly, corrupt, traitor, nobodi, want, ...
Name: stemmed_tokens, dtype: object
```

In [63]:
```
from sklearn.model_selection import train_test_split
# Train Test Split Function
def split_train_test(top_data_df_small, test_size=0.3, shuffle_state=True):
    X_train, X_test, Y_train, Y_test = train_test_split(top_data_df_small[[ 'text',  'stemmed_tokens'
]],
                                                        top_data_df_small['target'],
                                                        shuffle=shuffle_state,
                                                        test_size=test_size,
                                                        random_state=15)
    print("Value counts for Train sentiments")
    print(Y_train.value_counts())
    print("Value counts for Test sentiments")
    print(Y_test.value_counts())
    print(type(X_train))
    print(type(Y_train))
    X_train = X_train.reset_index()
    X_test = X_test.reset_index()
    Y_train = Y_train.to_frame()
    Y_train = Y_train.reset_index()
    Y_test = Y_test.to_frame()
    Y_test = Y_test.reset_index()
    print(X_train.head())
    return X_train, X_test, Y_train, Y_test

# Call the train_test_split
X_train, X_test, Y_train, Y_test = split_train_test(top_data_df_small)
```

```
Value counts for Train sentiments
0    4365
1    2177
Name: target, dtype: int64
Value counts for Test sentiments
0    1855
1     949
Name: target, dtype: int64
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
   index                                              text  \
0   8030                             least you are honest lmao
1   3722                            All that shit was hers URL
2   3522                             Cutie connection authentic
3   4232  What interview Evanne honest and right from th...
4   1678  Warriorscoach fundraising for gun control Brad...

                                       stemmed_tokens
0                       [least, you, ar, honest, lmao]
1                      [all, that, shit, wa, her, url]
2                             [cuti, connect, authent]
3  [what, interview, evann, honest, and, right, f...
4  [warriorscoach, fundrais, for, gun, control, b...
```

In [64]:

```python
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import torch
# Use cuda if present
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device available for running: ")
print(device)
```

```
Device available for running:
cpu
```

In [67]:

```python
from gensim.models import Word2Vec
size = 500
window = 3
min_count = 1
workers = 3
sg = 1

# Function to train word2vec model
def make_word2vec_model(top_data_df_small, padding=True, sg=1, min_count=1, size=500, workers=3, window
=3):
    if  padding:
        print(len(top_data_df_small))
        temp_df = pd.Series(top_data_df_small['stemmed_tokens']).values
        temp_df = list(temp_df)
        temp_df.append(['pad'])
        word2vec_file =   'word2vec_' + str(size) + '_PAD.model'
    else:
        temp_df = top_data_df_small['stemmed_tokens']
        word2vec_file =   'word2vec_' + str(size) + '.model'
    w2v_model = Word2Vec(temp_df, min_count = min_count, size = size, workers = workers, window = windo
w, sg = sg)

    w2v_model.save(word2vec_file)
    return w2v_model, word2vec_file

# Train Word2vec model
w2vmodel, word2vec_file = make_word2vec_model(top_data_df_small, padding=True, sg=sg, min_count=min_cou
nt, size=size, workers=workers, window=window)
```

```
9346
```

In [68]:

```python
# Function to get the output tensor
def make_target(label):
    if label == -1:
        return torch.tensor([0], dtype=torch.long, device=device)
    elif label == 0:
        return torch.tensor([1], dtype=torch.long, device=device)
    else:
        return torch.tensor([2], dtype=torch.long, device=device)
```

In [71]:

```python
EMBEDDING_SIZE = 500
NUM_FILTERS = 10
import gensim

class CnnTextClassifier(nn.Module):
    def __init__(self, vocab_size, num_classes, window_sizes=(1,2,3,5)):
        super(CnnTextClassifier, self).__init__()
        w2vmodel = gensim.models.KeyedVectors.load( 'word2vec_500_PAD.model')
        weights = w2vmodel.wv
        # With pretrained embeddings
        self.embedding = nn.Embedding.from_pretrained(torch.FloatTensor(weights.vectors), padding_idx=w
2vmodel.wv.vocab['pad'].index)
        # Without pretrained embeddings
        # self.embedding = nn.Embedding(vocab_size, EMBEDDING_SIZE)

        self.convs = nn.ModuleList([
                                   nn.Conv2d(1, NUM_FILTERS, [window_size, EMBEDDING_SIZE], padding=(wi
ndow_size - 1, 0))
                                   for window_size in window_sizes
        ])

        self.fc = nn.Linear(NUM_FILTERS * len(window_sizes), num_classes)

    def forward(self, x):
        x = self.embedding(x) # [B, T, E]

        # Apply a convolution + max_pool layer for each window size
        x = torch.unsqueeze(x, 1)
        xs = []
        for conv in self.convs:
            x2 = torch.tanh(conv(x))
            x2 = torch.squeeze(x2, -1)
            x2 = F.max_pool1d(x2, x2.size(2))
            xs.append(x2)
        x = torch.cat(xs, 2)

        # FC
        x = x.view(x.size(0), -1)
        logits = self.fc(x)

        probs = F.softmax(logits, dim = 1)

        return probs
```

In [73]:

```python
max_sen_len = top_data_df_small.stemmed_tokens.map(len).max()
padding_idx = w2vmodel.wv.vocab['pad'].index
def make_word2vec_vector_cnn(sentence):
    padded_X = [padding_idx for i in range(max_sen_len)]
    i = 0
    for word in sentence:
        if word not in w2vmodel.wv.vocab:
            padded_X[i] = 0
            print(word)
        else:
            padded_X[i] = w2vmodel.wv.vocab[word].index
        i += 1
    return torch.tensor(padded_X, dtype=torch.long, device=device).view(1, -1)
```

In [74]:

```python
NUM_CLASSES = 3
VOCAB_SIZE = len(w2vmodel.wv.vocab)

cnn_model = CnnTextClassifier(vocab_size=VOCAB_SIZE, num_classes=NUM_CLASSES)
cnn_model.to(device)
loss_function = nn.CrossEntropyLoss()
optimizer = optim.Adam(cnn_model.parameters(), lr=0.001)
num_epochs = 30

# Open the file for writing loss
loss_file_name = 'cnn_class_big_loss_with_padding.csv'
f = open(loss_file_name,'w')
f.write('iter, loss')
f.write('\n')
losses = []
cnn_model.train()
for epoch in range(num_epochs):
    print("Epoch" + str(epoch + 1))
    train_loss = 0
    for index, row in X_train.iterrows():
        # Clearing the accumulated gradients
        cnn_model.zero_grad()

        # Make the bag of words vector for stemmed tokens
        bow_vec = make_word2vec_vector_cnn(row['stemmed_tokens'])

        # Forward pass to get output
        probs = cnn_model(bow_vec)

        # Get the target label
        target = make_target(Y_train['target'][index])

        # Calculate Loss: softmax --> cross entropy loss
        loss = loss_function(probs, target)
        train_loss += loss.item()

        # Getting gradients w.r.t. parameters
        loss.backward()

        # Updating parameters
        optimizer.step()


    # if index == 0:
    #     continue
    print("Epoch ran :"+ str(epoch+1))
    f.write(str((epoch+1)) + "," + str(train_loss / len(X_train)))
    f.write('\n')
    train_loss = 0

torch.save(cnn_model, 'cnn_big_model_500_with_padding.pth')

f.close()
print("Input vector")
print(bow_vec.cpu().numpy())
print("Probs")
print(probs)
print(torch.argmax(probs, dim=1).cpu().numpy()[0])
```

```
Epoch1
Epoch ran :1
Epoch2
Epoch ran :2
Epoch3
Epoch ran :3
Epoch4
Epoch ran :4
Epoch5
Epoch ran :5
Epoch6
Epoch ran :6
Epoch7
Epoch ran :7
Epoch8
Epoch ran :8
Epoch9
Epoch ran :9
Epoch10
Epoch ran :10
Epoch11
Epoch ran :11
Epoch12
Epoch ran :12
Epoch13
Epoch ran :13
Epoch14
Epoch ran :14
Epoch15
Epoch ran :15
Epoch16
Epoch ran :16
Epoch17
Epoch ran :17
Epoch18
Epoch ran :18
Epoch19
Epoch ran :19
Epoch20
Epoch ran :20
Epoch21
Epoch ran :21
Epoch22
Epoch ran :22
Epoch23
Epoch ran :23
Epoch24
Epoch ran :24
Epoch25
Epoch ran :25
Epoch26
Epoch ran :26
Epoch27
Epoch ran :27
Epoch28
Epoch ran :28
Epoch29
Epoch ran :29
Epoch30
Epoch ran :30
Input vector
[[  235    13    16     0   928   172   358   244     7     7 10887 10887
  10887 10887 10887 10887 10887 10887 10887 10887 10887 10887 10887 10887
  10887 10887 10887 10887 10887 10887 10887 10887 10887 10887 10887 10887
  10887 10887 10887 10887 10887 10887 10887 10887 10887 10887 10887 10887
  10887]]
Probs
tensor([[1.1411e-11, 9.5633e-01, 4.3666e-02]], grad_fn=<SoftmaxBackward>)
1
```

In [79]:

```python
from sklearn.metrics import classification_report
bow_cnn_predictions = []
original_lables_cnn_bow = []
cnn_model.eval()
loss_df = pd.read_csv('cnn_class_big_loss_with_padding.csv')
print(loss_df.columns)
# loss_df.plot('loss')
with torch.no_grad():
    for index, row in X_test.iterrows():
        bow_vec = make_word2vec_vector_cnn(row['stemmed_tokens'])
        probs = cnn_model(bow_vec)
        _, predicted = torch.max(probs.data, 1)
        bow_cnn_predictions.append(predicted.cpu().numpy()[0])
        original_lables_cnn_bow.append(make_target(Y_test['target'][index]).cpu().numpy()[0])

print(classification_report(original_lables_cnn_bow,bow_cnn_predictions))
loss_file_name = 'cnn_class_big_loss_with_padding.csv'
loss_df = pd.read_csv(loss_file_name)
print(loss_df.columns)
plt_500_padding_30_epochs = loss_df[' loss'].plot()
fig = plt_500_padding_30_epochs.get_figure()
fig.savefig('loss_plt_500_padding_30_epochs.pdf')
```

```
Index(['iter', ' loss'], dtype='object')
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarnin
g: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Us
e `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
              precision    recall  f1-score   support

           1       0.66      1.00      0.80      1855
           2       0.00      0.00      0.00       949

    accuracy                           0.66      2804
   macro avg       0.33      0.50      0.40      2804
weighted avg       0.44      0.66      0.53      2804

Index(['iter', ' loss'], dtype='object')
```

In [88]:

```python
data_test =  pd.read_csv('transferlearning-dl-spring2020/test.csv')
```

In [89]:

```python
from gensim.utils import simple_preprocess
# Tokenize the text column to get the new column 'tokenized_text'
data_test['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in data_test['text']]
print(data_test['tokenized_text'].head(10))
```

```
0    [user, user, go, home, you, re, drunk, user, m...
1                  [user, user, oh, noes, tough, shit]
2    [user, canada, doesn, need, another, cuck, we,...
3    [user, user, user, it, should, scare, every, a...
4    [user, user, user, user, lol, throwing, the, b...
5                       [user, user, you, are, correct]
6    [user, user, kind, of, like, when, conservativ...
7    [the, only, thing, the, democrats, have, is, l...
8    [user, user, user, user, user, user, user, use...
9    [user, user, user, user, user, user, user, use...
Name: tokenized_text, dtype: object
```

In [90]:

```python
from gensim.parsing.porter import PorterStemmer
porter_stemmer = PorterStemmer()
# Get the stemmed_tokens
data_test['stemmed_tokens'] = [[porter_stemmer.stem(word) for word in tokens] for tokens in data_test[
'tokenized_text'] ]
data_test['stemmed_tokens'].head(10)
```

Out[90]:

```
0    [user, user, go, home, you, re, drunk, user, m...
1                   [user, user, oh, noe, tough, shit]
2    [user, canada, doesn, need, anoth, cuck, we, a...
3    [user, user, user, it, should, scare, everi, a...
4    [user, user, user, user, lol, throw, the, bull...
5                        [user, user, you, ar, correct]
6    [user, user, kind, of, like, when, conserv, wa...
7    [the, onli, thing, the, democrat, have, is, ly...
8    [user, user, user, user, user, user, user, use...
9    [user, user, user, user, user, user, user, use...
Name: stemmed_tokens, dtype: object
```

In [92]:

```python
from sklearn.model_selection import train_test_split
# Train Test Split Function
def split_train_test(top_data_df_small, test_size=1, shuffle_state=True):
    X_train, X_test, Y_train, Y_test = train_test_split(top_data_df_small[[ 'text',  'stemmed_tokens'
]],
                                                        top_data_df_small['target'],
                                                        shuffle=shuffle_state,
                                                        test_size=test_size,
                                                        random_state=15)
    print("Value counts for Train sentiments")
    print(Y_train.value_counts())
    print("Value counts for Test sentiments")
    print(Y_test.value_counts())
    print(type(X_train))
    print(type(Y_train))
    X_train = X_train.reset_index()
    X_test = X_test.reset_index()
    Y_train = Y_train.to_frame()
    Y_train = Y_train.reset_index()
    Y_test = Y_test.to_frame()
    Y_test = Y_test.reset_index()
    print(X_train.head())
    return X_train, X_test, Y_train, Y_test

# Call the train_test_split
X_test = data_test[[ 'text',  'stemmed_tokens']]
```

In [93]:

```python
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import torch
# Use cuda if present
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device available for running: ")
print(device)
```

```
Device available for running:
cpu
```

In [94]:

```python
from gensim.models import Word2Vec
size = 500
window = 3
min_count = 1
workers = 3
sg = 1

# Function to train word2vec model
def make_word2vec_model(top_data_df_small, padding=True, sg=1, min_count=1, size=500, workers=3, window
=3):
    if  padding:
        print(len(top_data_df_small))
        temp_df = pd.Series(top_data_df_small['stemmed_tokens']).values
        temp_df = list(temp_df)
        temp_df.append(['pad'])
        word2vec_file =   'word2vec_' + str(size) + '_PAD.model'
    else:
        temp_df = top_data_df_small['stemmed_tokens']
        word2vec_file =   'word2vec_' + str(size) + '.model'
    w2v_model = Word2Vec(temp_df, min_count = min_count, size = size, workers = workers, window = windo
w, sg = sg)

    w2v_model.save(word2vec_file)
    return w2v_model, word2vec_file

# Train Word2vec model
w2vmodel, word2vec_file = make_word2vec_model(X_test, padding=True, sg=sg, min_count=min_count, size=si
ze, workers=workers, window=window)
```

```
3894
```

In [97]:

```python
max_sen_len = X_test.stemmed_tokens.map(len).max()
padding_idx = w2vmodel.wv.vocab['pad'].index
def make_word2vec_vector_cnn(sentence):
    padded_X = [padding_idx for i in range(max_sen_len)]
    i = 0
    for word in sentence:
        if word not in w2vmodel.wv.vocab:
            padded_X[i] = 0
            print(word)
        else:
            padded_X[i] = w2vmodel.wv.vocab[word].index
        i += 1
    return torch.tensor(padded_X, dtype=torch.long, device=device).view(1, -1)

from sklearn.metrics import classification_report
bow_cnn_predictions = []
original_lables_cnn_bow = []
cnn_model.eval()
#loss_df = pd.read_csv('cnn_class_big_loss_with_padding.csv')
#print(loss_df.columns)
# loss_df.plot('loss')
with torch.no_grad():
    for index, row in X_test.iterrows():
        bow_vec = make_word2vec_vector_cnn(row['stemmed_tokens'])
        probs = cnn_model(bow_vec)
        _, predicted = torch.max(probs.data, 1)
        bow_cnn_predictions.append(predicted.cpu().numpy()[0])
#        original_lables_cnn_bow.append(make_target(Y_test['target'][index]).cpu().numpy()[0])

#print(classification_report(original_lables_cnn_bow,bow_cnn_predictions))
#loss_file_name = 'cnn_class_big_loss_with_padding.csv'
#loss_df = pd.read_csv(loss_file_name)
#print(loss_df.columns)
#plt_500_padding_30_epochs = loss_df[' loss'].plot()
#fig = plt_500_padding_30_epochs.get_figure()
#fig.savefig('loss_plt_500_padding_30_epochs.pdf')
```

```
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
...]
```

In [200]:
```
data_test['Target'] = bow_cnn_predictions
submission = data_test[['id','Target']]
submission.to_csv('cnn.csv', index=False) # writing data to a CSV file
submission
```

Out[200]:

	id	Target
0	90194	1
1	77444	1
2	13384	1
3	54920	1
4	56117	1
5	67757	1
6	12681	1
7	12609	1
8	70380	1
9	12108	1
10	14726	1
11	74477	1
12	49845	1
13	47311	1
14	75689	1
15	84102	1
16	10607	1
17	98992	1
18	53264	1
19	54842	1
20	48995	1
21	72353	1
22	50759	1
23	14574	1
24	93119	1
25	43133	1
26	59537	1
27	32317	1
28	76680	1
29	80561	1
...	...	...
3864	54190	1
3865	28037	1
3866	77430	1
3867	75815	1
3868	87290	1
3869	99475	1
3870	43323	1
3871	37666	1
3872	77905	1
3873	83400	1
3874	84081	1
3875	71649	1
3876	20841	1
3877	90959	1
3878	32598	1
3879	43964	1
3880	97745	1
3881	86716	1
3882	21033	1
3883	66832	1
3884	28996	1
3885	64713	1
3886	63482	1
3887	11132	1
3888	87416	1
3889	90041	1
3890	98824	1
3891	95338	1
3892	67210	1
3893	46552	1

3894 rows × 2 columns

In [ ]:
```
# LSTM for text classification
```

In [38]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding
from keras.optimizers import RMSprop
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping
%matplotlib inline
```

In [167]:
```
df = data
df.head()
```

Out[167]:

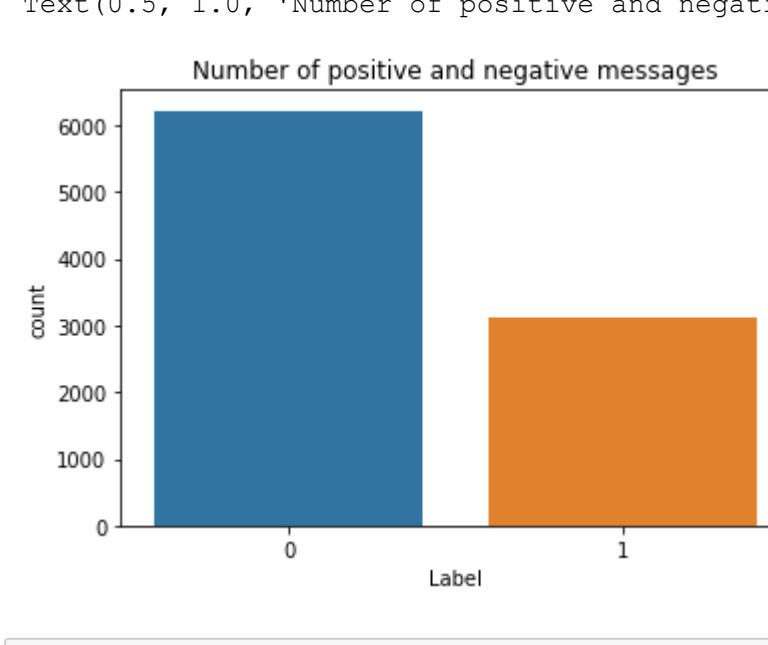
	id	text	target	tokenized_text	stemmed_tokens
0	86426	She should ask few native Americans what their...	1	[she, should, ask, few, native, americans, wha...	[she, should, ask, few, nativ, american, what,...
1	16820	Amazon investigating Chinese employees who are...	0	[amazon, investigating, chinese, employees, wh...	[amazon, investig, chines, employ, who, ar, s...
2	62688	Someone should veTaken this piece shit volcano	1	[someone, should, vetaken, this, piece, shit, ...	[someon, should, vetaken, thi, piec, shit, vol...
3	43605	Obama wanted liberals amp illegals move into r...	0	[obama, wanted, liberals, amp, illegals, move,...	[obama, want, liber, amp, illeg, move, into, r...
4	97670	Liberals are all Kookoo	1	[liberals, are, all, kookoo]	[liber, ar, all, kookoo]

In [168]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9346 entries, 0 to 9345
Data columns (total 5 columns):
id                 9346 non-null int64
text               9346 non-null object
target             9346 non-null int64
tokenized_text     9346 non-null object
stemmed_tokens     9346 non-null object
dtypes: int64(2), object(3)
memory usage: 365.2+ KB
```

In [169]:
```
sns.countplot(df.target)
plt.xlabel('Label')
plt.title('Number of positive and negative messages')
```

Out[169]: Text(0.5, 1.0, 'Number of positive and negative messages')



In [96]:
```
X = data.text
Y = data.target
le = LabelEncoder()
Y = le.fit_transform(Y)
Y = Y.reshape(-1,1)
```

In [97]:
```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.15)
```

In [98]:
```
max_words = 1000
max_len = 150
tok = Tokenizer(num_words=max_words)
tok.fit_on_texts(X_train)
sequences = tok.texts_to_sequences(X_train)
sequences_matrix = sequence.pad_sequences(sequences,maxlen=max_len)
```

In [99]:
```
def RNN():
    inputs = Input(name='inputs',shape=[max_len])
    layer = Embedding(max_words,50,input_length=max_len)(inputs)
    layer = LSTM(64)(layer)
    layer = Dense(256,name='FC1')(layer)
    layer = Activation('relu')(layer)
    layer = Dropout(0.5)(layer)
    layer = Dense(1,name='out_layer')(layer)
    layer = Activation('sigmoid')(layer)
    model = Model(inputs=inputs,outputs=layer)
    return model
```

In [100]:
```
model = RNN()
model.summary()
model.compile(loss='binary_crossentropy',optimizer=RMSprop(),metrics=['accuracy'])
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inputs (InputLayer)          (None, 150)               0
_________________________________________________________________
embedding_13 (Embedding)     (None, 150, 50)           50000
_________________________________________________________________
lstm_20 (LSTM)               (None, 64)                29440
_________________________________________________________________
FC1 (Dense)                  (None, 256)               16640
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
out_layer (Dense)            (None, 1)                 257
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0
=================================================================
Total params: 96,337
Trainable params: 96,337
Non-trainable params: 0
_________________________________________________________________
```

In [101]:
```
model.fit(sequences_matrix,Y_train,batch_size=128,epochs=10,
          validation_split=0.2)
```

```
Train on 6355 samples, validate on 1589 samples
Epoch 1/10
2688/6355 [===========>..................] - ETA: 30s - loss: 0.6564 - accuracy: 0.6533
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-101-74450c35e6d8> in <module>
      1 model.fit(sequences_matrix,Y_train,batch_size=128,epochs=10,
----> 2           validation_split=0.2)

/anaconda3/lib/python3.7/site-packages/keras/engine/training.py in fit(self, x, y, batch_size, epoch
s, verbose, callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight, initi
al_epoch, steps_per_epoch, validation_steps, validation_freq, max_queue_size, workers, use_multiproce
ssing, **kwargs)
   1237                                         steps_per_epoch=steps_per_epoch,
   1238                                         validation_steps=validation_steps,
-> 1239                                         validation_freq=validation_freq)
   1240
   1241     def evaluate(self,

/anaconda3/lib/python3.7/site-packages/keras/engine/training_arrays.py in fit_loop(model, fit_functio
n, fit_inputs, out_labels, batch_size, epochs, verbose, callbacks, val_function, val_inputs, shuffle,
 initial_epoch, steps_per_epoch, validation_steps, validation_freq)
    194                     ins_batch[i] = ins_batch[i].toarray()
    195
--> 196                 outs = fit_function(ins_batch)
    197                 outs = to_list(outs)
    198                 for l, o in zip(out_labels, outs):

/anaconda3/lib/python3.7/site-packages/tensorflow/python/keras/backend.py in __call__(self, inputs)
   3074
   3075     fetched = self._callable_fn(*array_vals,
-> 3076                                 run_metadata=self.run_metadata)
   3077     self._call_fetch_callbacks(fetched[-len(self._fetches):])
   3078     return nest.pack_sequence_as(self._outputs_structure,

/anaconda3/lib/python3.7/site-packages/tensorflow/python/client/session.py in __call__(self, *args, *
*kwargs)
   1437           ret = tf_session.TF_SessionRunCallable(
   1438               self._session._session, self._handle, args, status,
-> 1439               run_metadata_ptr)
   1440         if run_metadata:
   1441           proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)

KeyboardInterrupt:
```

In [175]:
```
test_sequences = tok.texts_to_sequences(X_test)
test_sequences_matrix = sequence.pad_sequences(test_sequences,maxlen=max_len)
```

In [177]:
```
accr = model.evaluate(test_sequences_matrix,Y_test)
```

```
1402/1402 [==============================] - 5s 4ms/step
```

In [180]:
```
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))
```

```
Test set
  Loss: 0.693
  Accuracy: 0.530
```

In [181]:
```
df = data_test
df.head()
```

Out[181]:

	id	text	tokenized_text	stemmed_tokens	Target
0	90194	@USER @USER Go home you're drunk!!! @USER #MAG...	[user, user, go, home, you, re, drunk, user, m...	[user, user, go, home, you, re, drunk, user, m...	0
1	77444	@USER @USER Oh noes! Tough shit.	[user, user, oh, noes, tough, shit]	[user, user, oh, noe, tough, shit]	0
2	13384	@USER Canada doesn't need another CUCK! We alr...	[user, canada, doesn, need, another, cuck, we,...	[user, canada, doesn, need, anoth, cuck, we, a...	0
3	54920	@USER @USER @USER It should scare every Americ...	[user, user, user, it, should, scare, every, a...	[user, user, user, it, should, scare, everi, a...	0
4	56117	@USER @USER @USER @USER LOL!!! Throwing the ...	[user, user, user, user, lol, throwing, the, b...	[user, user, user, user, lol, throw, the, bull...	0

In [182]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3894 entries, 0 to 3893
Data columns (total 5 columns):
id                 3894 non-null int64
text               3894 non-null object
tokenized_text     3894 non-null object
stemmed_tokens     3894 non-null object
Target             3894 non-null int64
dtypes: int64(2), object(3)
memory usage: 152.2+ KB
```

In [183]:
```
X = df.text
X_test = X
```

In [184]:
```
test_sequences = tok.texts_to_sequences(X_test)
test_sequences_matrix = sequence.pad_sequences(test_sequences,maxlen=max_len)
```

In [185]:
```
test_sequences_matrix.shape
```

Out[185]: (3894, 150)

In [186]:
```
Y_test
```

Out[186]:
```
array([[0],
       [0],
       [0],
       ...,
       [1],
       [0],
       [1]])
```

In [187]:
```
pred = model.predict(test_sequences_matrix)
```

In [188]:
```
predicted = []
for i in range(len(pred)):
    if pred[i] >= 0.5:
        predicted += [1]
    else:
        predicted += [0]
```

In [ ]:
```
predicted
```

In [ ]:
```
data_test['Target'] = predicted
submission = data_test[['id','Target']]
submission.to_csv('lstm.csv', index=False) # writing data to a CSV file
submission
```

In [201]:
```
pred = []
for i in range(3894):
    pred += [0]
```

In [ ]:
```
data_test['Target'] = pred
submission = data_test[['id','Target']]
submission.to_csv('random.csv', index=False) # writing data to a CSV file
submission
```

In [ ]:
```
# Bi-directional RNN for text classification
```

In [29]:
```
wordvec_df.shape
```

Out[29]: (9346, 200)

In [43]:
```
import torch

#Reproducing same results
SEED = 2019

#Torch
torch.manual_seed(SEED)

#Cuda algorithms
torch.backends.cudnn.deterministic = True
```

In [46]:
```
#loading custom dataset
training_data=data

#print preprocessed text
print(vars(training_data[0:1]))
```

```
{'_is_copy': <weakref at 0x1a30693f98; to 'DataFrame' at 0x1a30555a20>, '_data': BlockManager
Items: Index(['id', 'text', 'target'], dtype='object')
Axis 1: RangeIndex(start=0, stop=1, step=1)
IntBlock: slice(0, 4, 2), 2 x 1, dtype: int64
ObjectBlock: slice(1, 2, 1), 1 x 1, dtype: object, '_item_cache': {}}
```

In [49]:
```
import random
train_data,X_test, valid_data,Y_test = train_test_split(training_data['text'],training_data['target'],t
est_size=0.2)
```

In [50]:
```
# LSTM for text classification
```

In [118]:
```
# LSTM for sequence classification in the IMDB dataset
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000

X = data.text
Y = data.target
le = LabelEncoder()
Y = le.fit_transform(Y)
Y = Y.reshape(-1,1)

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.2)

max_words = 1000
max_len = 200
tok = Tokenizer(num_words=max_words)
tok.fit_on_texts(X_train)
sequences = tok.texts_to_sequences(X_train)
X_train = sequence.pad_sequences(sequences,maxlen=max_len)
print(X_train.shape)
print(y_train.shape)
#X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
#X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)
# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_len))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
model.fit(X_train, Y_train, epochs=10, batch_size=64)
# Final evaluation of the model
scores = model.evaluate(X_test, Y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
(7476, 200)
(25000,)
Model: "sequential_31"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_22 (Embedding)     (None, 200, 32)           160000
_________________________________________________________________
lstm_29 (LSTM)               (None, 100)               53200
_________________________________________________________________
dense_25 (Dense)             (None, 1)                 101
=================================================================
Total params: 213,301
Trainable params: 213,301
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
7476/7476 [==============================] - 120s 16ms/step - loss: 0.6291 - accuracy: 0.6651
Epoch 2/10
7476/7476 [==============================] - 112s 15ms/step - loss: 0.5440 - accuracy: 0.7289
Epoch 3/10
7476/7476 [==============================] - 113s 15ms/step - loss: 0.4807 - accuracy: 0.7746
Epoch 4/10
7476/7476 [==============================] - 125s 17ms/step - loss: 0.4601 - accuracy: 0.7925
Epoch 5/10
7476/7476 [==============================] - 111s 15ms/step - loss: 0.4495 - accuracy: 0.7968
Epoch 6/10
7476/7476 [==============================] - 105s 14ms/step - loss: 0.4337 - accuracy: 0.8091
Epoch 7/10
7476/7476 [==============================] - 106s 14ms/step - loss: 0.4179 - accuracy: 0.8146
Epoch 8/10
7476/7476 [==============================] - 105s 14ms/step - loss: 0.4056 - accuracy: 0.8234
Epoch 9/10
7476/7476 [==============================] - 106s 14ms/step - loss: 0.3928 - accuracy: 0.8261
Epoch 10/10
7476/7476 [==============================] - 112s 15ms/step - loss: 0.3918 - accuracy: 0.8277
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-118-32a60363e4a4> in <module>
     46 model.fit(X_train, Y_train, epochs=10, batch_size=64)
     47 # Final evaluation of the model
---> 48 scores = model.evaluate(X_test, Y_test, verbose=0)
     49 print("Accuracy: %.2f%%" % (scores[1]*100))

/anaconda3/lib/python3.7/site-packages/keras/engine/training.py in evaluate(self, x, y, batch_size, v
erbose, sample_weight, steps, callbacks, max_queue_size, workers, use_multiprocessing)
   1347             x, y,
   1348             sample_weight=sample_weight,
-> 1349             batch_size=batch_size)
   1350         # Prepare inputs, delegate logic to `test_loop`.
   1351         if self._uses_dynamic_learning_phase():

/anaconda3/lib/python3.7/site-packages/keras/engine/training.py in _standardize_user_data(self, x, y,
 sample_weight, class_weight, check_array_lengths, batch_size)
    577             feed_input_shapes,
    578             check_batch_axis=False,  # Don't enforce the batch size.
--> 579             exception_prefix='input')
    580
    581         if y is not None:

/anaconda3/lib/python3.7/site-packages/keras/engine/training_utils.py in standardize_input_data(data,
 names, shapes, check_batch_axis, exception_prefix)
    143                             ': expected ' + names[i] + ' to have shape ' +
    144                             str(shape) + ' but got array with shape ' +
--> 145                             str(data_shape))
    146     return data
    147

ValueError: Error when checking input: expected embedding_22_input to have shape (200,) but got array
 with shape (1,)
```

In [120]:
```
#tok.fit_on_texts(X_)
sequences = tok.texts_to_sequences(X_test)
X_test = sequence.pad_sequences(sequences,maxlen=max_len)


scores = model.evaluate(X_test, Y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 71.98%
```

In [ ]:
```
test_sequences = tok.texts_to_sequences(data_test['text'])
test_sequences_matrix = sequence.pad_sequences(test_sequences,maxlen=max_len)

test_sequences_matrix.shape

pred = model.predict(test_sequences_matrix)

predicted = []
for i in range(len(pred)):
    if pred[i] >= 0.5:
        predicted += [1]
    else:
        predicted += [0]

data_test['Target'] = predicted
submission = data_test[['id','Target']]
submission.to_csv('lstm2.csv', index=False) # writing data to a CSV file
submission
```

In [89]:
```
# LSTM For text Classification With Dropout
```

In [127]:
```python
# LSTM with Dropout for sequence classification in the IMDB dataset
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000

X = data.text
Y = data.target
le = LabelEncoder()
Y = le.fit_transform(Y)
Y = Y.reshape(-1,1)

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.2)

max_words = 1000
max_len = 200
tok = Tokenizer(num_words=max_words)
tok.fit_on_texts(X_train)
sequences = tok.texts_to_sequences(X_train)
X_train = sequence.pad_sequences(sequences,maxlen=max_len)
print(X_train.shape)
print(y_train.shape)


# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_len))
model.add(Dropout(0.2))
model.add(LSTM(100))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
model.fit(X_train, y_train, epochs=20, batch_size=64)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
(7476, 200)
(7476,)
Model: "sequential_34"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_25 (Embedding)     (None, 200, 32)           160000
_________________________________________________________________
dropout_6 (Dropout)          (None, 200, 32)           0
_________________________________________________________________
lstm_32 (LSTM)               (None, 100)               53200
_________________________________________________________________
dropout_7 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_28 (Dense)             (None, 1)                 101
=================================================================
Total params: 213,301
Trainable params: 213,301
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/20
7476/7476 [==============================] - 113s 15ms/step - loss: 0.6294 - accuracy: 0.6664
Epoch 2/20
7476/7476 [==============================] - 109s 15ms/step - loss: 0.5412 - accuracy: 0.7330
Epoch 3/20
7476/7476 [==============================] - 108s 15ms/step - loss: 0.4781 - accuracy: 0.7824
Epoch 4/20
7476/7476 [==============================] - 110s 15ms/step - loss: 0.4582 - accuracy: 0.7956
Epoch 5/20
7476/7476 [==============================] - 114s 15ms/step - loss: 0.4473 - accuracy: 0.8028
Epoch 6/20
7476/7476 [==============================] - 108s 14ms/step - loss: 0.4334 - accuracy: 0.8089
Epoch 7/20
7476/7476 [==============================] - 109s 15ms/step - loss: 0.4216 - accuracy: 0.8131
Epoch 8/20
7476/7476 [==============================] - 122s 16ms/step - loss: 0.4085 - accuracy: 0.8178
Epoch 9/20
7476/7476 [==============================] - 118s 16ms/step - loss: 0.4059 - accuracy: 0.8202
Epoch 10/20
7476/7476 [==============================] - 127s 17ms/step - loss: 0.3894 - accuracy: 0.8300
Epoch 11/20
7476/7476 [==============================] - 114s 15ms/step - loss: 0.3802 - accuracy: 0.8287
Epoch 12/20
7476/7476 [==============================] - 129s 17ms/step - loss: 0.3721 - accuracy: 0.8356
Epoch 13/20
7476/7476 [==============================] - 122s 16ms/step - loss: 0.3612 - accuracy: 0.8426
Epoch 14/20
7476/7476 [==============================] - 123s 16ms/step - loss: 0.3537 - accuracy: 0.8407
Epoch 15/20
7476/7476 [==============================] - 117s 16ms/step - loss: 0.3493 - accuracy: 0.8487
Epoch 16/20
7476/7476 [==============================] - 121s 16ms/step - loss: 0.3281 - accuracy: 0.8559
Epoch 17/20
7476/7476 [==============================] - 116s 15ms/step - loss: 0.3257 - accuracy: 0.8598
Epoch 18/20
7476/7476 [==============================] - 124s 17ms/step - loss: 0.3156 - accuracy: 0.8666
Epoch 19/20
7476/7476 [==============================] - 123s 16ms/step - loss: 0.3006 - accuracy: 0.8713
Epoch 20/20
7476/7476 [==============================] - 113s 15ms/step - loss: 0.3016 - accuracy: 0.8732

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-127-b855e3d580bc> in <module>
     43 model.fit(X_train, y_train, epochs=20, batch_size=64)
     44 # Final evaluation of the model
---> 45 scores = model.evaluate(X_test, y_test, verbose=0)
     46 print("Accuracy: %.2f%%" % (scores[1]*100))

/anaconda3/lib/python3.7/site-packages/keras/engine/training.py in evaluate(self, x, y, batch_size, v
erbose, sample_weight, steps, callbacks, max_queue_size, workers, use_multiprocessing)
   1347             x, y,
   1348             sample_weight=sample_weight,
-> 1349             batch_size=batch_size)
   1350         # Prepare inputs, delegate logic to `test_loop`.
   1351         if self._uses_dynamic_learning_phase():

/anaconda3/lib/python3.7/site-packages/keras/engine/training.py in _standardize_user_data(self, x, y,
 sample_weight, class_weight, check_array_lengths, batch_size)
    577             feed_input_shapes,
    578             check_batch_axis=False,  # Don't enforce the batch size.
--> 579             exception_prefix='input')
    580
    581         if y is not None:

/anaconda3/lib/python3.7/site-packages/keras/engine/training_utils.py in standardize_input_data(data,
 names, shapes, check_batch_axis, exception_prefix)
    143                             ': expected ' + names[i] + ' to have shape ' +
    144                             str(shape) + ' but got array with shape ' +
--> 145                             str(data_shape))
    146     return data
    147

ValueError: Error when checking input: expected embedding_25_input to have shape (200,) but got array
 with shape (1,)
```

In [128]:
```python
#tok.fit_on_texts(X_)
sequences = tok.texts_to_sequences(X_test)
X_test = sequence.pad_sequences(sequences,maxlen=max_len)


scores = model.evaluate(X_test, Y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))

test_sequences = tok.texts_to_sequences(data_test['text'])
test_sequences_matrix = sequence.pad_sequences(test_sequences,maxlen=max_len)

test_sequences_matrix.shape

pred = model.predict(test_sequences_matrix)

predicted = []
for i in range(len(pred)):
    if pred[i] >= 0.5:
        predicted += [1]
    else:
        predicted += [0]

data_test['Target'] = predicted
submission = data_test[['id','Target']]
submission.to_csv('lstm4.csv', index=False) # writing data to a CSV file
submission
```

```
Accuracy: 71.39%
```

Out[128]:

	id	Target
0	90194	0
1	77444	1
2	13384	0
3	54920	0
4	56117	1
5	67757	0
6	12681	0
7	12609	1
8	70380	1
9	12108	1
10	14726	0
11	74477	0
12	49845	1
13	47311	0
14	75689	0
15	84102	0
16	10607	0
17	98992	1
18	53264	1
19	54842	0
20	48995	0
21	72353	1
22	50759	0
23	14574	0
24	93119	0
25	43133	1
26	59537	0
27	32317	0
28	76680	0
29	80561	1
...	...	...
3864	54190	0
3865	28037	0
3866	77430	0
3867	75815	1
3868	87290	0
3869	99475	0
3870	43323	0
3871	37666	0
3872	77905	0
3873	83400	0
3874	84081	0
3875	71649	0
3876	20841	0
3877	90959	0
3878	32598	0
3879	43964	0
3880	97745	0
3881	86716	1
3882	21033	1
3883	66832	0
3884	28996	0
3885	64713	0
3886	63482	1
3887	11132	0
3888	87416	0
3889	90041	0
3890	98824	0
3891	95338	1
3892	67210	0
3893	46552	0

3894 rows × 2 columns

In [ ]:
```python
# LSTM and Convolutional Neural Network For Sequence Classification
```

In [130]:
```python
# LSTM and CNN for sequence classification in the IMDB dataset
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000

X = data.text
Y = data.target
le = LabelEncoder()
Y = le.fit_transform(Y)
Y = Y.reshape(-1,1)

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.2)

max_words = 1000
max_len = 200
tok = Tokenizer(num_words=max_words)
tok.fit_on_texts(X_train)
sequences = tok.texts_to_sequences(X_train)
X_train = sequence.pad_sequences(sequences,maxlen=max_len)
print(X_train.shape)
print(y_train.shape)

# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_len))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
model.fit(X_train, y_train, epochs=10, batch_size=64)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
(7476, 200)
(7476,)
Model: "sequential_36"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_27 (Embedding)     (None, 200, 32)           160000
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 200, 32)           3104
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 100, 32)           0
_________________________________________________________________
lstm_34 (LSTM)               (None, 100)               53200
_________________________________________________________________
dense_30 (Dense)             (None, 1)                 101
=================================================================
Total params: 216,405
Trainable params: 216,405
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
7476/7476 [==============================] - 76s 10ms/step - loss: 0.6315 - accuracy: 0.6663
Epoch 2/10
7476/7476 [==============================] - 65s 9ms/step - loss: 0.5387 - accuracy: 0.7334
Epoch 3/10
7476/7476 [==============================] - 66s 9ms/step - loss: 0.4650 - accuracy: 0.7875
Epoch 4/10
7476/7476 [==============================] - 59s 8ms/step - loss: 0.4393 - accuracy: 0.8039
Epoch 5/10
7476/7476 [==============================] - 61s 8ms/step - loss: 0.4195 - accuracy: 0.8121
Epoch 6/10
7476/7476 [==============================] - 61s 8ms/step - loss: 0.3923 - accuracy: 0.8222
Epoch 7/10
7476/7476 [==============================] - 58s 8ms/step - loss: 0.3610 - accuracy: 0.8404
Epoch 8/10
7476/7476 [==============================] - 59s 8ms/step - loss: 0.3101 - accuracy: 0.8670
Epoch 9/10
7476/7476 [==============================] - 59s 8ms/step - loss: 0.2623 - accuracy: 0.8953
Epoch 10/10
7476/7476 [==============================] - 68s 9ms/step - loss: 0.2010 - accuracy: 0.9240

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-130-2c043b74a2dd> in <module>
     43 model.fit(X_train, y_train, epochs=10, batch_size=64)
     44 # Final evaluation of the model
---> 45 scores = model.evaluate(X_test, y_test, verbose=0)
     46 print("Accuracy: %.2f%%" % (scores[1]*100))

/anaconda3/lib/python3.7/site-packages/keras/engine/training.py in evaluate(self, x, y, batch_size, v
erbose, sample_weight, steps, callbacks, max_queue_size, workers, use_multiprocessing)
   1347             x, y,
   1348             sample_weight=sample_weight,
-> 1349             batch_size=batch_size)
   1350         # Prepare inputs, delegate logic to `test_loop`.
   1351         if self._uses_dynamic_learning_phase():

/anaconda3/lib/python3.7/site-packages/keras/engine/training.py in _standardize_user_data(self, x, y,
 sample_weight, class_weight, check_array_lengths, batch_size)
    577             feed_input_shapes,
    578             check_batch_axis=False,  # Don't enforce the batch size.
--> 579             exception_prefix='input')
    580
    581         if y is not None:

/anaconda3/lib/python3.7/site-packages/keras/engine/training_utils.py in standardize_input_data(data,
 names, shapes, check_batch_axis, exception_prefix)
    143                             ': expected ' + names[i] + ' to have shape ' +
    144                             str(shape) + ' but got array with shape ' +
--> 145                             str(data_shape))
    146     return data
    147

ValueError: Error when checking input: expected embedding_27_input to have shape (200,) but got array
 with shape (1,)
```

In [131]:
```python
#tok.fit_on_texts(X_)
sequences = tok.texts_to_sequences(X_test)
X_test = sequence.pad_sequences(sequences,maxlen=max_len)


scores = model.evaluate(X_test, Y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))

test_sequences = tok.texts_to_sequences(data_test['text'])
test_sequences_matrix = sequence.pad_sequences(test_sequences,maxlen=max_len)

test_sequences_matrix.shape

pred = model.predict(test_sequences_matrix)

predicted = []
for i in range(len(pred)):
    if pred[i] >= 0.5:
        predicted += [1]
    else:
        predicted += [0]

data_test['Target'] = predicted
submission = data_test[['id','Target']]
submission.to_csv('lstm5.csv', index=False) # writing data to a CSV file
submission
```

```
Accuracy: 67.70%
```

Out[131]:

	id	Target
0	90194	1
1	77444	1
2	13384	0
3	54920	1
4	56117	1
5	67757	0
6	12681	0
7	12609	0
8	70380	0
9	12108	1
10	14726	1
11	74477	0
12	49845	1
13	47311	0
14	75689	0
15	84102	0
16	10607	0
17	98992	1
18	53264	1
19	54842	0
20	48995	1
21	72353	1
22	50759	0
23	14574	0
24	93119	0
25	43133	1
26	59537	0
27	32317	0
28	76680	1
29	80561	1
...	...	...
3864	54190	0
3865	28037	0
3866	77430	0
3867	75815	1
3868	87290	0
3869	99475	0
3870	43323	1
3871	37666	0
3872	77905	1
3873	83400	0
3874	84081	1
3875	71649	0
3876	20841	0
3877	90959	0
3878	32598	0
3879	43964	0
3880	97745	1
3881	86716	1
3882	21033	1
3883	66832	0
3884	28996	0
3885	64713	1
3886	63482	1
3887	11132	0
3888	87416	0
3889	90041	1
3890	98824	0
3891	95338	1
3892	67210	0
3893	46552	0

3894 rows × 2 columns

In [ ]:
```python
from random import random
from numpy import array
from numpy import cumsum
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import TimeDistributed
from keras.layers import Bidirectional


X = data.text
Y = data.target
le = LabelEncoder()
Y = le.fit_transform(Y)
Y = Y.reshape(-1,1)

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.2)

max_words = 1000
max_len = 200
tok = Tokenizer(num_words=max_words)
tok.fit_on_texts(X_train)
sequences = tok.texts_to_sequences(X_train)
X_train = sequence.pad_sequences(sequences,maxlen=max_len)
print(X_train.shape)
print(y_train.shape)



# define LSTM
model = Sequential()
model.add(Bidirectional(LSTM(20, return_sequences=True), input_shape=(max_len, 1)))
model.add(TimeDistributed(Dense(1, activation='sigmoid')))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# train LSTM
model.fit(X, y, epochs=1, batch_size=1, verbose=2)
# evaluate LSTM
X,y = get_sequence(n_timesteps)
yhat = model.predict_classes(X, verbose=0)
for i in range(n_timesteps):
        print('Expected:', y[0, i], 'Predicted', yhat[0, i])
```

In [ ]:
```python
#tok.fit_on_texts(X_)
sequences = tok.texts_to_sequences(X_test)
X_test = sequence.pad_sequences(sequences,maxlen=max_len)


scores = model.evaluate(X_test, Y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))

test_sequences = tok.texts_to_sequences(data_test['text'])
test_sequences_matrix = sequence.pad_sequences(test_sequences,maxlen=max_len)

test_sequences_matrix.shape

pred = model.predict(test_sequences_matrix)

predicted = []
for i in range(len(pred)):
    if pred[i] >= 0.5:
        predicted += [1]
    else:
        predicted += [0]

data_test['Target'] = predicted
submission = data_test[['id','Target']]
submission.to_csv('lstm2.csv', index=False) # writing data to a CSV file
submission
```