Disaster Tweets Prediction

Import Libraries

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from nltk.corpus import stopwords
from nltk.util import ngrams
from sklearn.feature_extraction.text import CountVectorizer
from collections import defaultdict
from collections import  Counter
plt.style.use('ggplot')
stop=set(stopwords.words('english'))
import re
from nltk.tokenize import word_tokenize
import gensim
import string
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from tqdm import tqdm
from keras.models import Sequential
from keras.layers import Embedding,LSTM,Dense,SpatialDropout1D
from keras.initializers import Constant
from sklearn.model_selection import train_test_split
from keras.optimizers import Adam
```

```
Using TensorFlow backend.
```

In [2]:
```
import os
#os.listdir('../input/glove-global-vectors-for-word-representation/glove.6B.100d.txt')
```

Load the data

In [3]:
```
tweet= pd.read_csv('../input/nlp-getting-started/train.csv')
test=pd.read_csv('../input/nlp-getting-started/test.csv')
tweet.head(3)
```

Out[3]:

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1

In [4]:
```
print('There are {} rows and {} columns in train'.format(tweet.shape[0],tweet.shape[1]))
print('There are {} rows and {} columns in train'.format(test.shape[0],test.shape[1]))
```

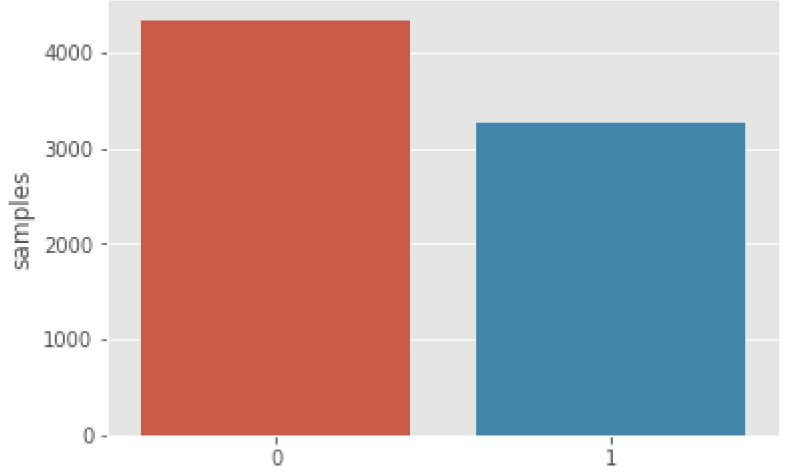
```
There are 7613 rows and 5 columns in train
There are 3263 rows and 4 columns in train
```

Class distribution

Before we begin with anything else,let's check the class distribution.There are only two classes 0 and 1.

In [5]:
```
x=tweet.target.value_counts()
sns.barplot(x.index,x)
plt.gca().set_ylabel('samples')
```

Out[5]:
```
Text(0, 0.5, 'samples')
```



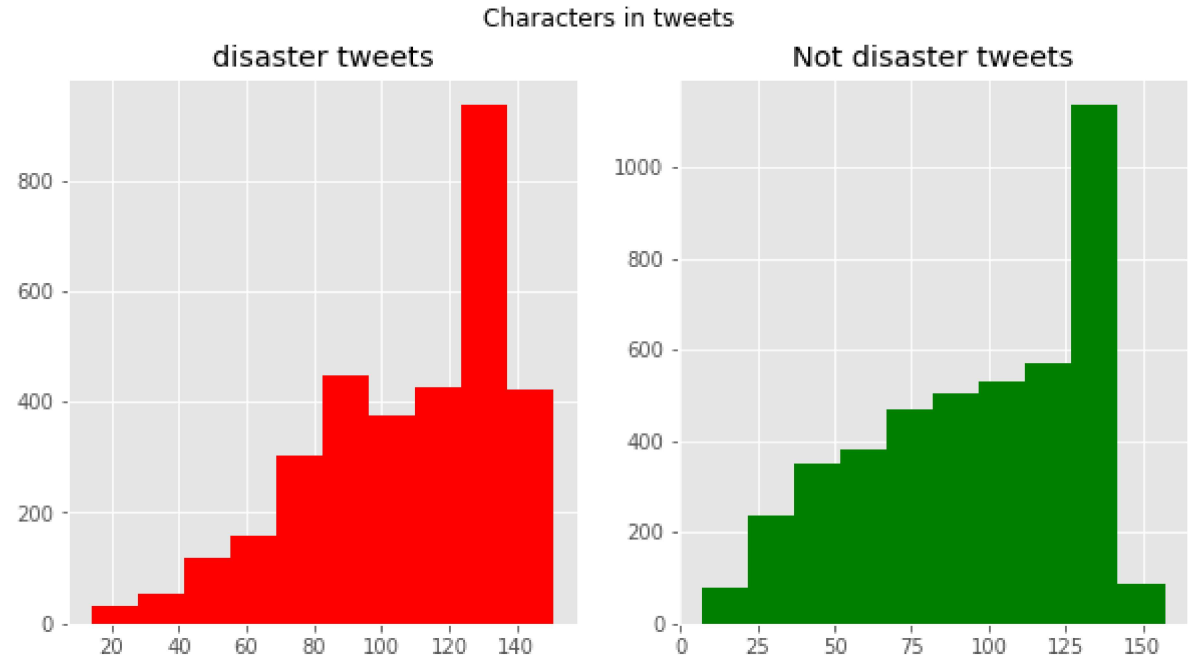
There are more tweets with class 0 (No disaster) than class 1 (disaster tweets)

Exploratory Data Analysis of tweets

First,we will do very basic analysis,that is character level,word level and sentence level analysis.

Number of characters in tweets

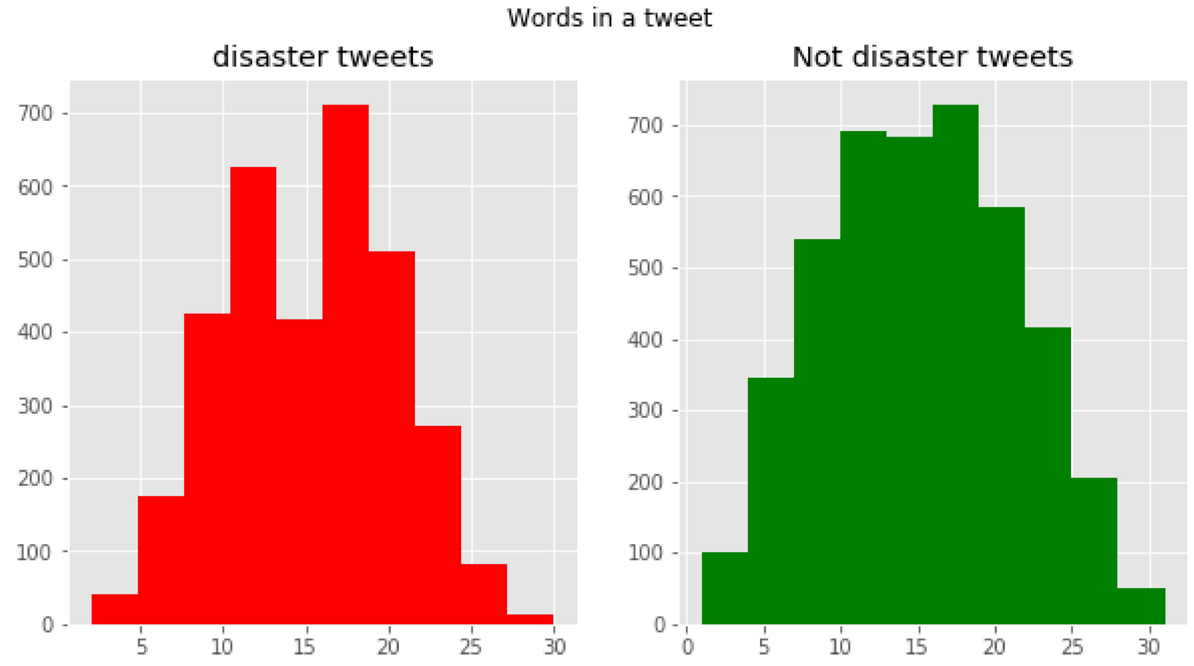
In [6]:
```
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(10,5))
tweet_len=tweet[tweet['target']==1]['text'].str.len()
ax1.hist(tweet_len,color='red')
ax1.set_title('disaster tweets')
tweet_len=tweet[tweet['target']==0]['text'].str.len()
ax2.hist(tweet_len,color='green')
ax2.set_title('Not disaster tweets')
fig.suptitle('Characters in tweets')
plt.show()
```



The distribution of both seems to be almost same.120 t0 140 characters in a tweet are the most common among both.

Number of words in a tweet

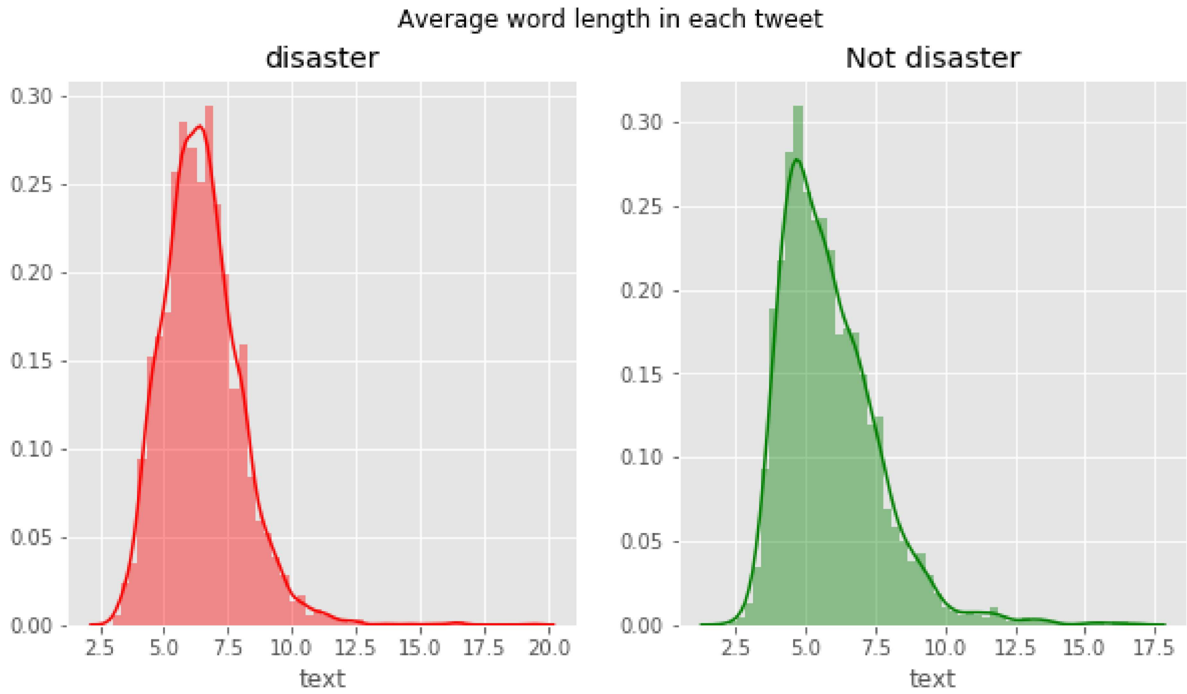
In [7]:
```
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(10,5))
tweet_len=tweet[tweet['target']==1]['text'].str.split().map(lambda x: len(x))
ax1.hist(tweet_len,color='red')
ax1.set_title('disaster tweets')
tweet_len=tweet[tweet['target']==0]['text'].str.split().map(lambda x: len(x))
ax2.hist(tweet_len,color='green')
ax2.set_title('Not disaster tweets')
fig.suptitle('Words in a tweet')
plt.show()
```



Average word length in a tweet

In [8]:
```
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(10,5))
word=tweet[tweet['target']==1]['text'].str.split().apply(lambda x : [len(i) for i in x])
sns.distplot(word.map(lambda x: np.mean(x)),ax=ax1,color='red')
ax1.set_title('disaster')
word=tweet[tweet['target']==0]['text'].str.split().apply(lambda x : [len(i) for i in x])
sns.distplot(word.map(lambda x: np.mean(x)),ax=ax2,color='green')
ax2.set_title('Not disaster')
fig.suptitle('Average word length in each tweet')
```

Out[8]:
```
Text(0.5, 0.98, 'Average word length in each tweet')
```




```
In [9]: def create_corpus(target):
    corpus=[]

    for x in tweet[tweet['target']==target]['text'].str.split():
        for i in x:
            corpus.append(i)
    return corpus
```

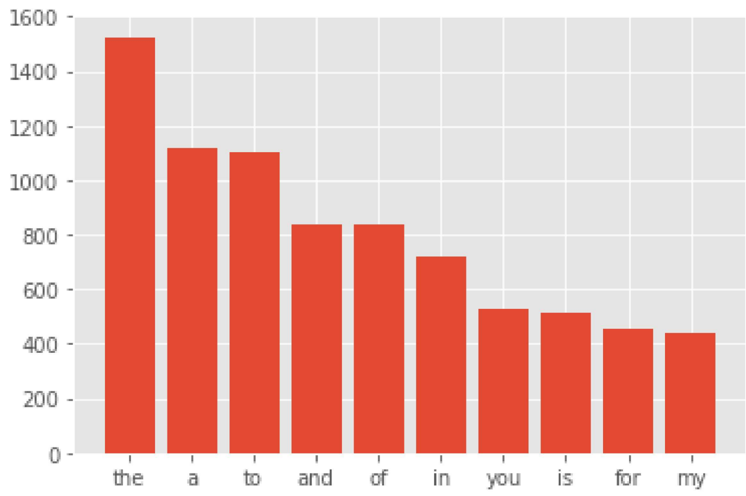
Common stopwords in tweets

First we will analyze tweets with class 0.

```
In [10]: corpus=create_corpus(0)

dic=defaultdict(int)
for word in corpus:
    if word in stop:
        dic[word]+=1

top=sorted(dic.items(), key=lambda x:x[1],reverse=True)[:10]
```

```
In [11]: x,y=zip(*top)
plt.bar(x,y)
```

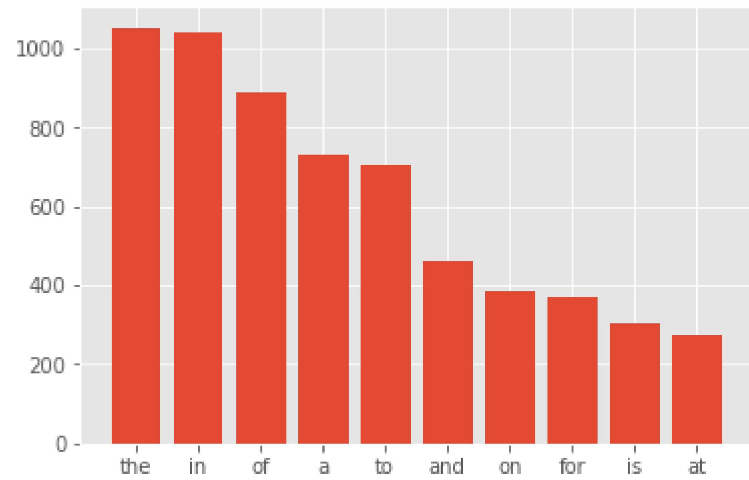
Out[11]: <BarContainer object of 10 artists>



Now,we will analyze tweets with class 1.

```
In [12]: corpus=create_corpus(1)

dic=defaultdict(int)
for word in corpus:
    if word in stop:
        dic[word]+=1

top=sorted(dic.items(), key=lambda x:x[1],reverse=True)[:10]



x,y=zip(*top)
plt.bar(x,y)
```

Out[12]: <BarContainer object of 10 artists>



In both of them,"the" dominates which is followed by "a" in class 0 and "in" in class 1.

Analyzing punctuations.

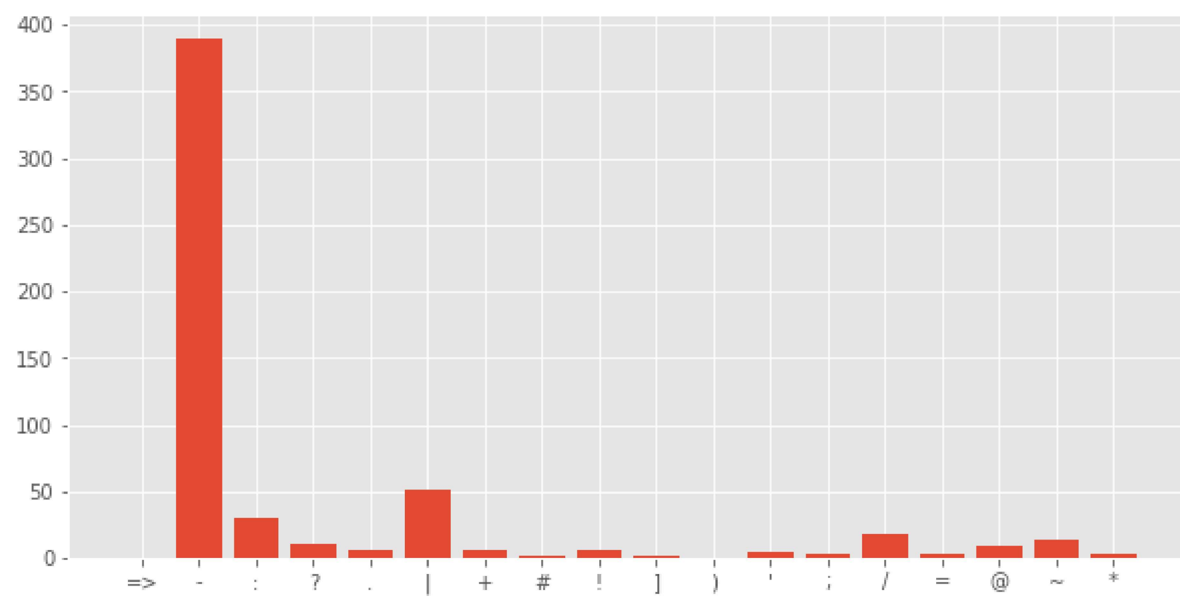
First let's check tweets indicating real disaster.

In [13]:

```python
plt.figure(figsize=(10,5))
corpus=create_corpus(1)

dic=defaultdict(int)
import string
special = string.punctuation
for i in (corpus):
    if i in special:
        dic[i]+=1

x,y=zip(*dic.items())
plt.bar(x,y)
```

Out[13]:

```
<BarContainer object of 18 artists>
```



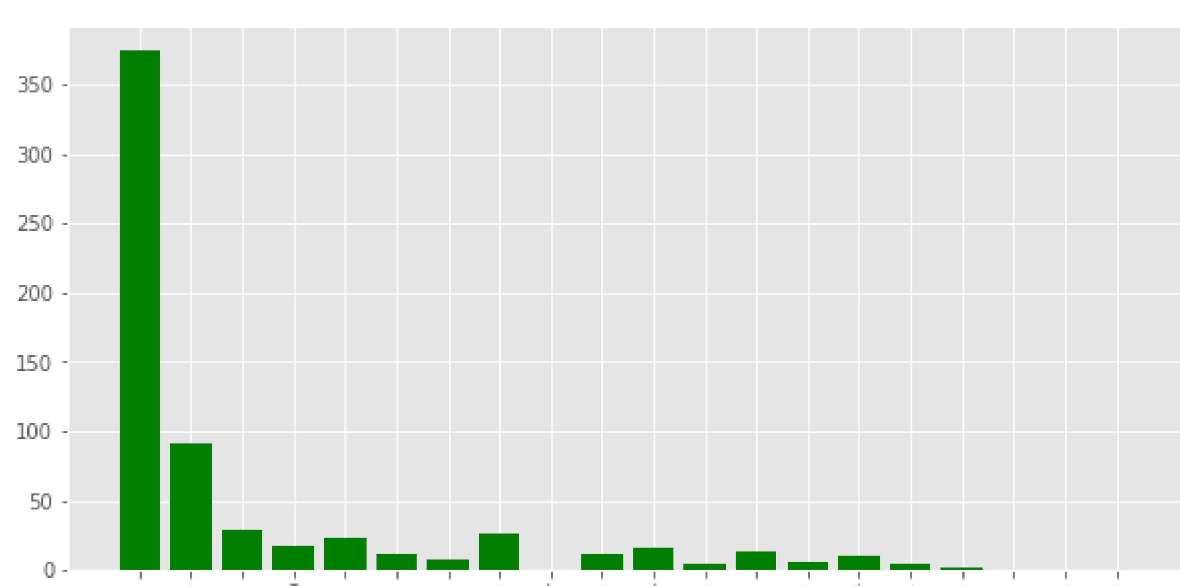
Now,we will move on to class 0.

In [14]:

```python
plt.figure(figsize=(10,5))
corpus=create_corpus(0)

dic=defaultdict(int)
import string
special = string.punctuation
for i in (corpus):
    if i in special:
        dic[i]+=1

x,y=zip(*dic.items())
plt.bar(x,y,color='green')
```

Out[14]:

```
<BarContainer object of 20 artists>
```

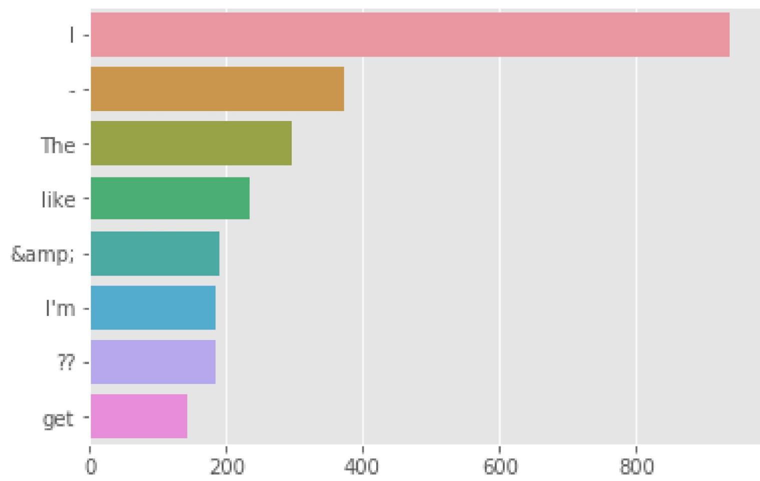


Common words

In [15]:
```
counter=Counter(corpus)
most=counter.most_common()
x=[]
y=[]
for word,count in most[:40]:
    if (word not in stop) :
        x.append(word)
        y.append(count)
```

In [16]:
```
sns.barplot(x=y,y=x)
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x797b48e686d8>



Lot of cleaning needed.

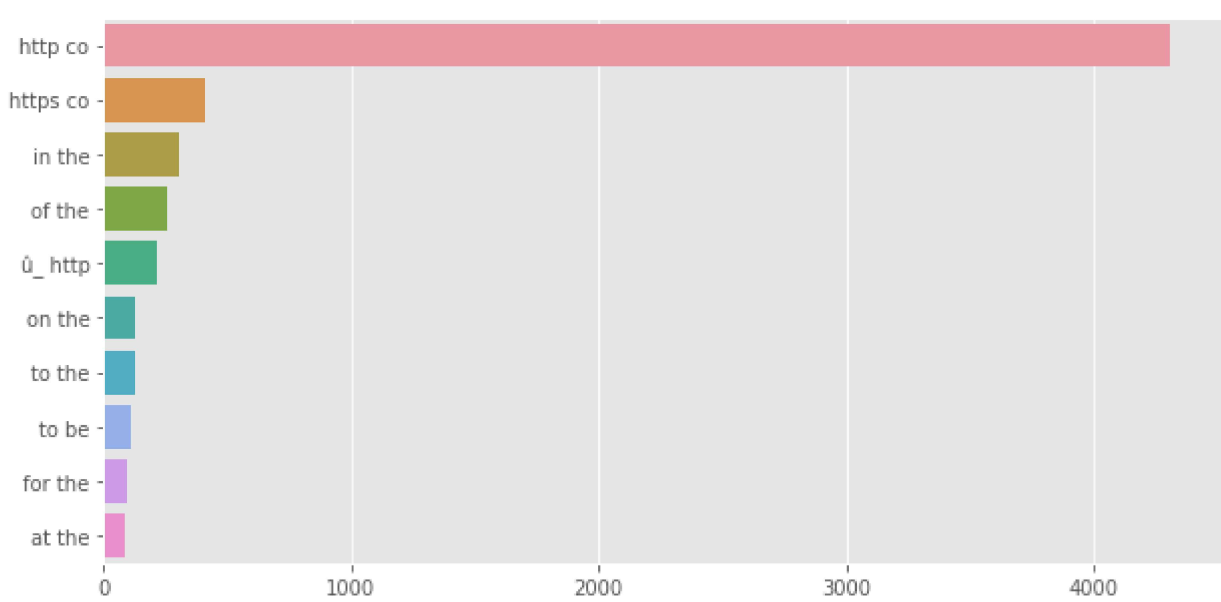
Ngram analysis

we will do a bigram (n=2) analysis over the tweets.Let's check the most common bigrams in tweets.

In [17]:
```
def get_top_tweet_bigrams(corpus, n=None):
    vec = CountVectorizer(ngram_range=(2, 2)).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
```

In [18]:
```
plt.figure(figsize=(10,5))
top_tweet_bigrams=get_top_tweet_bigrams(tweet['text'])[:10]
x,y=map(list,zip(*top_tweet_bigrams))
sns.barplot(x=y,y=x)
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x797b49208128>



We will need lot of cleaning here..

Data Cleaning

As we know,twitter tweets always have to be cleaned before we go onto modelling.So we will do some basic cleaning such as spelling correction,removing punctuations,removing html tags and emojis etc.

In [19]:
```
df=pd.concat([tweet,test])
df.shape
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:1: FutureWarning: Sorting because non-concatenation axis i
s not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  """Entry point for launching an IPython kernel.
```

Out[19]:

```
(10876, 5)
```

Removing urls

In [20]:

```python
example="http://nlp-getting-started"
```

In [ ]:

```python
def remove_URL(text):
    url = re.compile(r'https?://\S+|www\.\S+')
    return url.sub(r'',text)

remove_URL(example)
```

In [22]:

```python
df['text']=df['text'].apply(lambda x : remove_URL(x))
```

Removing HTML tags

In [23]:

```python
example = """<div>
<h1>Real or Fake</h1>
<p>K </p>
<a href="https://www.k.com/c/nlp-getting-started">getting started</a>
</div>"""
```

In [ ]:

```python
def remove_html(text):
    html=re.compile(r'<.*?>')
    return html.sub(r'',text)
print(remove_html(example))
```

In [25]:

```python
df['text']=df['text'].apply(lambda x : remove_html(x))
```

Romoving Emojis

In [26]:

```python
def remove_emoji(text):
    emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)

remove_emoji("Omg another Earthquake 😔😔")
```

Out[26]:

```
'Omg another Earthquake '
```

In [27]:

```python
df['text']=df['text'].apply(lambda x: remove_emoji(x))
```

Removing punctuations

In [28]:

```python
def remove_punct(text):
    table=str.maketrans('','',string.punctuation)
    return text.translate(table)

example="I am a #king"
print(remove_punct(example))
```

```
I am a king
```

In [29]:

```python
df['text']=df['text'].apply(lambda x : remove_punct(x))
```

Spelling Correction

use `pyspellcheker` to do that.

In [30]:

```
!pip install pyspellchecker
```

```
Collecting pyspellchecker
  Downloading https://files.pythonhosted.org/packages/74/2e/dd08caa20287695631e7267aa6ad0dbd1990cae32323db1620cc5cb549b
0/pyspellchecker-0.7.2-py3-none-any.whl (3.4MB)
     |████████████████████████████████| 3.4MB 6.3MB/s eta 0:00:01
Installing collected packages: pyspellchecker
Successfully installed pyspellchecker-0.7.2
```

In [31]:

```python
from spellchecker import SpellChecker
```

```
spell = SpellChecker()
def correct_spellings(text):
    corrected_text = []
    misspelled_words = spell.unknown(text.split())
    for word in text.split():
        if word in misspelled_words:
            corrected_text.append(spell.correction(word))
        else:
            corrected_text.append(word)
    return " ".join(corrected_text)

text = "corect me plese"
correct_spellings(text)
```

Out[31]:
```
'correct me plese'
```

In [32]:
```
#df['text']=df['text'].apply(lambda x : correct_spellings(x)#)
```

GloVe for Vectorization

Use GloVe pretrained corpus model to represent our words.It is available in 3 varieties :50D ,100D and 200 Dimentional.We will try 100 D here.

In [33]:
```
def create_corpus(df):
    corpus=[]
    for tweet in tqdm(df['text']):
        words=[word.lower() for word in word_tokenize(tweet) if((word.isalpha()==1) & (word not in stop))]
        corpus.append(words)
    return corpus
```

In [34]:
```
corpus=create_corpus(df)
```

```
100%|██████████| 10876/10876 [00:02<00:00, 4405.49it/s]
```

In [35]:
```
embedding_dict={}
with open('../input/glove-global-vectors-for-word-representation/glove.6B.100d.txt','r') as f:
    for line in f:
        values=line.split()
        word=values[0]
        vectors=np.asarray(values[1:],'float32')
        embedding_dict[word]=vectors
f.close()
```

In [36]:
```
MAX_LEN=50
tokenizer_obj=Tokenizer()
tokenizer_obj.fit_on_texts(corpus)
sequences=tokenizer_obj.texts_to_sequences(corpus)

tweet_pad=pad_sequences(sequences,maxlen=MAX_LEN,truncating='post',padding='post')
```

In [37]:
```
word_index=tokenizer_obj.word_index
print('Number of unique words:',len(word_index))
```

```
Number of unique words: 20342
```

In [38]:
```
num_words=len(word_index)+1
embedding_matrix=np.zeros((num_words,100))

for word,i in tqdm(word_index.items()):
    if i > num_words:
        continue

    emb_vec=embedding_dict.get(word)
    if emb_vec is not None:
        embedding_matrix[i]=emb_vec
```

```
100%|██████████| 20342/20342 [00:00<00:00, 321799.12it/s]
```

Baseline Model

In [39]:
```
model=Sequential()

embedding=Embedding(num_words,100,embeddings_initializer=Constant(embedding_matrix),
                   input_length=MAX_LEN,trainable=False)

model.add(embedding)
model.add(SpatialDropout1D(0.2))
model.add(LSTM(64, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))


optimzer=Adam(learning_rate=1e-5)

model.compile(loss='binary_crossentropy',optimizer=optimzer,metrics=['accuracy'])
```

In [40]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 50, 100)           2034300   
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 50, 100)           0         
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                42240     
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65        
=================================================================
Total params: 2,076,605
Trainable params: 42,305
Non-trainable params: 2,034,300
_________________________________________________________________
```

In [41]:
```
train=tweet_pad[:tweet.shape[0]]
test=tweet_pad[tweet.shape[0]:]
```

In [42]:
```
X_train,X_test,y_train,y_test=train_test_split(train,tweet['target'].values,test_size=0.15)
print('Shape of train',X_train.shape)
print("Shape of Validation ",X_test.shape)
```

```
Shape of train (6471, 50)
Shape of Validation  (1142, 50)
```

In [43]:
```
history=model.fit(X_train,y_train,batch_size=4,epochs=15,validation_data=(X_test,y_test),verbose=2)
```

```
Train on 6471 samples, validate on 1142 samples
Epoch 1/15
 - 61s - loss: 0.6914 - accuracy: 0.5665 - val_loss: 0.6883 - val_accuracy: 0.5841
Epoch 2/15
 - 60s - loss: 0.6695 - accuracy: 0.5823 - val_loss: 0.5976 - val_accuracy: 0.7215
Epoch 3/15
 - 60s - loss: 0.6016 - accuracy: 0.7044 - val_loss: 0.5347 - val_accuracy: 0.7820
Epoch 4/15
 - 60s - loss: 0.5815 - accuracy: 0.7271 - val_loss: 0.5133 - val_accuracy: 0.7872
Epoch 5/15
 - 61s - loss: 0.5663 - accuracy: 0.7385 - val_loss: 0.5037 - val_accuracy: 0.7881
Epoch 6/15
 - 61s - loss: 0.5597 - accuracy: 0.7381 - val_loss: 0.4973 - val_accuracy: 0.7977
Epoch 7/15
 - 62s - loss: 0.5615 - accuracy: 0.7421 - val_loss: 0.4966 - val_accuracy: 0.7907
Epoch 8/15
 - 62s - loss: 0.5563 - accuracy: 0.7476 - val_loss: 0.4885 - val_accuracy: 0.7951
Epoch 9/15
 - 62s - loss: 0.5449 - accuracy: 0.7552 - val_loss: 0.4858 - val_accuracy: 0.7986
Epoch 10/15
 - 61s - loss: 0.5482 - accuracy: 0.7497 - val_loss: 0.4835 - val_accuracy: 0.7977
Epoch 11/15
 - 61s - loss: 0.5413 - accuracy: 0.7557 - val_loss: 0.4814 - val_accuracy: 0.7977
Epoch 12/15
 - 62s - loss: 0.5368 - accuracy: 0.7591 - val_loss: 0.4801 - val_accuracy: 0.8021
Epoch 13/15
 - 61s - loss: 0.5332 - accuracy: 0.7634 - val_loss: 0.4770 - val_accuracy: 0.7968
Epoch 14/15
 - 61s - loss: 0.5404 - accuracy: 0.7572 - val_loss: 0.4767 - val_accuracy: 0.8004
Epoch 15/15
 - 61s - loss: 0.5318 - accuracy: 0.7634 - val_loss: 0.4747 - val_accuracy: 0.8004
```

Final result

In [44]:
```
sample_result=pd.read_csv('../input/nlp-getting-started/sample_result.csv')
```

In [45]:
```
y_pre=model.predict(test)
y_pre=np.round(y_pre).astype(int).reshape(3263)
sub=pd.DataFrame({'id':sample_sub['id'].values.tolist(),'target':y_pre})
result.to_csv('result.csv',index=False)
```

In [46]:
```
result.head()
```

Out[46]:

	id	target
0	0	1
1	2	1
2	3	1
3	9	1
4	11	1