Coursera course: Introduction to Deep Learning

Week 4 assignment: NLP Disaster Tweets Kaggle Mini-Project

Natural Language Processing with Disaster Tweets: Predict which Tweets are about real disasters and which ones are not

University of Colorado Boulder

Taught by:Geena Kim , Assistant Teaching

Solution by: Miguel Duque B.

Date: March, 2023

Deliverable:

A Jupyter notebook with a description of the problem/data, exploratory data analysis (EDA) procedure, analysis (model building and training), result, and discussion/conclusion.

Table of contents

1. Brief description of the problem and data

This notebook is based on the data from the kaggle competition: Natural Language Processing with Disaster Tweets (https://www.kaggle.com/competitions/nlp-getting-started).

Problem:

> In this competition, you're challenged to build a machine learning model that predicts which Tweets are about real disasters and which one's aren't. You'll have access to a dataset of 10,000 tweets that were hand classified. If this is your first time working on an NLP problem, we've created a quick tutorial to get you up and running.

> Disclaimer: The dataset for this competition contains text that may be considered profane, vulgar, or offensive.
>
> Acknowledgments This dataset was created by the company figure-eight and originally shared on their 'Data For Everyone' website here.
>
> Tweet source: https://twitter.com/AnyOtherAnnaK/status/629195955506708480 (https://twitter.com/AnyOtherAnnaK/status/629195955506708480)

Data:

> Each sample in the train and test set has the following information:
>
> - id - a unique identifier for each tweet
> - text - the text of the tweet
> - location - the location the tweet was sent from (may be blank)
> - keyword - a particular keyword from the tweet (may be blank)
> - target - in train.csv only, this denotes whether a tweet is about a real disaster (1) or not (0)

More insights:

- There are 7613 tweets in the training data set, which is balanced.
- The test set contains 3263 samples where we should predict the target (0 or 1).
- This is a binary classification problem
- Location and keyword are categorical variables containing strings and missing values. Tree-based models are a good option for these.
- Texts will be processed with some RNN after cleaning and embedding them.

Import libraries

In [1]:
```
import os
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
import pickle
from sklearn.model_selection import train_test_split
from tqdm import tqdm
```

2. Procedure

Plan for analysis using RNN with LSTM or GRU

The procedure followed has been inspired by this tutorial on patent text predictions with RNNs (https://towardsdatascience.com/recurrent-neural-networks-by-example-in-python-ffd204f99470), with some modifications.

0. Inspect and visualize the data. We will also handle missing values.
1. Clean text in tweets before computing word embeddings: remove urls, user names,...
2. Tokenize and convert text from list of strings into list of lists of integers (sequences) using Keras Tokenizer
3. Load in pre-trained embeddings (Glove model trained on tweets) to compute a embedding matrix
4. Split train data into a training subset and a cross validation subset.
5. Build a basic machine learning model as baseline, which uses only keyword and location
6. Create features to pass to an RNN model: a sequence of words is defined
7. Build an RNN model with Embedding, LSTM (or GRU) cell, and Dense layers
8. Train model to predict next work in sequence
9. Make predictions by passing in starting sequence

Other models for comparison purposes

- A Baseline model is one that predicts the most frequent category
- A tree-based model based on `keyword` and `location`
- RNN models with stacked LSTM or GRU cells
- We can also create a mixed RNN model that takes both clean `text` and one-hot encoded categorical variables corresponding to `keyword` and `location`

The original competition maximizes the F1-score, but we will be maximizing accuracy during training, since the data classes are balanced.

3. Exploratory Data Analysis (EDA) — Inspect, Visualize and Clean the Data

3.1 Inspect and visualize data

Check available files in input folder:

In [2]:
```
print('List of input files:\n')
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
List of input files:

/kaggle/input/nlp-getting-started/sample_submission.csv
/kaggle/input/nlp-getting-started/train.csv
/kaggle/input/nlp-getting-started/test.csv
/kaggle/input/glove-twitter-pickles-27b-25d-50d-100d-200d/glove.twitter.27B.25d.pkl
/kaggle/input/glove-twitter-pickles-27b-25d-50d-100d-200d/glove.twitter.27B.50d.pkl
/kaggle/input/glove-twitter-pickles-27b-25d-50d-100d-200d/glove.twitter.27B.200d.pkl
/kaggle/input/glove-twitter-pickles-27b-25d-50d-100d-200d/glove.twitter.27B.100d.pkl
```

Remarks:

- We have 3 .csv files from the competition dataset. We will be working with the train.csv file.
- Because we are working natural language processing and tweets, I have added some pickle files that contain word embeddings.
- These are glove word embeddings from Stanford trained on 2B tweets (https://nlp.stanford.edu/projects/glove/).
- We will need to preprocess tweets, in the same way as the Stanford project did. They have published a Ruby code for text cleaning (https://nlp.stanford.edu/projects/glove/preprocess-twitter.rb).

Helper functions

In [3]:
```
#-------------------- Helper functions for histogram count annotation ----------------------------
def add_histogram_values(ax): [ax.bar_label(remove_0_tags_for_histograms(b)) for b in ax.containers]
def remove_0_tags_for_histograms(ax_container):
    ind = np.where(ax_container.datavalues>0)[0]
    ax_container.datavalues = ax_container.datavalues[ind]
    ax_container.patches = [ax_container.patches[i] for i in ind]
    return ax_container
```

Read data

In [4]:
```
df_train = pd.read_csv('/kaggle/input/nlp-getting-started/train.csv')
df_test = pd.read_csv('/kaggle/input/nlp-getting-started/test.csv')
```

Show training dataset

In [5]:
```
print('Train set')
display(df_train)
display(df_train.info())
```

```
Train set
```

	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
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1
...	...	...	...	...	...
7608	10869	NaN	NaN	Two giant cranes holding a bridge collapse int...	1
7609	10870	NaN	NaN	@aria_ahrary @TheTawniest The out of control w...	1
7610	10871	NaN	NaN	M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt...	1
7611	10872	NaN	NaN	Police investigating after an e-bike collided ...	1
7612	10873	NaN	NaN	The Latest: More Homes Razed by Northern Calif...	1

7613 rows × 5 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7613 entries, 0 to 7612
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   id        7613 non-null   int64 
 1   keyword   7552 non-null   object
 2   location  5080 non-null   object
 3   text      7613 non-null   object
 4   target    7613 non-null   int64 
dtypes: int64(2), object(3)
memory usage: 297.5+ KB

None
```

Show test data set

In [6]:
```
print('Test set')
display(df_test)
display(df_test.info())
```

```
Test set
```

	id	keyword	location	text
0	0	NaN	NaN	Just happened a terrible car crash
1	2	NaN	NaN	Heard about #earthquake is different cities, s...
2	3	NaN	NaN	there is a forest fire at spot pond, geese are...
3	9	NaN	NaN	Apocalypse lighting. #Spokane #wildfires
4	11	NaN	NaN	Typhoon Soudelor kills 28 in China and Taiwan
...	...	...	...	...
3258	10861	NaN	NaN	EARTHQUAKE SAFETY LOS ANGELES ÛÒ SAFETY FASTE...
3259	10865	NaN	NaN	Storm in RI worse than last hurricane. My city...
3260	10868	NaN	NaN	Green Line derailment in Chicago http://t.co/U...
3261	10874	NaN	NaN	MEG issues Hazardous Weather Outlook (HWO) htt...
3262	10875	NaN	NaN	#CityofCalgary has activated its Municipal Eme...

3263 rows × 4 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3263 entries, 0 to 3262
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   id        3263 non-null   int64 
 1   keyword   3237 non-null   object
 2   location  2158 non-null   object
 3   text      3263 non-null   object
dtypes: int64(1), object(3)
memory usage: 102.1+ KB

None
```

There aresome missing values in `kewword` and `location` columns

Explore some tweets

In [7]:
```
print('Some random tweets:\n')
for row in np.random.choice(len(df_train), size=15, replace=False):
    print('(id={}, label={}), text: {}'
          .format(df_train.loc[row,'id'],df_train.loc[row,'target'],df_train.loc[row,'text']))
```

```
Some random tweets:

(id=1316, label=0), text: Bloody Mary in the sink. Beet juice http://t.co/LUigmHMa1i
(id=7148, label=1), text: STERLING-SCOTT on the Red Carpet at a fundraiser for 'OSO Mudslide' https://t.c
o/mA4ra7AtqL http://t.co/cg579wlDnE
(id=6213, label=1), text: California School Bus Hijacker Parole Stands http://t.co/kPIVXGjNqt #sacramento
(id=9710, label=0), text: Maaaaan I love Love Without Tragedy by @rihanna I wish she made the whole song
(id=9619, label=0), text: Beautiful lightning as seen from plane window http://t.co/5CwUyLnFUm http://t.c
o/1tyYqFz13D
(id=8644, label=0), text: Talk on GOZ is fantastic. Most interesting fact so far is that they manually bou
ght all the .ru domains to sinkhole rather than seek co-op.
(id=4003, label=0), text: I'm a disaster?? https://t.co/VCV73BUaCZ
(id=1272, label=0), text: @Anthxvy runs in the blood
(id=1814, label=1), text: Charred remains of homes on 2nd St in Manchester NH after fire rips through 2 bu
ildings damaging neighboring homes http://t.co/Ja3W1S3tmr
(id=8902, label=1), text: Sassy city girl country hunk stranded in Smoky Mountain snowstorm #AoMS http://
t.co/nkKcTttsD9 #ibooklove #bookboost
(id=2207, label=0), text: @SyringeToAnger åÇ and probably even more. But some disagreements with General R
oss and the catastrophic occurrence made something clear. åÈ
(id=10427, label=0), text: In @edfringe? We highly recommend @M00NF00L #Titania @Summerhallery A whirlwind
reimagining /Shakespeare's Midsummer https://t.co/iIAIGZkbnJ
(id=1921, label=1), text: Killing Black Babies at Planned Parenthood where's the demonstrations looting of
PP burning down the buildings Black Babies Lives Matter
(id=753, label=0), text: 2 TIX 10/3 Frozen Fury XVII: Los Angeles Kings v Avalanche 103 Row:AA MGM Grand h
ttp://t.co/kBtZZZG2Tp
(id=4908, label=0), text: I'm ready to explode! http://t.co/OwJe3i6yGN
```

Tweet contain distinctive features from formal texts, such as:

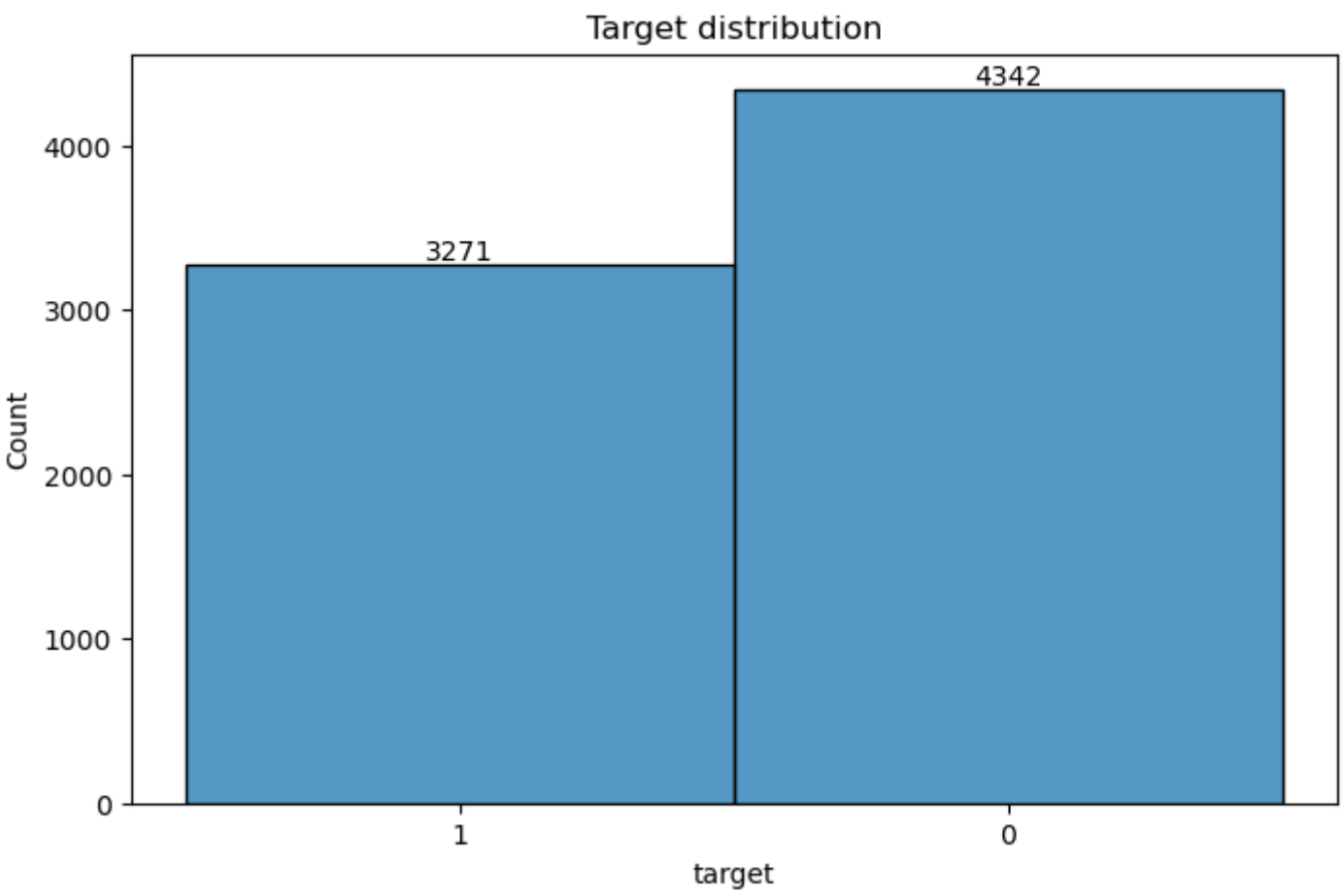
- URLs
- User names
- Hashtags
- Emojis,
- Some words are all in capital letters.

Target distribution

In [8]:
```
print('Target distribution (percentage):')
display(np.round(df_train['target'].value_counts(normalize=True)*100),2)
#-----------------------------------------------------------------------------------------------
fig, ax = plt.subplots(figsize=(8, 5))
ax2 = sns.histplot(
    data = df_train['target'].astype(str),
#     x = 'target',
#     hue = 'target',
#     palette = 'colorblind',
    legend = False,
    ).set(
        title = 'Target distribution');
add_histogram_values(ax)
```

```
Target distribution (percentage):


0    57.0
1    43.0
Name: target, dtype: float64
```

2



The data set is balanced

Locations

In [9]:
```
print('There are {} different locations in training set, with {} missing locations:'.format(
    len(df_train['location'].unique()),
    df_train['location'].isna().sum()))
df_train['location'].value_counts()
```

There are 3342 different locations in training set, with 2533 missing locations:

Out[9]:
```
USA                    104
New York                71
United States           50
London                  45
Canada                  29
                      ...
MontrÌ©al, QuÌ©bec       1
Montreal                1
ÌÏT: 6.4682,3.18287      1
Live4Heed??             1
Lincoln                 1
Name: location, Length: 3341, dtype: int64
```

In [10]:
```
print('There are {} different locations in test set, with {} missing locations:'.format(
    len(df_test['location'].unique()),
    df_test['location'].isna().sum()))
df_test['location'].value_counts()
```

There are 1603 different locations in test set, with 1105 missing locations:

Out[10]:
```
New York                  38
USA                       37
Worldwide                 16
United States             15
London                    13
                          ..
Medford, NJ                1
Quezon City                1
LanÌ¼s                      1
USA,Washington,Seattle     1
Brussels, Belgium          1
Name: location, Length: 1602, dtype: int64
```

Keywords

In [11]:
```
print('There are {} different keywords in training set, with {} missing keywords:'.format(
    len(df_train['keyword'].unique()),
    df_train['keyword'].isna().sum()))
df_train['keyword'].value_counts()
```

```
There are 222 different keywords in training set, with 61 missing keywords:
```

Out[11]:
```
fatalities              45
deluge                  42
armageddon              42
sinking                 41
damage                  41
                        ..
forest%20fire           19
epicentre               12
threat                  11
inundation              10
radiation%20emergency    9
Name: keyword, Length: 221, dtype: int64
```

In [12]:
```
print('There are {} different keywords in test set, with {} missing keywords:'.format(
    len(df_test['keyword'].unique()),
    df_test['keyword'].isna().sum()))
df_test['keyword'].value_counts()
```

```
There are 222 different keywords in test set, with 26 missing keywords:
```

Out[12]:
```
deluged               23
demolished            22
rubble                22
first%20responders    21
seismic               21
                      ..
threat                 5
fatalities             5
forest%20fire          5
inundation             4
epicentre              1
Name: keyword, Length: 221, dtype: int64
```

¿Are all locations from test set also in training set?

In [13]:

```
print('Locations in test set not found in train set:')
s = set(df_test['location'])
s1 = s - set(df_train['location'])
print('There are {} locations in test set not found in train set. This means {:.0%} of the test set locations
are not in train set'.format(len(s1),len(s1)/len(s)))
print('\nlist of such locations:')
print(s1)
```

Locations in test set not found in train set:
There are 1180 locations in test set not found in train set. This means 74% of the test set locations are not in train set

list of such locations:
{'Glendale, AZ', 'Turin', 'Harare', 'Pridelands, IN', '{Daughter of Ares} {Eighteen}', 'nipple squad makes me happy', 'riverwest, milw ', '? the Foothills of SC ?', 'Earthrealm', 'trust none..', 'Okinawa', 'hyejeong?taehyung', 'IRELAND ', 'Italia', 'Quezon City, PHILIPPINES', 'Yemen', 'NYC/LI/NJ/LHV', 'Diantara Temlen Laksani', 'Townsville', 'Secane, PA', 'Mississauga', 'Anime World', 'Saint Andrews, Scotland', 'Malta; Gozo', 'Dublin, Ireland. ', 'From Boston, for New England', 'Doo Doo Boy Island', 'Mtl', 'Halifax, Nova Scotia, Canada', 'St Louis', 'War Drobe, Spare Oom', 'NYC ?? PA', 'Acey mountain islanddåÇTorontoåÈ', 'Grand Rapids, Mich.', 'somewhere or other', 'Florence, SC', '#TeamHKNgang ', 'Paddington, London', 'Citizen of the World', '3/12 + Jake Miller', 'BurBank, CA', 'Oro Valley, AZ', 'kolkata', 'support all girls!', 'Brantford, Ontario', 'Seattle/Snohomish/Redlands', "Trapin'", 'Staffordshire', 'NoPa, San Francisco', 'baltimore ', '@rejxctedmgc is my Harry?', 'Boulder via DC', 'Newburgh, NY', 'Romans 1:16', 'Lawrence, MA', 'Melbourne VIC Australia', 'ICO Arkansas', 'philadelphia, pa', 'Lismore, New South Wales', 'Swat Pakistan', ' Kaijo High School', 'Texas, Forever ', 'North West', '408 N. Western, Wenatchee, WA ', 'TÌÁchira, Venezuela', 'Your Backyard', 'FRM JERZ LIVING IN DADE COUNTY', 'St. Catherine, Jamaica, W.I.', 'DC Metro Area', 'Brentwood uk', 'Cincinnati', '#Global', 'Wonderland', 'Cali', '22714 Ventura Blvd. WHills, CA', '??????????? :P', 'Boston/NY/Montreal/London', 'The Floor', 'Serving Oklahoma', 'Outreach Africa', 'Texas Hill Country', 'the beautiful Northwest (USA)', 'bishop amat', 'District of Columbia, USA', 'ShloMotion', 'Arlington VA (DC Area)', 'The North Coast', 'Los Angeles, CA 90045', 'Merseyside', 'Kurdistan ', 'SantibaÌ±ez de Vidriales ZAMORA', '#Bitcoinland', 'Fantasy Land', 'Morgantown, WV', 'Mass.', "Ellixton London's 33rd Borough", 'trinity nc', '#ALLblacklivesmatter', 'Harlem', 'Roermond', 'Georgetown,Guyana', 'Hudson Valley,NY', 'Kampala, Uganda', 'chillin', 'UK.', 'California Central Valley', 'mombasa', 'LaPorte, IN', 'melbs 1/2 #rham', 'Bridgetown, Barbados', 'Vienna, Austria, Europe', 'Twitter.', 'Be Earth', 'Forest Grove, OR', 'Tweets by David ', 'Manhattan, New York', 'Toronto #6ixSideMafia', 'Arlington', 'Pembroke Dock, Wales', 'Lampe, MO', 'Pea Ridge, WV', 'Cin City', 'west coast ', 'Carthage, OH', 'Cedar City, Utah', 'Carlow, Ireland', 'Thessaloniki, Greece', 'KNOXVILLE TN USA', 'Williamsport, PA', 'ON, Canada', 'Amos,Quebec,Canada', "Maryland/Myrtle Beach CCU'19", 'Kruibeke, BelgiÌÇ', 'Antioch, CA', 'trashcan somewhere in hell', 'Accrington, Lancashire', 'Jaipur, Rajasthan', 'Albequerque', 'San Diegohhjhhhhhghghpjg', 'stars/pens/caps/hawks', 'Niger State', 'Fallen TX', "Sky's The limit (B.I.G) ????", 'venezuela', 'Los Angeles, Califnordia', '626 - 702', 'Bay Area, California ', 'In outer space', 'IG: 94fijiwater', '????????', 'N.Y.C', 'Highway to Hell', 'The Three Broomsticks ', '   Manchester/Nantwich', 'K-Town ', 'Southern Cali', 'San Gabriel Valley, CA', 'beyond time and space', 'Capital Federal, Argentina', 'Detroit, Mi', 'Death City, Nevada', 'Louisiana the real La', 'Japan ', 'Pune, Maharashtra India', 'Bangkok, Thailand.', 'Strathmore, California', 'Edinburgh/ West Yorkshire', 'New Bedford, MA', 'Lake Charles, LA', 'Piscataway, NJ', 'Lemon Grove, CA', 'San Bernardo - Chile', 'HamptonRoads, Virginia', 'St. Catharines', 'Derbyshire', 'Terre Haute', 'YesNaija.com', 'FL', 'West Hampstead, London NW6', 'Worcester', 'sad', '#Capulets #5SOSfam #5quadfam ', '336', 'Santos, SÌ£o Paulo', 'bucky barnes hell', 'Seek for LIGHT today!!!', 'Hannover', 'Little Rock', '#BVSTRONG', 'LastLevelPress.com', 'Pasadena, CA', 'L.A', 'West Virginia', 'Guayaquil, Ecuador', '???\x89Ûç???\x89Ûçs??\x89Ûç????\x89Ûç????????s', 'arizona', '2014.02.14~ing', 'Nashville, Tenn.', "Wouldn't u like to know", 'Dublin,Ireland', 'Northeast, Ohio', 'Metro Detroit', 'Indian Trail, North Carolina', 'Gurgaon', '@kiarakardashhh | Fayetteville', '#sundaunited', 'Rocky Mountains Colorado', 'Kano', 'Worldwalking', 'Bucks', 'Brussels, Belgium', 'Unauthentically British UK', 'Cape Town, South Africa', 'Monaghan, Ireland', 'Seraphim Vault, Cosmodrone', 'Staten Island, NY', 'Free Hanseatic City of Bremen, Germany', 'Philippines :)', 'The Dalles, Oregon', '~always in motion~', 'Nigeria|| Lagos', 'Keeper of the TriForce ', 'ITUNES RADIO [Hiphop/Rap]', 'On The Net', 'Johnstown, NY', 'Interwebs', 'Lansing ?? Ypsilanti ', 'Islamabad Pakistan', 'Emirate', 'Eldora, IA', 'West Baltimore ', 'Berisso, Argentina', 'Richmond, VA & Bristow, VA', 'TAIZ - YEMEN', 'isabel beatriz paras de leon', 'Colorado, The Mile High City', 'Twin Falls, Idaho, 83301', '#IzzoWorld', 'Pedophilia', 'Globally 1-844-360-WISE', 'Wayne, NJ', 'Fukushima, JAPAN', 'Bloemfontein, Free State', 'SeattIe', '#uniteblue', 'Akron, Ohio ', 'ÌÏT: 58.193556,-5.334943', ' ', 'Niels Groeneveld / RedSocks', '65 Skelmersdale Lane', 'Florence, South Carolina', 'London / Herts', 'ÌÏT: 41.373061,-71.942237', 'Ann Arbor, MI', 'Phuket', 'BogotÌÁ, Colombia', ' Cairo', 'Quezon City', 'New Orleans', 'USA - Global Online Sales ', 'Brinscall, England', 'Golden British Columbia Canada', 'Washington, DC area', 'North Druid Hills, GA', 'Monroe, OH', 'St.Louis', 'SÌ©te, France, (foto c.1968)', 'Iowa', 'England UK', ' Birmingham.', 'united states', 'Kentucky', 'almonds', 'ÌÏT: 7.384559,3.8793718', 'Chattanooga, TN', 'Saint John, N.B, Canada ', 'real world', ' Stah-koomi-tapii-akii', 'Sunderland/The Saaf', 'Ottawa, ON, Canada', 'HEAVEN', 'Goldsboro, NC', 'Sales Specialist~ Worldwide ', 'Johnson County, Kansas', 'salem ma', 'Paranaque City, National Capital Region', 'Greater Vancouver, British Columbia', "Where e'er the mood takes me", '10 hours from pluto', 'Eugene, OR.', 'Republic of Harper ', 'Moncton, NB', 'Outworld', 'MANAGEMENT MULTIMEDIA', 'UberlÌçndia - MG - Brasil', 'ham ham clubhouse', '?? Newport , RI ??', 'Arlington\n', 'Utica , New York (Upstate) ', 'Deerfield Beach Fl

orida', 'Port Townsend, WA', 'Herbville', 'Edwardsville, IL', 'GO Bucks!', 'Lehigh Acres, FL', 'probably b
argos', 'upstate NY', 'PrincetonAve', 'Red Deer, Alberta, Canada', 'mn |7-18-13|8-27-14|7-26-15|', 'LND Gr
eatest City In the World', 'Ex Astris Scientia. TSSADID.', '36702 State Road 52, Dade City', 'Hollywood, C
a', 'Brussels', 'Shangri-La', 'Batam', 'Fredericton, NB', 'mullingar ireland', 'born on september 1st', 'K
yiv (Kiev), Ukraine (????, ????, ???????)', 'Revolutionary Road, USA', 'kansas USA', 'Namma Bengaluru', "3
05 but I'm So St. Louis...", 'Vineland, NJ, USA', 'Land of the Free', 'ny || ga', 'Gauteng Heidelberg,Rata
nda', 'Kibworth Beauchamp, England', 'Kocaeli-Izmit', 'Banner Elk, NC', '???/??', 'Tokio / Tokyo', 'Tri-St
ate', 'Albany, Western Australia', 'kentucky', 'Plymouth, England', 'Loreto College, Mullingar', ' way way
way up ??', 'Here, There & Everywhere..', 'jds ', 'Plymouth', 'Amsterdam, Worldwide', 'For a healthier, ha
ppier YOU!', 'daily ? 18 ? ?', 'Metropolis, TX', 'Ì\x81th Cliath, Ìäire', 'Southend-on-Sea, England', 'Wes
t Monroe, Louisiana', 'Between Manchester and Lille.', 'Salem, OR', 'London. ', 'Sherbrooke & MontrÌ©al',
'Texas, United States', 'Raleigh, North Carolina, USA', 'From Harlem to Duke ! ', '| INDIA |', '@KEYTNC3 @
KCOY', 'Concord, North Carolina', 'Springfield, MO', 'Oregon & SW Washington', 'who fuckin knows', 'omaha
neblastya', 'PNW', 'Woodbridge, VA ', 'Sussex, UK', 'Alexander, Iowa', 'Inwood ontario', 'Moreno City - Bu
enos Aires', 'Rome', 'Garland, Texas', 'Pico Rivera, CA', 'N.E.Ohio', 'Dublin ', 'crying', 'quantico', 'Mo
ntana Misery', 'EspaÌ±a/Catalunya/Girona', 'england', 'Summerside', 'West Virginia ', 'Georgia/Florida ',
'Central New Jersey', 'Hollywood, California', 'Leanbox?', 'East la Mirada, CA', 'Keffi', 'San Luis Obisp
o, California', 'United States, Ohio', 'West Coast', 'Chicagoland and the world!', 'FEMA REGION 2', '(quee
r, trans, he/him, black)', 'Kill Devil Hills', 'Fargo, ND | North of Normal ', '32935', 'Wherever the musi
c takes me ', 'Lakeland Fl', 'Toledo, OH ', 'Prague, CZ, the Planet of Fans', 'Modishthaan', 'Last Legs, N
ew Zealand', 'Bengaluru, India', 'bkk', 'South of Boston', 'Botswana Gaborone', 'Alberta, Canada', 'Long I
sland, New York', 'Wakefield, England', 'Plovdiv, Bulgaria', 'Midland, MI', 'Midland, Mi', 'Monroe,LA', 'P
arkway, CA', 'DMZ, AZ', 'Try looking at the map?', 'land-where-everything-sucks', 'istanbul', 'centre of a
ttention', 'Tucson, Arizona', 'Mansfield, Ohio', 'would rather be in my room', 'Brooksville, Florida', '?a
rsehole squad?', 'Curitiba-PR', 'Dallas, Texas', 'Astoria, NY', 'Morganton GA', 'South Park
??????????????????', '62', 'Sporting capital of the World', 'EVERYWHERE, USA', 'Dalton', 'atl, ga', 'FL/N
J', 'No', 'darwins, au ', 'Saint Petersburg, FL', 'Camberville (Bostonish)', 'USA!', 'jonas/lovato/bieber/
5sos', 'South East Florida', 'Dunedin, New Zealand', 'Mobile, AL', 'Kaneville', '9 Benua Ltd.', 'Canaduh',
'BGSU', 'Journey ', 'Nanda Parbat', 'S.P.Brasil', 'Twin Cities, MN', 'Stamford, CT', 'MDS af ?', 'Dreamvil
le | Hoolagan', 'Washington DC region', 'new jersey', 'm a a l k ', 'Rochester Minnesota', 'Mainer missing
Guatemala', 'Dallas', 'Modesto, CA', 'Wellington, FL', 'between here and there', 'Bellingham, WA', 'Tokyo
JAPAN', 'Sandbach Cheshire UK ', 'shanghai', 'Whoop Ass, Georgia', 'Redding ', 'Los Angeles via Pittsburg
h', "At Grandmother Willow's", 'live from the low end', 'Time for NON VIOLENT dissent! ', 'Burnley, Lancas
hire', 'West Coast, Canada', 'Utopia ', 'My house probably', 'wolverhampton, low hill', 'Ontario CA', 'Ven
ice, Sweden', 'Surrey, BC', 'St. Louie', "Ton's Ì¡ead town åÈ Tx", 'Chiba,very close to Tokyo.', 'Davis, C
A', 'Android Land', 'Rome, IT', 'Coos Bay, OR', 'Long Beach, Dublin, Amsterdam', 'Algonquin, IL', 'globetr
otter', 'Cowtown, Caliii !!', 'Barnsley', 'Hagerstown, MD 21742', 'Crawley, England', 'Philadelphia; World
War I', 'Philippians 4:13', "sam's town", 'Fishkill, NY', 'Brisbane, Australia', 'Lagos, Nigeria.', 'Unive
rsity of Oklahoma', 'Hustle Flow Nation', 'Flat Rock', 'Delta, Nigeria', 'Oh.', "'schland", 'clearlake',
'Beacon Hills ~ The Glade', 'Dili. East Timor', 'Fairbanks Alaska', 'argentina', '?? | pittsburgh ', 'West
Palm Beach, FL', 'Edmonton Alberta Canada', 'Hartford, CT', 'Polska', 'Syracuse, NY', 'Covering the U.S.
A.', 'Spring Lake Park, MN', 'Leeds, England, United Kingdom', 'FL.CO.HI.NJ', 'yakima, wa', 'baltimore mar
yland', 'Richmond, VA, USA', 'Please follow and RT! :)', 'qui tacet consentire videtur', 'http://t.co/mOY6
lmu', 'Nj ?? Az ', 'British Columbia', 'Hermiston, Oregon', 'New Hampshire \x89Ûç WMUR ', 'Grand Rapids, M
I', '?illinois?', 'Fano (Italy)', 'Eastbourne East Sussex ', 'PPCC ', 'Wickford', 'Cheltenham', 'Indianapo
lis, Indiana', 'CLT via NOLA via MO via MN', 'Sheeps Clothing', 'Bikini Bottom', 'ÌÏT: 50.953278,-113.9787
85', 'Gislaved', 'Issaquah, WA', 'Back of the beyond, Lesotho', 'Roseland', 'Virginia (USA)', 'Lawrence, K
S', 'UiTM, Shah Alam', 'Salt Lake City', 'Do Not Follow Me, Am I a Bot.', '9.25.14?8.5.15?10.6.15 | gen?',
'Calgary ', 'Nigeria,West African', 'Miami - Support NAVY #SEALS ', 'Logansport, Indiana', 'wherever Wolf
Blitzer is', 'started acc 1.9.15 2:25 pm', 'Northampton', 'astral plane', 'Dark Night. ???? ??', 'Head of
the Family', 'Qwuank', 'Sherman ave South Bronx #STF', 'usa', 'Milano', 'Irvine, CA', 'High Desert', 'Real
ity Based World', "East Carolina University'19 ??", '35% gay. 65% water. ', 'Shrewsbury, England', 'AZ ? T
X ? CA ? IL ?', 'Bobba Island ', 'Ohio land of no sun', 'vb/norfolk,va', 'Nor Cal', "Your Girl's Pussy",
'Cape Coral, FL USA', 'NYC, ??, VAN', 'Lithgow, NSW, Australia', 'Detroit, MI ', 'Sericita, Minas Gerais',
'KATONG PLAZA #02-10', 'among the socially awkward ?', 'Northwest Arkansas', 'My World', 'Paradise ', 'Mil
an, Italy', '#Kashmir', 'Seattle Washington', 'somewhere in time', 'Andhra Pradesh, India', 'Sunny Florid
a', 'West Midlands, England', 'Ottawa', 'On The Island: Nassau, Bahamas', 'Lexington, KY', 'South Africa B
raamfontein', 'Wheeler,Wis.', 'U.S.A.', 'instagram- joeyparmenterr', 'Jefferson, GA', 'Toulouse Haute-Garo
nne France', 'Conway, AR', 'In my baby with pie', 'State of Jefferson', 'The other side of the mirror', 'B
exhill', ' ? they/them ?', 'Kurnool', 'Narbin city ', '6/24/12 7/21/13 8/22/14 7/17/15', 'Wolfgangmuzic@g
mail.com', 'Huntington Beach, CA', 'Kent, United Kingdom', 'South East Asia', 'Amman', 'bristol ', 'Deep i
n the heart of LibLand', 'Chinade, Nigeria', 'Free Palestine | Save Gaza', 'IG: xbougiebri', 'HELL', '17 y
ear old anti-theist.', 'Ventura County', 'Canberra (mostly)', 'St Albans', 'Perth, Scotland', 'madrid', 'N

orthwest Jersey', 'Almost there ', 'University, FL', 'Denver,CO USA', 'Jabalpur', 'Amsterdam NL or Greenwi
ch USA', '/ Kattappana, Kerala ', 'Franklinton - BR - Houston', 'rest easy angel @datboiiharri.', 'Panama
City Beach, Florida', 'My Own Little Corner', 'U.S.A. ', 'My own little world.', 'Wellington City, New Zea
land', 'Temple, GA', 'slytherin /', 'Outer Rim', 'U.A.E.', 'Bedford Stuyvesant, Brooklyn', 'State College
Pa', 'Kalispell, Montana', 'Jacksonville, Florida', 'DC & MoCo', 'US of Eh', 'Carrboro, NC', '8.27.14 & 7.
28.15', 'Pune', 'Rahway, NJ', 'chapel hill, cary n.c.', 'Toronto, Ontario, Canada', 'Canton Ohio', 'Mystic
Falls', 'Orange County, California ', 'Kanto, Japan', 'Vault 101, Fallout', 'Az', '|Elsmere| Wilmington, D
E.', 'http://wingssilverwork.com', 'fairly local', 'East Texas', 'BLF (MP)/ PTA (GP)/ PHB(LIMPS)', 'Hutchi
nson, MN', 'Stone Ridge, VA', 'it was yesterday :(', 'france', 'Denver|Boulder|Ft. Collins, CO', 'San Anto
ne', 'Manisto Trap House ', 'Pittsburgh, PA', 'ÌÏT: 35.353272,-78.553744', 'someplace safe', '#RoshanPakis
tan', 'psalms 27:1', 'Las Baegas', 'IL?MI', 'Over the Hills and Far Away', 'bliss', 'Somewhere in the Inte
rnet...', 'Salford, Greater Manchester', 'California USA', 'Home is wherever I am ', 'Sydney - Australia',
'Ecruteak City, Johto', 'Ohio...yep we said it.', 'Duel Academia', 'Cape Coral Fl', 'Portugal-Spain-Indone
sia', 'Whitley Bay, England', 'NWA & River Valley', 'In erotic world ', 'Yellowknife NT Canada', 'High in
Prague with Aya', 'burger king with usher', 'bestatriz \x89Ûç29/4 \x89Ûç #askdemigod ', 'Sunmy Melbourne,
England', 'Canada,Ontario', 'Atrapada en el mundo.', 'Salem, MA', 'Montana', 'Yorkshire and London', 'Stil
l on my laptop', 'Broward, Miami, Tally', 'Greece', 'dancer - Peter Pan panto 2015 ', 'Elko, Nevada', 'OHI
O', 'moon ', 'Northwest', 'Superior, WI', 'These United States', 'Chillicothe, OH', 'www.facebook.com/Rand
irobics', 'Scout Team ', 'GARRUS?', 'Orlando, FL ', 'lagos , Usa', 'St. Charles MO', 'Southeast Michigan
(HOTH)', 'Queens', 'In Range ', 'SIUC | 618', 'Reaching for the stars ', 'Under the rain...', 'All Motorwa
ys, UK', 'Paonia, Colorado', 'Lucknow', 'Isle of Patmos', 'Harlem New York', 'Auckland NZ', 'California mo
st of the time', 'your mom', 'cucumber squad | she/her | ', '424 N. FAIRFAX AVE. 90036', 'East Coast ', 'u
ncanny valley', 'Any town USA', 'Wandering', 'Kansas KS', 'Eugene, OR', 'In my head.....', 'Alexandria, V
A ', 'North Hollywood, CA. (SoCal)', 'Springfield Tn', 'Poughkeepsie, NY', 'Ngayogyakarta Hadiningrat ',
'Lahore, Pakistan', 'Tinley Park, IL', 'Holland,MI', 'Valusia', 'Bolton, England', 'HTX iNNeRvErSe', 'finl
and ', 'OKC', 'tacompton, washington ', 'Cary, North Carolina', 'Washington DC, an airport ', 'Calabasas,
CA', 'West Covina CA', 'tarn et garonne/lot et garonne', 'Seoul, Republic of Korea', "Niall's place | SAF
12 SQUAD |", 'Gap', 'Las Vegas ', 'our galaxy', 'boca raton', 'XIX | 5SOS | Ed Sheeran |', 'Windsor ??CT
', 'Manado,Sulawesi Utara', 'Wherever the #Sooners are', 'CLEVELAND', 'patna', 'with my music and air head
s', '121 N La Salle St, Suite 500', 'sam', 'SW London (RBK)', 'LOST Angeles lol', 'Winston Salem NC', 'War
aq', 'SDF-1 Macross S. Ataria Island', 'Sunny Southern California', 'Shellharbour, Wollongong', '557619',
'Columbia, MO', '????? ?? ?????? -????? ????', 'VH1 Soul in a BET World ', 'Currently Somewhere On Earth',
'Searching for Rivendell', 'Nevada', 'North Devon, UK ', 'she/her \x89Ûç 19 \x89Ûç poland', 'That London',
'norton ', 'Santa Cruz ', 'Ottawa Canada', 'Conover, NC', 'City of Bacoor, CALABARZON PHL', 'Sunny South A
frica', 'Jammu Kashmir ', 'Connecticut, USA', 'Washington, DC and on planes', 'El Cerrito, CA', 'Salem, Or
egon', '53.337853,-6.288693', 'Enemy of the state ', "sarah's dreads", 'Metro, Lampung ~ Balikpapan', 'Cal
gary and USA', '3-Jan', 'Greenville, SC', 'magodo', 'Cramerton, NC', 'Canada | #LUX', ' Aomori', 'Greenwic
h, London', 'University of Maryland', 'Quezon City, National Capital Region', 'Gresham, OR', 'palm spraang
s', 'SÌ£o Vicente, SÌ£o Paulo', 'probably in hell ', 'D.C.', 'Mumbai, India.', "N 32å¡39' 0'' / W 97å¡16'
0'", 'Brooklyn, NY and Norfolk, Va', 'Babylon Up-On Hudson', 'Poyth WesssterrrnnOystrayahhh', 'Desloge, M
o', 'Waterloo, Ontario, Canada', '3rd terrestrial planet, Sun', 'Upper St. Clair', 'Quezon, Philippines',
'Rowlett, TX', 'House of El', '770 to Benedict College ', 'VI~D[M]V', 'Portland', 'Central Jersey', 'Tuls
a', 'Ponyville,Equestria ', 'Punjab, Pakistan', "luisa's heart", 'SQU\\/D, uk', 'st. louis, missouri', 'Co
lumbus, Ohio', 'garbage disposal bc im trash', 'Free and democratic Britain', 'Minnesota', 'Muskegon, MI',
'TO YOUR CHOSEN DESTINATION', ' ?becky?chloe?', '@ArgentinaLiars ?| willbradley', 'twitch.tv/dgn_esport
s', 'Hull ma', 'Limerick, Ireland.', 'she/her ', 'North Vernon, IN', 'UAE', 'Lansing, MI USA', 'NJ, Amerik
kka', 'Bowling Green, Ky', 'SweizyLand', 'Aragua', 'Bentonville, Arkansas', 'Denver, Co', 'Love Reiss', 'w
ww.dorsavi.com', 'The best side of middlesbrough', 'Hounslow, London', 'Queens, New York', 'Nr Great Misse
nden, Bucks', 'in your dreams', 'New York City & Mpls/St. Paul', 'Karma ', 'festac,Lagos,Nigeria', '$$$',
'Hodesto, Cuntlifornia', 'Frankfurt, Germany', 'Jon Bellion | Luke Christopher', 'Cardiff', "ZIKKO'S HQ",
'Nap town', 'hayling ', 'North East America', 'manchester, UK', 'PG', 'Digitosphere', 'Michigan ?? South C
arolina', 'St Marys, OH', 'MS', 'INEQUITY IS INJUSTICE || NJ', 'Budgam kashmir', 'In your hearts and min
ds', '55 Wall Street', 'A glass case of emotions', 'Comox Valley Courtenay, BC ', 'Uganda l Africa l Worl
dwide. ', 'Saint-Petersburg', 'eileenborut,webster, tx', 'Chicago, Illinois ', 'Sudbury', 'USA,Washingto
n,Seattle', 'Sea of Green', 'Stone Mountain, GA', 'ID', 'Bharat.', "In ma daddy nd mummy's. Hart..", 'Neth
erverse', 'Cardiff/London/NYC/Warwick', 'Ur BaQ', 'Austin,Tx', 'Lake Hefner Bike Trail', 'That place', 'Bi
deford, England', 'Hubli, Karnataka', 'lagos', 'Bedford', 'Egypt', 'Tampa Bay. Fire the cannons', 'Sagina
w, Mich.', 'ÌÏT: 41.106046,-80.657836', 'teen top ? jongsuk ? exo ? bts', 'pa', 'Under Ya Skin', 'ÌÏT: 39.
168519,-119.766123', 'Sunshine Coast of BC', 'Blackfield, England', 'leicester', 'Down, down, baby...,S
C.', 'Vernal, Utah', 'Vancouver ', 'cyberspace', 'Instagram:marissatunis', 'Greer, SC', 'scarborough, onta
rio', 'Wales Cymru', 'Montreal,QC', 'The Court of Public Opinion.', 'Michigander, USA', 'Buzz City ', 'The
Forbidden Forest', 'Tuksa, OK', 'Pleasantville, NY', 'Gornal - England', 'The Real World', 'inland empire
ca', 'Stuck In Traffic', 'SHIBUYA TOKYO JAPAN', 'Beal Feirste Northern Ireland', 'Hawkes Bay, New Zealan

d', 'North Tonawanda, NY', 'Where is my mind?', 'Blog', '130515 \x89Ûç Gallavich.', 'Derbyshire, England', '1/11 numberjacks squad', 'Sawangan 1, Central Java', 'London/Lagos', 'faisalabad', 'she/her? \x89Ûç tr \x89Ûç jordan', 'Windy City, Land of the Snakes', 'West Powelton, Philadelphia', 'ig: j.nessaa', 'GroÌÙdeutsches Reich', 'Oman ', 'House of the Rising Sun', 'AUSTRALIA', 'earth', 'Liverpool, England', 'Haifa, Israel', 'Phoenix, AZ ', '#HDYNATION', 'w/ @_ridabot, probably', 'ID where potatoes grow', 'Denver Colorado', 'toronto \x89Ûç unicorn island ', 'Virginia Beach, VA USA', 'Edmonton', 'Northern Scandinavia', '#ViewsAreMyOwn #IBackTheBlue', 'Winchester, UK', 'HYPE RESSHA HYPE RESSHA', 'Greater Boston Area', 'Indianapolis', 'Nepal', 'BHUBANESWAR', 'foothills', 'Dirty Bay, CA', 'somewhere over the rainbow', 'Missoula, MT', 'seattle', 'vine: woah stelena ', 'Manchester,England', 'UK & Oversees', 'bang bang bang', 'uchiha', 'Beijing,China', 'Tuscumbia, AL', 'Blue Nation Naija', 'Honeymoon Tour 03.26.15?', '?x??p = ?/2', 'North Coast of O-H-I-O', 'Homewood, PA', 'Ittihad .f.c', 'New England ????', 'the Great White North', 'San Francisco, Los Angeles', 'Somewhere on the Earth', 'Playing: HL2: EP1, Dust: AET', 'Swaggdad, Irock', 'Tampa, Florida', 'Davis', 'Ventura, CA', 'chi town   ', 'Louisville Kentucky', 'Berkeley, CA 94703, USA ? ', 'Muscat', 'Minneapolis St. Paul Minnesota', 'Mighty Tempe, Arizona', 'Istanbul, TÌ_rkiye', 'behind them oranges', 'The Land of Make Believe', 'Tampa FL', 'Milan, Lombardy', '(SoCal)', 'Colwyn Bay, Wales', 'WARNING: I TWEET NSFW AND NSFL', 'Mechanicsville, VA', '   Waiheke Island', "Don't stalk me- thanks", 'Northeast United States ', 'OUTERSPACE', '   FLA', 'MNL, Philippines', 'they/them or she/her', '#DCjacobin', 'Jeddah, Makkah Al Mukarrama', 'Lufkin.Texas,USA', 'Rexburg ID', 'ocsf', '301|804', 'Montana/North Carolina', 'North Bellmore, NY', 'The Interwebs!', 'Downtown New York', 'gateway regional hs', 'RELEASE THE REIS ', 'Baroda', 'å◎hicago', '28.709672,-97.376514', 'kyiv. ukraine', 'CANADA ', 'Rhyming', 'Stgo, Chile', 'Nashville, Music City, USA', 'a pool of my own tears', 'edmonton', 'sending rude things to heather', 'portsmouth ', 'Central #Ohio', 'Leeds ', "N 51å¡25' 0'' / W 0å¡45' 0''", '?????????, ?????-?????????', 'Fredericksburg, VA', 'Motown, WV', 'downriver.', 'Bodmin, Cornwall', 'SP - Brasil #1', ' Elizabeth, NJ', 'Atlanta, GA (kind of)', 'Highlands Ranch CO', 'Portland, Oregon, USA', 'Rice Lake, WI/Toronto, ON', 'Bieber Fever UK', 'Capitol Hill, Washington', 'New york', 'Twitter', 'Boca Raton, FL', 'Crescent Moon w/ Wook', 'O-Town , The left end', 'Lagos,nigeria', 'Pretoria, South Africa', 'In A Multi-Fandom', 'Austin Texas, Ontario Canada', 'The New Way To Surf!', 'Behavioral Analysis Unit (BAU)', 'Lakerland', 'Washington D.C. / Istanbul', 'NJ, USA', 'Sheffield.?', 'dam squad 4 lyf', 'Swansea', 'HUSTON TEXAS 1-844-360-WISE', 'Hiding in a cardboard box.', 'Wolverhampton ', 'Windsor ON Canada', 'Arizona, USA', 'San Antonio', '1937 Germany ', 'Los Angeles/ Las Vegas/ Boston', 'Ankeny', 'Kansas ', 'LanÌ¼s', 'Upper Dicker, England', 'Bangalore,India', 'Sittwe', 'phx az', 'St. Petersburg. Fla.', 'Mii Facebook', 'Bending the elements.', 'YYC', 'Raleigh', 'VA', 'Out and about ', 'West Michigan', '#NAME?', 'Lawn, PA', 'Rhode Island, USA', 'charlotte, nc', 'San Jose State University ', 'The Open road!', 'North Korea', 'Piedmont, CA', 'All Round The World', 'threeonefive. ', 'Shanatic/GGian/SRKian', 'Europa', 'Pasadena, TX', 'Baker, Louisiana', 'south wales valleys', 'MIA', 'The Woodlands, TX', 'MakoHaru Hell', 'University of Alabama', 'Berkeley, CA', 'South Wairarapa/Wellington ', '?orth, SG. ', 'Beaumont, Texas', 'Planet Earth (mainly) #Neuland', '??????????', 'my USA, åµ???? ?aÌÙ? ', 'Trujillo, Peru', 'Rock Hill , SC ', 'Palacio, Madrid', 'lbtidronegirlshell ', '? In your head ?', 'Bath, England', ';)', 'Runcorn', 'Rochester NY', 'Fargo, ND', 'Fort Worth, TX', 'lake worth, Fl', 'Cambridge ', '1996???????????', 'Kearney (area) Nebraska', 'Everywhere.. ', 'Nanaimo, BC', 'Philadelphia & Worldwide', 'Rapid City & The Black Hills', 'Udaipur', 'Tampa Bay', 'Duo lane', 'Stockport, Uk', 'The Land of Pleasant Living', 'YouTube Channel', 'Harvard Square, Cambridge, MA'}

¿Are all keywords from test set also in training set?

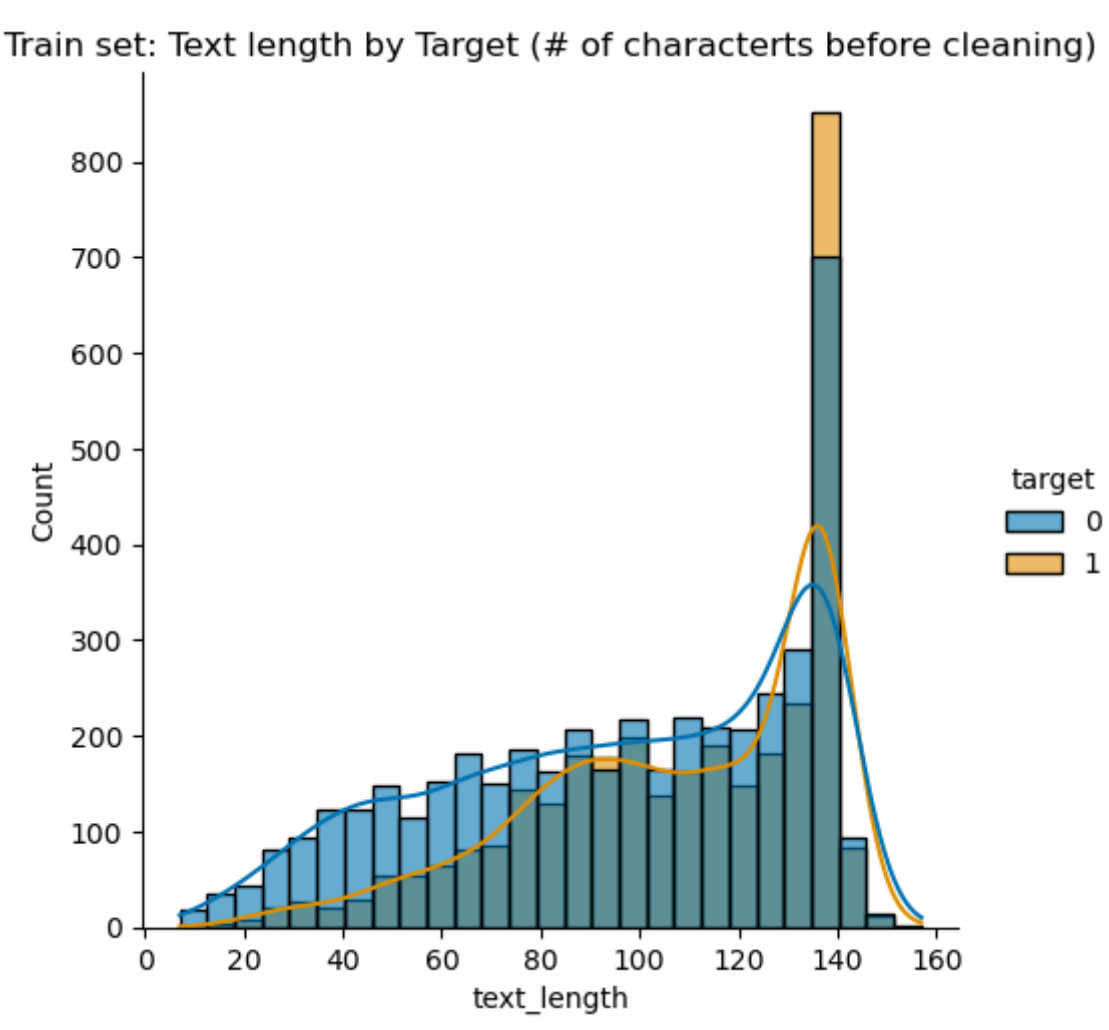
In [14]:

```
print('Keywords in test set not found in train set:')
s =  set(df_test['keyword'])
s1 = s- set(df_train['keyword'])
print('There are {} keywords in test set not found in train set. This means {:.0%} of the test set keywords are not in train set'.format(len(s1),len(s1)/len(s)))

print('\nlist of such keywords:')
print(s1)
```

```
Keywords in test set not found in train set:
There are 0 keywords in test set not found in train set. This means 0% of the test set keywords are not in train set

list of such keywords:
set()
```

Distributions

In [15]:
```
df_train['text_length'] = df_train['text'].apply(len)
df_test['text_length'] = df_test['text'].apply(len)

# fig, ax = plt.subplots(figsize=(8, 5))
ax2 = sns.displot(
    data = df_train,
    x = 'text_length',
    hue = 'target',
    palette = 'colorblind',
    kde = True,
    alpha = 0.6,
    legend = True,
    ).set(
        title = 'Train set: Text length by Target (# of characterts before cleaning)');
```

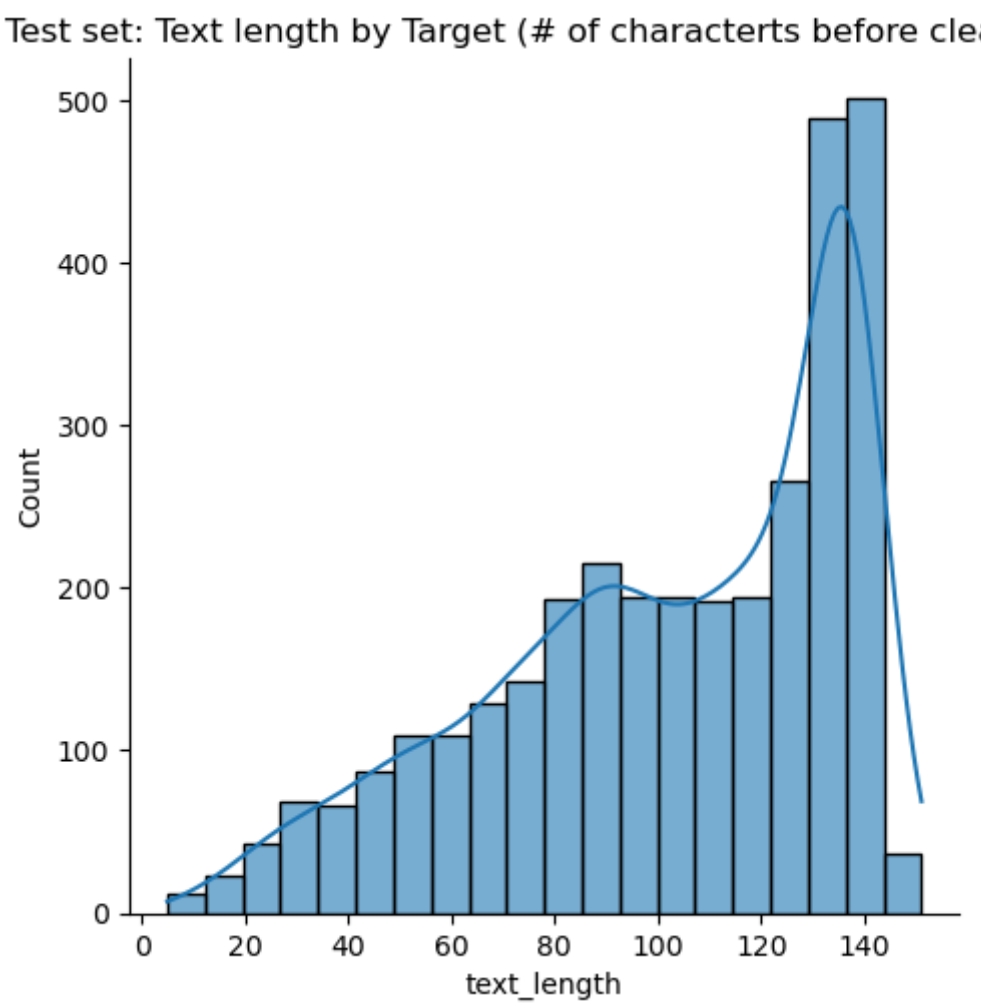


In [16]:
```
df_train.describe()
```

Out[16]:

	id	target	text_length
count	7613.000000	7613.00000	7613.000000
mean	5441.934848	0.42966	101.037436
std	3137.116090	0.49506	33.781325
min	1.000000	0.00000	7.000000
25%	2734.000000	0.00000	78.000000
50%	5408.000000	0.00000	107.000000
75%	8146.000000	1.00000	133.000000
max	10873.000000	1.00000	157.000000

In [17]:
```
ax2 = sns.displot(
    data = df_test,
    x = 'text_length',
#     hue = 'target',
#     palette = 'colorblind',
    kde = True,
    alpha = 0.6,
    legend = True,
    ).set(
        title = 'Test set: Text length by Target (# of characterts before cleaning)');
```



In [18]:
```
df_test.describe()
```

Out[18]:

	id	text_length
count	3263.000000	3263.000000
mean	5427.152927	102.108183
std	3146.427221	33.972158
min	0.000000	5.000000
25%	2683.000000	78.000000
50%	5500.000000	109.000000
75%	8176.000000	134.000000
max	10875.000000	151.000000

¿Is location or keyword correlated with target?

Let us first replace unknown locations and keywords with "unknown".

Next, let us see if knowing the location or keyword gives us information about the target. We do this by examining the conditional probability p(target|location) or p(target|keyword) for the top values. From the histograms, we expect to have similar bar heights for target=0 and target=1 if there is no correlation.

In [19]:
```
# Check if 'unknown' is already in dataframe
print('Is the word "unknown" already a location?:','unknown' in set(df_train['location']))
print('Is the word "unknown" already a keyword?:','unknown' in set(df_train['keyword']))
```

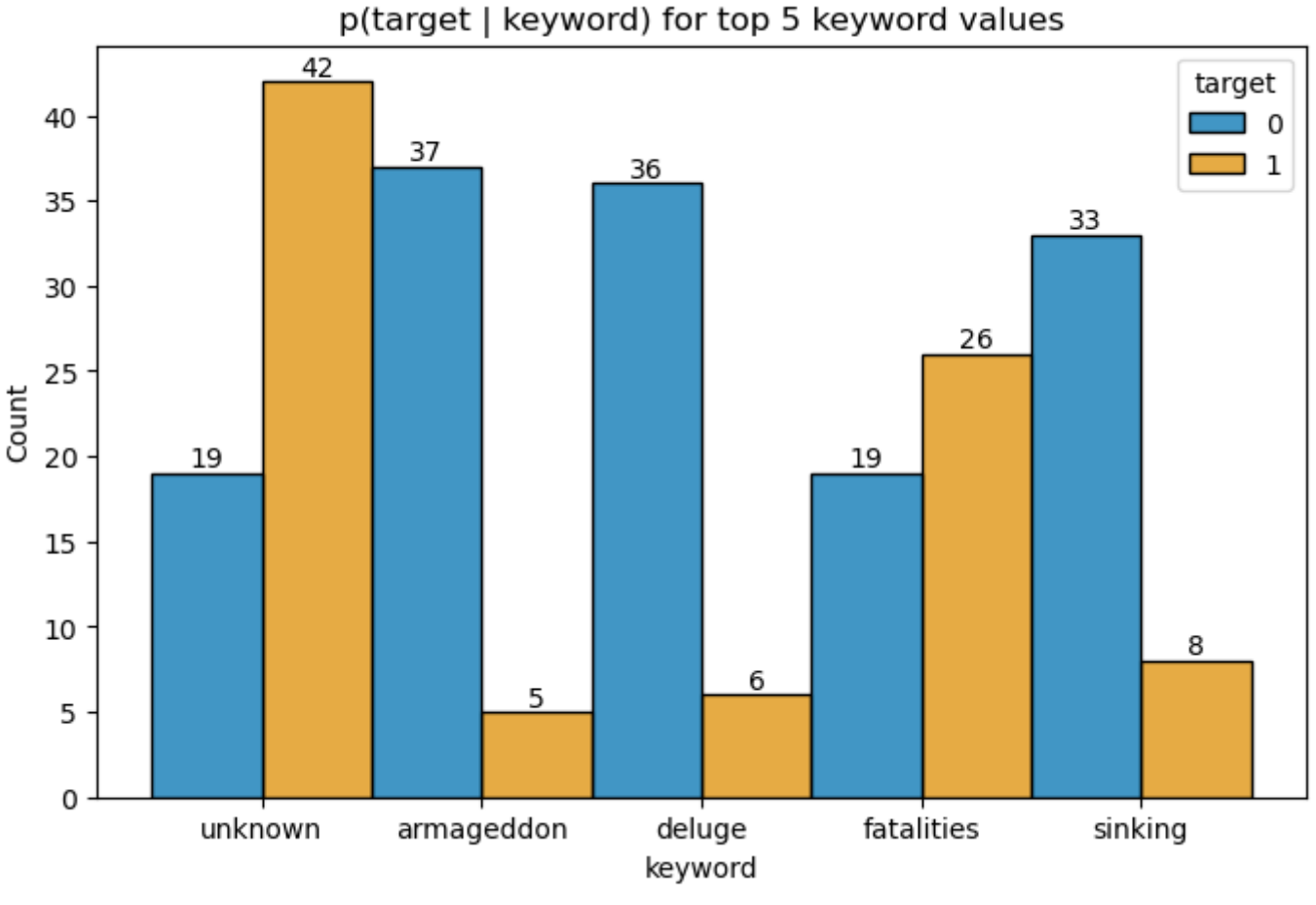
```
Is the word "unknown" already a location?: False
Is the word "unknown" already a keyword?: False
```

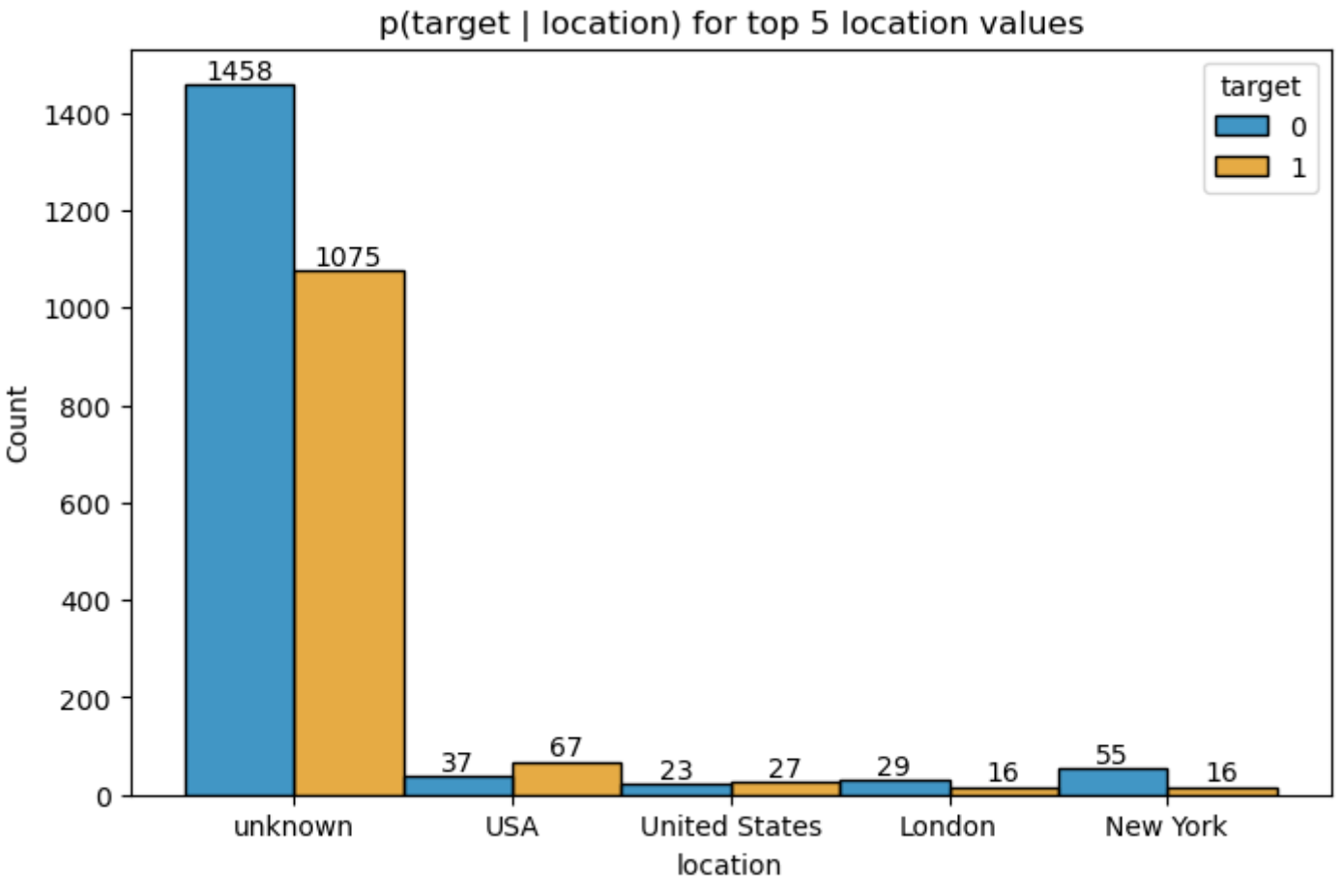
In [20]:
```
def impute_unknown(df):
    df['location'].fillna('unknown',inplace = True)
    df['keyword'].fillna('unknown',inplace = True)
impute_unknown(df_train)
impute_unknown(df_test)
print('Imputing the word "unkown" has been imputed for NaNs in location and kewyword columns')
```

```
Imputing the word "unkown" has been imputed for NaNs in location and kewyword columns
```

In [21]:

```
def hist_plot_top_vals(df,col_top,col_hue='target',ntop=5):
    top_n_vals = df[col_top].value_counts()[0:ntop].index
    df_top = df[df[col_top].isin(top_n_vals)]
    fig, ax = plt.subplots(figsize=(8, 5))
    title = 'p({} | {}) for top {} {} values'.format(
        col_hue, col_top, ntop, col_top)
    ax2 = sns.histplot(
        data = df_top,
        x = col_top,
        hue = col_hue,
        palette = 'colorblind',
        legend = True,
        multiple = 'dodge',
        ).set(
            title = title);
    add_histogram_values(ax)
#-----------------------------------------------------------------------------------------------
hist_plot_top_vals(df_train,'keyword')
hist_plot_top_vals(df_train,'location')
```





Conclusions from data inspection

- There are >7000 samples to train from
- Without precprocessing text, we have from 5 to 157 characters. Our model should be able to work with a variable number of inputs.

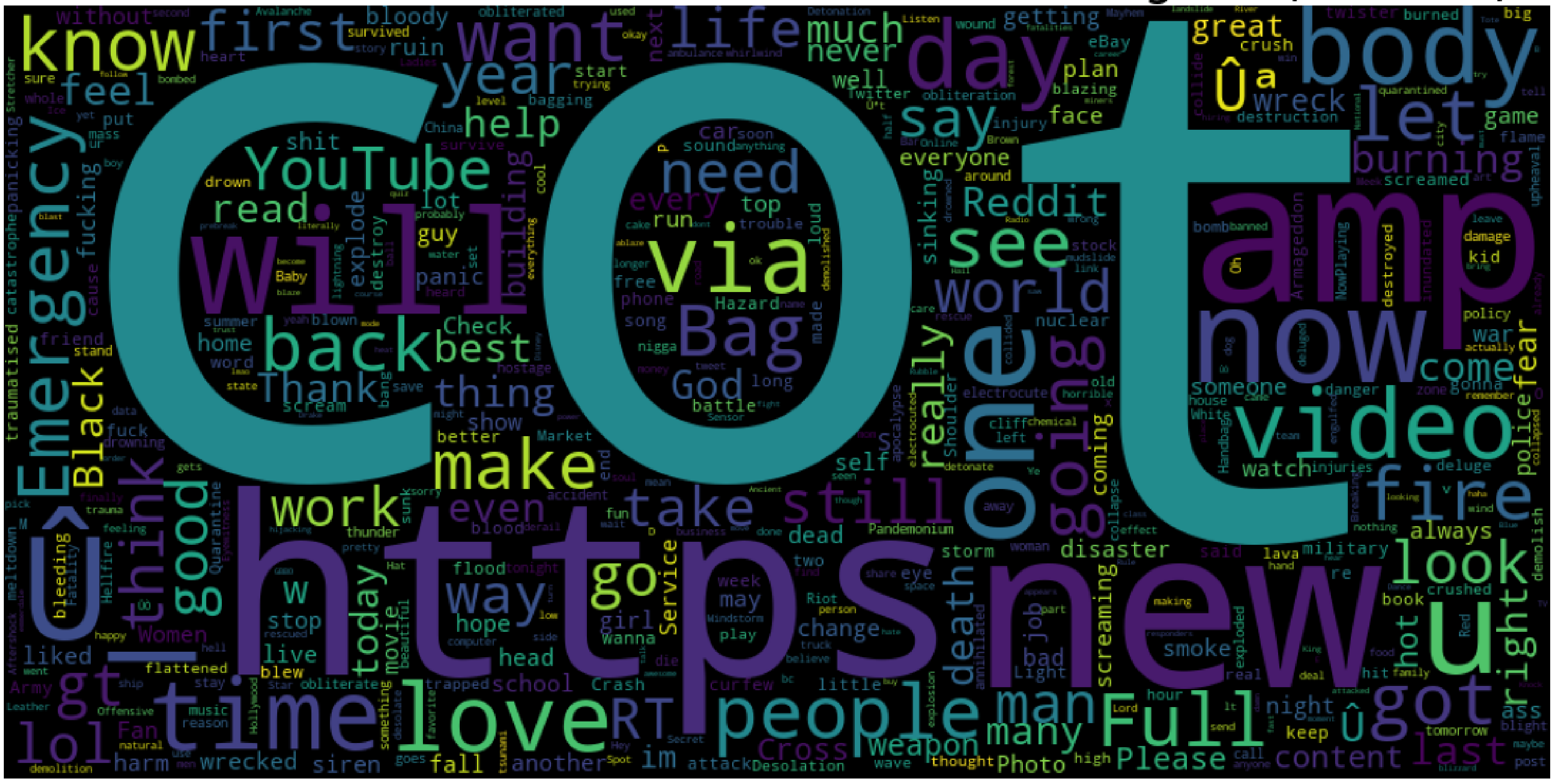
Main words per category

In [22]:
```
from wordcloud import WordCloud
```

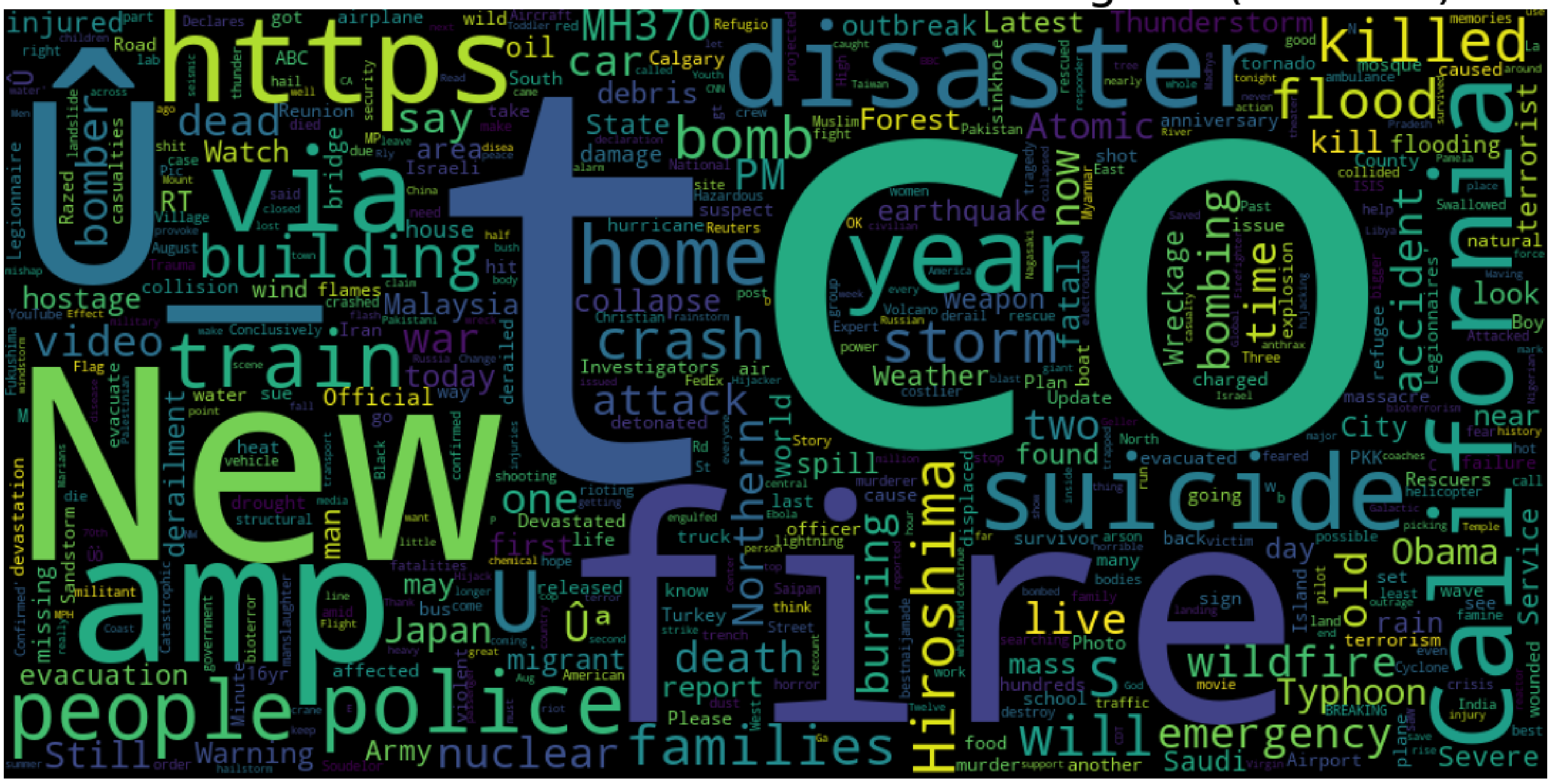
In [23]:

```
def plot_word_cloud(df,colname,target_filter=None,title='word could',num_words=500):
    if target_filter is not None:
        texts = df.loc[df['target']==target_filter,colname]
    else:
        texts = df[colname]

    one_text = " ".join(tweet for tweet in texts)
    wc = WordCloud(max_words = num_words , width = 1000 , height = 500,
              collocations=False).generate(one_text)
    plt.figure(figsize = (14,14))
    plt.axis("off")
    plt.imshow(wc)
    plt.title(title,fontsize=25)
plot_word_cloud(df_train,'text',0,'Word cloud for non disaster tweets in training set (label=0)')
plot_word_cloud(df_train,'text',1,'Word cloud for disaster tweets in training set (label=1)')
plot_word_cloud(df_train,'location',0,'Word cloud for non disaster locations in training set (label=0)',200)
plot_word_cloud(df_train,'location',1,'Word cloud for disaster locations in training set (label=1)',200)
plot_word_cloud(df_train,'keyword',0,'Word cloud for non disaster keywords in training set (label=0)',200)
plot_word_cloud(df_train,'keyword',1,'Word cloud for disaster keywords in training set (label=1)',200)
```

Word cloud for non disaster tweets in training set (label=0)

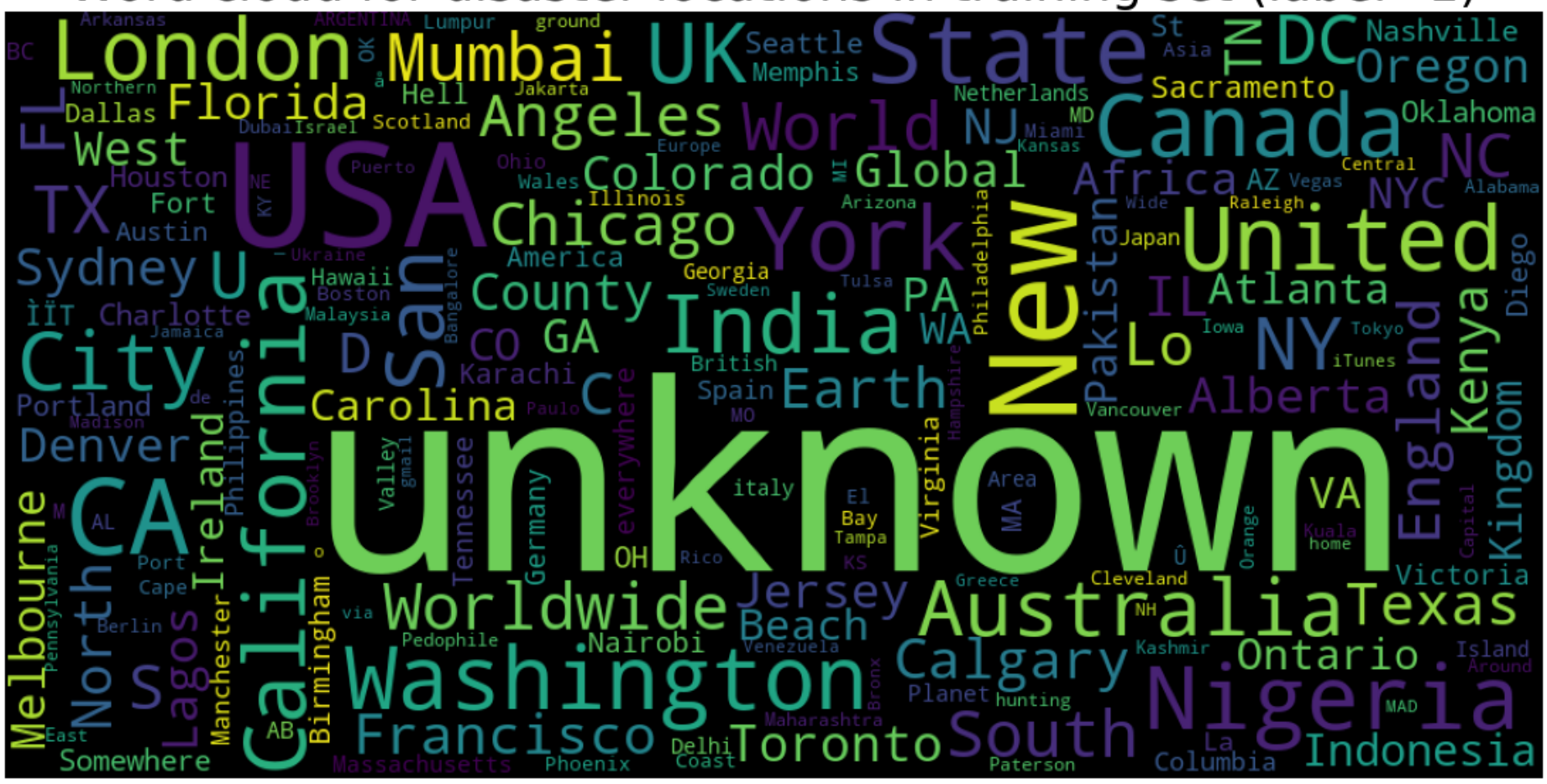


Word cloud for disaster tweets in training set (label=1)

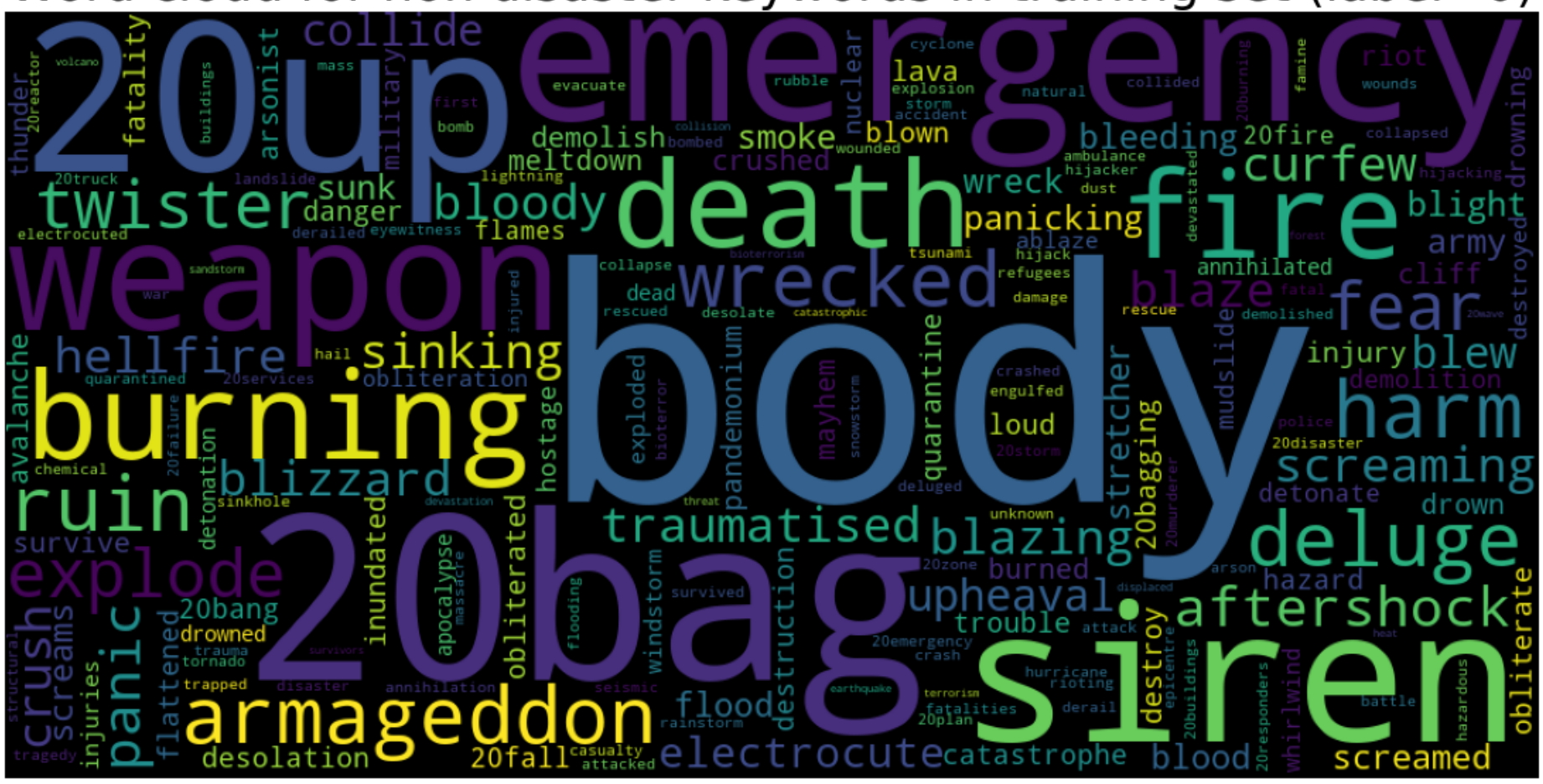


Word cloud for non disaster locations in training set (label=0)

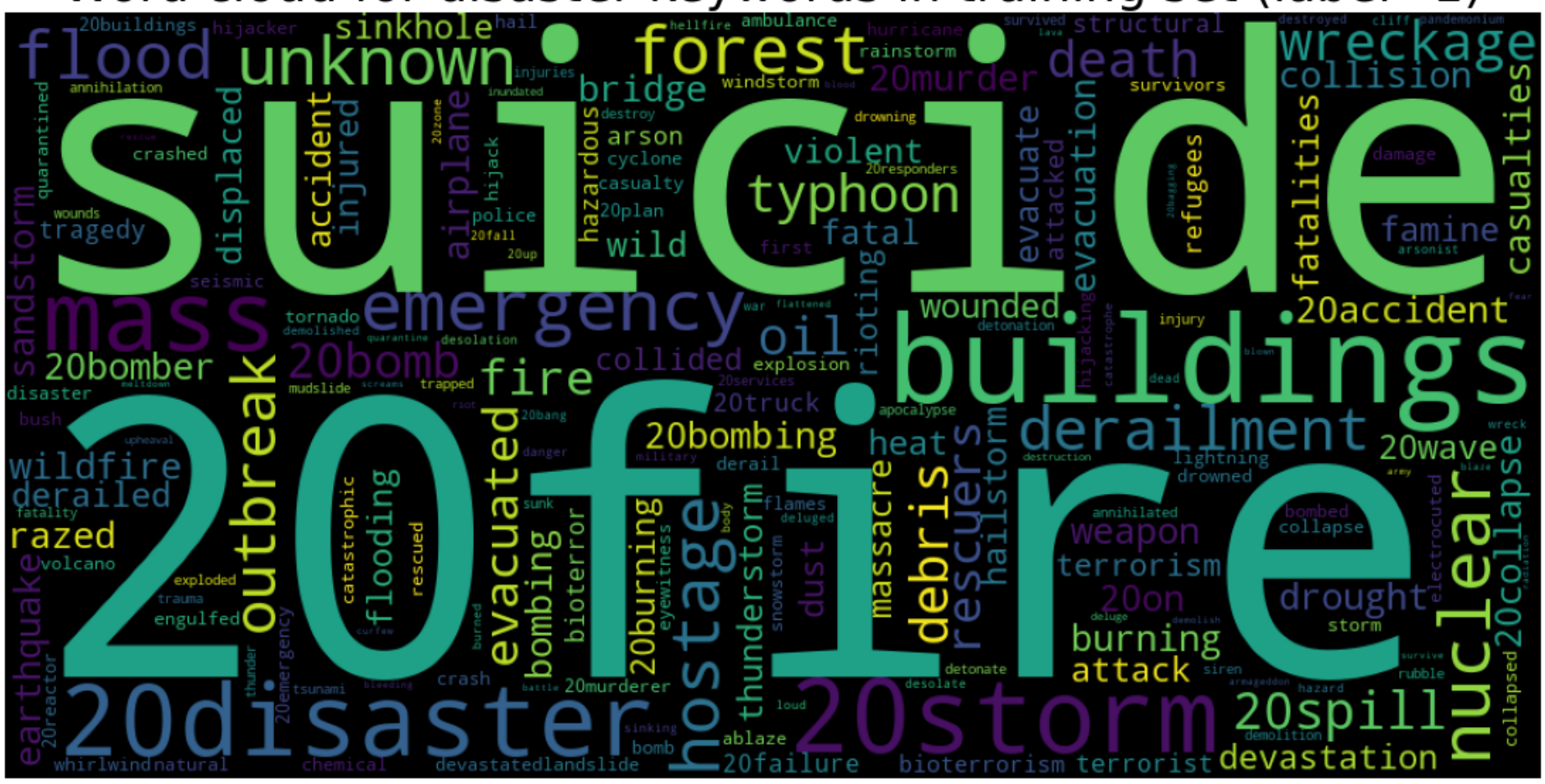
Word cloud for disaster locations in training set (label=1)



Word cloud for non disaster keywords in training set (label=0)



Word cloud for disaster keywords in training set (label=1)



3.2 Clean text data

Clean Tweets

In order to clean tweets I have adapted the original Ruby code for text cleaning (https://nlp.stanford.edu/projects/glove/preprocess-twitter.rb) as much as I could. Some parts I modified were:

- Find user name with regex (https://stackoverflow.com/questions/2304632/regex-for-twitter-usernamehttps://stackoverflow.com/questions/2304632/regex-for-twitter-username)
- Find URL (https://stackoverflow.com/questions/11331982/how-to-remove-any-url-within-a-string-in-python)

The cleaning procedure for text involves regex expressios for dealing with:

- URLs
- User names
- Hashtags
- Emojis,
- Words in capital letters.

In [24]:
```
import re
```

In [25]:

```
def clean_text(text):
    """Apply a list of regex replacements to clean tweets"""

    def hashtag_sub(hashtag):
        """Clean hashtags"""
        hashtag_body = hashtag.group(1) # remove the "#" from string
        if hashtag_body.isupper():
            # Hashtag is all uppercase
            result = "<HASHTAG> "+ hashtag_body + " <ALLCAPS>"
        else:
            # Split hashtag on uppercase letters
            result = "<HASHTAG> "+ ' '.join(re.split(r'(?=[A-Z])', hashtag_body))
            result = re.sub('\s{2,}',' ',result) # remove double whitespaces
        return result

    # Parts of text smileys
    eyes = "[8:=;]"
    nose = "['`\-]?"

    replacements = [
    ("(https?:\/\/)(\s)*(www\.)?(\s)*((\w|\s)+\.)*([\w\-\s]+\/)*([\w\-]+)((\?)?[\w\s]*=\s*[\w\%&]*)*","<URL
>"), # Replace URL
    ("/"," / "), # Force splitting words appended with slashes (once we tokenized the URLs, of course)
    ("(?<=^|(?<=[^a-zA-Z0-9-_\.]))@([A-Za-z]+[A-Za-z0-9-_]+)", "<USER>"), # Replace User name
    (rf"(?i){eyes}{nose}[)d]+|[)d]+{nose}{eyes}", "<SMILE>"), # Replace ':)' and alike
    (rf"(?i){eyes}{nose}p+", "<LOLFACE>"), # Replace ':P' and alike
    (rf"{eyes}{nose}\(+|\)+{nose}{eyes}", "<SADFACE>"), # Replace ':(' and alike
    (rf"{eyes}{nose}[\/|l*]", "<NEUTRALFACE>"), # not sure if this is working
    ("<3","<HEART>"), # replace heart emoji <3
    ("[-+]?[.\d]*[\d]+[:,.\d]*", " <NUMBER> "), #" Replace numbers"
    (r'#(\w+)',hashtag_sub), # Detect and split hashtags
    ("([!?.]){2,}",r"\1 <REPEAT>"),  # Mark punctuation repetitions (eg. "!!!" => "! <REPEAT>"
    (r"\b(\S*?)(.)\2{2,}\b",r"\1\2 <ELONG>"), # Mark elongated words (eg. "wayyyy" => "way <ELONG>"
    (r"(?<!\<)(\b[A-Z]{2,})",lambda m: m[0].lower() + ' <ALLCAPS>') # Detect words in capital letters that do n
ot start with "<" (modified)
    ]
    for old, new in replacements:
        text = re.sub(old, new, text)
    return text

added_tags = ['<URL>',
              '<USER>',
              '<SMILE>',
              '<LOLFACE>',
              '<SADFACE>',
              '<NEUTRALFACE>',
              '<NUMBER>',
              '<HASHTAG>',
              '<ALLCAPS>',
              '<REPEAT>',
              '<ELONG>']

# Show how it words with random samples
print('-'*60,'\n   Random examples of tweets before and after cleaning:\n','-'*60)
for id in np.random.choice(len(df_train), size=10, replace=False):
    text = df_train['text'][id]
    cl = clean_text(text)
    if cl is not text:
        print(text,'\n => ', clean_text(text))
```

```
------------------------------------------------------------
   Random examples of tweets before and after cleaning:
 ------------------------------------------------------------
#breaking #LA Refugio oil spill may have been costlier bigger than projected http://t.co/5ueCmcv2Pk
 =>  <HASHTAG> breaking <HASHTAG> la <ALLCAPS> <ALLCAPS> Refugio oil spill may have been costlier bigger t
han projected <URL>
#hot  Reddit's new content policy goes into effect many horrible subreddits banned or quarantined http://
t.co/nGKrZPza45 #prebreak #best
 =>  <HASHTAG> hot  Reddit's new content policy goes into effect many horrible subreddits banned or quaran
tined <URL> <HASHTAG> prebreak <HASHTAG> best
Nueva favorita: EmergeNCY feat. The Chemical Brothers / My Bits http://t.co/MET4YtZMFB @DeezerColombia
 =>  Nueva favorita: EmergeNCY feat. The Chemical Brothers  /  My Bits <URL> <USER>
11:30BST traffic: A10&gt;Paris A40 Geneva A7 Mons A1 Hamburg A2&gt;Hanover A5 Karlsruhe Gotthard n/b  htt
p://t.co/yoi9tOCxiQ
 =>   <NUMBER> bst <ALLCAPS> traffic: A <NUMBER> &gt<LOLFACE>aris A <NUMBER>  Geneva A <NUMBER>  Mons A <N
UMBER>  Hamburg A <NUMBER> &gt;Hanover A <NUMBER>  Karlsruhe Gotthard n / b  <URL>
What if every 5000 wins in ranked play gave you a special card back.. Would be cool for the long te Û_ htt
p://t.co/vq3yaB2j8N
 =>  What if every  <NUMBER>  wins in ranked play gave you a special card back. <REPEAT> Would be cool for
the long te Û_ <URL>
Wreckage 'Conclusively Confirmed' as From MH370: Malaysia PM: Investigators and the families of those who
were... http://t.co/KuKmAL605a
 =>  Wreckage 'Conclusively Confirmed' as From mh <ALLCAPS> <NUMBER>  Malaysia pm <ALLCAPS>: Investigators
and the families of those who were. <REPEAT> <URL>
MH370: debris found on reunion island. ?? #sad #tragedy #innocent #crash #mh370
 =>  mh <ALLCAPS> <NUMBER>  debris found on reunion island. ? <REPEAT> <HASHTAG> sad <HASHTAG> tragedy <HA
SHTAG> innocent <HASHTAG> crash <HASHTAG> mh <NUMBER>
It's Time To Reduce Gun Deaths http://t.co/ilADQEBxPn
 =>  It's Time To Reduce Gun Deaths <URL>
@Eric_Tsunami worry about yourself
 =>  <USER> worry about yourself
```

In [26]:

```
# Apply text cleaning function to df_train and df_test
df_train["clean_text"] = df_train["text"].apply(clean_text)
df_test["clean_text"] = df_test["text"].apply(clean_text)
```

3.3 Tokenize words

The use of keras Tokenizer and word glove word embeddings used here is inspired by this work (https://towardsdatascience.com/recurrent-neural-networks-by-example-in-python-ffd204f99470). The main differences are:

- We will use glove vectors trained on twitter. A similar cleaning method was implemented for consistency.
- We will predicta binary output, not the next word
- The sequence of words taken as input is sometimes too short, se we should be able to pass blank words.
- When training the tokenizer, let us add the word '<blank>' to the vocabulary.

We will use Keras Tokenizer to split words. Each word will be mapped to an integer value in the `word_idx` dictionary (the reverse mapping is available as `idx_word`). The RNN will be fed with a sequence of integers representing words. Suppose we choose

In [27]:

```
from keras.preprocessing.text import Tokenizer

def preprocess(df,tokenizer):
    """ Tokenize clean_text column in df """
    df['sequences'] = tokenizer.texts_to_sequences(df['clean_text'])
    df['seq_len'] = df['sequences'].apply(len)


# Create tokenizer
tokenizer = Tokenizer(num_words=None,
                      filters = '\t\n', # These are included in our embedding dict: '#$%&/()*+-<>=[\\]^_}`{|}'
                      lower = True,
                      split = ' ')
# Extract text to train tokenizer
tokenizer_train_strings = df_train['clean_text'].to_list()
tokenizer_train_strings.append('<blank>') # add this word to handle short sequences

# Fit tokenizer
tokenizer.fit_on_texts(tokenizer_train_strings);

# Get mappings of integers to wotds and reb
idx_word = tokenizer.index_word
word_idx = {word:idx for idx, word in idx_word.items()}

num_words = len(idx_word) + 1

# Apply tokenizer to transform data
preprocess(df_train,tokenizer)
preprocess(df_test,tokenizer)

# Show some results
print('-'*70,'\n Here are some random examples of tokenized tweets as integer lists:\n','-'*70)
for id in np.random.choice(len(df_train), size=10, replace=False):
    print(df_train['clean_text'][id])
    print(' =>',df_train['sequences'][id])

# Show stats
print('\n'*3,'Stats for training data:')
display(df_train.describe())
print('\nStats for test data:')
display(df_test.describe())
print('\nAfter tokenizing, {} different words were found in training data'.format(num_words))
```

```
--------------------------------------------------------------------------
 Here are some random examples of tokenized tweets as integer lists:
 --------------------------------------------------------------------------
I wanna tweet a 'niall thx for not making me was to electrocute myself' tweet but I'm scared I'll jinx it
 => [13, 400, 887, 7, 11997, 1912, 15, 40, 585, 42, 31, 9, 631, 11998, 887, 37, 51, 1015, 418, 11999, 24]
i decided to take a break from my emotional destruction to watch tangled then watch desolation of smaug
 => [13, 2674, 9, 169, 7, 1248, 27, 19, 1426, 377, 9, 143, 11108, 154, 143, 572, 10, 1548]
Learning from the Legacy of a Catastrophic Eruption - The New Yorker <URL>
 => [1810, 27, 3, 2271, 10, 7, 455, 2018, 17, 3, 57, 3168, 2]
New Women Handbag Faux Leather Ladies Shoulder Tote Cross Body Bag Large Satchel - Full re Û_ <URL> <URL>
 => [57, 522, 1106, 1230, 799, 932, 529, 1105, 262, 92, 368, 931, 2225, 17, 124, 622, 2, 2]
We walk the plank of a sinking ship
 => [53, 1227, 3, 16581, 10, 7, 329, 835]
<USER> <USER> @_OliviaAnn_ I was looking for you guys on the live stream. I'm guessing the evacuation cost
you the front?
 => [6, 6, 12483, 13, 31, 1029, 15, 18, 765, 16, 3, 221, 5469, 51, 12484, 3, 254, 1848, 18, 3, 12485]
Drake is really body bagging meek
 => [1035, 14, 173, 92, 561, 981]
If I survive tonight. I wouldn't change one thing. ? <REPEAT>
 => [52, 13, 379, 1312, 13, 2363, 526, 70, 3132, 25, 11]
Wen I finally get the girl I want I'm flooding yall wit all pics of us Rs tho
 => [5615, 13, 715, 55, 3, 382, 13, 146, 51, 260, 2596, 3427, 47, 1521, 10, 93, 2219, 1511]
Photo: Bath &amp;amp; Body Works cosmetic bag in periwinkle blue with copper piping along the top and four cor
ners. <REPEAT> <URL>
 => [875, 2560, 39, 92, 3755, 8616, 368, 8, 8617, 701, 20, 8618, 8619, 1406, 3, 215, 12, 833, 8620, 11, 2]



 Stats for training data:
```

	id	target	text_length	seq_len
count	7613.000000	7613.00000	7613.000000	7613.000000
mean	5441.934848	0.42966	101.037436	16.764613
std	3137.116090	0.49506	33.781325	6.745955
min	1.000000	0.00000	7.000000	1.000000
25%	2734.000000	0.00000	78.000000	12.000000
50%	5408.000000	0.00000	107.000000	17.000000
75%	8146.000000	1.00000	133.000000	22.000000
max	10873.000000	1.00000	157.000000	51.000000

```
 Stats for test data:
```

	id	text_length	seq_len
count	3263.000000	3263.000000	3263.000000
mean	5427.152927	102.108183	15.323016
std	3146.427221	33.972158	6.550926
min	0.000000	5.000000	0.000000
25%	2683.000000	78.000000	11.000000
50%	5500.000000	109.000000	15.000000
75%	8176.000000	134.000000	20.000000
max	10875.000000	151.000000	55.000000

```
 After tokenizing, 18829 different words were found in training data
```

In [28]:
```
blank_id = word_idx['<blank>']
print('Missing words will be mapped to id: ',blank_id,'<blank>')
num_words+=1 # update number of words
```

```
Missing words will be mapped to id:  18828 <blank>
```

Remarks

- For some tokenizations, the tweets may become empty. We should be able to produce a sequence of blank words.

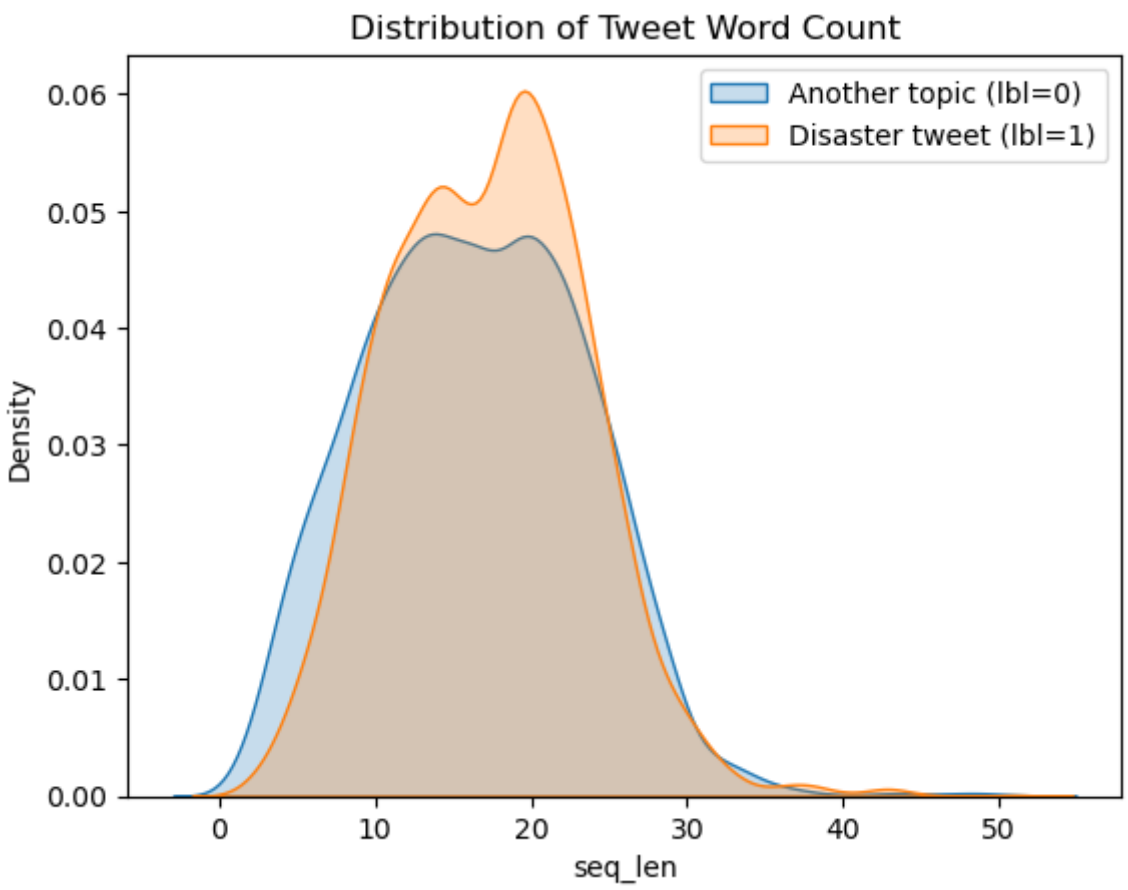
In [29]:
```
print('Most commong words:')
[print(id+1,idx_word[id+1]) for id in range(10)];
```

```
Most commong words:
1 <allcaps>
2 <url>
3 the
4 <hashtag>
5 <number>
6 <user>
7 a
8 in
9 to
10 of
```

In [30]:
```
max_words_per_text = df_train['seq_len'].max()
print('Let us try a model with a maximum number of {} words'.format(max_words_per_text))
```

```
Let us try a model with a maximum number of 51 words
```

In [31]:
```
# create graphs
sns.kdeplot(df_train['seq_len'][df_train['target'] == 0], fill=True, label = 'Another topic (lbl=0)');
sns.kdeplot(df_train['seq_len'][df_train['target'] == 1], fill=True, label = 'Disaster tweet (lbl=1)');
# set title and plot
plt.title('Distribution of Tweet Word Count')
plt.legend();
plt.show()
```



Because the method we used for tokenizing converts the text to lowercase, let us update the list of modify the list of added tags. Also, let us include the word <blank> in our list of added tags

In [32]:
```
added_tags = [tag.lower() for tag in added_tags]
if '<blank>' not in added_tags:
    added_tags.append('<blank>')
added_tags
```

Out[32]:
```
['<url>',
 '<user>',
 '<smile>',
 '<lolface>',
 '<sadface>',
 '<neutralface>',
 '<number>',
 '<hashtag>',
 '<allcaps>',
 '<repeat>',
 '<elong>',
 '<blank>']
```

3.4 Use Glove to get a pretrained embedding matrix for tweets

The University of Stanford had published glove word embeddings trained on 2B tweets (https://nlp.stanford.edu/projects/glove/). We will use those word embeddings in our RNN model.

Notice that a similar cleaning procedure has been implemented here, where tags such as "<USER>" are added.

In [33]:
```
# Load word embeddings from Stanford glove trained on twitter data
embeddings_filename = '/kaggle/input/glove-twitter-pickles-27b-25d-50d-100d-200d/glove.twitter.27B.100d.pkl'
print('Loading word embeddings from Glovec trained on twitter data')
with open(embeddings_filename, 'rb') as f:
    embeddings = pickle.load(f)
embed_dim = len(embeddings['the'])
print('{} word embedding vectors of dimension {} were loaded'.format(len(embeddings),embed_dim))
```

```
Loading word embeddings from Glovec trained on twitter data
1193514 word embedding vectors of dimension 100 were loaded
```

Check tags

Let us now verify that the following special tags we included during text cleaning are also available in the word embeddings dictionary.

In [34]:

```
# Verify that the following words are included in the word embeddings dictionary:
{tag.lower(): embeddings.get(tag.lower(),'not found') for tag in added_tags}
```

Out[34]:


```
{'<url>': array([ 0.33292  , -0.54203  , -0.11322  , -0.27659  , -0.026186 ,
        -0.82087  , -0.21403  , -0.16494  ,  0.25512  ,  0.73279  ,
        -0.28273  ,  0.30562  , -2.3428   ,  0.57922  ,  0.28763  ,
        -0.99739  , -0.092065 , -0.38625  , -0.056712 ,  0.22435  ,
        -0.30217  ,  0.092644 ,  0.23267  ,  0.053841 ,  0.5248   ,
        -1.4725   , -0.21217  ,  0.60497  ,  1.0452   ,  0.0067332,
        -0.50481  , -1.1028   , -1.1451   ,  0.3235   ,  0.89579  ,
        -0.42413  , -0.17208  , -0.086374 , -0.62347  , -0.25032  ,
        -0.91658  ,  0.17972  , -0.015353 ,  0.52888  ,  1.3924   ,
         0.34129  ,  0.36661  ,  0.052747 , -0.3025   , -1.0406   ,
        -0.31652  , -1.1544   , -0.42162  , -0.17128  ,  0.89121  ,
         0.19002  ,  0.080897 ,  0.41373  , -0.094438 ,  0.18039  ,
         0.54055  , -0.98818  , -0.23105  , -0.36734  ,  0.21701  ,
        -0.27166  , -0.20397  ,  0.48816  ,  0.28361  ,  0.078591 ,
         0.6951   , -0.18313  ,  0.7354   ,  0.10051  , -0.49952  ,
        -0.88482  , -1.4918   ,  0.91632  , -0.24593  , -0.69987  ,
         1.2849   ,  0.13556  ,  0.21557  , -0.36722  ,  0.70076  ,
         0.0094641, -0.53055  ,  0.75533  , -0.09199  , -0.30924  ,
        -0.1522   , -0.26417  ,  0.065998 , -0.93542  ,  0.69566  ,
        -0.31284  ,  0.36152  , -0.11462  ,  1.2662   , -0.39507  ],
       dtype=float32),
 '<user>': array([ 0.63006  ,  0.65177  ,  0.25545  ,  0.018593 ,  0.043094 ,
         0.047194 ,  0.23218  ,  0.11613  ,  0.17371  ,  0.40487  ,
         0.022524 , -0.076731 , -2.2911   ,  0.094127 ,  0.43293  ,
         0.041801 ,  0.063175 , -0.64486  , -0.43657  ,  0.024114 ,
        -0.082989 ,  0.21686  , -0.13462  , -0.22336  ,  0.39436  ,
        -2.1724   , -0.39544  ,  0.16536  ,  0.39438  , -0.35182  ,
        -0.14996  ,  0.10502  , -0.45937  ,  0.27729  ,  0.8924   ,
        -0.042313 , -0.009345 ,  0.55017  ,  0.095521 ,  0.070504 ,
        -1.1781   ,  0.013723 ,  0.17742  ,  0.74142  ,  0.17716  ,
         0.038468 , -0.31684  ,  0.08941  ,  0.20557  , -0.34328  ,
        -0.64303  , -0.878    , -0.16293  , -0.055925 ,  0.33898  ,
         0.60664  , -0.2774   ,  0.33626  ,  0.21603  , -0.11051  ,
         0.0058673, -0.64757  , -0.068222 , -0.77414  ,  0.13911  ,
        -0.15851  , -0.61885  , -0.10192  , -0.47     ,  0.19787  ,
         0.42175  , -0.18458  ,  0.080581 , -0.22545  , -0.065129 ,
        -0.15328  ,  0.087726 , -0.18817  , -0.08371  ,  0.21779  ,
         0.97899  ,  0.1092   ,  0.022705 , -0.078234 ,  0.15595  ,
         0.083105 , -0.6824   ,  0.57469  , -0.19942  ,  0.50566  ,
        -0.18277  ,  0.37721  , -0.12514  , -0.42821  , -0.81075  ,
        -0.39326  , -0.17386  ,  0.55096  ,  0.64706  , -0.6093   ],
       dtype=float32),
 '<smile>': array([-0.10517 ,  0.040971, -0.21829 ,  0.32066 ,  0.37484 , -0.095511,
        -0.35746 ,  0.44272 , -0.6136  ,  0.6219  , -0.31894 ,  0.14336 ,
        -2.3681  , -0.2219  , -0.13295 ,  0.093323,  0.43702 , -0.039794,
        -1.2155  ,  0.62244 , -0.019664,  0.31255 , -0.44484 ,  0.23797 ,
         0.10417 , -1.8545  , -0.31294 ,  0.029292,  0.8802  , -0.40601 ,
        -0.4129  ,  0.046448, -0.74438 ,  0.54033 ,  1.1012  ,  0.37489 ,
         0.83674 ,  0.31185 , -0.046693, -0.36435 , -1.5303  ,  0.12056 ,
         0.30849 ,  1.0116  ,  0.55789 , -0.096525, -0.47763 , -0.37219 ,
         0.14489 , -0.13876 , -1.061   ,  0.18065 ,  0.34758 , -0.19935 ,
         0.46133 , -0.12942 ,  0.032383,  0.2349  ,  0.68034 ,  0.24369 ,
         0.074733, -0.61693 ,  0.71277 , -0.10145 ,  0.24318 ,  0.088283,
        -1.1927  , -0.6972  , -0.16482 ,  0.13287 ,  0.12679 ,  0.32985 ,
         0.23037 , -0.29643 ,  0.08321 , -0.24909 , -0.5865  ,  0.19051 ,
         0.17579 ,  0.16169 ,  0.77838 ,  0.29354 , -0.021284, -0.0446  ,
         0.94447 , -0.14804 , -0.013783,  0.10019 , -0.14515 , -0.014627,
        -0.35524 , -0.49568 ,  0.28256 ,  0.018774, -0.56505 , -1.4321  ,
         0.45923 , -0.30029 ,  0.24039 , -0.61567 ], dtype=float32),
 '<lolface>': array([ 3.0474e-01,  1.6262e-01, -4.9555e-02,  6.0789e-01,  1.0302e-01,
        -2.6600e-01, -5.1048e-01,  5.1823e-01,  3.2592e-01,  1.1673e+00,
        -3.4397e-01,  1.8624e-01, -2.3911e+00, -5.3897e-02,  6.1946e-01,
```

```
        3.7955e-01,  3.1288e-01, -2.4494e-01, -8.8671e-01,  6.1179e-01,
        1.1424e-01,  2.5152e-01, -4.6366e-02, -7.4213e-02,  4.8907e-01,
       -1.8586e+00, -4.5058e-01, -2.7000e-01,  4.7579e-01, -2.6060e-01,
        1.3911e-01,  4.6296e-01, -3.1068e-01,  1.8103e-01,  1.0816e+00,
       -1.4753e-01,  4.8119e-01,  5.7829e-01,  5.6871e-01, -3.3607e-01,
       -2.2851e+00,  1.8152e-01,  2.7723e-01,  7.1770e-01,  2.5227e-01,
       -4.6089e-01, -1.0334e+00, -1.1434e-01,  1.3947e-01,  1.1658e-01,
       -2.0298e-01,  5.3492e-02,  3.5397e-01, -1.4793e-01,  4.0172e-01,
       -2.0700e-01,  4.5904e-01,  3.1330e-01,  4.8474e-01,  2.4530e-01,
       -2.6958e-01, -2.7581e-01,  5.4761e-01, -2.1307e-01,  5.2283e-01,
        3.9076e-03, -8.6639e-01, -8.0684e-01, -1.3644e-01,  1.1548e+00,
        9.0165e-02,  6.2264e-01, -2.4754e-02, -5.2488e-01, -1.5977e-01,
        2.9119e-01,  1.2625e-01,  7.3378e-01,  4.2280e-01,  1.2157e-01,
        5.8120e-02,  3.0225e-02,  7.4797e-03, -4.3154e-01,  5.0918e-01,
       -3.6598e-01, -2.5798e-01, -7.2860e-02, -2.1596e-01,  9.1094e-02,
       -4.7654e-01,  4.6654e-01, -1.9509e-01,  4.1691e-01, -5.9629e-01,
       -1.1427e+00,  1.0167e-01, -1.0811e-03, -3.8643e-01, -5.2512e-02],
      dtype=float32),
'<sadface>': array([ 2.6377e-01,  1.8511e-03, -5.9860e-02,  6.6940e-01, -6.8349e-02,
       -2.9919e-01,  1.4382e-01,  9.4750e-01,  7.6036e-02,  1.0483e+00,
        2.3552e-01, -1.3251e-01, -1.9145e+00, -5.3988e-02,  1.4995e-01,
        1.9039e-01, -4.3906e-02, -1.5691e-01, -1.5348e+00,  2.3418e-01,
       -3.2457e-01,  2.7490e-01,  3.7731e-02,  8.8474e-01,  7.1045e-01,
       -2.3357e+00, -5.6250e-01, -5.7004e-01,  4.1061e-01,  9.3879e-02,
       -5.8222e-01,  8.9418e-02, -5.3740e-01,  1.9604e-01, -2.8836e-02,
        3.2341e-01,  4.4855e-01, -6.9937e-03,  1.4190e-01,  1.9635e-01,
       -1.7599e+00,  2.8597e-01,  1.5377e-02,  1.4502e-01,  1.5943e-01,
       -6.0169e-02, -4.9848e-01,  6.9717e-02,  4.3245e-01,  5.0008e-01,
       -8.6326e-01,  5.8527e-01,  7.3028e-01,  8.0656e-02,  5.0974e-01,
       -1.0826e-01, -2.6449e-01,  6.2259e-01,  2.3726e-01,  1.9843e-01,
       -1.4528e-01,  7.9233e-02,  8.4865e-01, -7.1887e-01,  3.5394e-01,
       -4.8404e-01, -9.9357e-01, -6.0757e-01, -2.8485e-02,  3.8812e-01,
       -3.0863e-02,  5.2672e-01,  4.7227e-01, -6.4386e-01,  5.9189e-01,
       -2.8705e-01, -4.0863e-01, -1.9282e-01, -2.5404e-01, -1.9800e-01,
        5.5163e-01, -2.3015e-01,  5.4695e-01, -7.4863e-01,  8.8645e-01,
        4.0946e-01,  9.9944e-02,  2.6639e-02,  2.3622e-01, -1.0407e-02,
       -6.1643e-01, -5.6303e-01,  7.3868e-01,  7.7589e-01, -6.2232e-01,
       -1.0046e+00, -1.2030e-01, -1.8292e-01, -2.7916e-03,  7.9693e-02],
      dtype=float32),
'<neutralface>': array([-2.5479e-01,  2.8597e-01, -1.2619e-01,  2.5982e-01,  3.7647e-01,
       -6.7438e-01,  8.6313e-02,  8.8655e-01,  2.1197e-01,  6.9159e-01,
       -3.3638e-01,  3.1030e-01, -1.6839e+00, -2.9957e-01, -2.5205e-01,
       -7.8191e-02,  4.0166e-01,  1.6907e-01, -1.1209e+00,  1.3442e-01,
       -3.4402e-01,  5.8807e-02,  8.0312e-02,  6.7621e-01,  7.0206e-01,
       -1.9263e+00, -5.2007e-01, -1.0018e-01,  6.9666e-01, -7.0083e-01,
       -6.3174e-01, -2.0926e-02,  1.7226e-01,  6.7697e-01,  9.0173e-01,
        2.7986e-01,  4.0568e-01,  2.9203e-01,  2.1259e-01,  3.0557e-01,
       -1.8288e+00,  3.1255e-01,  7.0031e-01,  8.3574e-01, -9.3650e-02,
       -1.2959e-02, -5.3268e-01, -5.5866e-01,  3.1307e-01, -2.5560e-02,
       -1.4545e+00,  1.3959e-01,  7.6462e-01, -5.8686e-01,  1.2002e+00,
        5.4177e-01,  6.7749e-02,  4.0517e-01,  4.8637e-01,  2.9130e-01,
       -3.6712e-01, -7.2702e-01,  6.4753e-01,  1.1999e-01,  3.2981e-01,
       -3.3899e-01, -1.5756e+00, -9.6979e-01, -5.9517e-01,  7.7967e-01,
        2.7689e-01,  5.4691e-01, -1.7624e-01, -4.9526e-01,  4.3184e-01,
       -1.5255e-01,  1.4487e-01, -2.4103e-01,  5.4403e-02, -1.0292e-02,
       -4.9593e-02,  3.1088e-01,  1.5390e-01,  3.1279e-01,  5.6050e-01,
        9.1607e-02,  1.3037e-01,  8.4531e-01,  1.0339e-01,  4.8569e-02,
       -2.2754e-01,  1.5461e-02,  1.9204e-01,  7.7970e-01, -1.2721e-03,
       -1.5250e+00,  2.5172e-01, -5.0175e-01, -1.9778e-01, -1.6067e-01],
      dtype=float32),
'<number>': array([ 0.41436 , -0.066986,  0.47485 , -0.6039  ,  0.13103 ,  0.40182 ,
       -1.0205  ,  0.55811 ,  0.030847,  0.88296 , -0.20015 ,  0.30581 ,
       -2.2959  ,  0.27628 , -0.5745  , -0.65363 ,  0.015981, -0.078754,
```

```
        0.29041 , -0.10476 , -0.32831 , -0.60674 ,  0.12636 , -0.15849 ,
       -0.1296  , -1.624   , -0.43042 , -0.8912  ,  0.96105 ,  0.81197 ,
        0.40817 , -0.61422 , -0.12197 ,  0.38899 ,  1.7551  ,  0.67989 ,
       -0.26323 , -0.72813 ,  0.047063, -0.090324, -2.3259  ,  0.11788 ,
       -0.5689  , -0.058212, -0.087544, -0.77046 , -0.31418 , -0.70277 ,
        0.3376  , -0.42649 , -0.71333 , -0.67349 , -0.31198 ,  0.62908 ,
        0.85631 ,  0.42757 ,  0.36119 , -0.49708 , -0.31234 ,  0.64581 ,
       -0.10335 , -0.60543 , -0.12391 ,  0.31221 , -0.17433 ,  0.60417 ,
       -0.31664 ,  0.27621 ,  0.70393 , -0.75601 ,  0.14436 , -0.14913 ,
        0.13205 ,  0.6797  ,  0.27813 , -0.95859 , -0.5562  ,  0.21086 ,
        0.6172  , -0.68459 ,  1.8256  , -0.48079 , -0.060844, -0.15566 ,
        0.022667, -0.39432 , -0.2795  ,  0.72409 ,  0.55935 ,  0.82758 ,
       -0.44102 ,  0.091789,  0.22119 ,  0.36782 ,  0.83826 , -0.5405  ,
        0.030171, -0.48167 ,  0.68211 ,  0.36674 ], dtype=float32),
'<hashtag>': array([ 1.2632e-01, -6.7049e-01,  3.6077e-01, -5.9244e-01, -2.0898e-01,
       -3.8867e-01,  6.0406e-01,  1.1256e-01,  1.7291e-01,  7.5089e-01,
       -4.1904e-01, -2.4512e-01, -2.5029e+00,  4.4884e-02, -1.1681e-01,
       -6.5162e-01, -8.2035e-01,  2.7720e-01, -6.3401e-01,  1.0864e+00,
        2.0465e-01,  7.9896e-01, -2.4333e-01, -7.5383e-01,  7.8801e-01,
       -1.6382e+00, -6.4087e-01,  1.0505e+00,  9.0609e-01,  1.2356e-01,
       -1.3677e+00, -9.1896e-01, -6.2909e-01,  1.6805e-01,  1.7366e+00,
       -5.0556e-01, -3.4257e-01, -3.7407e-02, -7.7894e-02,  3.5249e-02,
       -1.0114e+00,  7.7162e-02, -1.3277e-01, -6.0730e-01,  1.4327e+00,
        2.1274e-01,  4.9006e-01, -1.2465e-01, -3.1849e-01, -4.8117e-01,
       -7.1897e-01, -3.1361e-01,  8.2422e-02, -4.0866e-01,  4.5471e-01,
        2.0882e-01,  8.7377e-02, -3.5028e-01, -1.1497e-01,  3.2167e-01,
        2.0904e-01, -5.9784e-01, -6.4296e-01,  9.2053e-02, -1.7296e-01,
       -7.9641e-01, -4.9189e-01,  6.0002e-01,  6.8936e-01, -6.8849e-02,
        3.1007e-01, -3.6550e-01,  3.2889e-01,  5.4283e-01, -3.7797e-02,
        8.4533e-01, -1.1667e+00,  1.2164e+00,  8.6391e-02, -4.8441e-01,
        1.2335e+00,  9.4050e-01,  9.0539e-01,  6.3121e-02,  4.5760e-01,
        1.1538e+00, -1.1640e+00,  1.0212e+00, -2.4575e-01, -4.1846e-01,
        3.0621e-01, -4.9738e-01, -3.0436e-02, -2.0296e-01, -3.4046e-01,
       -1.6077e-03, -5.0538e-01,  6.6966e-01,  1.0386e+00,  3.3380e-01],
      dtype=float32),
'<allcaps>': array([ 0.33044 ,  0.36213 , -0.15892 , -0.47508 , -0.063629, -0.18372 ,
        0.13071 ,  0.61767 , -0.24933 ,  1.4177  ,  0.14176 ,  0.38365 ,
       -1.6181  ,  0.17322 ,  0.15272 , -0.77282 , -0.60746 ,  0.34523 ,
        0.13082 ,  0.41413 ,  0.33098 ,  0.51875 ,  0.75675 , -0.56653 ,
        1.3991  , -1.2663  ,  0.054918,  0.50918 ,  0.81151 ,  0.73381 ,
       -1.0978  , -1.2046  , -1.659   ,  0.70671 ,  1.7311  , -0.31473 ,
       -0.23074 , -0.61805 ,  0.3129  ,  0.2892  , -0.78312 , -0.052375,
        0.22623 , -0.093728,  1.2496  ,  0.059133, -0.14123 , -0.24111 ,
       -0.46753 , -0.90309 , -1.5381  , -0.42466 , -0.22135 , -0.33583 ,
        0.72838 ,  0.75743 ,  0.051164, -0.44487 , -0.54041 ,  0.98763 ,
       -0.096493, -1.4522  , -0.16171 ,  0.41627 , -0.72265 , -0.86155 ,
       -0.40381 , -0.18222 , -0.16954 , -0.19077 ,  0.95275 , -0.035938,
        0.18976 ,  0.44652 , -0.43185 , -0.088494, -0.75584 ,  0.87691 ,
       -0.38125 , -1.4013  ,  0.68165 , -0.083688,  0.69105 ,  0.58954 ,
        0.2336  ,  0.39261 , -0.83894 ,  2.0978  ,  0.13161 , -0.21419 ,
       -0.52524 ,  0.26011 ,  0.2959  , -0.093153, -0.01044 , -0.6742  ,
       -0.13762 , -0.028998,  0.56561 ,  0.17777 ], dtype=float32),
'<repeat>': array([ 1.5955e-01,  2.6989e-01,  2.0858e-01,  3.0543e-01, -7.8765e-02,
        5.7171e-01, -1.1096e-01,  4.0187e-01,  1.5700e-01,  7.0075e-01,
        1.7793e-01, -1.6058e-01, -2.0023e+00,  3.4172e-01,  6.3570e-02,
       -2.6815e-04, -8.2191e-02, -1.2240e-01, -5.3392e-01, -6.3666e-01,
       -6.5473e-02,  2.2979e-01,  3.8609e-03,  1.4067e-01,  7.0308e-01,
       -1.7196e+00, -2.6129e-01, -3.9572e-01,  8.6836e-01, -2.5373e-01,
       -6.0829e-01, -6.9807e-02, -4.3805e-01,  5.8822e-01,  1.6711e-01,
        2.8068e-01,  4.7312e-01,  8.9204e-02,  3.6153e-02, -2.4280e-01,
       -1.1554e+00,  1.4460e-01, -5.6071e-01, -2.4045e-02,  6.9194e-01,
        6.4904e-02,  5.4029e-01,  2.9806e-01, -5.3121e-02, -2.4850e-02,
       -1.1096e+00, -4.4980e-02,  5.0792e-01,  1.2867e-02,  5.8161e-01,
```

```
         2.5465e-01, -1.2493e-01,  4.5379e-01,  3.6407e-01,  4.7338e-02,
        -1.6498e-01, -2.5395e-01,  3.4337e-01,  2.4567e-01,  3.6126e-01,
        -9.0730e-02, -1.8550e-02, -8.3714e-01, -2.9615e-01,  5.0266e-01,
         7.9340e-01,  3.9469e-01,  2.4411e-01, -9.6259e-02,  1.1570e-01,
        -2.5216e-01, -2.1291e-01,  4.1922e-01, -3.6031e-01, -4.0823e-01,
         4.2863e-01,  5.1523e-02, -2.4388e-01, -8.5650e-01,  4.9252e-01,
         5.5513e-01, -3.6313e-01,  5.9899e-01, -1.5685e-01,  2.4068e-01,
        -2.7657e-01, -3.9037e-01, -3.9130e-02, -3.0325e-03, -5.4934e-01,
         8.9335e-02,  3.9065e-01, -1.1433e-01,  3.2797e-01, -6.6378e-01],
       dtype=float32),
'<elong>': array([ 1.3843e-01,  9.9803e-02,  1.7756e-02,  8.0437e-01,  6.0603e-02,
        -2.6381e-01,  4.0485e-01,  3.8277e-01,  7.3432e-02,  8.6143e-01,
        -7.3322e-01,  4.1529e-01, -1.6406e+00,  1.0813e-01, -2.1912e-01,
        -7.7445e-01, -2.2763e-01, -7.1287e-01, -9.0651e-01,  2.6695e-02,
        -5.2923e-02,  7.9648e-02,  7.4934e-01,  4.1577e-01,  8.2849e-01,
        -2.4916e+00, -4.3309e-02,  8.4579e-02,  5.4622e-01,  2.2592e-01,
        -6.3423e-01, -8.2994e-03, -6.0767e-01,  1.5099e-01, -7.4703e-02,
         6.2276e-01, -3.6558e-01,  2.7889e-01,  2.5673e-01, -3.0723e-01,
        -1.9189e+00,  7.2002e-01, -5.2408e-04,  2.0431e-01,  6.3296e-02,
         7.9136e-02, -2.0141e-01,  4.8301e-01,  3.1460e-01,  4.6127e-01,
        -1.9753e+00, -6.0828e-01,  4.1193e-01,  4.3439e-03,  1.2191e+00,
         5.6115e-01, -9.8016e-02,  7.6635e-01,  5.4969e-01,  2.0520e-01,
        -7.5093e-01, -5.7189e-01,  8.2580e-02, -2.7617e-01,  1.4465e-01,
        -5.4270e-01, -9.2662e-01, -7.3692e-01, -4.0466e-02,  1.0021e-01,
         4.0175e-01,  2.7227e-01, -2.7547e-01,  2.7186e-01, -4.7587e-03,
        -5.6713e-01,  2.1039e-01,  4.3011e-01, -3.8225e-01,  5.2998e-02,
         8.3440e-01,  1.2591e-01, -2.6968e-01, -4.3603e-01,  2.7717e-01,
         7.7154e-01, -3.3810e-01, -1.8748e-01, -1.3012e-01, -6.6589e-02,
         6.7188e-03,  2.7435e-01, -2.5802e-01, -7.9546e-02, -3.6391e-01,
        -3.6738e-02,  2.3538e-01, -1.0575e-01,  2.8356e-01, -6.3926e-01],
       dtype=float32),
'<blank>': 'not found'}
```

Handle blanks

We can see that the word embeddings taken from twitter already contain all the tags introduced during text cleaning. The tag "<blank>" was added here to fill the slots of the RNN where text is too short. We have two options at this point:

1. Map it to a vector of zeros
2. Map it to a vector of a related word such as blank or empty

Notice that unrecognized words in our model will be passed to a mask that will assign the vector 0 to them. However, an unrecognized word is different from an empty slot, so let us distinguish them by using the second approach here.

In [35]:
```
embeddings['<blank>'] = embeddings['empty']
```

In [36]:
```
# Build a matrix to store word embeddings
embedding_matrix = np.zeros((num_words,embed_dim))
print('The embedding matrix has shape {}'.format(embedding_matrix.shape))
for i, word in enumerate(word_idx.keys()):
    vector = embeddings.get(word,None)
    if vector is not None:
        embedding_matrix[i+1,:] = vector
```

```
The embedding matrix has shape (18830, 100)
```

In [37]:
```
df_train_sub, df_cv_sub = train_test_split(df_train, test_size=0.2, stratify=df_train['target'])
print('The training data set was split into {} entries for training and {} entries for cross validation'
      .format(len(df_train_sub), len(df_cv_sub)))
```

```
The training data set was split into 6090 entries for training and 1523 entries for cross validation
```

4. Models

Let us start by finding the expected accuracy of a very naive model: predicting the maximum category always.

In [38]:
```
from sklearn import metrics
```

Targets

In [39]:
```
y_train = df_train_sub['target']
y_cv = df_cv_sub['target']
```

Baseline: predict most frequent

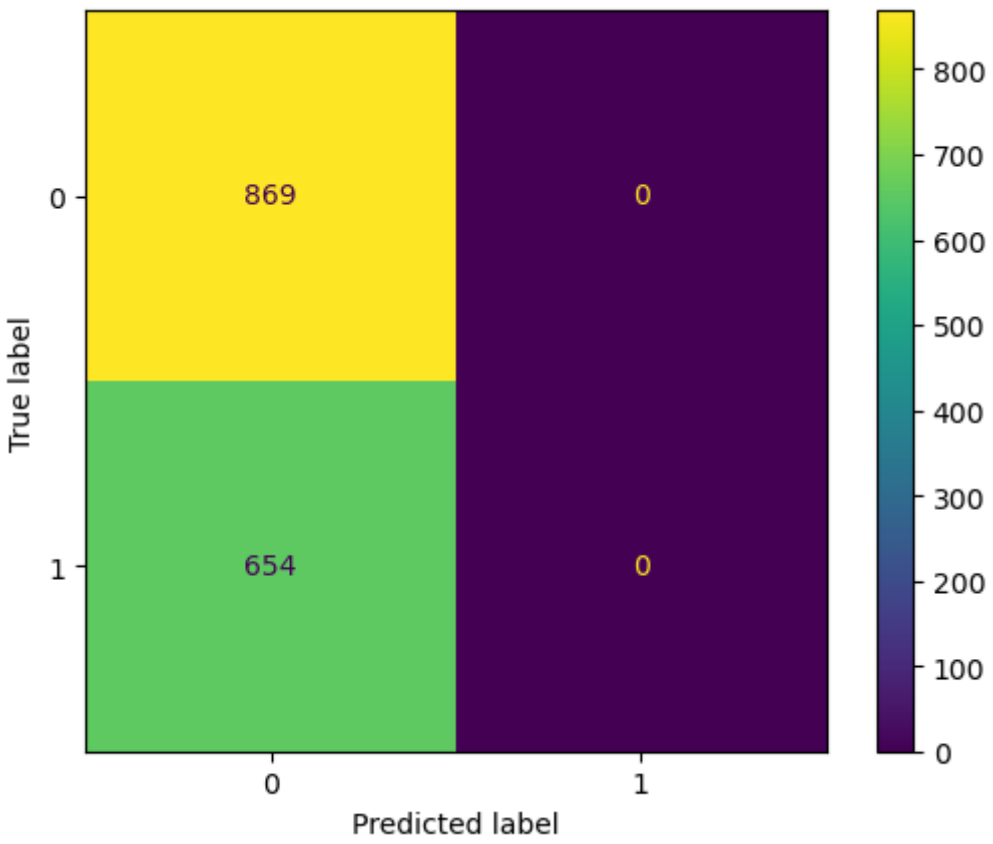
In [40]:
```
most_frequent = y_train.value_counts().index[0]
y_pred_cv = df_cv_sub['target']*0 + most_frequent
print(y_pred_cv)
```

```
4358    0
4331    0
1779    0
5446    0
383     0
       ..
3905    0
303     0
4966    0
2428    0
6568    0
Name: target, Length: 1523, dtype: int64
```

In [41]:
```
def display_conf_max_and_acc(y_true,y_pred,model_name):
    acc = metrics.accuracy_score(y_true,y_pred)
    cm=metrics.confusion_matrix(y_true,y_pred);
    print('Cross validation accuracy of model `{}` is {:.2%}'.format(model_name,acc))
    metrics.ConfusionMatrixDisplay(cm).plot();
    return acc
```

In [42]:
```
name = 'Baseline - Predict most frequent category'
acc = display_conf_max_and_acc(y_cv,y_pred_cv,name)
```

```
Cross validation accuracy of model `Baseline - Predict most frequent category` is 57.06%
```



In [43]:
```
models = [] # All model results will be stored here

# Baseline model
model = {'model_name': name,
    'cv_acc': acc}

models.append(model)
print("Model: {}\nCross Val accuracy: {:.2%}".format(model['model_name'],model['cv_acc']))
```

```
Model: Baseline - Predict most frequent category
Cross Val accuracy: 57.06%
```

4.1 Basic model using keyword and location only

Let us use a tree based model to predict target without using the text column. We would normally use one-hot encoding, but with trees one can directly use ordinal encoding (https://towardsdatascience.com/one-hot-encoding-is-making-your-tree-based-ensembles-worse-heres-why-d64b282b5769)

Create Gradient boosting model

In [44]:
```
from sklearn import preprocessing
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import GradientBoostingClassifier as GB
from sklearn.model_selection import GridSearchCV
```

In [45]:
```
cols_to_encode = ['keyword','location']
ct = ColumnTransformer(
    [("ordinal_encoder",
      preprocessing.OrdinalEncoder(handle_unknown='use_encoded_value',unknown_value=-1),
      cols_to_encode)],
#     remainder="passthrough"
)
ct = ct.fit(df_train[cols_to_encode])


X_train_gb = ct.transform(df_train_sub[cols_to_encode])
X_cv_gb = ct.transform(df_cv_sub[cols_to_encode])
X_test_gb = ct.transform(df_test[cols_to_encode])
```

In [46]:
```
# np.concatenate([X_train_gb,X_cv_gb]).shape
# # np.concatenate([y_train,y_cv])
```

In [47]:

```
# Train and use hyperarameter optimization
# Note: we want to use the same cross validation split here and in the Keras model,
# and for this we use a PredefinedSplit with one fold.
from sklearn.model_selection import PredefinedSplit

clf = GB(n_estimators=200,
         learning_rate=1.5,
         max_depth=1)

param_grid = {'n_estimators': [50,150,300],
              'learning_rate': [0.05, 0.1, 0.2],
             'max_depth':[2,3,5]}
print('X_train_gb:',X_train_gb.shape,type(X_train_gb))
print('X_cv_gb:',X_train_gb.shape,type(X_cv_gb))
print('y_train:',y_train.shape,type(y_train))
print('y_cv:',y_cv.shape,type(y_cv))
xx = np.concatenate([X_train_gb,X_cv_gb])
yy = np.concatenate([y_train,y_cv])
test_fold =np.zeros_like(yy);
test_fold[0:X_train_gb.shape[0]]=-1
hyper_search = GridSearchCV(clf,
                            param_grid,
                            cv = PredefinedSplit(test_fold),
                            refit = True,
                            verbose = 3)
# hyper_search.fit(X_train_gb,y_train)

hyper_search.fit(xx,yy)
```

```
X_train_gb: (6090, 2) <class 'numpy.ndarray'>
X_cv_gb: (6090, 2) <class 'numpy.ndarray'>
y_train: (6090,) <class 'pandas.core.series.Series'>
y_cv: (1523,) <class 'pandas.core.series.Series'>
Fitting 1 folds for each of 27 candidates, totalling 27 fits
[CV 1/1] END learning_rate=0.05, max_depth=2, n_estimators=50;, score=0.601 total time=   0.2s
[CV 1/1] END learning_rate=0.05, max_depth=2, n_estimators=150;, score=0.618 total time=   0.5s
[CV 1/1] END learning_rate=0.05, max_depth=2, n_estimators=300;, score=0.655 total time=   0.9s
[CV 1/1] END learning_rate=0.05, max_depth=3, n_estimators=50;, score=0.603 total time=   0.2s
[CV 1/1] END learning_rate=0.05, max_depth=3, n_estimators=150;, score=0.648 total time=   0.6s
[CV 1/1] END learning_rate=0.05, max_depth=3, n_estimators=300;, score=0.682 total time=   1.2s
[CV 1/1] END learning_rate=0.05, max_depth=5, n_estimators=50;, score=0.646 total time=   0.3s
[CV 1/1] END learning_rate=0.05, max_depth=5, n_estimators=150;, score=0.689 total time=   1.0s
[CV 1/1] END learning_rate=0.05, max_depth=5, n_estimators=300;, score=0.698 total time=   2.0s
[CV 1/1] END learning_rate=0.1, max_depth=2, n_estimators=50;, score=0.617 total time=   0.1s
[CV 1/1] END learning_rate=0.1, max_depth=2, n_estimators=150;, score=0.659 total time=   0.4s
[CV 1/1] END learning_rate=0.1, max_depth=2, n_estimators=300;, score=0.697 total time=   0.9s
[CV 1/1] END learning_rate=0.1, max_depth=3, n_estimators=50;, score=0.617 total time=   0.2s
[CV 1/1] END learning_rate=0.1, max_depth=3, n_estimators=150;, score=0.676 total time=   0.6s
[CV 1/1] END learning_rate=0.1, max_depth=3, n_estimators=300;, score=0.707 total time=   1.2s
[CV 1/1] END learning_rate=0.1, max_depth=5, n_estimators=50;, score=0.682 total time=   0.3s
[CV 1/1] END learning_rate=0.1, max_depth=5, n_estimators=150;, score=0.693 total time=   1.0s
[CV 1/1] END learning_rate=0.1, max_depth=5, n_estimators=300;, score=0.697 total time=   2.0s
[CV 1/1] END learning_rate=0.2, max_depth=2, n_estimators=50;, score=0.634 total time=   0.2s
[CV 1/1] END learning_rate=0.2, max_depth=2, n_estimators=150;, score=0.690 total time=   0.4s
[CV 1/1] END learning_rate=0.2, max_depth=2, n_estimators=300;, score=0.705 total time=   0.9s
[CV 1/1] END learning_rate=0.2, max_depth=3, n_estimators=50;, score=0.664 total time=   0.2s
[CV 1/1] END learning_rate=0.2, max_depth=3, n_estimators=150;, score=0.700 total time=   0.6s
[CV 1/1] END learning_rate=0.2, max_depth=3, n_estimators=300;, score=0.705 total time=   1.2s
[CV 1/1] END learning_rate=0.2, max_depth=5, n_estimators=50;, score=0.692 total time=   0.3s
[CV 1/1] END learning_rate=0.2, max_depth=5, n_estimators=150;, score=0.701 total time=   1.0s
[CV 1/1] END learning_rate=0.2, max_depth=5, n_estimators=300;, score=0.698 total time=   2.0s
```

Out[47]:

```
GridSearchCV(cv=PredefinedSplit(test_fold=array([-1, -1, ...,  0,  0])),
             estimator=GradientBoostingClassifier(learning_rate=1.5,
                                                  max_depth=1,
                                                  n_estimators=200),
             param_grid={'learning_rate': [0.05, 0.1, 0.2],
                         'max_depth': [2, 3, 5],
                         'n_estimators': [50, 150, 300]},
             verbose=3)
```

In [48]:

```
print('Best hyperparameters:\n',hyper_search.best_params_)
print('Best score:\n',hyper_search.best_score_)
```

```
Best hyperparameters:
 {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 300}
Best score:
 0.706500328299409
```

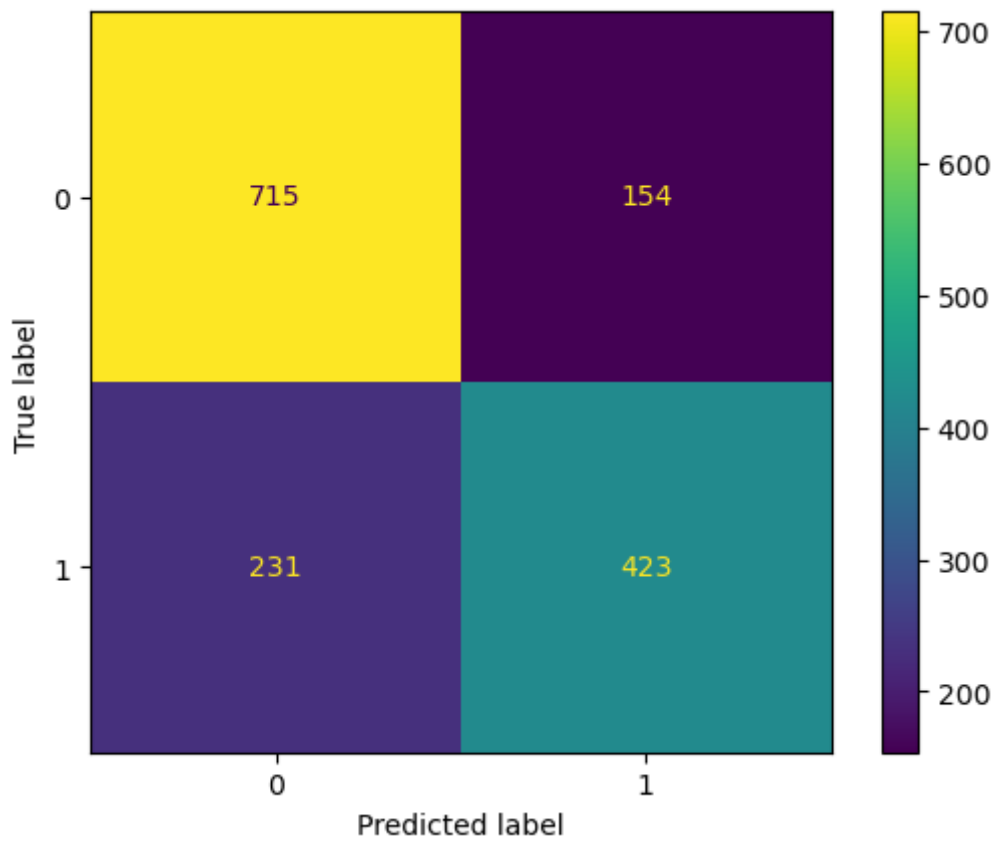
In [49]:

```
clf = hyper_search.best_estimator_
print(clf)
```

```
GradientBoostingClassifier(n_estimators=300)
```

In [50]:
```
# Results
y_pred_cv = clf.predict(X_cv_gb)
name = 'Gradient Boosting using keyword and location only'
acc = display_conf_max_and_acc(y_cv,y_pred_cv,name)
```

```
Cross validation accuracy of model `Gradient Boosting using keyword and location only` is 74.72%
```



In [51]:
```
# Simple gradient boosting model
model = {'model_name': name,
    'cv_acc': acc,
        'model': clf}

models.append(model)
print("Model: {}\nCross Val accuracy: {:.2%}".format(model['model_name'],model['cv_acc']))
```

```
Model: Gradient Boosting using keyword and location only
Cross Val accuracy: 74.72%
```

In [52]:
```
pd.DataFrame(models)
```

Out[52]:

	model_name	cv_acc	model
0	Baseline - Predict most frequent category	0.570584	NaN
1	Gradient Boosting using keyword and location only	0.747209	([DecisionTreeRegressor(criterion='friedman_ms...

4.2 Recursive neural network model with one LSTM layer based on text only

Prepare text input vector for RNNs as a sequence of integers based on tokenizer

We will be using Glove word embeddings trained on twitter data. The RNN model will be fed with a sequence of integers representing words. Instead of padding zeros, we will pad embeddings from a desired word to the left of the text sequence.

In [53]:
```
model_word_slots = max_words_per_text
print('The RNN model will be using {} words per text'.format(model_word_slots))
```

```
The RNN model will be using 51 words per text
```

In [54]:
```
# We could use pad_sequences from keras to pad zeros, but another approach is used here
def prepare_input_data_from_one_seq(seq:list,blank_id:int=0,desired_len:int=20):
    """
    Given a sequence of numbers, return a row of desired length padding blank_id on the left of seq
    """
    x = np.zeros(desired_len,dtype=np.int32)
    x.fill(blank_id)
    n = np.min([len(seq),desired_len])
    if n>0:
        x[-n:] = np.array(seq)[:n]
    return x
def prepare_input_data(df,blank_id, n_words):
    """ Compute X for our neural network model"""
    s = lambda row: prepare_input_data_from_one_seq(row,blank_id,n_words)
    return np.stack(df['sequences'].apply(s))

# Compute X from dataframes
X_train_rnn = prepare_input_data(df_train_sub,blank_id, model_word_slots)
X_cv_rnn = prepare_input_data(df_cv_sub,blank_id, model_word_slots)
X_test_rnn = prepare_input_data(df_test,blank_id, model_word_slots)
# Reshape labels into 2d
y_train_rnn = y_train.values.reshape(-1,1)
y_cv_rnn = y_cv.values.reshape(-1,1)

print("Dimensions: \n Xtrain: {}\n y_train_rnn: {}".format(X_train_rnn.shape,y_train_rnn.shape))
```

```
Dimensions:
 Xtrain: (6090, 51)
 y_train_rnn: (6090, 1)
```

In [55]:
```
np.random.randint(8)
```

Out[55]:
```
2
```

In [56]:

```
print('Display random sample of sequence fed to model:')
k = np.random.randint(X_train_rnn.shape[0])
print('Original text:')
print(df_train_sub['text'].iloc[k])
print('Sequence as words:')
[idx_word[id] for id in X_train_rnn[k]]
```

```
Display random sample of sequence fed to model:
Original text:
Reuters Top News: PHOTOS: The Rocky Fire has grown into California's most ... - http://t.co/qwrRfDGXCc #Ne
wsInTweets http://t.co/sstj2bEpqn
Sequence as words:
```

Out[56]:

```
['<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 '<blank>',
 'reuters',
 'top',
 'news:',
 'photos',
 '<allcaps>:',
 'the',
 'rocky',
 'fire',
 'has',
 'grown',
 'into',
 "california's",
 'most',
 '.',
 '<repeat>',
 '-',
 '<url>',
 '<hashtag>',
 'news',
 'in',
 'tweets',
 '<url>']
```

Create RNN model with Keras Sequential API

In [57]:

```
from keras import Sequential, layers, Input, Model
from keras.models import load_model
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.utils import plot_model
from keras.optimizers import Adam, Nadam
```

In [58]:

```
def create_rnn_model(name:str='my_rnn_model',
                     num_RNN_cell_layers:int=1,
                     dim_RNN_cell=32,
                     dim_dense_lyr = 16,
                    learning_rate = 5e-4,
                    dropout = 0.4,
                    rnn_cell_type = 'LSTM'):
    model = Sequential(name=name)
    model.add(layers.Embedding(input_dim = num_words, # number of words in vocabulary
                       input_length = model_word_slots, # number of words in input
                       output_dim = embed_dim, # glove vector length
                       weights = [embedding_matrix], # precomputed weights from glove
                       trainable = False,
                       mask_zero = True))
    # map unknowun values to 0
    model.add(layers.Masking(mask_value = 0.0))

    # Optional Intermediate RNN (LSTM or GRU) layers
    RNN_cell_types = {'LSTM':layers.LSTM,
                      'GRU': layers.GRU}
    rnn_cell = RNN_cell_types[rnn_cell_type]
    if num_RNN_cell_layers>1:
        for i in range(num_RNN_cell_layers-1):
            model.add(rnn_cell(dim_RNN_cell,
                           recurrent_dropout = dropout,
                           dropout = dropout,
                           return_sequences=True))
    # Last RNN cell layer
    model.add(rnn_cell(dim_RNN_cell))
    return_sequences=True

    # Fully connected layer
    model.add(layers.Dense(dim_dense_lyr, activation='relu'))

    # Dropout for regularization
    model.add(layers.Dropout(dropout))

    # Output layer
    model.add(layers.Dense(1, activation='sigmoid',name = 'output'))

    model.compile(
    optimizer=Nadam(learning_rate), loss='binary_crossentropy', metrics=['accuracy'])


    return model

rnn_model1 = create_rnn_model('RNN_single_LSTM_cell_only_text',1,32,16)

rnn_model1.summary()
```

```
Model: "RNN_single_LSTM_cell_only_text"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 51, 100)           1883000

 masking (Masking)           (None, 51, 100)           0

 lstm (LSTM)                 (None, 32)                17024

 dense (Dense)               (None, 16)                528

 dropout (Dropout)           (None, 16)                0

 output (Dense)              (None, 1)                 17

=================================================================
Total params: 1,900,569
Trainable params: 17,569
Non-trainable params: 1,883,000
_________________________________________________________________
```

In [59]:

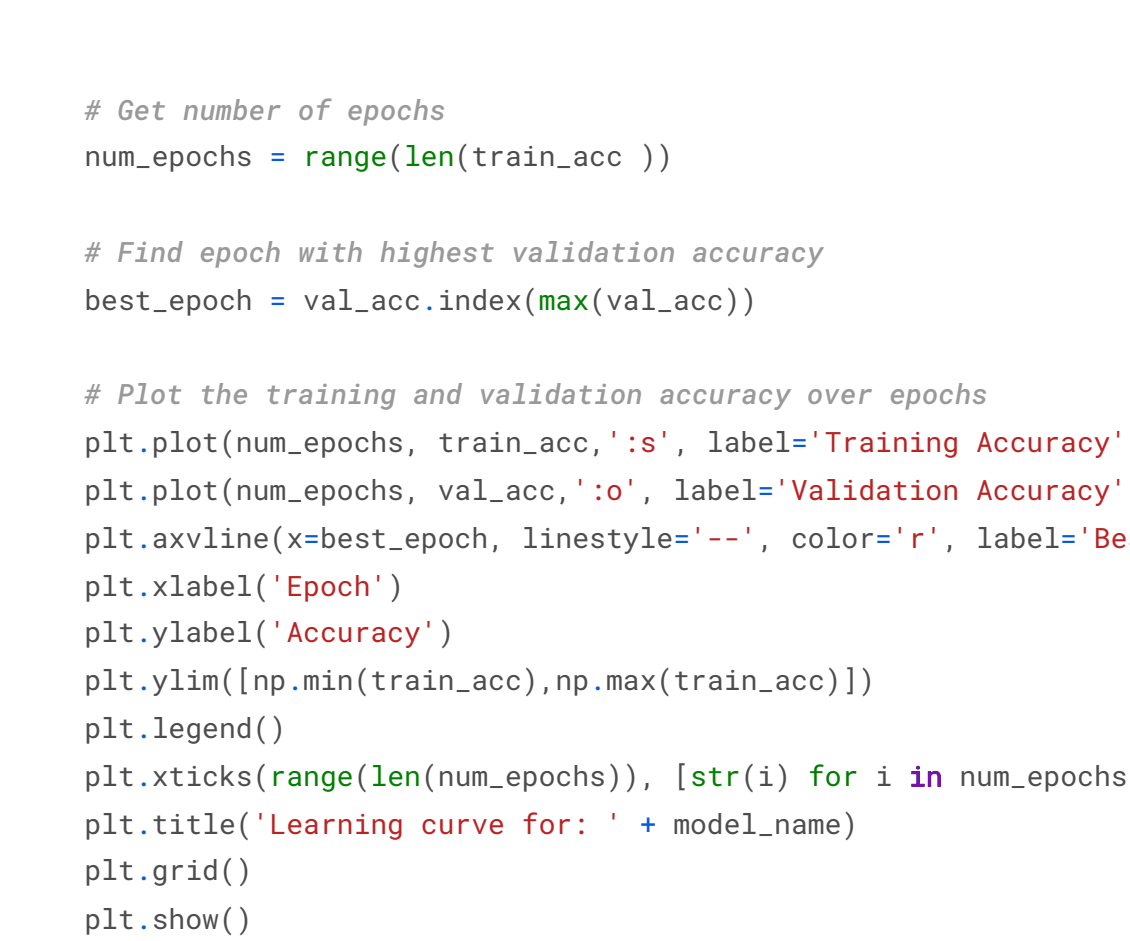
```
# Create callbacks
models_path = '../models'
model_save_path = os.path.join(models_path,rnn_model1.name+'.h5')
callbacks = [EarlyStopping(monitor='val_loss', patience=5),
             ModelCheckpoint(model_save_path, save_best_only=True,
                             save_weights_only=False)]
```

In [60]:
```
print('Training {} model on texts only...\n'.format(rnn_model1.name))
history = rnn_model1.fit(X_train_rnn,  y_train_rnn,
                   batch_size=64, epochs=150,
                   callbacks=callbacks,
                   validation_data=(X_cv_rnn, y_cv_rnn))
print('...Done')
```

```
Training RNN_single_LSTM_cell_only_text model on texts only...

Epoch 1/150
96/96 [==============================] - 10s 51ms/step - loss: 0.6127 - accuracy: 0.6677 - val_loss: 0.517
8 - val_accuracy: 0.7636
Epoch 2/150
96/96 [==============================] - 3s 35ms/step - loss: 0.5118 - accuracy: 0.7677 - val_loss: 0.4933
- val_accuracy: 0.7820
Epoch 3/150
96/96 [==============================] - 3s 35ms/step - loss: 0.4843 - accuracy: 0.7888 - val_loss: 0.4693
- val_accuracy: 0.7866
Epoch 4/150
96/96 [==============================] - 3s 34ms/step - loss: 0.4663 - accuracy: 0.7987 - val_loss: 0.4721
- val_accuracy: 0.7965
Epoch 5/150
96/96 [==============================] - 3s 34ms/step - loss: 0.4582 - accuracy: 0.8043 - val_loss: 0.4689
- val_accuracy: 0.7892
Epoch 6/150
96/96 [==============================] - 3s 35ms/step - loss: 0.4486 - accuracy: 0.8108 - val_loss: 0.4644
- val_accuracy: 0.7958
Epoch 7/150
96/96 [==============================] - 3s 35ms/step - loss: 0.4374 - accuracy: 0.8126 - val_loss: 0.4614
- val_accuracy: 0.7833
Epoch 8/150
96/96 [==============================] - 3s 35ms/step - loss: 0.4347 - accuracy: 0.8177 - val_loss: 0.4542
- val_accuracy: 0.7991
Epoch 9/150
96/96 [==============================] - 3s 34ms/step - loss: 0.4252 - accuracy: 0.8207 - val_loss: 0.4556
- val_accuracy: 0.7958
Epoch 10/150
96/96 [==============================] - 3s 36ms/step - loss: 0.4202 - accuracy: 0.8248 - val_loss: 0.4550
- val_accuracy: 0.8004
Epoch 11/150
96/96 [==============================] - 3s 35ms/step - loss: 0.4159 - accuracy: 0.8243 - val_loss: 0.4652
- val_accuracy: 0.7892
Epoch 12/150
96/96 [==============================] - 3s 35ms/step - loss: 0.4088 - accuracy: 0.8296 - val_loss: 0.4544
- val_accuracy: 0.7984
Epoch 13/150
96/96 [==============================] - 3s 34ms/step - loss: 0.4061 - accuracy: 0.8328 - val_loss: 0.4798
- val_accuracy: 0.7754
...Done
```

In [61]:

```
def plot_learning_curves(history,model_name):
    """
    Plots the training and validation loss over epochs from a Keras history object.
    """

    # Get training and validation loss values from the history object
    train_acc  = history.history['accuracy']
    val_acc = history.history['val_accuracy']



    # Get number of epochs
    num_epochs = range(len(train_acc ))

    # Find epoch with highest validation accuracy
    best_epoch = val_acc.index(max(val_acc))

    # Plot the training and validation accuracy over epochs
    plt.plot(num_epochs, train_acc,':s', label='Training Accuracy')
    plt.plot(num_epochs, val_acc,':o', label='Validation Accuracy')
    plt.axvline(x=best_epoch, linestyle='--', color='r', label='Best Model')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.ylim([np.min(train_acc),np.max(train_acc)])
    plt.legend()
    plt.xticks(range(len(num_epochs)), [str(i) for i in num_epochs])
    plt.title('Learning curve for: ' + model_name)
    plt.grid()
    plt.show()
plot_learning_curves(history,rnn_model1.name)
```

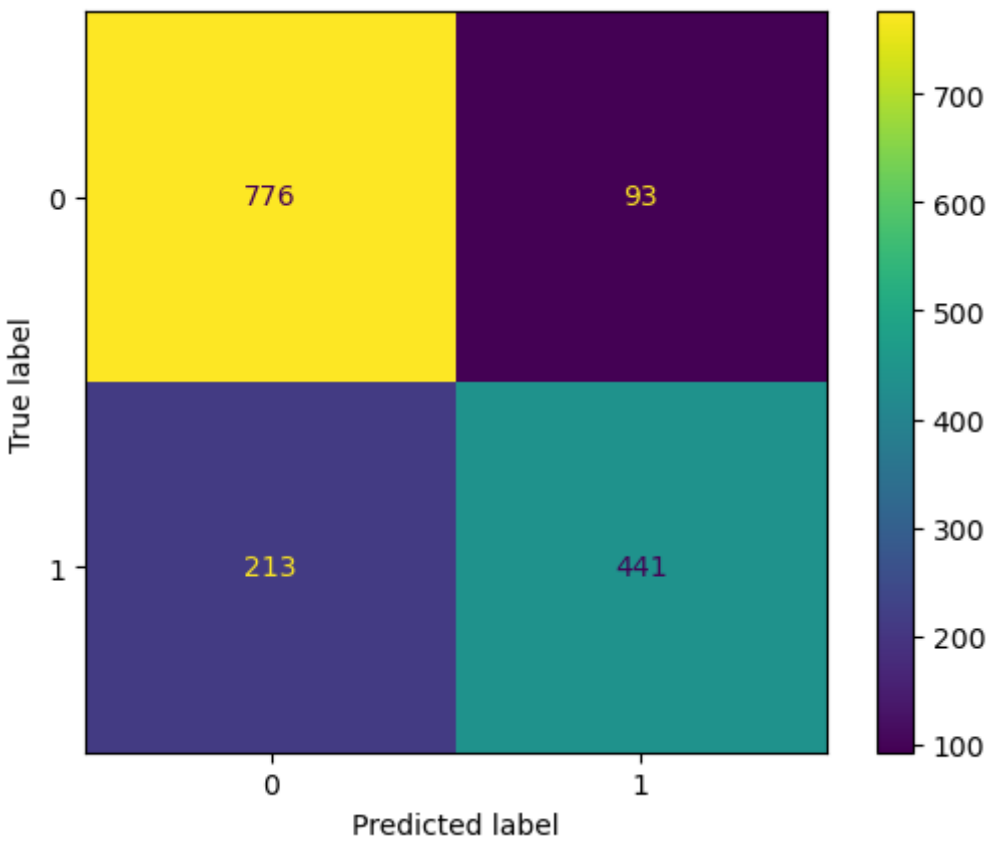


In [62]:

```
# Reload best model
rnn_model1 = load_model(model_save_path)
```

In [63]:
```
print('Computing model predictions...')
y_prob_cv = rnn_model1.predict(X_cv_rnn)
y_pred_cv = np.round(y_prob_cv)

acc = display_conf_max_and_acc(y_cv,y_pred_cv,rnn_model1.name)
```

```
Computing model predictions...
48/48 [==============================] - 2s 10ms/step
Cross validation accuracy of model `RNN_single_LSTM_cell_only_text` is 79.91%
```



In [64]:
```
# Recursive neural network LSTM model using text only
model = {'model_name': rnn_model1.name,
    'cv_acc': acc,
        'model': rnn_model1}

models.append(model)
print("Model: {}\nCross Val accuracy: {:.2%}".format(model['model_name'],model['cv_acc']))
```

```
Model: RNN_single_LSTM_cell_only_text
Cross Val accuracy: 79.91%
```

In [65]:
```
pd.DataFrame(models)
```

Out[65]:

	model_name	cv_acc	model
0	Baseline - Predict most frequent category	0.570584	NaN
1	Gradient Boosting using keyword and location only	0.747209	([DecisionTreeRegressor(criterion='friedman_ms...
2	RNN_single_LSTM_cell_only_text	0.799081	<keras.engine.sequential.Sequential object at ...

4.3 RNN model based on text only (hyperparameter optimization with LSTM and GRU)

Let us perform hyperaprameter optimization. We will optimize the number of stacked RNN layers and try both GRU and LSTM cells.

In [66]:
```
from kerastuner.engine.hyperparameters import HyperParameters
from kerastuner.tuners import Hyperband
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:1: DeprecationWarning: `import kerastuner` is deprecated, please use `import keras_tuner`.
  """Entry point for launching an IPython kernel.
```

In [67]:
```
# Define the hyperparameter search space
model_name = 'RNN_tuned_text_only2'

def build_model(hp):
    """
    Build model for keras tuner.
    Remove comments to tune other hyperparameters (will take several hours)
    Some parts may be commented for shorter running times
    """
    num_RNN_cell_layers = hp.Int('num_RNN_cell_layers', 1, 3);
#     dim_RNN_cell = hp.Int('dim_RNN_cell', 16, 64);
#     dim_dense_lyr = hp.Int('dim_dense_lyr', 8, 32);
#     learning_rate = hp.Float('learning_rate', 1e-3, 5e-2, sampling='log');
    dropout = hp.Float('dropout', min_value=0.2, max_value=0.5, step=0.2);
    rnn_cell_type = hp.Choice('rnn_cell_type', ['LSTM','GRU']);

    return create_rnn_model(model_name,
                            num_RNN_cell_layers=num_RNN_cell_layers,
#                             dim_RNN_cell=dim_RNN_cell,
#                             dim_dense_lyr=dim_dense_lyr,
#                             learning_rate=learning_rate,
                            dropout=dropout,
                            rnn_cell_type = rnn_cell_type)

# Create callbacks
models_path = '../hypertuning_models'
model_save_path = os.path.join(model_name+'.h5')
callbacks = [EarlyStopping(monitor='val_loss', patience=5),
             ModelCheckpoint(model_save_path, save_best_only=True,
                             save_weights_only=False)]

# Create the tuner and search for the best hyperparameters
hp = HyperParameters();


tuner = Hyperband(
    build_model,
    hyperparameters=hp,
    objective='val_accuracy',
    hyperband_iterations=2,
    directory='Keras_Hypertuning',
    project_name='RNN_hypertune'
)
```

In [68]:
```
# Tune model hyperparameters using cross validation
tuner.search(X_train_rnn, y_train_rnn,
             epochs = 50,
             batch_size=64,
             validation_data=(X_cv_rnn, y_cv_rnn),
             callbacks=callbacks)
```

```
Trial 12 Complete [00h 00m 16s]
val_accuracy: 0.753775417804718

Best val_accuracy So Far: 0.7793828248977661
Total elapsed time: 00h 06m 04s
```

In [69]:
```
# Create callbacks
models_path = '../models'
model_save_path = os.path.join(models_path,rnn_model1.name+'.h5')
callbacks = [EarlyStopping(monitor='val_loss', patience=5),
             ModelCheckpoint(model_save_path, save_best_only=True,
save_weights_only=False)]
```

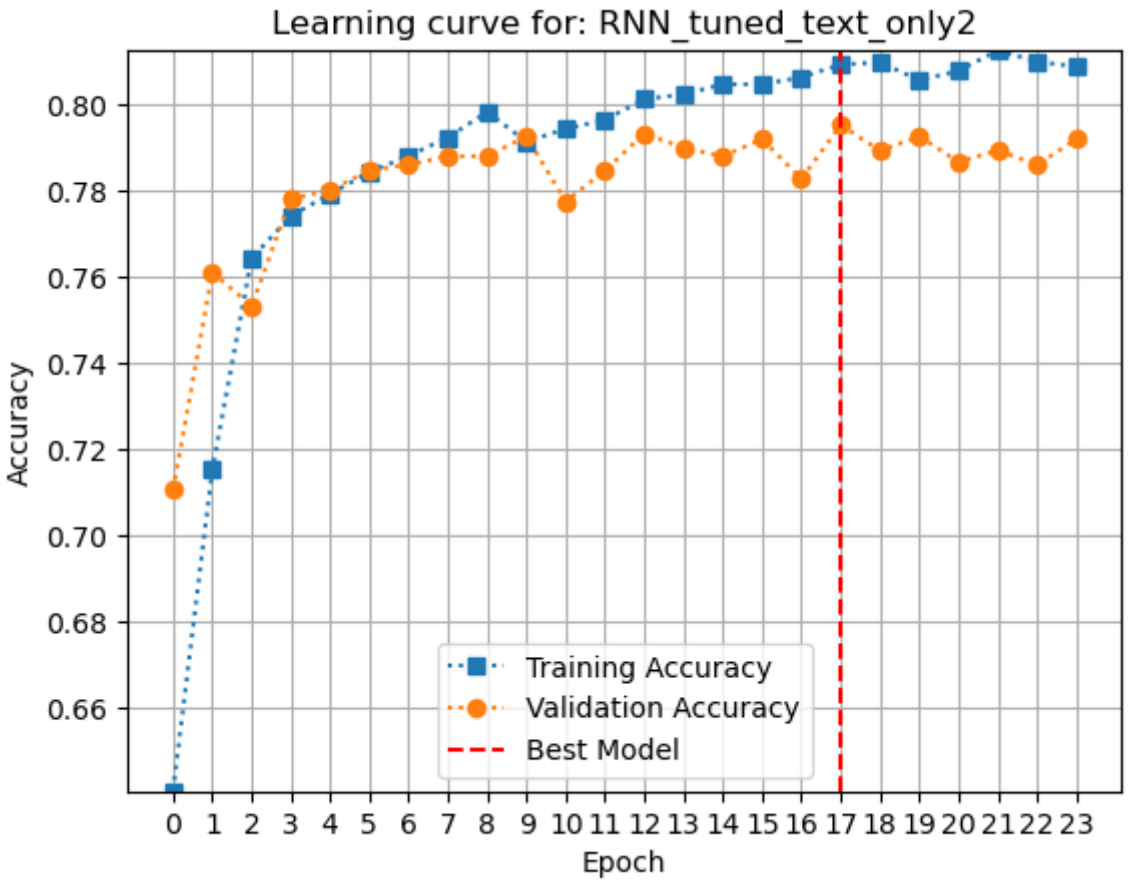
In [70]:

```
# Get top hyperparameters.
best_hp = tuner.get_best_hyperparameters()[0]
print("best hyperparameters:\n",best_hp.values)
# Build the model with the best hp.
rnn_model2 = build_model(best_hp)
rnn_model2.summary()
print('Training {} model on texts only...\n'.format(rnn_model2.name))
history = rnn_model2.fit(X_train_rnn,  y_train_rnn,
                    batch_size=64, epochs=150,
                    callbacks=callbacks,
                    validation_data=(X_cv_rnn, y_cv_rnn))
print('Done')
```

```
best hyperparameters:
 {'num_RNN_cell_layers': 2, 'dropout': 0.4, 'rnn_cell_type': 'GRU', 'tuner/epochs': 2, 'tuner/initial_epoc
h': 0, 'tuner/bracket': 4, 'tuner/round': 0}
Model: "RNN_tuned_text_only2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 51, 100)           1883000

 masking_1 (Masking)         (None, 51, 100)           0

 gru (GRU)                   (None, 51, 32)            12864

 gru_1 (GRU)                 (None, 32)                6336

 dense_1 (Dense)             (None, 16)                528

 dropout_1 (Dropout)         (None, 16)                0

 output (Dense)              (None, 1)                 17

=================================================================
Total params: 1,902,745
Trainable params: 19,745
Non-trainable params: 1,883,000
_________________________________________________________________
Training RNN_tuned_text_only2 model on texts only...

Epoch 1/150
96/96 [==============================] - 18s 105ms/step - loss: 0.6395 - accuracy: 0.6406 - val_loss: 0.58
16 - val_accuracy: 0.7104
Epoch 2/150
96/96 [==============================] - 8s 88ms/step - loss: 0.5647 - accuracy: 0.7151 - val_loss: 0.5088
- val_accuracy: 0.7610
Epoch 3/150
96/96 [==============================] - 8s 85ms/step - loss: 0.5193 - accuracy: 0.7642 - val_loss: 0.5482
- val_accuracy: 0.7531
Epoch 4/150
96/96 [==============================] - 8s 84ms/step - loss: 0.5065 - accuracy: 0.7741 - val_loss: 0.4878
- val_accuracy: 0.7781
Epoch 5/150
96/96 [==============================] - 8s 88ms/step - loss: 0.5018 - accuracy: 0.7791 - val_loss: 0.4776
- val_accuracy: 0.7800
Epoch 6/150
96/96 [==============================] - 8s 85ms/step - loss: 0.4871 - accuracy: 0.7842 - val_loss: 0.4720
- val_accuracy: 0.7846
Epoch 7/150
96/96 [==============================] - 8s 85ms/step - loss: 0.4827 - accuracy: 0.7880 - val_loss: 0.4725
- val_accuracy: 0.7859
Epoch 8/150
96/96 [==============================] - 8s 85ms/step - loss: 0.4793 - accuracy: 0.7923 - val_loss: 0.4675
- val_accuracy: 0.7879
Epoch 9/150
96/96 [==============================] - 8s 86ms/step - loss: 0.4653 - accuracy: 0.7982 - val_loss: 0.4797
- val_accuracy: 0.7879
Epoch 10/150
96/96 [==============================] - 8s 84ms/step - loss: 0.4718 - accuracy: 0.7913 - val_loss: 0.4680
- val_accuracy: 0.7925
Epoch 11/150
96/96 [==============================] - 8s 83ms/step - loss: 0.4635 - accuracy: 0.7943 - val_loss: 0.4684
- val_accuracy: 0.7774
Epoch 12/150
```

```
96/96 [==============================] - 8s 85ms/step - loss: 0.4616 - accuracy: 0.7964 - val_loss: 0.4642
- val_accuracy: 0.7846
Epoch 13/150
96/96 [==============================] - 8s 88ms/step - loss: 0.4628 - accuracy: 0.8011 - val_loss: 0.4636
- val_accuracy: 0.7932
Epoch 14/150
96/96 [==============================] - 8s 85ms/step - loss: 0.4583 - accuracy: 0.8023 - val_loss: 0.4611
- val_accuracy: 0.7899
Epoch 15/150
96/96 [==============================] - 8s 85ms/step - loss: 0.4545 - accuracy: 0.8046 - val_loss: 0.4577
- val_accuracy: 0.7879
Epoch 16/150
96/96 [==============================] - 8s 85ms/step - loss: 0.4526 - accuracy: 0.8046 - val_loss: 0.4627
- val_accuracy: 0.7919
Epoch 17/150
96/96 [==============================] - 8s 88ms/step - loss: 0.4537 - accuracy: 0.8062 - val_loss: 0.4560
- val_accuracy: 0.7827
Epoch 18/150
96/96 [==============================] - 8s 85ms/step - loss: 0.4428 - accuracy: 0.8092 - val_loss: 0.4521
- val_accuracy: 0.7951
Epoch 19/150
96/96 [==============================] - 8s 84ms/step - loss: 0.4486 - accuracy: 0.8097 - val_loss: 0.4503
- val_accuracy: 0.7892
Epoch 20/150
96/96 [==============================] - 8s 84ms/step - loss: 0.4455 - accuracy: 0.8056 - val_loss: 0.4533
- val_accuracy: 0.7925
Epoch 21/150
96/96 [==============================] - 8s 87ms/step - loss: 0.4411 - accuracy: 0.8077 - val_loss: 0.4573
- val_accuracy: 0.7866
Epoch 22/150
96/96 [==============================] - 8s 83ms/step - loss: 0.4375 - accuracy: 0.8125 - val_loss: 0.4631
- val_accuracy: 0.7892
Epoch 23/150
96/96 [==============================] - 8s 84ms/step - loss: 0.4341 - accuracy: 0.8099 - val_loss: 0.4557
- val_accuracy: 0.7859
Epoch 24/150
96/96 [==============================] - 8s 83ms/step - loss: 0.4343 - accuracy: 0.8087 - val_loss: 0.4691
- val_accuracy: 0.7919
Done
```

In [71]:
```
plot_learning_curves(history,rnn_model2.name)
```

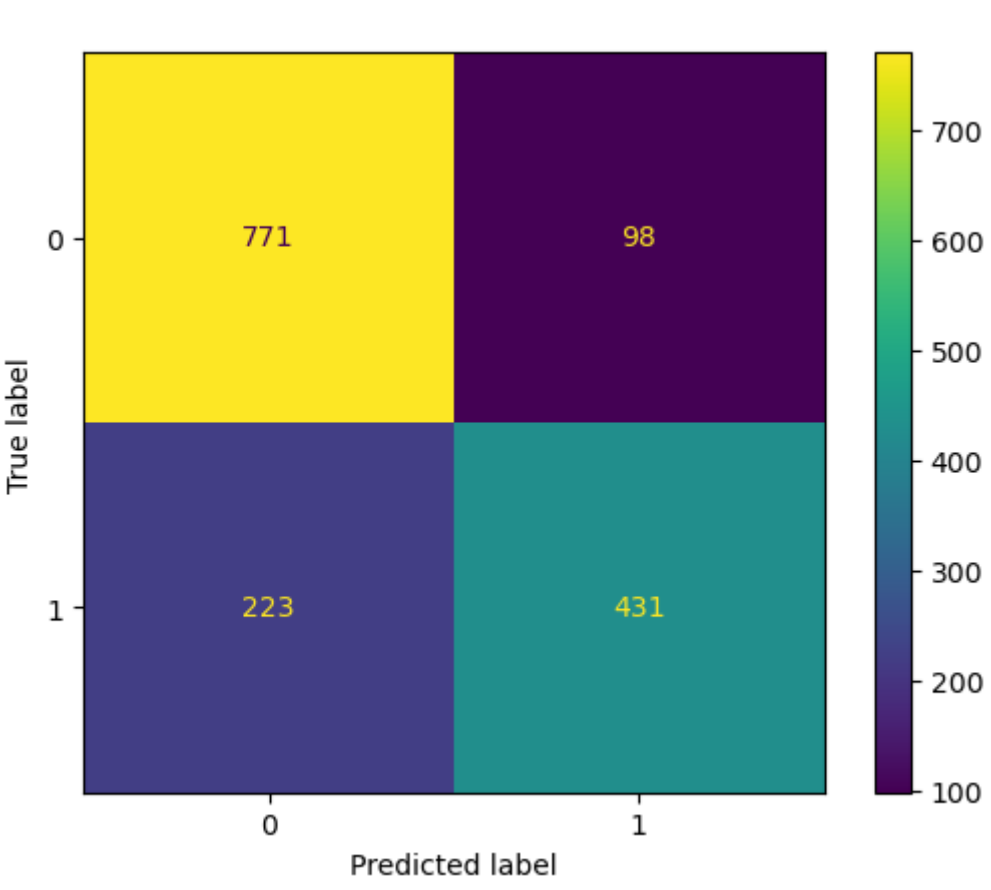


In [72]:
```
# Reload best model
rnn_model2 = load_model(model_save_path)
```

In [73]:
```
# !rm -rf /kaggle/working/Keras_Hypertuning # Remove model (use this to restart keras tuner)
```

In [74]:
```
print('Computing model predictions...')
y_prob_cv = rnn_model2.predict(X_cv_rnn)
y_pred_cv = np.round(y_prob_cv)
acc = display_conf_max_and_acc(y_cv,y_pred_cv,rnn_model2.name)
```

```
Computing model predictions...
48/48 [==============================] - 2s 16ms/step
Cross validation accuracy of model `RNN_tuned_text_only2` is 78.92%
```



In [75]:
```
# Recursive neural network LSTM model using text only
model = {'model_name': rnn_model2.name,
    'cv_acc': acc,
        'model': rnn_model2}
models.append(model)
print("Model: {}\nCross Val accuracy: {:.2%}".format(model['model_name'],model['cv_acc']))
```

```
Model: RNN_tuned_text_only2
Cross Val accuracy: 78.92%
```

In [76]:
```
pd.DataFrame(models)
```

Out[76]:

	model_name	cv_acc	model
0	Baseline - Predict most frequent category	0.570584	NaN
1	Gradient Boosting using keyword and location only	0.747209	([DecisionTreeRegressor(criterion='friedman_ms...
2	RNN_single_LSTM_cell_only_text	0.799081	<keras.engine.sequential.Sequential object at ...
3	RNN_tuned_text_only2	0.789232	<keras.engine.sequential.Sequential object at ...

4.4 RNN model with one LSTM layer based on text, keyword and location

Prepare inputs

We can use Keras functional API (https://keras.io/guides/functional_api/https://keras.io/guides/functional_api/) to accept integers from `text` mapping as input to an LSTM and regular dense layer to process `location` and `keyword`. However, this time it is more appropriate to use one-hot encoding for `keyword` and `location`

There are two many locations that appear only once in the training data. For our One hot encoding of this column, we will represent only values that are seen at least 2 times in the training data. The rest will be mapped to a vector of zeros, as well as any unknown data.

In [77]:
```
# Select only frequent values in location column
min_loc_freq = 2
s = df_train['location'].value_counts()
frequent_location_values = (s[s>=2]).index.to_list()
print("Out of {} 'location' different values, we are keeping {} values that appear at least {} times in the fu
ll training set."
      .format(len(s),len(frequent_location_values),min_loc_freq))
```

```
Out of 3342 'location' different values, we are keeping 524 values that appear at least 2 times in the ful
l training set.
```

In [78]:
```
cols_to_encode = ['keyword','location']
ct = ColumnTransformer(
    [("one_hot_encoder_keyword",
      preprocessing.OneHotEncoder(handle_unknown='ignore',sparse=False),
      ['keyword']),
    ("one_hot_encoder_location",
      preprocessing.OneHotEncoder(categories = [frequent_location_values], handle_unknown='ignore'),
     ['location'])
    ])

ct = ct.fit(df_train[cols_to_encode])
X_train_rnn2_kl = ct.transform(df_train_sub[cols_to_encode])
X_cv_rnn2_kl = ct.transform(df_cv_sub[cols_to_encode])
X_test_rnn2_kl = ct.transform(df_test[cols_to_encode])

num_kl_columns = X_train_rnn2_kl.shape[1]
print('keyword and location have been mapped to {} columns with one hot encoding.'
      .format(num_kl_columns))

print('Input data shapes: \n X_train_rnn (text): {}\n X_train_rnn2_kl (keyword and location): {} \n target:
{}'
      .format(X_train_rnn.shape,X_train_rnn2_kl.shape,y_train_rnn.shape))
```

```
keyword and location have been mapped to 746 columns with one hot encoding.
Input data shapes:
 X_train_rnn (text): (6090, 51)
 X_train_rnn2_kl (keyword and location): (6090, 746)
 target: (6090, 1)
```

Create model using functional API from Keras

In [79]:

```
def functional_keras_model(name='my_Functional_model'):
    txt_inputs = Input(shape=(model_word_slots,), name='text')
    kl_inputs = Input(shape=(num_kl_columns,), name="keyword_location")


    x = layers.Embedding(input_dim=num_words, # number of words in vocabulary
                         input_length=model_word_slots, # number of words in input
                         output_dim=embed_dim, # glove vector length
                         weights=[embedding_matrix], # precomputed weights from glove
                         trainable=False,
                         mask_zero=True)(txt_inputs)

    x = layers.Masking(mask_value=0.0)(x) # map unknown values to 0

    x = layers.LSTM(32, return_sequences=False)(x) # RNN LSTM layer

    x = layers.Dense(32, activation='relu')(x) # Fully connected layer
    x = layers.LayerNormalization()(x)
    x = layers.Dropout(0.5)(x) # Dropout for regularization

    y = kl_inputs
    for dense_layer_dim in [128, 64,32]:
        y = layers.Dense(dense_layer_dim, activation='relu')(y)
        y = layers.LayerNormalization()(y)
        y = layers.Dropout(0.3)(y)

    y = layers.Flatten()(y)

    xy = layers.concatenate([x, y],name='merge_text_and_keywords_locations')

    output = layers.Dense(1, activation='sigmoid',name='predict_label')(xy) # Output layer

    model = Model(inputs=[txt_inputs,kl_inputs],
                  outputs=output,
                  name=name)
    return model

rnn_model_func1 = functional_keras_model("RNN_text_plus_keyword_and_location")
rnn_model_func1.compile(
    optimizer=Nadam(learning_rate=5e-4), loss='binary_crossentropy', metrics=['accuracy'])
rnn_model_func1.summary()
```

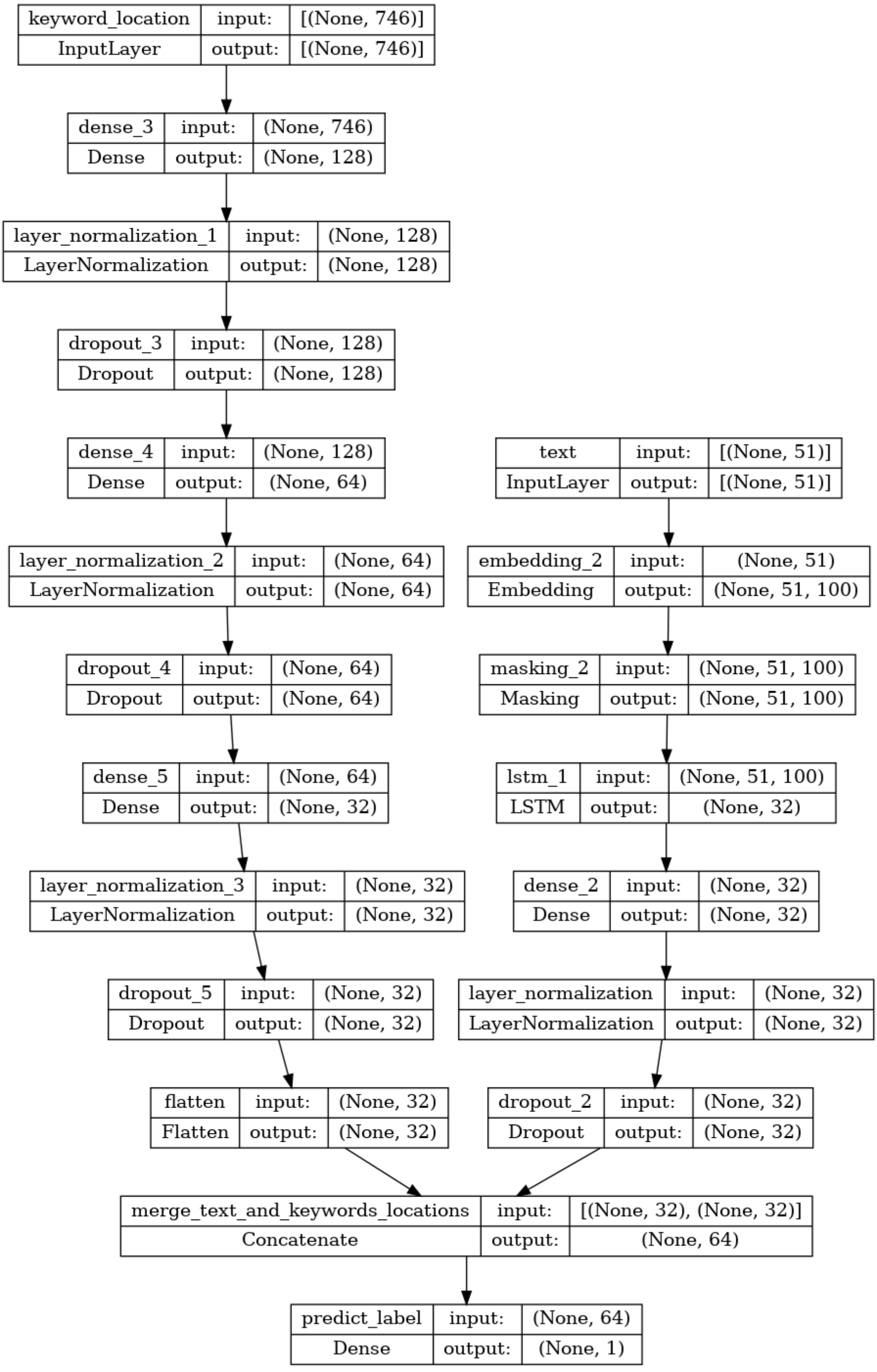
```
Model: "RNN_text_plus_keyword_and_location"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 keyword_location (InputLayer)  [(None, 746)]        0           []

 dense_3 (Dense)                (None, 128)          95616       ['keyword_location[0][0]']

 layer_normalization_1 (LayerNo  (None, 128)         256         ['dense_3[0][0]']
 rmalization)

 dropout_3 (Dropout)            (None, 128)          0           ['layer_normalization_1[0][0]']

 text (InputLayer)              [(None, 51)]         0           []

 dense_4 (Dense)                (None, 64)           8256        ['dropout_3[0][0]']

 embedding_2 (Embedding)        (None, 51, 100)      1883000     ['text[0][0]']

 layer_normalization_2 (LayerNo  (None, 64)          128         ['dense_4[0][0]']
 rmalization)

 masking_2 (Masking)            (None, 51, 100)      0           ['embedding_2[0][0]']

 dropout_4 (Dropout)            (None, 64)           0           ['layer_normalization_2[0][0]']

 lstm_1 (LSTM)                  (None, 32)           17024       ['masking_2[0][0]']

 dense_5 (Dense)                (None, 32)           2080        ['dropout_4[0][0]']

 dense_2 (Dense)                (None, 32)           1056        ['lstm_1[0][0]']

 layer_normalization_3 (LayerNo  (None, 32)          64          ['dense_5[0][0]']
 rmalization)

 layer_normalization (LayerNorm  (None, 32)          64          ['dense_2[0][0]']
 alization)

 dropout_5 (Dropout)            (None, 32)           0           ['layer_normalization_3[0][0]']

 dropout_2 (Dropout)            (None, 32)           0           ['layer_normalization[0][0]']

 flatten (Flatten)              (None, 32)           0           ['dropout_5[0][0]']

 merge_text_and_keywords_locati  (None, 64)          0           ['dropout_2[0][0]',
 ons (Concatenate)                                                'flatten[0][0]']

 predict_label (Dense)          (None, 1)            65          ['merge_text_and_keywords_locatio
                                                                 ns[0][0]']

==================================================================================================
Total params: 2,007,609
Trainable params: 124,609
Non-trainable params: 1,883,000
__________________________________________________________________________________________________
```

In [80]:

```
# Create callbacks
models_path = '../models'
model_save_path = os.path.join(models_path,rnn_model_func1.name+'.h5')
callbacks = [EarlyStopping(monitor='val_loss', patience=5),
             ModelCheckpoint(model_save_path, save_best_only=True,
                             save_weights_only=False)]
```

In [81]:
```
plot_model(rnn_model_func1, rnn_model_func1.name + ".png", show_shapes=True)
```

Out[81]:

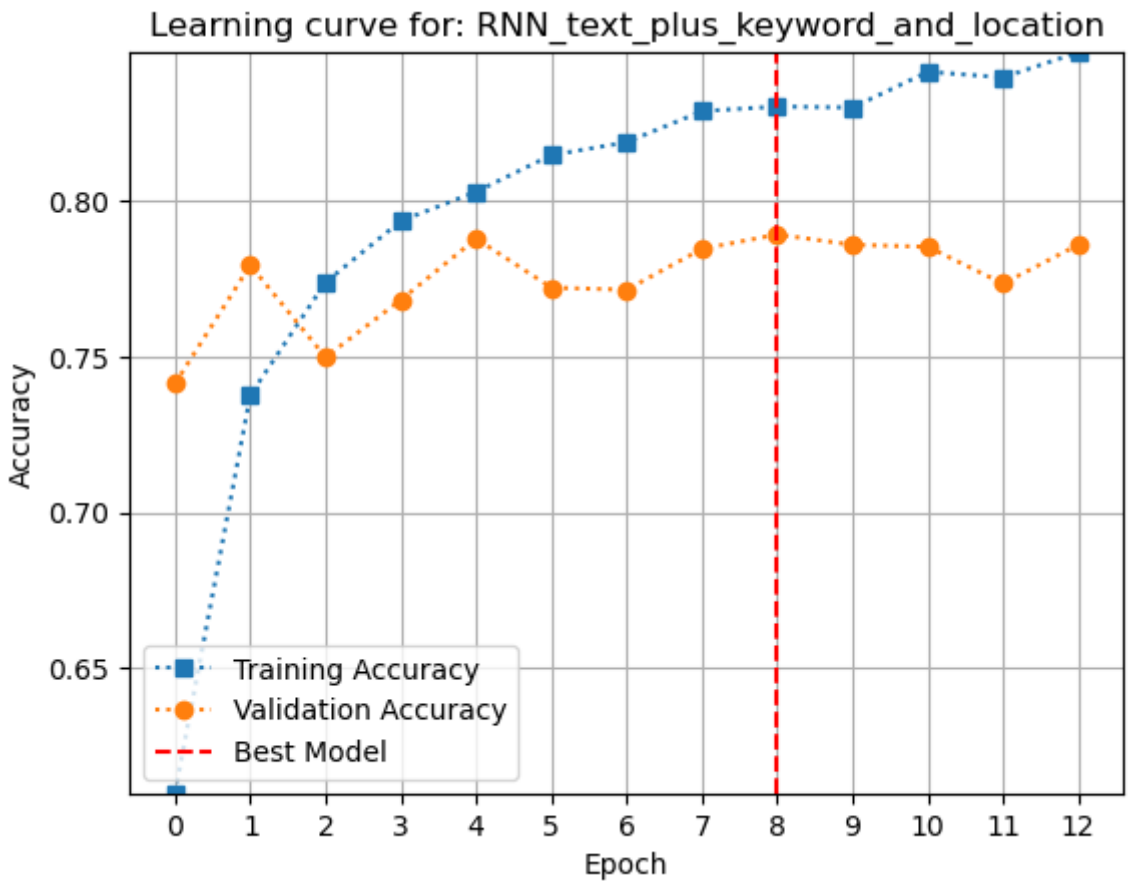


Train model

In [82]:
```
print('Training {} model on texts only...\n'.format(rnn_model_func1.name))
history = rnn_model_func1.fit(x={'text':X_train_rnn, 'keyword_location':X_train_rnn2_kl},  y = y_train_rnn,
                    batch_size=64, epochs=50,
                    callbacks=callbacks,
                    validation_data=({'text':X_cv_rnn,'keyword_location': X_cv_rnn2_kl}, y_cv_rnn)
                            )
print('...Done')
```

```
Training RNN_text_plus_keyword_and_location model on texts only...

Epoch 1/50
96/96 [==============================] - 12s 56ms/step - loss: 0.7597 - accuracy: 0.6097 - val_loss: 0.554
1 - val_accuracy: 0.7413
Epoch 2/50
96/96 [==============================] - 4s 40ms/step - loss: 0.5566 - accuracy: 0.7376 - val_loss: 0.4943
- val_accuracy: 0.7794
Epoch 3/50
96/96 [==============================] - 4s 37ms/step - loss: 0.4982 - accuracy: 0.7739 - val_loss: 0.5149
- val_accuracy: 0.7498
Epoch 4/50
96/96 [==============================] - 4s 38ms/step - loss: 0.4738 - accuracy: 0.7936 - val_loss: 0.4933
- val_accuracy: 0.7682
Epoch 5/50
96/96 [==============================] - 4s 39ms/step - loss: 0.4490 - accuracy: 0.8030 - val_loss: 0.4888
- val_accuracy: 0.7879
Epoch 6/50
96/96 [==============================] - 4s 39ms/step - loss: 0.4315 - accuracy: 0.8148 - val_loss: 0.4839
- val_accuracy: 0.7722
Epoch 7/50
96/96 [==============================] - 4s 37ms/step - loss: 0.4173 - accuracy: 0.8189 - val_loss: 0.4921
- val_accuracy: 0.7715
Epoch 8/50
96/96 [==============================] - 4s 38ms/step - loss: 0.4028 - accuracy: 0.8289 - val_loss: 0.4794
- val_accuracy: 0.7846
Epoch 9/50
96/96 [==============================] - 4s 40ms/step - loss: 0.3947 - accuracy: 0.8304 - val_loss: 0.4814
- val_accuracy: 0.7892
Epoch 10/50
96/96 [==============================] - 4s 38ms/step - loss: 0.3915 - accuracy: 0.8300 - val_loss: 0.4886
- val_accuracy: 0.7859
Epoch 11/50
96/96 [==============================] - 4s 37ms/step - loss: 0.3784 - accuracy: 0.8415 - val_loss: 0.4885
- val_accuracy: 0.7853
Epoch 12/50
96/96 [==============================] - 4s 39ms/step - loss: 0.3730 - accuracy: 0.8397 - val_loss: 0.5038
- val_accuracy: 0.7735
Epoch 13/50
96/96 [==============================] - 4s 38ms/step - loss: 0.3669 - accuracy: 0.8478 - val_loss: 0.4867
- val_accuracy: 0.7859
...Done
```

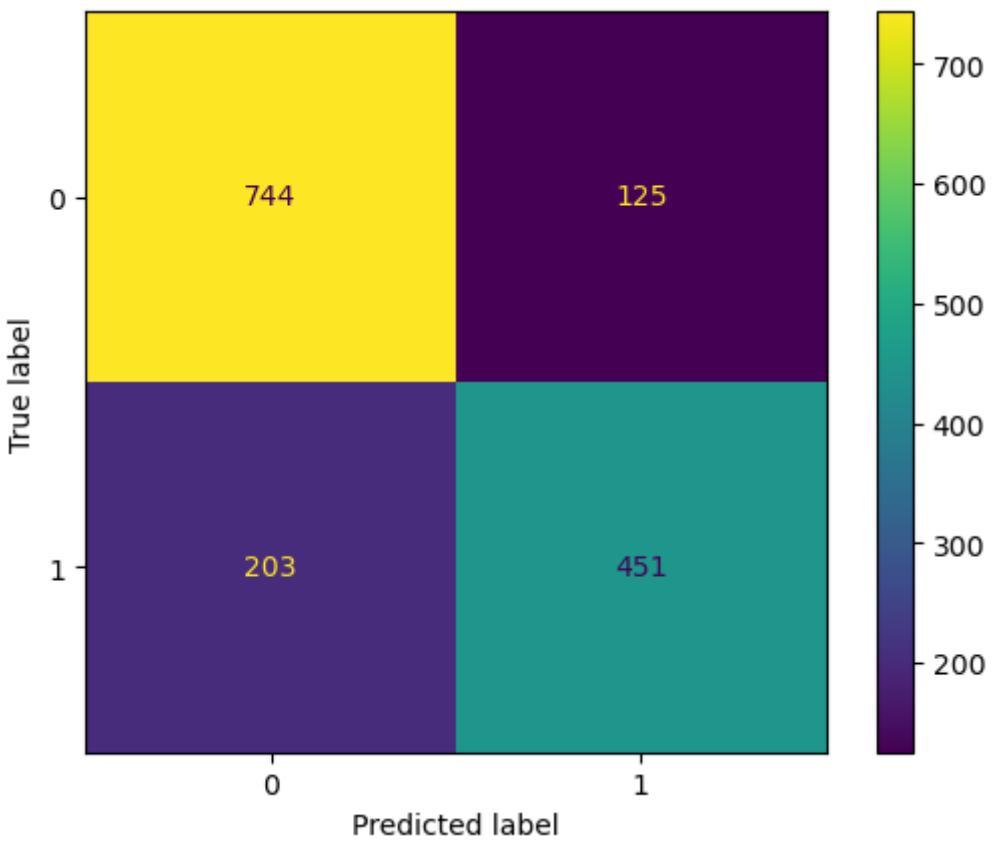
In [83]:
```
plot_learning_curves(history,rnn_model_func1.name)
```



In [84]:
```
# Reload best model
rnn_model_func1 = load_model(model_save_path)
```

In [85]:
```
print('Computing model predictions...')
y_prob_cv = rnn_model_func1.predict({'text':X_cv_rnn,'keyword_location': X_cv_rnn2_kl})
y_pred_cv = np.round(y_prob_cv)
acc = display_conf_max_and_acc(y_cv,y_pred_cv,rnn_model_func1.name)
```

```
Computing model predictions...
48/48 [==============================] - 2s 11ms/step
Cross validation accuracy of model `RNN_text_plus_keyword_and_location` is 78.46%
```



In [86]:
```
# Recursive neural network LSTM model using text only
model = {'model_name': rnn_model_func1.name,
    'cv_acc': acc,
        'model': rnn_model_func1}

models.append(model)
print("Model: {}\nCross Val accuracy: {:.2%}".format(model['model_name'],model['cv_acc']))
```

```
Model: RNN_text_plus_keyword_and_location
Cross Val accuracy: 78.46%
```

In [87]:
```
pd.DataFrame(models)
```

Out[87]:

	model_name	cv_acc	model
0	Baseline - Predict most frequent category	0.570584	NaN
1	Gradient Boosting using keyword and location only	0.747209	([DecisionTreeRegressor(criterion='friedman_ms...
2	RNN_single_LSTM_cell_only_text	0.799081	<keras.engine.sequential.Sequential object at ...
3	RNN_tuned_text_only2	0.789232	<keras.engine.sequential.Sequential object at ...
4	RNN_text_plus_keyword_and_location	0.784636	<keras.engine.functional.Functional object at ...

Discussion

- We are training on accuracy because the classes are balanced.
- Gradient boosting is able to reach around 73% accuracy using only keyword and location columns. Trees don't really need one hot encoding for categorical variables.
- Using the current word embedding, RNNs based on text only and GRU or LSTM cells can reach around 79% accuracy. Keras tuner found better results with LSTM than with GRU cells.
- Using an RNN that can handle both text, keyword an location columns only provided a small increase in accuracy. I tried other hyperparameter values but did not run keras tuner.

Further work

- Maybe we could use data augmentation by concatenating tweets with same label.
- Using a different word embedding may prove useful as well. We could also train models from different word embedding and use bagging to increase accuracy.
- Use RNN as a feature extractor and pass it to gradient boosting with the location and keyword columns.

Select best model

In [88]:
```
best_model_from_val_acc = max(models, key=lambda x: x['cv_acc'])
print("Best Model: {}\nCross Val accuracy: {:.2%}".format(best_model_from_val_acc['model_name'],best_model_fro
m_val_acc['cv_acc']))

best_model = best_model_from_val_acc['model']
```

```
Best Model: RNN_single_LSTM_cell_only_text
Cross Val accuracy: 79.91%
```

5. Submit results

In [89]:

```
print('Computing submission file')
# Prepare submission file
df_submit = df_test.copy()

try:
    # Keras RNN text only
    df_submit['target'] = np.round(best_model.predict(X_test_rnn),0).astype(np.int64)

except:
    try:
        # Keras RNN text + keyword + location
        df_submit['target'] = np.round(best_model.predict({'text':X_test_rnn,'keyword_location': X_test_rnn2_k
l}),0).astype(np.int64)
    except:
        # Sklearn Gradient Boosting
        df_submit['target'] = np.round(best_model.predict(X_test_gb),0).astype(np.int64)


df_submit = df_submit.loc[:,['id','target']]
df_submit
```

```
Computing submission file
102/102 [==============================] - 1s 10ms/step
```

Out[89]:

	id	target
0	0	1
1	2	1
2	3	0
3	9	1
4	11	1
...	...	...
3258	10861	1
3259	10865	0
3260	10868	1
3261	10874	1
3262	10875	0

3263 rows × 2 columns

In [90]:

```
# Competition Submission
df_submit.to_csv('submission.csv', index=False)
```