```
import gc
import re
import string
import operator
from collections import defaultdict

import numpy as np
import pandas as pd
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

import matplotlib.pyplot as plt
import seaborn as sns

import tokenization
from wordcloud import STOPWORDS

from sklearn.model_selection import StratifiedKFold, StratifiedShuffleSplit
from sklearn.metrics import precision_score, recall_score, f1_score

import tensorflow as tf
import tensorflow_hub as hub
from tensorflow import keras
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.layers import Dense, Input, Dropout, GlobalAveragePooling1D
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, Callback

SEED = 1337

df_train = pd.read_csv('../input/nlp-getting-started/train.csv', dtype={'id': np.int16, 'target':
df_test = pd.read_csv('../input/nlp-getting-started/test.csv', dtype={'id': np.int16})

print('Training Set Shape = {}'.format(df_train.shape))
print('Training Set Memory Usage = {:.2f} MB'.format(df_train.memory_usage().sum() / 1024**2))
print('Test Set Shape = {}'.format(df_test.shape))
print('Test Set Memory Usage = {:.2f} MB'.format(df_test.memory_usage().sum() / 1024**2))
```

```
Training Set Shape = (7613, 5)
Training Set Memory Usage = 0.20 MB
Test Set Shape = (3263, 4)
Test Set Memory Usage = 0.08 MB
```

1. Keyword and Location

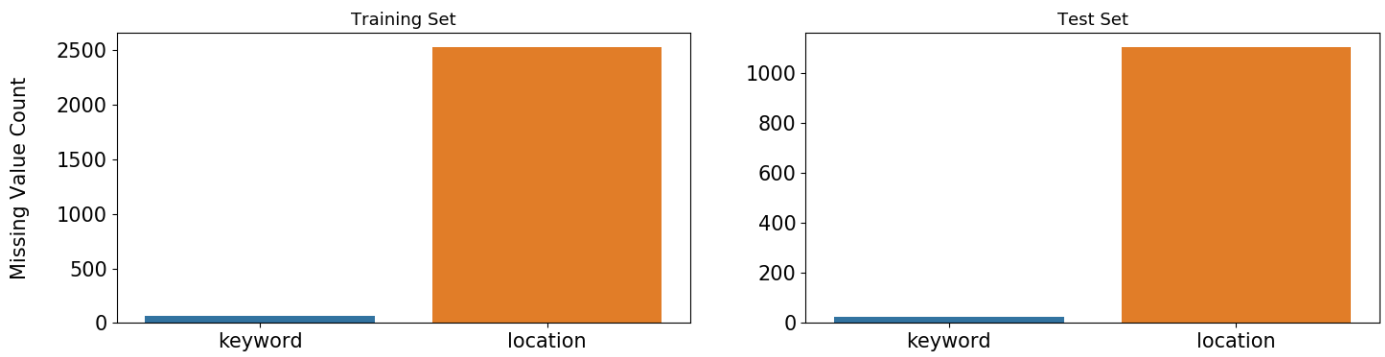
1.1 Missing Values

Both training and test set have same ratio of missing values in `keyword` and `location`.

- **0.8%** of `keyword` is missing in both training and test set
- **33%** of `location` is missing in both training and test set

Since missing value ratios between training and test set are too close, **they are most probably taken from the same sample**. Missing values in those features are filled with `no_keyword` and `no_location` respectively.

```
missing_cols = ['keyword', 'location']

fig, axes = plt.subplots(ncols=2, figsize=(17, 4), dpi=100)

sns.barplot(x=df_train[missing_cols].isnull().sum().index, y=df_train[missing_cols].isnull().sum()
sns.barplot(x=df_test[missing_cols].isnull().sum().index, y=df_test[missing_cols].isnull().sum().v

axes[0].set_ylabel('Missing Value Count', size=15, labelpad=20)
axes[0].tick_params(axis='x', labelsize=15)
axes[0].tick_params(axis='y', labelsize=15)
axes[1].tick_params(axis='x', labelsize=15)
axes[1].tick_params(axis='y', labelsize=15)

axes[0].set_title('Training Set', fontsize=13)
axes[1].set_title('Test Set', fontsize=13)

plt.show()

for df in [df_train, df_test]:
    for col in ['keyword', 'location']:
        df[col] = df[col].fillna(f'no_{col}')
```



1.2 Cardinality and Target Distribution

Locations are not automatically generated, they are user inputs. That's why `location` is very dirty and there are too many unique values in it. It shouldn't be used as a feature.

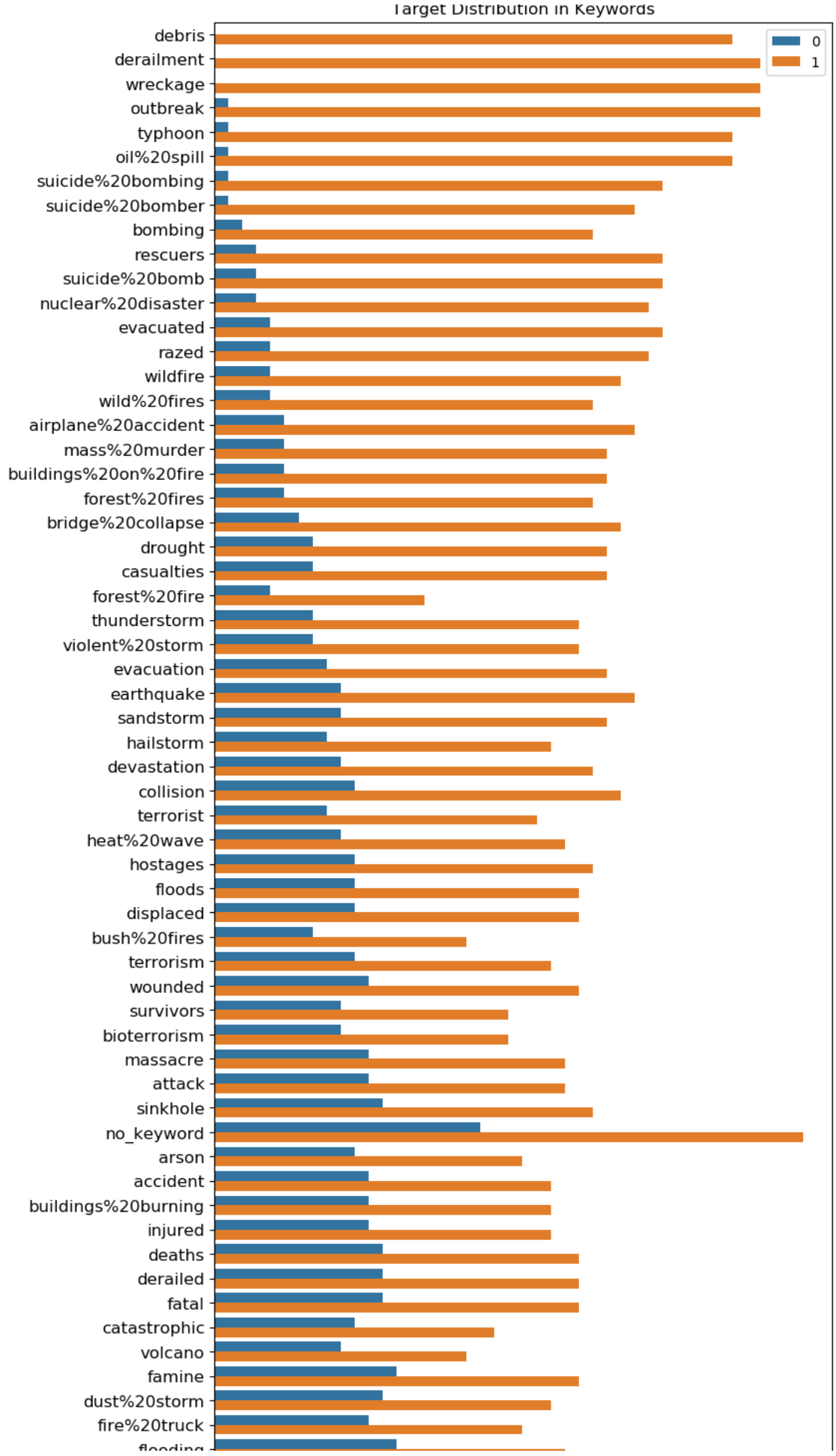
Fortunately, there is signal in `keyword` because some of those words can only be used in one context. Keywords have very different tweet counts and target means. `keyword` can be used as a feature by itself or as a word added to the text. Every single keyword in training set exists in test set. If training and test set are from the same sample, it is also possible to use target encoding on `keyword`.

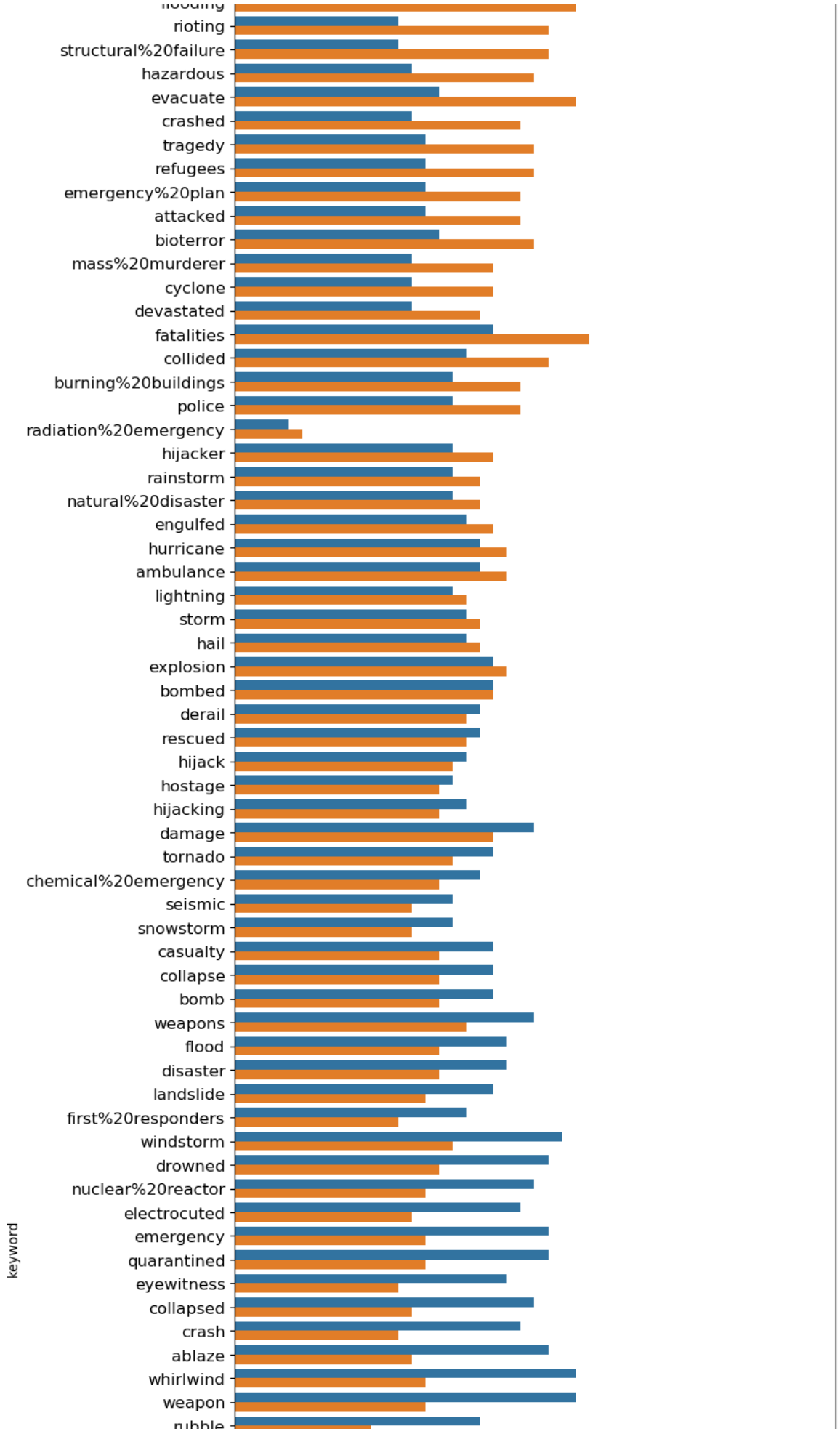
```
print(f'Number of unique values in keyword = {df_train["keyword"].nunique()} (Training) - {df_test
print(f'Number of unique values in location = {df_train["location"].nunique()} (Training) - {df_te
```

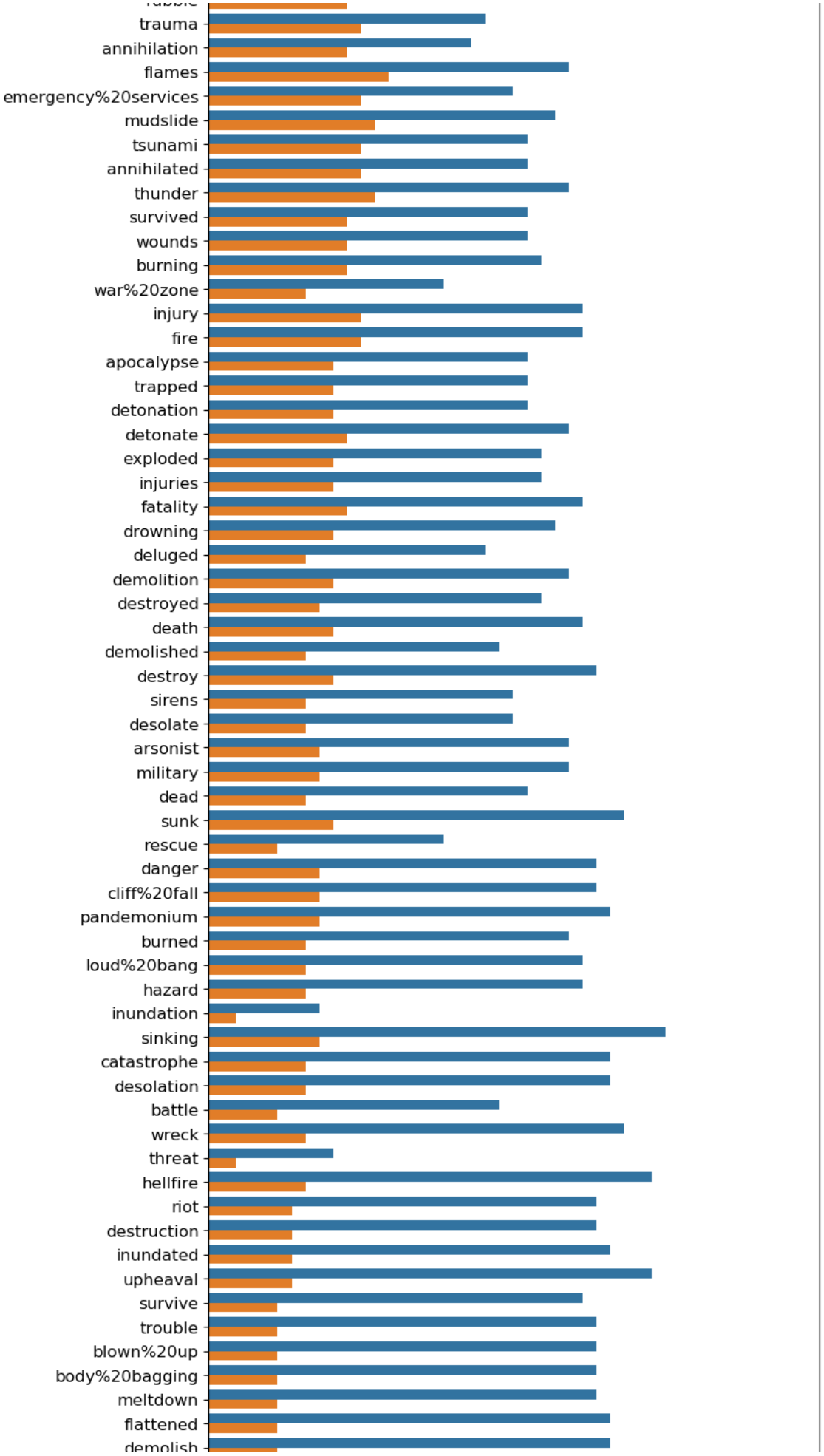
```
Number of unique values in keyword = 222 (Training) - 222 (Test)
Number of unique values in location = 3342 (Training) - 1603 (Test)
```

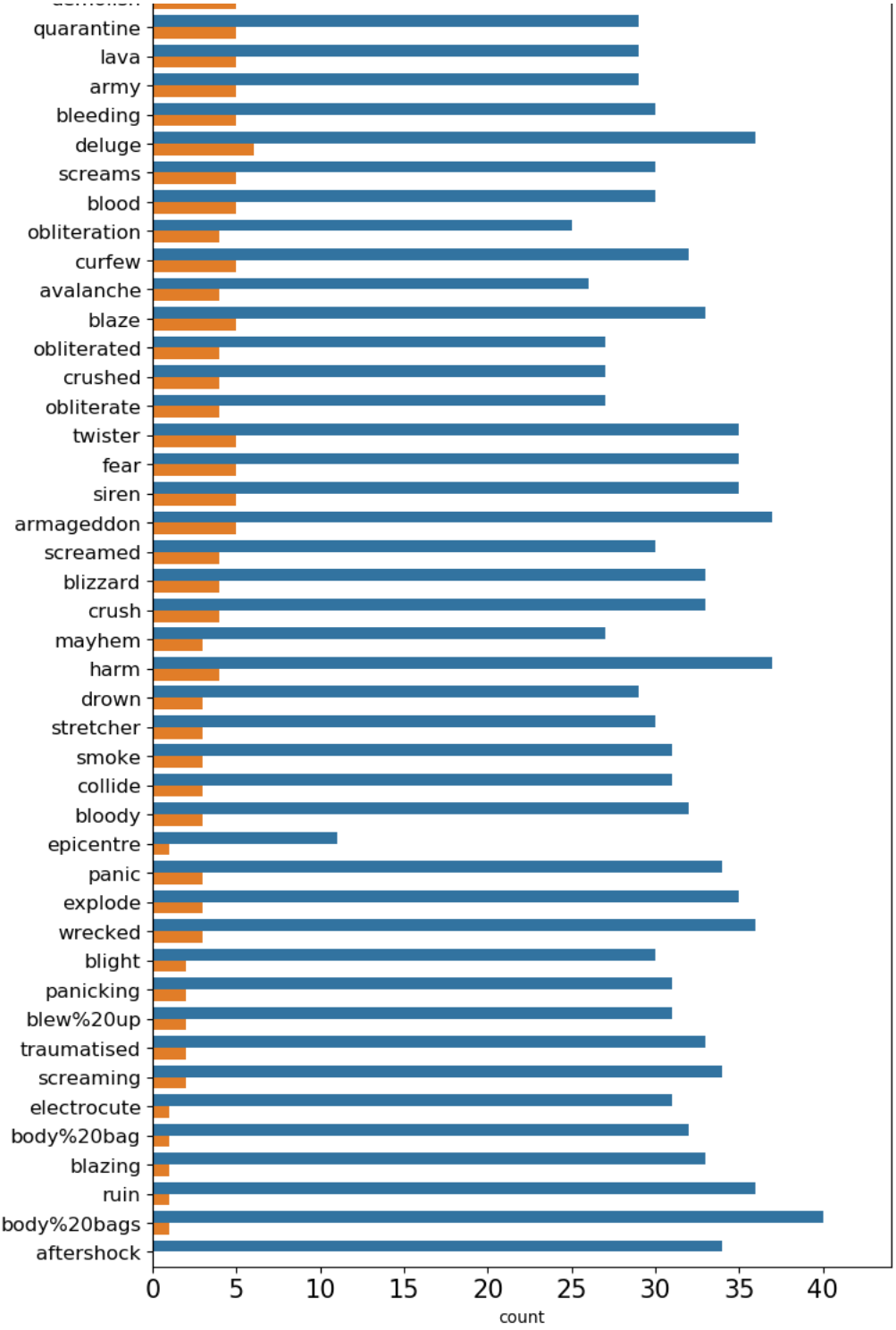
```
df_train['target_mean'] = df_train.groupby('keyword')['target'].transform('mean')

fig = plt.figure(figsize=(8, 72), dpi=100)

sns.countplot(y=df_train.sort_values(by='target_mean', ascending=False)['keyword'],
```

```
                hue=df_train.sort_values(by='target_mean', ascending=False)['target'])

    plt.tick_params(axis='x', labelsize=15)
    plt.tick_params(axis='y', labelsize=12)
    plt.legend(loc=1)
    plt.title('Target Distribution in Keywords')

    plt.show()

    df_train.drop(columns=['target_mean'], inplace=True)
```


Target Distribution in Keywords
0
1
debris
derailment
wreckage
outbreak
typhoon
oil%20spill
suicide%20bombing
suicide%20bomber
bombing
rescuers
suicide%20bomb
nuclear%20disaster
evacuated
razed
wildfire
wild%20fires
airplane%20accident
mass%20murder
buildings%20on%20fire
forest%20fires
bridge%20collapse
drought
casualties
forest%20fire
thunderstorm
violent%20storm
evacuation
earthquake
sandstorm
hailstorm
devastation
collision
terrorist
heat%20wave
hostages
floods
displaced
bush%20fires
terrorism
wounded
survivors
bioterrorism
massacre
attack
sinkhole
no_keyword
arson
accident
buildings%20burning
injured
deaths
derailed
fatal
catastrophic
volcano
famine
dust%20storm
fire%20truck


rioting
structural%20failure
hazardous
evacuate
crashed
tragedy
refugees
emergency%20plan
attacked
bioterror
mass%20murderer
cyclone
devastated
fatalities
collided
burning%20buildings
police
radiation%20emergency
hijacker
rainstorm
natural%20disaster
engulfed
hurricane
ambulance
lightning
storm
hail
explosion
bombed
derail
rescued
hijack
hostage
hijacking
damage
tornado
chemical%20emergency
seismic
snowstorm
casualty
collapse
bomb
weapons
flood
disaster
landslide
first%20responders
windstorm
drowned
nuclear%20reactor
electrocuted
emergency
quarantined
eyewitness
collapsed
crash
ablaze
whirlwind
weapon
keyword


trauma
annihilation
flames
emergency%20services
mudslide
tsunami
annihilated
thunder
survived
wounds
burning
war%20zone
injury
fire
apocalypse
trapped
detonation
detonate
exploded
injuries
fatality
drowning
deluged
demolition
destroyed
death
demolished
destroy
sirens
desolate
arsonist
military
dead
sunk
rescue
danger
cliff%20fall
pandemonium
burned
loud%20bang
hazard
inundation
sinking
catastrophe
desolation
battle
wreck
threat
hellfire
riot
destruction
inundated
upheaval
survive
trouble
blown%20up
body%20bagging
meltdown
flattened
demolish


quarantine
lava
army
bleeding
deluge
screams
blood
obliteration
curfew
avalanche
blaze
obliterated
crushed
obliterate
twister
fear
siren
armageddon
screamed
blizzard
crush
mayhem
harm
drown
stretcher
smoke
collide
bloody
epicentre
panic
explode
wrecked
blight
panicking
blew%20up
traumatised
screaming
electrocute
body%20bag
blazing
ruin
body%20bags
aftershock
0
5
10
15
20
25
30
35
40
count

2. Meta Features

Distributions of meta features in classes and datasets can be helpful to identify disaster tweets. It looks like disaster tweets are written in a more formal way with longer words compared to non-disaster tweets because most of them are coming from news agencies. Non-disaster tweets have more typos than disaster tweets because they are coming from individual users. The meta features used for the analysis are;

- `word_count` number of words in text
- `unique_word_count` number of unique words in text
- `stop_word_count` number of stop words in text
- `url_count` number of urls in text
- `mean_word_length` average character count in words
- `char_count` number of characters in text
- `punctuation_count` number of punctuations in text
- `hashtag_count` number of hashtags (#) in text
- `mention_count` number of mentions (@) in text

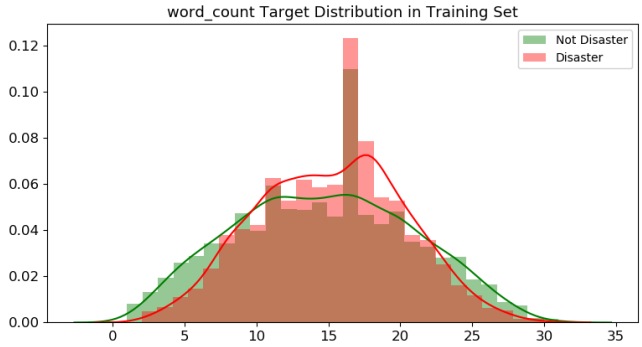
```
# word_count
df_train['word_count'] = df_train['text'].apply(lambda x: len(str(x).split()))
df_test['word_count'] = df_test['text'].apply(lambda x: len(str(x).split()))

# unique_word_count
df_train['unique_word_count'] = df_train['text'].apply(lambda x: len(set(str(x).split())))
df_test['unique_word_count'] = df_test['text'].apply(lambda x: len(set(str(x).split())))

# stop_word_count
df_train['stop_word_count'] = df_train['text'].apply(lambda x: len([w for w in str(x).lower().spli
df_test['stop_word_count'] = df_test['text'].apply(lambda x: len([w for w in str(x).lower().split(

# url_count
df_train['url_count'] = df_train['text'].apply(lambda x: len([w for w in str(x).lower().split() if
df_test['url_count'] = df_test['text'].apply(lambda x: len([w for w in str(x).lower().split() if '

# mean_word_length
df_train['mean_word_length'] = df_train['text'].apply(lambda x: np.mean([len(w) for w in str(x).sp
df_test['mean_word_length'] = df_test['text'].apply(lambda x: np.mean([len(w) for w in str(x).spli

# char_count
df_train['char_count'] = df_train['text'].apply(lambda x: len(str(x)))
df_test['char_count'] = df_test['text'].apply(lambda x: len(str(x)))

# punctuation_count
df_train['punctuation_count'] = df_train['text'].apply(lambda x: len([c for c in str(x) if c in st
df_test['punctuation_count'] = df_test['text'].apply(lambda x: len([c for c in str(x) if c in stri

# hashtag_count
df_train['hashtag_count'] = df_train['text'].apply(lambda x: len([c for c in str(x) if c == '#']))
df_test['hashtag_count'] = df_test['text'].apply(lambda x: len([c for c in str(x) if c == '#']))

# mention_count
df_train['mention_count'] = df_train['text'].apply(lambda x: len([c for c in str(x) if c == '@']))
df_test['mention_count'] = df_test['text'].apply(lambda x: len([c for c in str(x) if c == '@']))
```

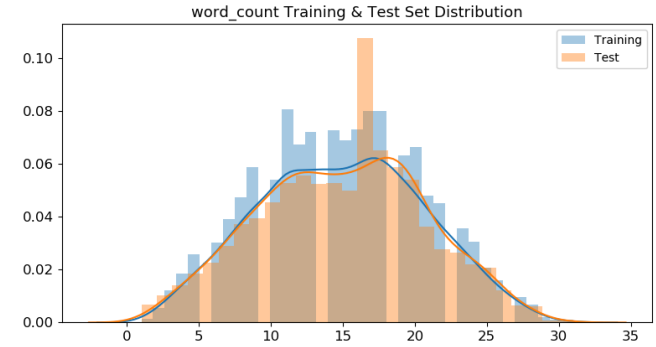
All of the meta features have very similar distributions in training and test set which also proves that training and test set are taken from the same sample.

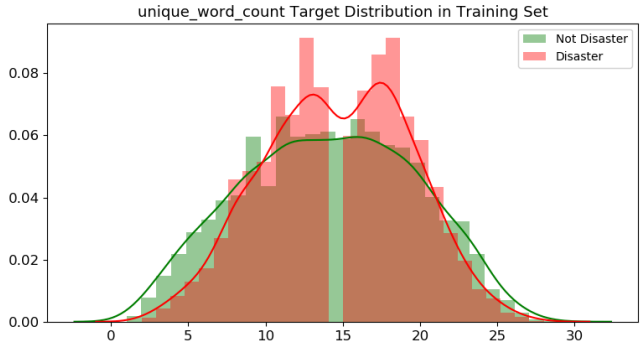
All of the meta features have information about target as well, but some of them are not good enough such as `url_count`, `hashtag_count` and `mention_count`.

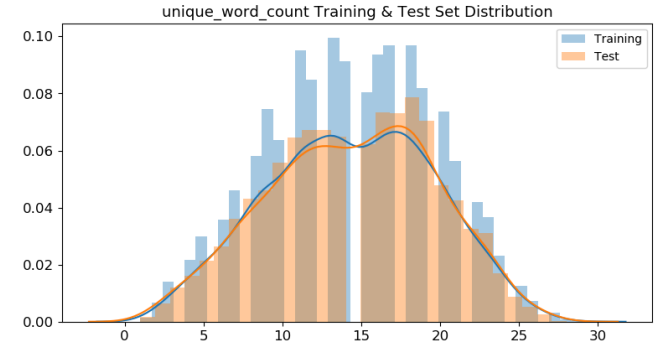
On the other hand, `word_count`, `unique_word_count`, `stop_word_count`, `mean_word_length`, `char_count`, `punctuation_count` have very different distributions for disaster and non-disaster tweets. Those features might be useful in models.

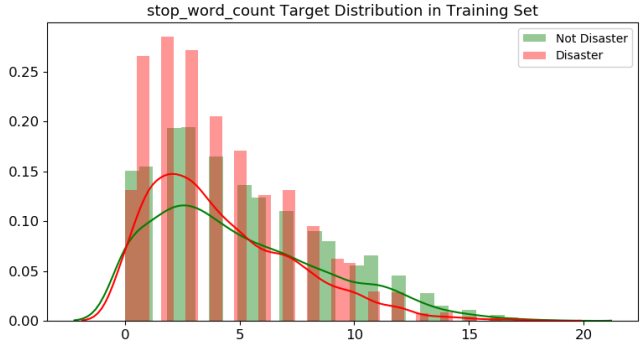
```
METAFEATURES = ['word_count', 'unique_word_count', 'stop_word_count', 'url_count', 'mean_word_leng
                'char_count', 'punctuation_count', 'hashtag_count', 'mention_count']
DISASTER_TWEETS = df_train['target'] == 1

fig, axes = plt.subplots(ncols=2, nrows=len(METAFEATURES), figsize=(20, 50), dpi=100)

for i, feature in enumerate(METAFEATURES):
    sns.distplot(df_train.loc[~DISASTER_TWEETS][feature], label='Not Disaster', ax=axes[i][0], col
    sns.distplot(df_train.loc[DISASTER_TWEETS][feature], label='Disaster', ax=axes[i][0], color='r

    sns.distplot(df_train[feature], label='Training', ax=axes[i][1])
    sns.distplot(df_test[feature], label='Test', ax=axes[i][1])

    for j in range(2):
        axes[i][j].set_xlabel('')
        axes[i][j].tick_params(axis='x', labelsize=12)
        axes[i][j].tick_params(axis='y', labelsize=12)
        axes[i][j].legend()

    axes[i][0].set_title(f'{feature} Target Distribution in Training Set', fontsize=13)
    axes[i][1].set_title(f'{feature} Training & Test Set Distribution', fontsize=13)

plt.show()
```

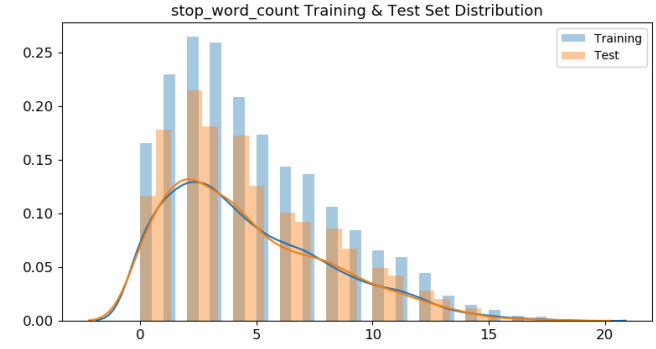

word_count Target Distribution in Training Set
Not Disaster
Disaster

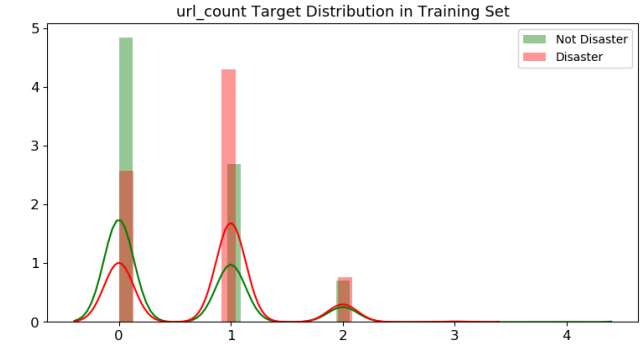

word_count Training & Test Set Distribution
Training
Test

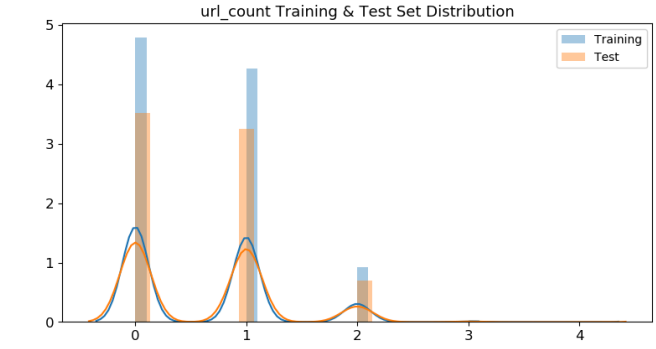

unique_word_count Target Distribution in Training Set
Not Disaster
Disaster

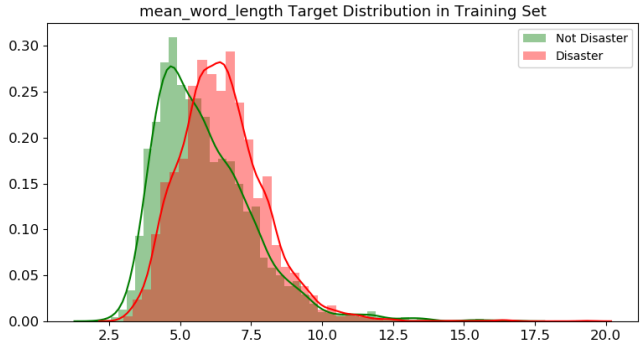

unique_word_count Training & Test Set Distribution
Training
Test

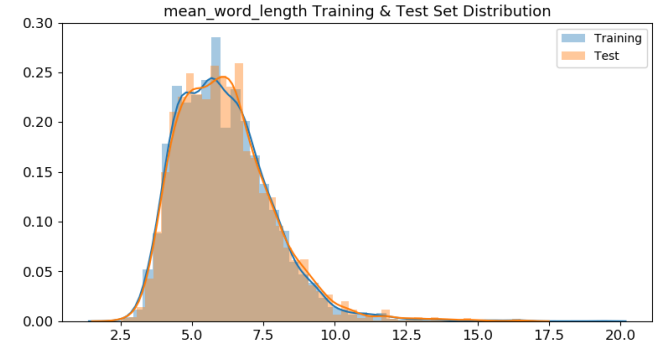

stop_word_count Target Distribution in Training Set
Not Disaster
Disaster

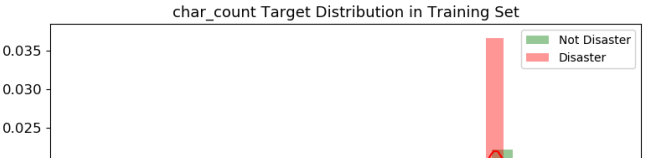

stop_word_count Training & Test Set Distribution
Training
Test

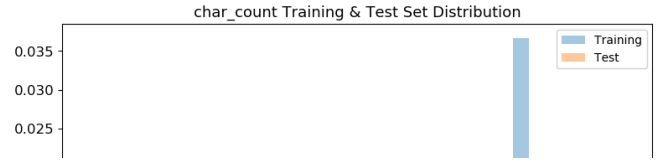

url_count Target Distribution in Training Set
Not Disaster
Disaster

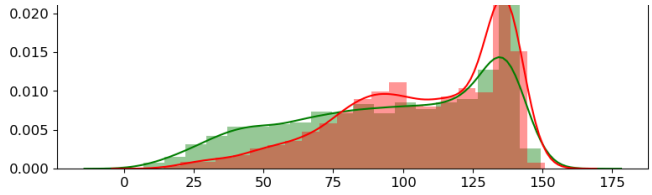

url_count Training & Test Set Distribution
Training
Test

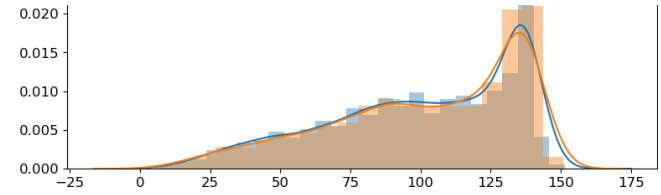

mean_word_length Target Distribution in Training Set
Not Disaster
Disaster

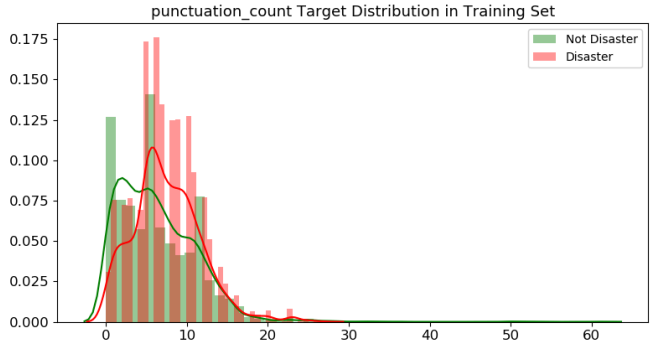

mean_word_length Training & Test Set Distribution
Training
Test

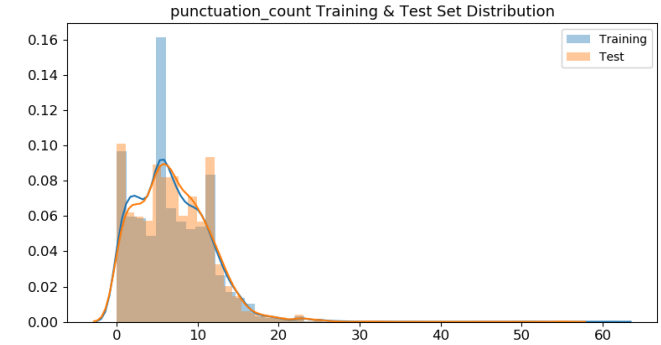

char_count Target Distribution in Training Set
Not Disaster
Disaster

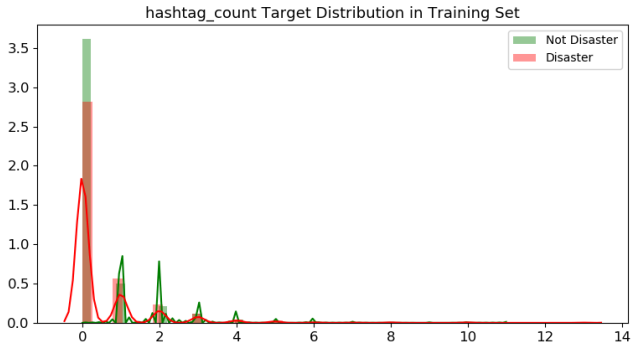

char_count Training & Test Set Distribution
Training
Test

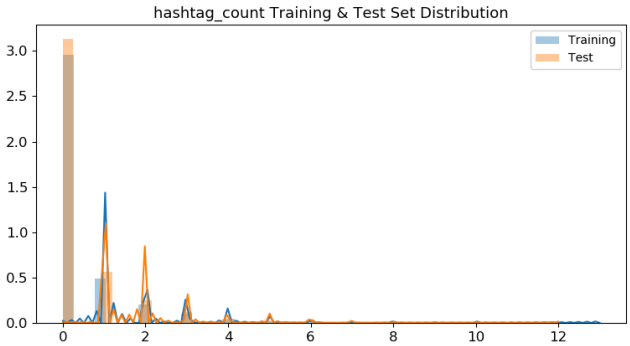


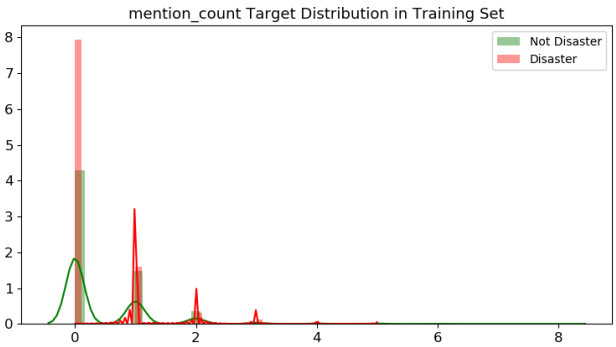


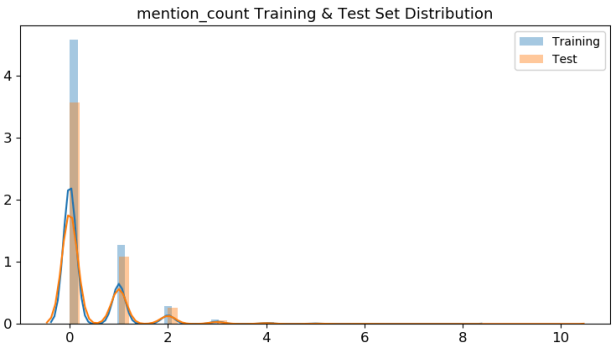

punctuation_count Target Distribution in Training Set
Not Disaster
Disaster


punctuation_count Training & Test Set Distribution
Training
Test


hashtag_count Target Distribution in Training Set
Not Disaster
Disaster


hashtag_count Training & Test Set Distribution
Training
Test


mention_count Target Distribution in Training Set
Not Disaster
Disaster

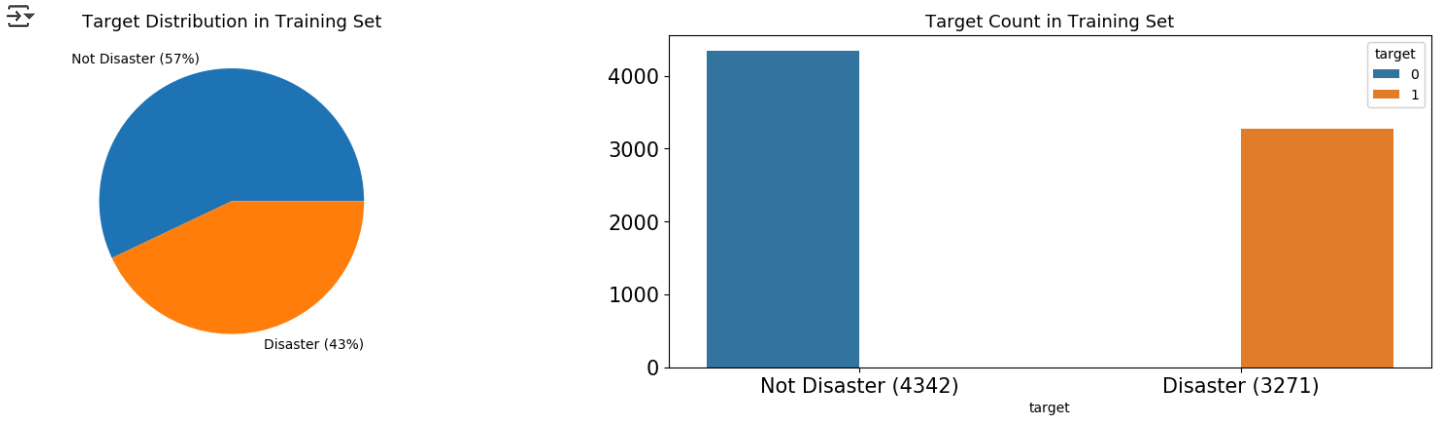

mention_count Training & Test Set Distribution
Training
Test

3. Target and N-grams

3.1 Target

Class distributions are **57%** for **0** (Not Disaster) and **43%** for **1** (Disaster). Classes are almost equally separated so they don't require any stratification by `target` in cross-validation.

```
fig, axes = plt.subplots(ncols=2, figsize=(17, 4), dpi=100)
plt.tight_layout()

df_train.groupby('target').count()['id'].plot(kind='pie', ax=axes[0], labels=['Not Disaster (57%)'
sns.countplot(x=df_train['target'], hue=df_train['target'], ax=axes[1])

axes[0].set_ylabel('')
axes[1].set_ylabel('')
axes[1].set_xticklabels(['Not Disaster (4342)', 'Disaster (3271)'])
axes[0].tick_params(axis='x', labelsize=15)
axes[0].tick_params(axis='y', labelsize=15)
axes[1].tick_params(axis='x', labelsize=15)
axes[1].tick_params(axis='y', labelsize=15)

axes[0].set_title('Target Distribution in Training Set', fontsize=13)
axes[1].set_title('Target Count in Training Set', fontsize=13)

plt.show()
```




```
def generate_ngrams(text, n_gram=1):
    token = [token for token in text.lower().split(' ') if token != '' if token not in STOPWORDS]
    ngrams = zip(*[token[i:] for i in range(n_gram)])
    return [' '.join(ngram) for ngram in ngrams]

N = 100

# Unigrams
disaster_unigrams = defaultdict(int)
nondisaster_unigrams = defaultdict(int)

for tweet in df_train[DISASTER_TWEETS]['text']:
```

```
        for word in generate_ngrams(tweet):
            disaster_unigrams[word] += 1

    for tweet in df_train[~DISASTER_TWEETS]['text']:
        for word in generate_ngrams(tweet):
            nondisaster_unigrams[word] += 1

    df_disaster_unigrams = pd.DataFrame(sorted(disaster_unigrams.items(), key=lambda x: x[1])[::-1])
    df_nondisaster_unigrams = pd.DataFrame(sorted(nondisaster_unigrams.items(), key=lambda x: x[1])[::

    # Bigrams
    disaster_bigrams = defaultdict(int)
    nondisaster_bigrams = defaultdict(int)

    for tweet in df_train[DISASTER_TWEETS]['text']:
        for word in generate_ngrams(tweet, n_gram=2):
            disaster_bigrams[word] += 1

    for tweet in df_train[~DISASTER_TWEETS]['text']:
        for word in generate_ngrams(tweet, n_gram=2):
            nondisaster_bigrams[word] += 1

    df_disaster_bigrams = pd.DataFrame(sorted(disaster_bigrams.items(), key=lambda x: x[1])[::-1])
    df_nondisaster_bigrams = pd.DataFrame(sorted(nondisaster_bigrams.items(), key=lambda x: x[1])[::-1

    # Trigrams
    disaster_trigrams = defaultdict(int)
    nondisaster_trigrams = defaultdict(int)

    for tweet in df_train[DISASTER_TWEETS]['text']:
        for word in generate_ngrams(tweet, n_gram=3):
            disaster_trigrams[word] += 1

    for tweet in df_train[~DISASTER_TWEETS]['text']:
        for word in generate_ngrams(tweet, n_gram=3):
            nondisaster_trigrams[word] += 1

    df_disaster_trigrams = pd.DataFrame(sorted(disaster_trigrams.items(), key=lambda x: x[1])[::-1])
    df_nondisaster_trigrams = pd.DataFrame(sorted(nondisaster_trigrams.items(), key=lambda x: x[1])[::
```

3.2 Unigrams

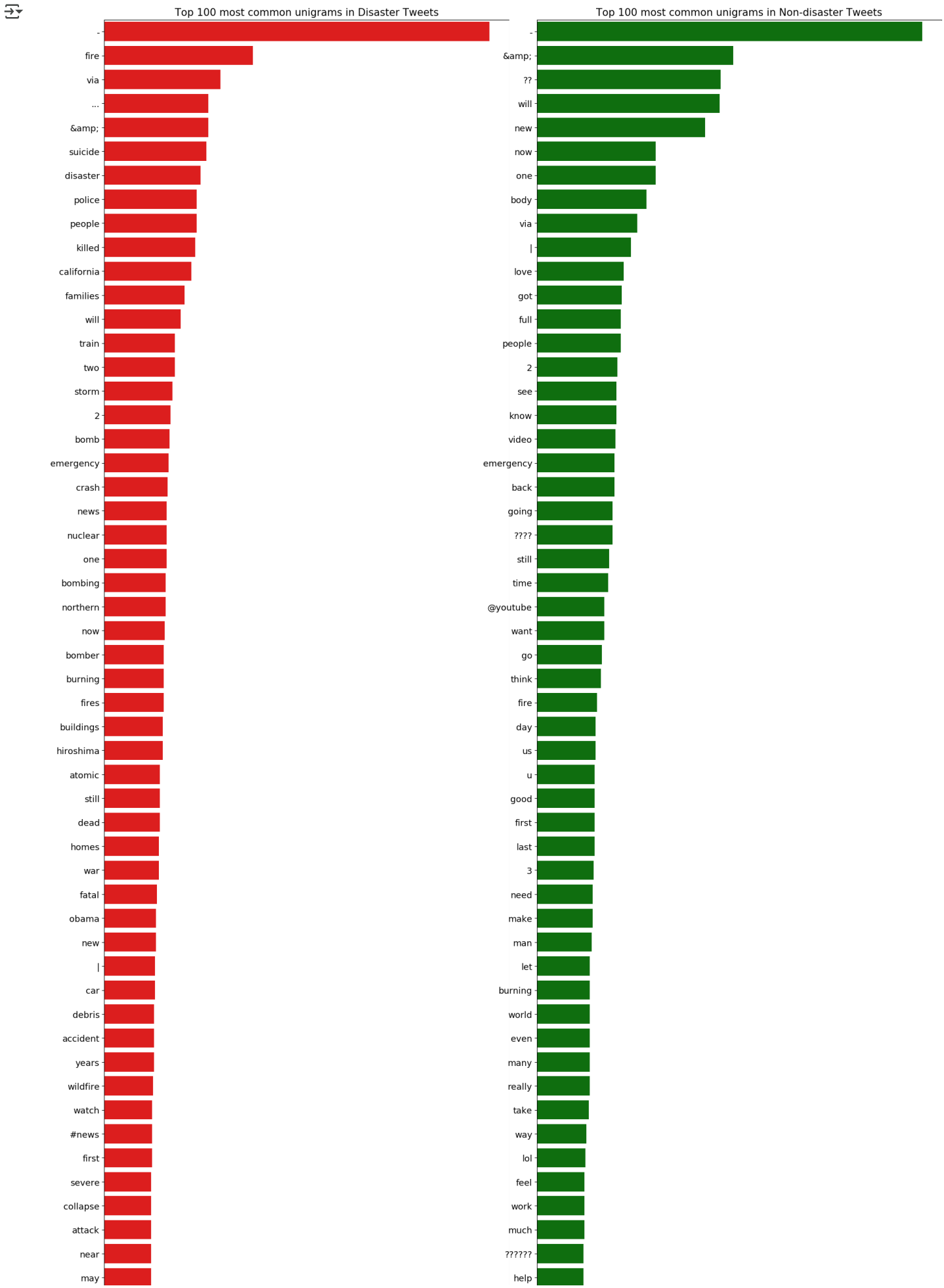
Most common unigrams exist in **both classes** are mostly punctuations, stop words or numbers. It is better to clean them before modelling since they don't give much information about `target`.

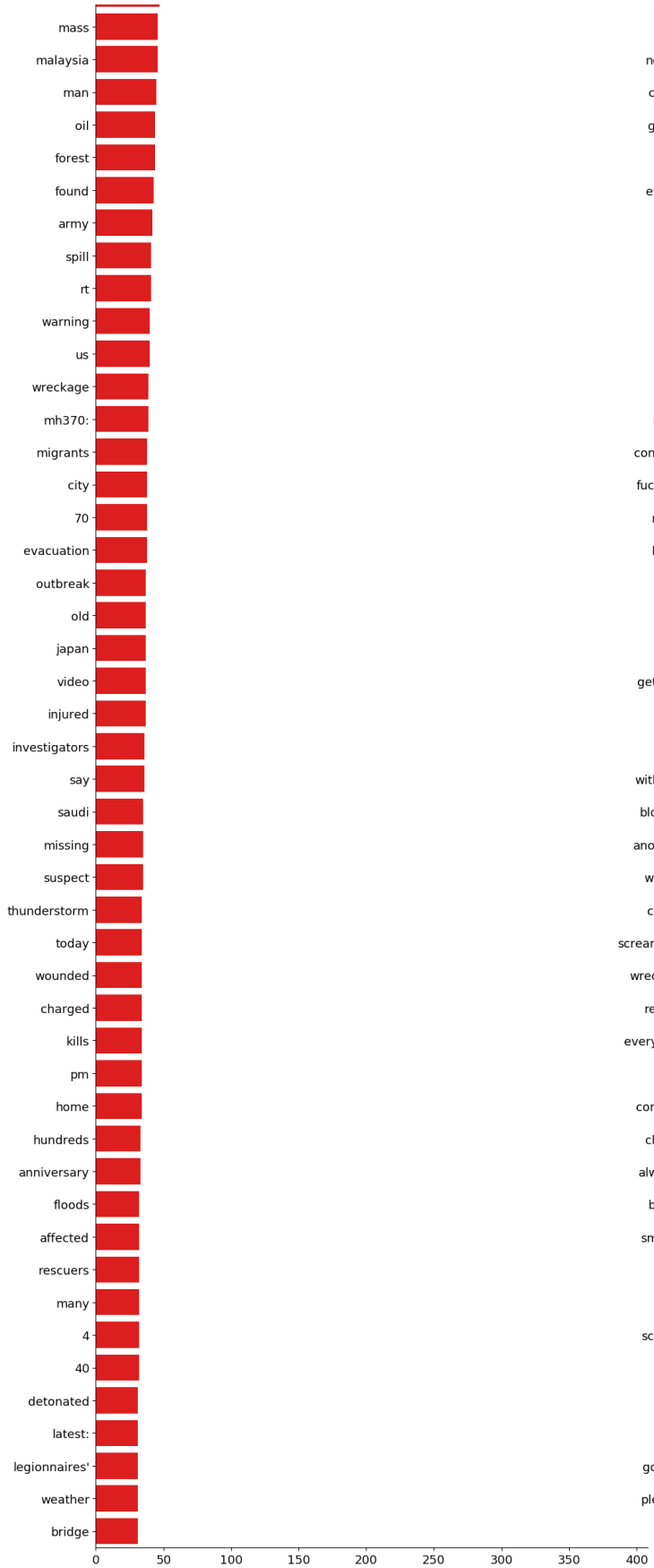
Most common unigrams in **disaster** tweets are already giving information about disasters. It is very hard to use some of those words in other contexts.

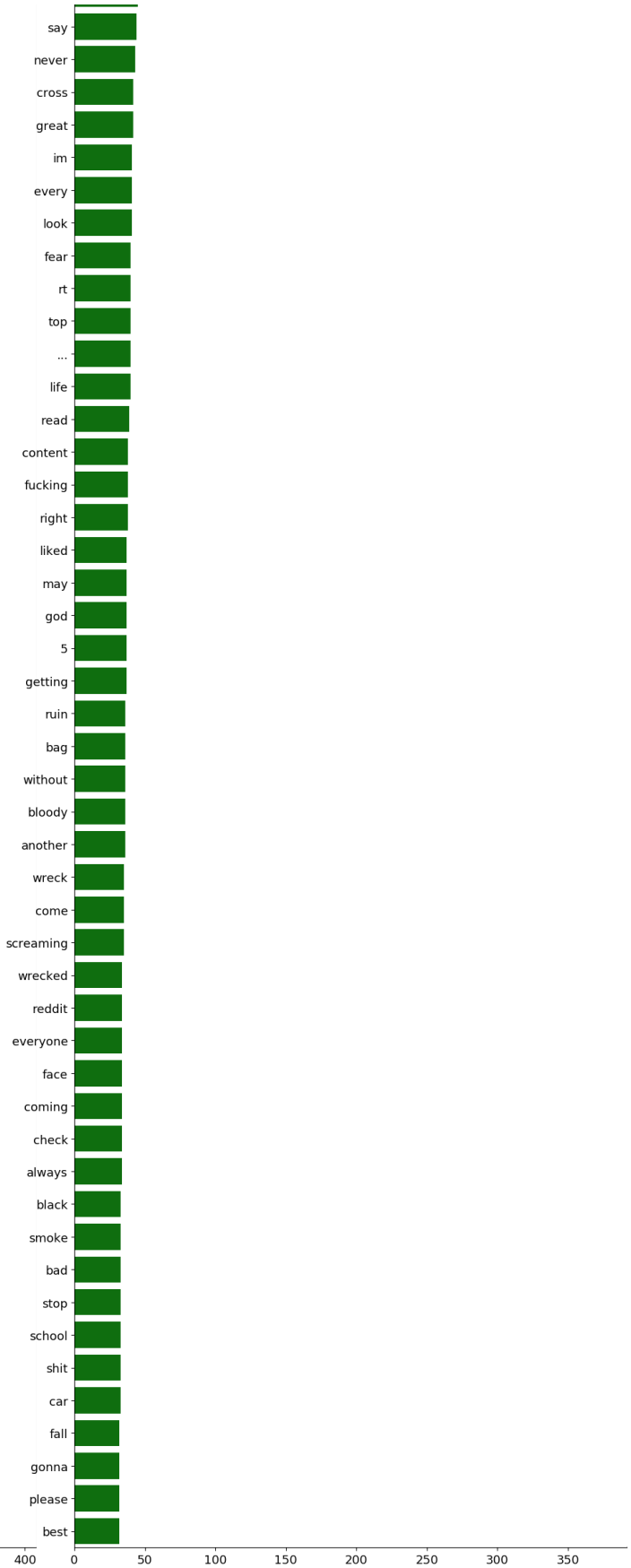
Most common unigrams in **non-disaster** tweets are verbs. This makes sense because most of those sentences have informal active structure since they are coming from individual users.

```
fig, axes = plt.subplots(ncols=2, figsize=(18, 50), dpi=100)
plt.tight_layout()

sns.barplot(y=df_disaster_unigrams[0].values[:N], x=df_disaster_unigrams[1].values[:N], ax=axes[0]
sns.barplot(y=df_nondisaster_unigrams[0].values[:N], x=df_nondisaster_unigrams[1].values[:N], ax=a

for i in range(2):
    axes[i].spines['right'].set_visible(False)
```

```
        axes[i].set_xlabel('')
        axes[i].set_ylabel('')
        axes[i].tick_params(axis='x', labelsize=13)
        axes[i].tick_params(axis='y', labelsize=13)

    axes[0].set_title(f'Top {N} most common unigrams in Disaster Tweets', fontsize=15)
    axes[1].set_title(f'Top {N} most common unigrams in Non-disaster Tweets', fontsize=15)

    plt.show()
```


Top 100 most common unigrams in Disaster Tweets
-
fire
via
...
&
suicide
disaster
police
people
killed
california
families
will
train
two
storm
2
bomb
emergency
crash
news
nuclear
one
bombing
northern
now
bomber
burning
fires
buildings
hiroshima
atomic
still
dead
homes
war
fatal
obama
new
|
car
debris
accident
years
wildfire
watch
#news
first
severe
collapse
attack
near
may
Top 100 most common unigrams in Non-disaster Tweets
-
&
??
will
new
now
one
body
via
|
love
got
full
people
2
see
know
video
emergency
back
going
????
still
time
@youtube
want
go
think
fire
day
us
u
good
first
last
3
need
make
man
let
burning
world
even
many
really
take
way
lol
feel
work
much
??????
help


mass
malaysia
man
oil
forest
found
army
spill
rt
warning
us
wreckage
mh370:
migrants
city
70
evacuation
outbreak
old
japan
video
injured
investigators
say
saudi
missing
suspect
thunderstorm
today
wounded
charged
kills
pm
home
hundreds
anniversary
floods
affected
rescuers
many
4
40
detonated
latest:
legionnaires'
weather
bridge
0
50
100
150
200
250
300
350
400


say
never
cross
great
im
every
look
fear
rt
top
...
life
read
content
fucking
right
liked
may
god
5
getting
ruin
bag
without
bloody
another
wreck
come
screaming
wrecked
reddit
everyone
face
coming
check
always
black
smoke
bad
stop
school
shit
car
fall
gonna
please
best
0
50
100
150
200
250
300
350

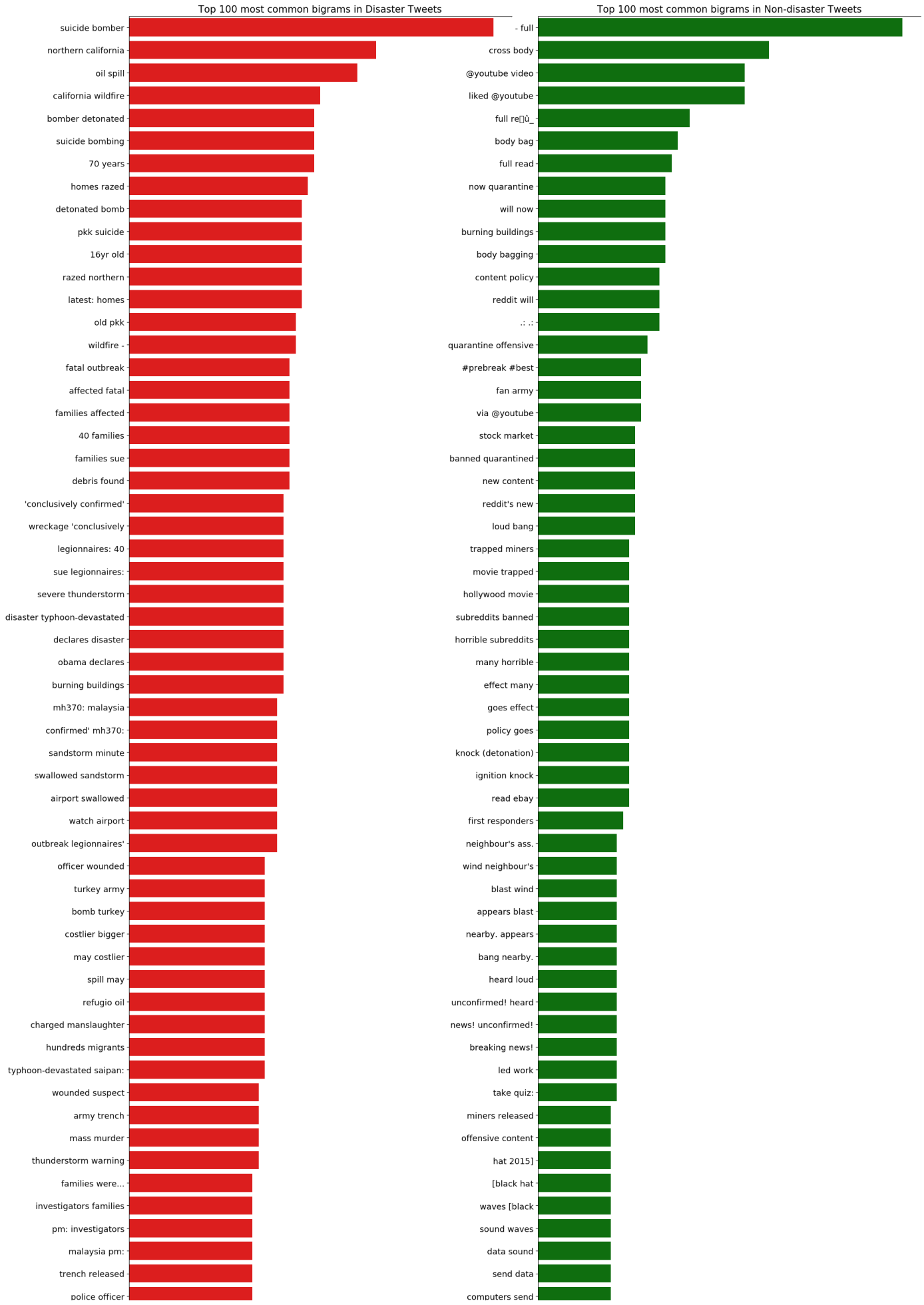
3.3 Bigrams

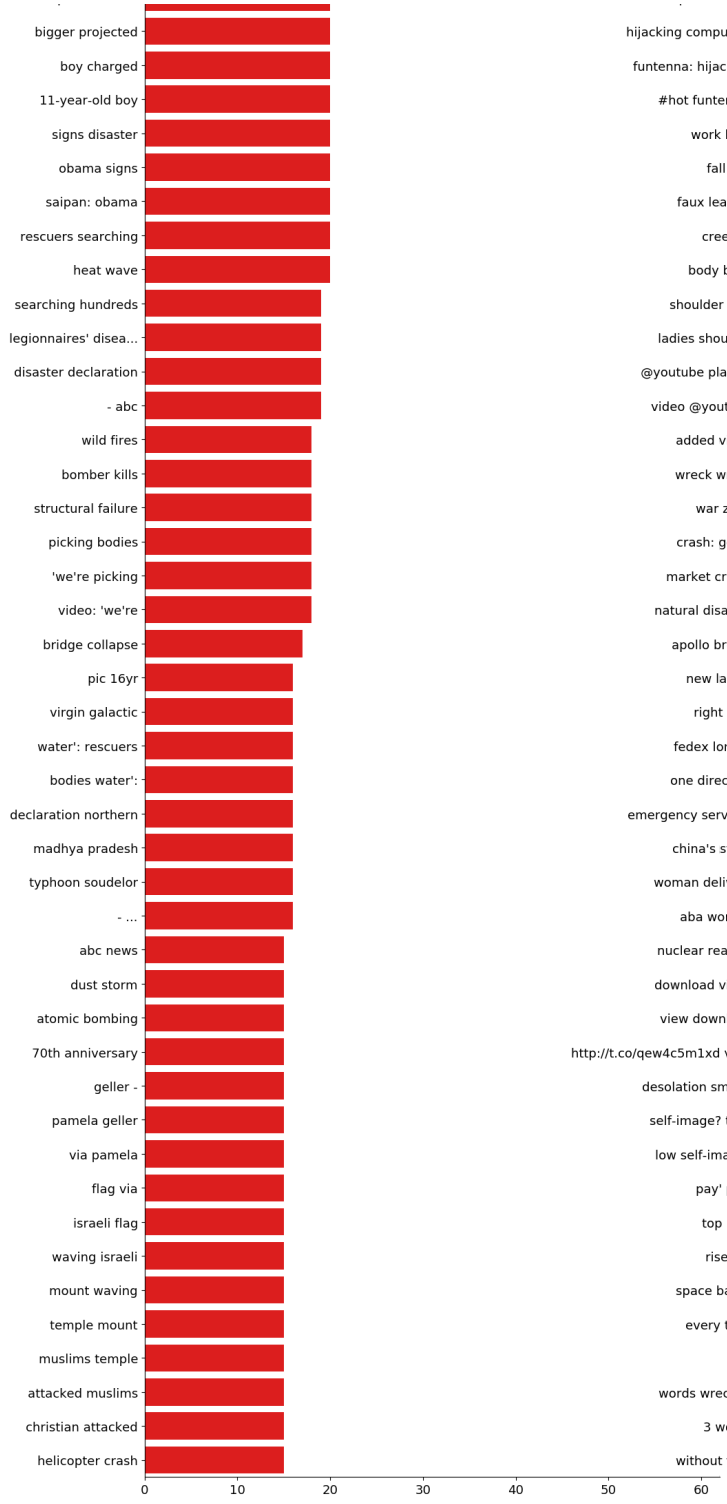
There are no common bigrams exist in **both classes** because the context is clearer.

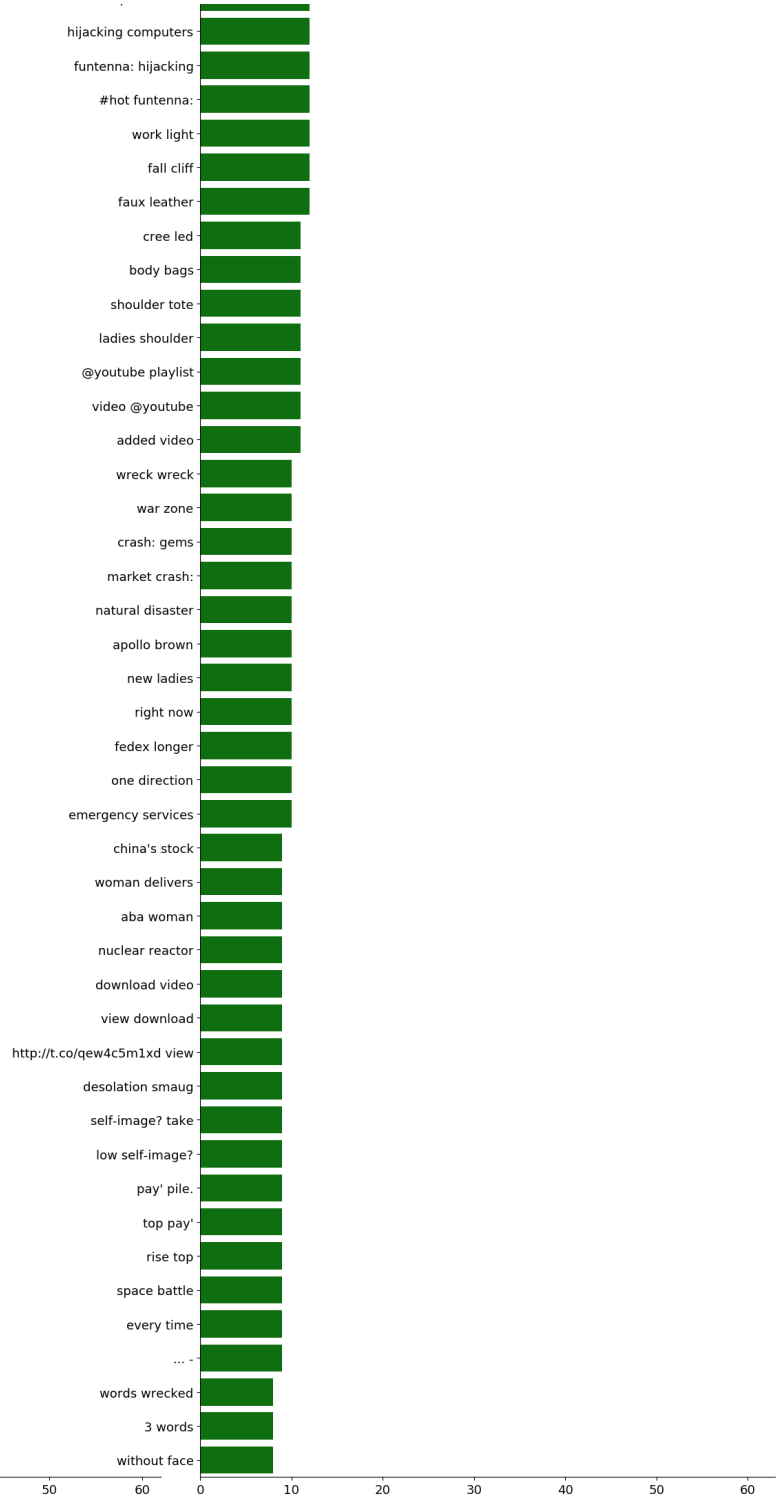
Most common bigrams in **disaster** tweets are giving more information about the disasters than unigrams, but punctuations have to be stripped from words.

Most common bigrams in **non-disaster** tweets are mostly about reddit or youtube, and they contain lots of punctuations. Those punctuations have to be cleaned out of words as well.

```
fig, axes = plt.subplots(ncols=2, figsize=(18, 50), dpi=100)
plt.tight_layout()

sns.barplot(y=df_disaster_bigrams[0].values[:N], x=df_disaster_bigrams[1].values[:N], ax=axes[0],
sns.barplot(y=df_nondisaster_bigrams[0].values[:N], x=df_nondisaster_bigrams[1].values[:N], ax=axe

for i in range(2):
    axes[i].spines['right'].set_visible(False)
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')
    axes[i].tick_params(axis='x', labelsize=13)
    axes[i].tick_params(axis='y', labelsize=13)

axes[0].set_title(f'Top {N} most common bigrams in Disaster Tweets', fontsize=15)
axes[1].set_title(f'Top {N} most common bigrams in Non-disaster Tweets', fontsize=15)

plt.show()
```


Top 100 most common bigrams in Disaster Tweets
suicide bomber
northern california
oil spill
california wildfire
bomber detonated
suicide bombing
70 years
homes razed
detonated bomb
pkk suicide
16yr old
razed northern
latest: homes
old pkk
wildfire -
fatal outbreak
affected fatal
families affected
40 families
families sue
debris found
'conclusively confirmed'
wreckage 'conclusively
legionnaires: 40
sue legionnaires:
severe thunderstorm
disaster typhoon-devastated
declares disaster
obama declares
burning buildings
mh370: malaysia
confirmed' mh370:
sandstorm minute
swallowed sandstorm
airport swallowed
watch airport
outbreak legionnaires'
officer wounded
turkey army
bomb turkey
costlier bigger
may costlier
spill may
refugio oil
charged manslaughter
hundreds migrants
typhoon-devastated saipan:
wounded suspect
army trench
mass murder
thunderstorm warning
families were...
investigators families
pm: investigators
malaysia pm:
trench released
police officer
Top 100 most common bigrams in Non-disaster Tweets
- full
cross body
@youtube video
liked @youtube
full re□û_
body bag
full read
now quarantine
will now
burning buildings
body bagging
content policy
reddit will
.: .:
quarantine offensive
#prebreak #best
fan army
via @youtube
stock market
banned quarantined
new content
reddit's new
loud bang
trapped miners
movie trapped
hollywood movie
subreddits banned
horrible subreddits
many horrible
effect many
goes effect
policy goes
knock (detonation)
ignition knock
read ebay
first responders
neighbour's ass.
wind neighbour's
blast wind
appears blast
nearby. appears
bang nearby.
heard loud
unconfirmed! heard
news! unconfirmed!
breaking news!
led work
take quiz:
miners released
offensive content
hat 2015]
[black hat
waves [black
sound waves
data sound
send data
computers send


bigger projected
boy charged
11-year-old boy
signs disaster
obama signs
saipan: obama
rescuers searching
heat wave
searching hundreds
legionnaires' disea...
disaster declaration
- abc
wild fires
bomber kills
structural failure
picking bodies
'we're picking
video: 'we're
bridge collapse
pic 16yr
virgin galactic
water': rescuers
bodies water':
declaration northern
madhya pradesh
typhoon soudelor
- ...
abc news
dust storm
atomic bombing
70th anniversary
geller -
pamela geller
via pamela
flag via
israeli flag
waving israeli
mount waving
temple mount
muslims temple
attacked muslims
christian attacked
helicopter crash
0
10
20
30
40
50
60


hijacking computers
funtenna: hijacking
#hot funtenna:
work light
fall cliff
faux leather
cree led
body bags
shoulder tote
ladies shoulder
@youtube playlist
video @youtube
added video
wreck wreck
war zone
crash: gems
market crash:
natural disaster
apollo brown
new ladies
right now
fedex longer
one direction
emergency services
china's stock
woman delivers
aba woman
nuclear reactor
download video
view download
http://t.co/qew4c5m1xd view
desolation smaug
self-image? take
low self-image?
pay' pile.
top pay'
rise top
space battle
every time
... -
words wrecked
3 words
without face
0
10
20
30
40
50
60

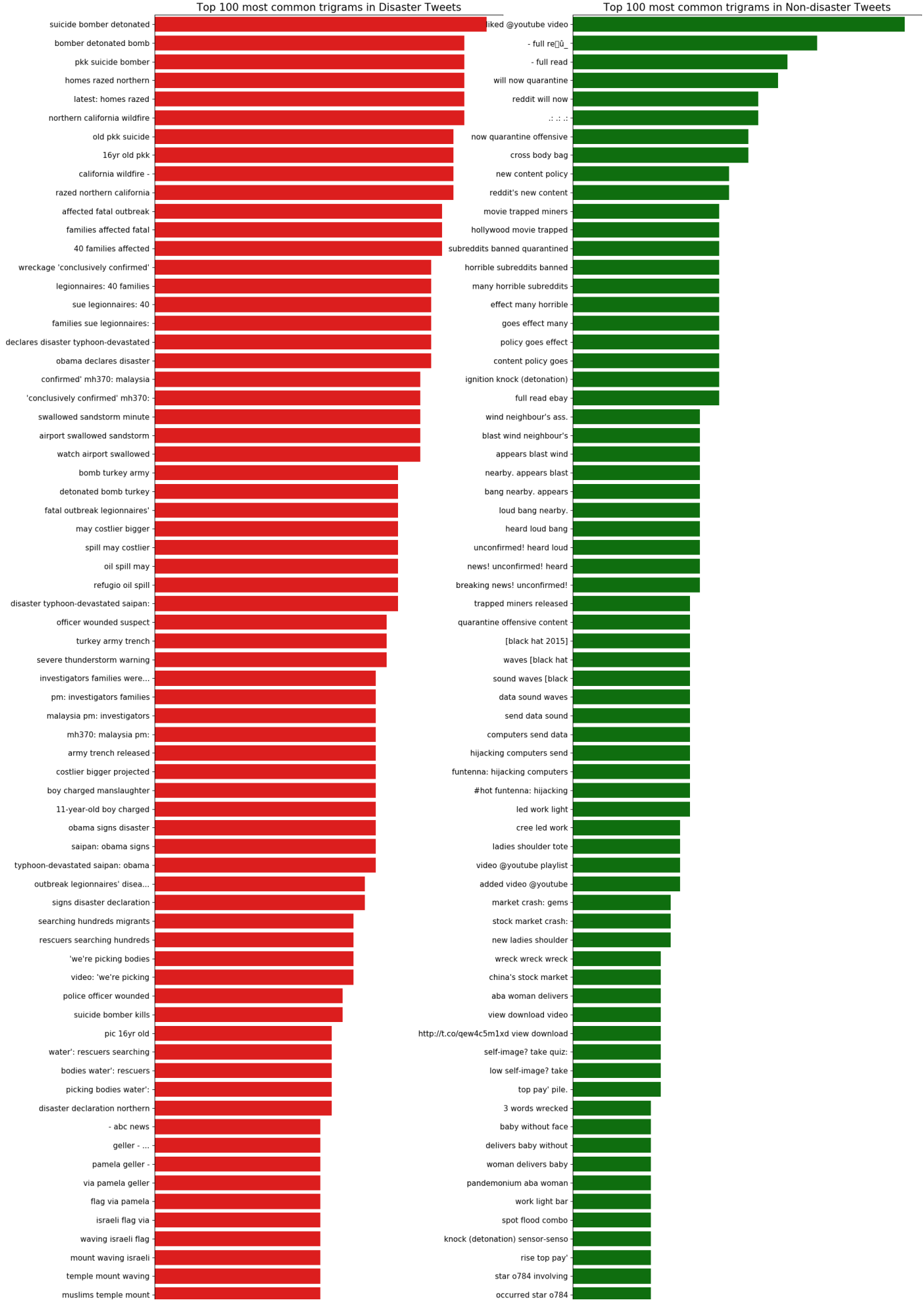
3.4 Trigrams

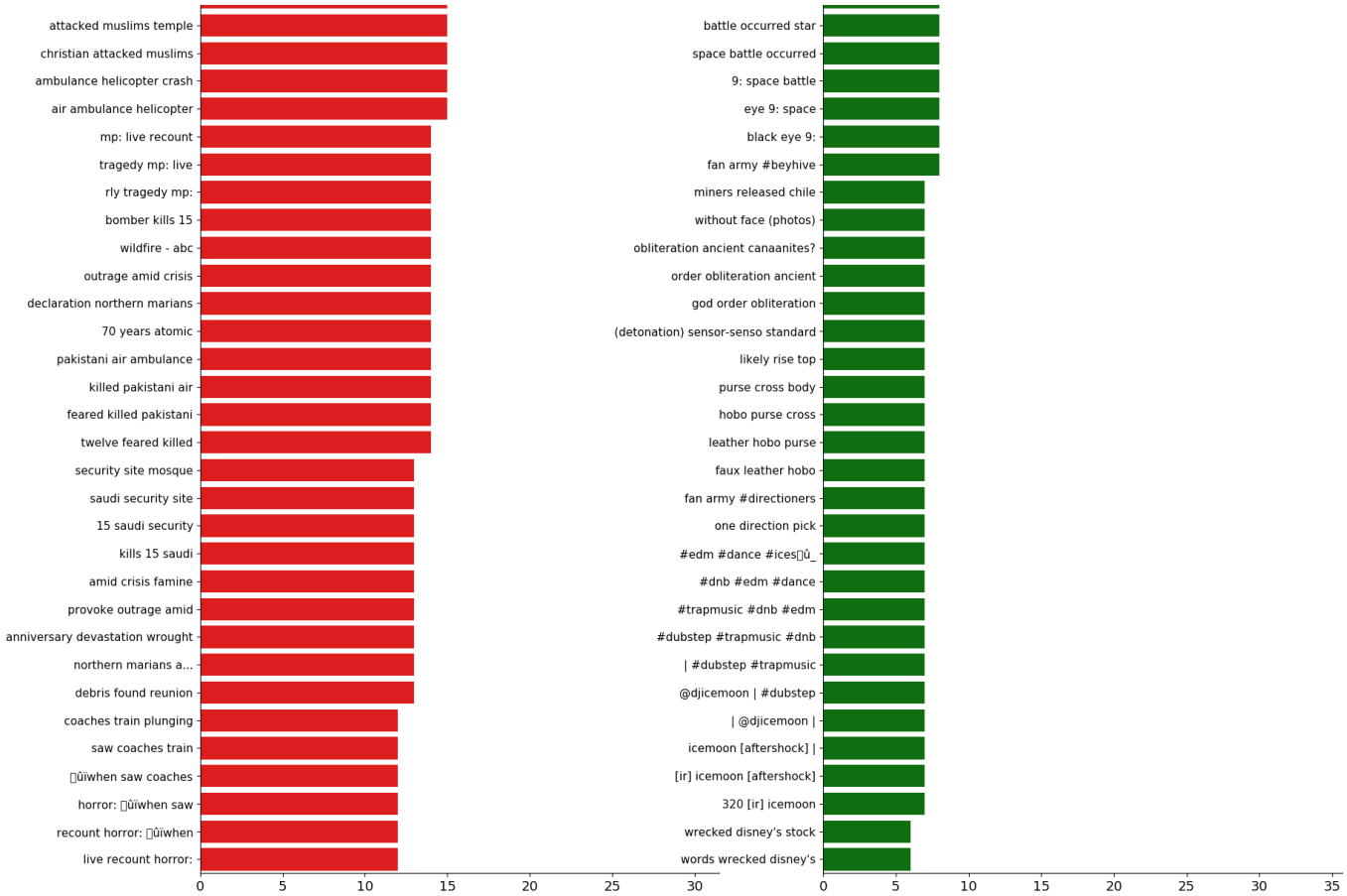
There are no common trigrams exist in **both classes** because the context is clearer.

Most common trigrams in **disaster** tweets are very similar to bigrams. They give lots of information about disasters, but they may not provide any additional information along with bigrams.

Most common trigrams in **non-disaster** tweets are also very similar to bigrams, and they contain even more punctuations.

```
fig, axes = plt.subplots(ncols=2, figsize=(20, 50), dpi=100)

sns.barplot(y=df_disaster_trigrams[0].values[:N], x=df_disaster_trigrams[1].values[:N], ax=axes[0]
sns.barplot(y=df_nondisaster_trigrams[0].values[:N], x=df_nondisaster_trigrams[1].values[:N], ax=a

for i in range(2):
    axes[i].spines['right'].set_visible(False)
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')
    axes[i].tick_params(axis='x', labelsize=13)
    axes[i].tick_params(axis='y', labelsize=11)

axes[0].set_title(f'Top {N} most common trigrams in Disaster Tweets', fontsize=15)
axes[1].set_title(f'Top {N} most common trigrams in Non-disaster Tweets', fontsize=15)

plt.show()
```


Top 100 most common trigrams in Disaster Tweets
suicide bomber detonated
bomber detonated bomb
pkk suicide bomber
homes razed northern
latest: homes razed
northern california wildfire
old pkk suicide
16yr old pkk
california wildfire -
razed northern california
affected fatal outbreak
families affected fatal
40 families affected
wreckage 'conclusively confirmed'
legionnaires: 40 families
sue legionnaires: 40
families sue legionnaires:
declares disaster typhoon-devastated
obama declares disaster
confirmed' mh370: malaysia
'conclusively confirmed' mh370:
swallowed sandstorm minute
airport swallowed sandstorm
watch airport swallowed
bomb turkey army
detonated bomb turkey
fatal outbreak legionnaires'
may costlier bigger
spill may costlier
oil spill may
refugio oil spill
disaster typhoon-devastated saipan:
officer wounded suspect
turkey army trench
severe thunderstorm warning
investigators families were...
pm: investigators families
malaysia pm: investigators
mh370: malaysia pm:
army trench released
costlier bigger projected
boy charged manslaughter
11-year-old boy charged
obama signs disaster
saipan: obama signs
typhoon-devastated saipan: obama
outbreak legionnaires' disea...
signs disaster declaration
searching hundreds migrants
rescuers searching hundreds
'we're picking bodies
video: 'we're picking
police officer wounded
suicide bomber kills
pic 16yr old
water': rescuers searching
bodies water': rescuers
picking bodies water':
disaster declaration northern
- abc news
geller - ...
pamela geller -
via pamela geller
flag via pamela
israeli flag via
waving israeli flag
mount waving israeli
temple mount waving
muslims temple mount
Top 100 most common trigrams in Non-disaster Tweets
liked @youtube video
- full re□û_
- full read
will now quarantine
reddit will now
.: .: .:
now quarantine offensive
cross body bag
new content policy
reddit's new content
movie trapped miners
hollywood movie trapped
subreddits banned quarantined
horrible subreddits banned
many horrible subreddits
effect many horrible
goes effect many
policy goes effect
content policy goes
ignition knock (detonation)
full read ebay
wind neighbour's ass.
blast wind neighbour's
appears blast wind
nearby. appears blast
bang nearby. appears
loud bang nearby.
heard loud bang
unconfirmed! heard loud
news! unconfirmed! heard
breaking news! unconfirmed!
trapped miners released
quarantine offensive content
[black hat 2015]
waves [black hat
sound waves [black
data sound waves
send data sound
computers send data
hijacking computers send
funtenna: hijacking computers
#hot funtenna: hijacking
led work light
cree led work
ladies shoulder tote
video @youtube playlist
added video @youtube
market crash: gems
stock market crash:
new ladies shoulder
wreck wreck wreck
china's stock market
aba woman delivers
view download video
http://t.co/qew4c5m1xd view download
self-image? take quiz:
low self-image? take
top pay' pile.
3 words wrecked
baby without face
delivers baby without
woman delivers baby
pandemonium aba woman
work light bar
spot flood combo
knock (detonation) sensor-senso
rise top pay'
star o784 involving
occurred star o784


attacked muslims temple
christian attacked muslims
ambulance helicopter crash
air ambulance helicopter
mp: live recount
tragedy mp: live
rly tragedy mp:
bomber kills 15
wildfire - abc
outrage amid crisis
declaration northern marians
70 years atomic
pakistani air ambulance
killed pakistani air
feared killed pakistani
twelve feared killed
security site mosque
saudi security site
15 saudi security
kills 15 saudi
amid crisis famine
provoke outrage amid
anniversary devastation wrought
northern marians a...
debris found reunion
coaches train plunging
saw coaches train
ûïwhen saw coaches
horror: ûïwhen saw
recount horror: ûïwhen
live recount horror:
0
5
10
15
20
25
30
battle occurred star
space battle occurred
9: space battle
eye 9: space
black eye 9:
fan army #beyhive
miners released chile
without face (photos)
obliteration ancient canaanites?
order obliteration ancient
god order obliteration
(detonation) sensor-senso standard
likely rise top
purse cross body
hobo purse cross
leather hobo purse
faux leather hobo
fan army #directioners
one direction pick
#edm #dance #ices û_
#dnb #edm #dance
#trapmusic #dnb #edm
#dubstep #trapmusic #dnb
| #dubstep #trapmusic
@djicemoon | #dubstep
| @djicemoon |
icemoon [aftershock] |
[ir] icemoon [aftershock]
320 [ir] icemoon
wrecked disney's stock
words wrecked disney's
0
5
10
15
20
25
30
35

4. Embeddings and Text Cleaning

4.1 Embeddings Coverage

When you have pre-trained embeddings, doing standard preprocessing steps might not be a good idea because some of the valuable information can be lost. It is better to get vocabulary as close to embeddings as possible. In order to do that, train vocab and test vocab are created by counting the words in tweets.

Text cleaning is based on the embeddings below:

- GloVe-300d-840B
- FastText-Crawl-300d-2M

```
%%time

glove_embeddings = np.load('../input/pickled-glove840b300d-for-10sec-loading/glove.840B.300d.pkl',
fasttext_embeddings = np.load('../input/pickled-crawl300d2m-for-kernel-competitions/crawl-300d-2M.
```

```
CPU times: user 14.2 s, sys: 5.2 s, total: 19.4 s
Wall time: 19.8 s
```

Words in the intersection of vocab and embeddings are stored in `covered` along with their counts. Words in vocab that don't exist in embeddings are stored in `oov` along with their counts. `n_covered` and `n_oov` are total number of counts and they are used for calculating coverage percentages.

Both GloVe and FastText embeddings have more than **50%** vocabulary and **80%** text coverage without cleaning. GloVe and FastText coverage are very close but GloVe has slightly higher coverage.

```
def build_vocab(X):

    tweets = X.apply(lambda s: s.split()).values
    vocab = {}

    for tweet in tweets:
        for word in tweet:
            try:
                vocab[word] += 1
            except KeyError:
                vocab[word] = 1
    return vocab


def check_embeddings_coverage(X, embeddings):

    vocab = build_vocab(X)

    covered = {}
    oov = {}
    n_covered = 0
    n_oov = 0

    for word in vocab:
        try:
            covered[word] = embeddings[word]
            n_covered += vocab[word]
        except:
            oov[word] = vocab[word]
            n_oov += vocab[word]
```

```
    vocab_coverage = len(covered) / len(vocab)
    text_coverage = (n_covered / (n_covered + n_oov))

    sorted_oov = sorted(oov.items(), key=operator.itemgetter(1))[::-1]
    return sorted_oov, vocab_coverage, text_coverage

train_glove_oov, train_glove_vocab_coverage, train_glove_text_coverage = check_embeddings_coverage
test_glove_oov, test_glove_vocab_coverage, test_glove_text_coverage = check_embeddings_coverage(df
print('GloVe Embeddings cover {:.2%} of vocabulary and {:.2%} of text in Training Set'.format(trai
print('GloVe Embeddings cover {:.2%} of vocabulary and {:.2%} of text in Test Set'.format(test_glo

train_fasttext_oov, train_fasttext_vocab_coverage, train_fasttext_text_coverage = check_embeddings
test_fasttext_oov, test_fasttext_vocab_coverage, test_fasttext_text_coverage = check_embeddings_co
print('FastText Embeddings cover {:.2%} of vocabulary and {:.2%} of text in Training Set'.format(t
print('FastText Embeddings cover {:.2%} of vocabulary and {:.2%} of text in Test Set'.format(test_
```

```
GloVe Embeddings cover 52.06% of vocabulary and 82.68% of text in Training Set
GloVe Embeddings cover 57.21% of vocabulary and 81.85% of text in Test Set
FastText Embeddings cover 51.52% of vocabulary and 81.84% of text in Training Set
FastText Embeddings cover 56.55% of vocabulary and 81.12% of text in Test Set
```

4.2 Text Cleaning

Tweets require lots of cleaning but it is inefficient to clean every single tweet because that would consume too much time. A general approach must be implemented for cleaning.

- The most common type of words that require cleaning in `oov` have punctuations at the start or end. Those words doesn't have embeddings because of the trailing punctuations. Punctuations `#`, `@`, `!`, `?`, `+`, `&`, `-`, `$`, `=`, `<`, `>`, `|`, `{`, `}`, `^`, `'`, `(`, `)`, `[`, `]`, `*`, `%`, `...`, `'`, `.`, `:`, `;` are separated from words
- Special characters that are attached to words are removed completely
- Contractions are expanded
- Urls are removed
- Character entity references are replaced with their actual symbols
- Typos and slang are corrected, and informal abbreviations are written in their long forms
- Some words are replaced with their acronyms and some words are grouped into one
- Finally, hashtags and usernames contain lots of information about the context but they are written without spaces in between words so they don't have embeddings. Informational usernames and hashtags should be expanded but there are too many of them. I expanded as many as I could, but it takes too much time to run `clean` function after adding those replace calls.

```
%%time

def clean(tweet):

    # Special characters
    tweet = re.sub(r"\x89Û_", "", tweet)
    tweet = re.sub(r"\x89ÛÒ", "", tweet)
    tweet = re.sub(r"\x89ÛÓ", "", tweet)
    tweet = re.sub(r"\x89ÛÏWhen", "When", tweet)
    tweet = re.sub(r"\x89ÛÏ", "", tweet)
    tweet = re.sub(r"China\x89Ûªs", "China's", tweet)
    tweet = re.sub(r"let\x89Ûªs", "let's", tweet)
    tweet = re.sub(r"\x89Û÷", "", tweet)
    tweet = re.sub(r"\x89Ûª", "", tweet)
    tweet = re.sub(r"\x89Û\x9d", "", tweet)
    tweet = re.sub(r"å_", "", tweet)
    tweet = re.sub(r"\x89Û¢", "", tweet)
    tweet = re.sub(r"\x89Û¢åÊ", "", tweet)
    tweet = re.sub(r"fromåÊwounds", "from wounds", tweet)
    tweet = re.sub(r"åÊ", "", tweet)
```

```
    tweet = re.sub(r"åÈ", "", tweet)
    tweet = re.sub(r"JapÌ_n", "Japan", tweet)
    tweet = re.sub(r"Ì©", "e", tweet)
    tweet = re.sub(r"å¨", "", tweet)
    tweet = re.sub(r"SuruÌ¤", "Suruc", tweet)
    tweet = re.sub(r"åÇ", "", tweet)
    tweet = re.sub(r"å£3million", "3 million", tweet)
    tweet = re.sub(r"åÀ", "", tweet)

    # Contractions
    tweet = re.sub(r"he's", "he is", tweet)
    tweet = re.sub(r"there's", "there is", tweet)
    tweet = re.sub(r"We're", "We are", tweet)
    tweet = re.sub(r"That's", "That is", tweet)
    tweet = re.sub(r"won't", "will not", tweet)
    tweet = re.sub(r"they're", "they are", tweet)
    tweet = re.sub(r"Can't", "Cannot", tweet)
    tweet = re.sub(r"wasn't", "was not", tweet)
    tweet = re.sub(r"don\x89Ûªt", "do not", tweet)
    tweet = re.sub(r"aren't", "are not", tweet)
    tweet = re.sub(r"isn't", "is not", tweet)
    tweet = re.sub(r"What's", "What is", tweet)
    tweet = re.sub(r"haven't", "have not", tweet)
    tweet = re.sub(r"hasn't", "has not", tweet)
    tweet = re.sub(r"There's", "There is", tweet)
    tweet = re.sub(r"He's", "He is", tweet)
    tweet = re.sub(r"It's", "It is", tweet)
    tweet = re.sub(r"You're", "You are", tweet)
    tweet = re.sub(r"I'M", "I am", tweet)
    tweet = re.sub(r"shouldn't", "should not", tweet)
    tweet = re.sub(r"wouldn't", "would not", tweet)
    tweet = re.sub(r"i'm", "I am", tweet)
    tweet = re.sub(r"I\x89Ûªm", "I am", tweet)
    tweet = re.sub(r"I'm", "I am", tweet)
    tweet = re.sub(r"Isn't", "is not", tweet)
    tweet = re.sub(r"Here's", "Here is", tweet)
    tweet = re.sub(r"you've", "you have", tweet)
    tweet = re.sub(r"you\x89Ûªve", "you have", tweet)
    tweet = re.sub(r"we're", "we are", tweet)
    tweet = re.sub(r"what's", "what is", tweet)
    tweet = re.sub(r"couldn't", "could not", tweet)
    tweet = re.sub(r"we've", "we have", tweet)
    tweet = re.sub(r"it\x89Ûªs", "it is", tweet)
    tweet = re.sub(r"doesn\x89Ûªt", "does not", tweet)
    tweet = re.sub(r"It\x89Ûªs", "It is", tweet)
    tweet = re.sub(r"Here\x89Ûªs", "Here is", tweet)
    tweet = re.sub(r"who's", "who is", tweet)
    tweet = re.sub(r"I\x89Ûªve", "I have", tweet)
    tweet = re.sub(r"y'all", "you all", tweet)
    tweet = re.sub(r"can\x89Ûªt", "cannot", tweet)
    tweet = re.sub(r"would've", "would have", tweet)
    tweet = re.sub(r"it'll", "it will", tweet)
    tweet = re.sub(r"we'll", "we will", tweet)
    tweet = re.sub(r"wouldn\x89Ûªt", "would not", tweet)
    tweet = re.sub(r"We've", "We have", tweet)
    tweet = re.sub(r"he'll", "he will", tweet)
    tweet = re.sub(r"Y'all", "You all", tweet)
    tweet = re.sub(r"Weren't", "Were not", tweet)
```

```
    tweet = re.sub(r"Didn't", "Did not", tweet)
    tweet = re.sub(r"they'll", "they will", tweet)
    tweet = re.sub(r"they'd", "they would", tweet)
    tweet = re.sub(r"DON'T", "DO NOT", tweet)
    tweet = re.sub(r"That\x89Ûªs", "That is", tweet)
    tweet = re.sub(r"they've", "they have", tweet)
    tweet = re.sub(r"i'd", "I would", tweet)
    tweet = re.sub(r"should've", "should have", tweet)
    tweet = re.sub(r"You\x89Ûªre", "You are", tweet)
    tweet = re.sub(r"where's", "where is", tweet)
    tweet = re.sub(r"Don\x89Ûªt", "Do not", tweet)
    tweet = re.sub(r"we'd", "we would", tweet)
    tweet = re.sub(r"i'll", "I will", tweet)
    tweet = re.sub(r"weren't", "were not", tweet)
    tweet = re.sub(r"They're", "They are", tweet)
    tweet = re.sub(r"Can\x89Ûªt", "Cannot", tweet)
    tweet = re.sub(r"you\x89Ûªll", "you will", tweet)
    tweet = re.sub(r"I\x89Ûªd", "I would", tweet)
    tweet = re.sub(r"let's", "let us", tweet)
    tweet = re.sub(r"it's", "it is", tweet)
    tweet = re.sub(r"can't", "cannot", tweet)
    tweet = re.sub(r"don't", "do not", tweet)
    tweet = re.sub(r"you're", "you are", tweet)
    tweet = re.sub(r"i've", "I have", tweet)
    tweet = re.sub(r"that's", "that is", tweet)
    tweet = re.sub(r"i'll", "I will", tweet)
    tweet = re.sub(r"doesn't", "does not", tweet)
    tweet = re.sub(r"i'd", "I would", tweet)
    tweet = re.sub(r"didn't", "did not", tweet)
    tweet = re.sub(r"ain't", "am not", tweet)
    tweet = re.sub(r"you'll", "you will", tweet)
    tweet = re.sub(r"I've", "I have", tweet)
    tweet = re.sub(r"Don't", "do not", tweet)
    tweet = re.sub(r"I'll", "I will", tweet)
    tweet = re.sub(r"I'd", "I would", tweet)
    tweet = re.sub(r"Let's", "Let us", tweet)
    tweet = re.sub(r"you'd", "You would", tweet)
    tweet = re.sub(r"It's", "It is", tweet)
    tweet = re.sub(r"Ain't", "am not", tweet)
    tweet = re.sub(r"Haven't", "Have not", tweet)
    tweet = re.sub(r"Could've", "Could have", tweet)
    tweet = re.sub(r"youve", "you have", tweet)
    tweet = re.sub(r"donå«t", "do not", tweet)

    # Character entity references
    tweet = re.sub(r"&gt;", ">", tweet)
    tweet = re.sub(r"&lt;", "<", tweet)
    tweet = re.sub(r"&amp;", "&", tweet)

    # Typos, slang and informal abbreviations
    tweet = re.sub(r"w/e", "whatever", tweet)
    tweet = re.sub(r"w/", "with", tweet)
    tweet = re.sub(r"USAgov", "USA government", tweet)
    tweet = re.sub(r"recentlu", "recently", tweet)
    tweet = re.sub(r"Ph0tos", "Photos", tweet)
    tweet = re.sub(r"amirite", "am I right", tweet)
    tweet = re.sub(r"exp0sed", "exposed", tweet)
    tweet = re.sub(r"<3", "love", tweet)
```

```
    tweet = re.sub(r"amageddon", "armageddon", tweet)
    tweet = re.sub(r"Trfc", "Traffic", tweet)
    tweet = re.sub(r"8/5/2015", "2015-08-05", tweet)
    tweet = re.sub(r"WindStorm", "Wind Storm", tweet)
    tweet = re.sub(r"8/6/2015", "2015-08-06", tweet)
    tweet = re.sub(r"10:38PM", "10:38 PM", tweet)
    tweet = re.sub(r"10:30pm", "10:30 PM", tweet)
    tweet = re.sub(r"16yr", "16 year", tweet)
    tweet = re.sub(r"lmao", "laughing my ass off", tweet)
    tweet = re.sub(r"TRAUMATISED", "traumatized", tweet)

    # Hashtags and usernames
    tweet = re.sub(r"IranDeal", "Iran Deal", tweet)
    tweet = re.sub(r"ArianaGrande", "Ariana Grande", tweet)
    tweet = re.sub(r"camilacabello97", "camila cabello", tweet)
    tweet = re.sub(r"RondaRousey", "Ronda Rousey", tweet)
    tweet = re.sub(r"MTVHottest", "MTV Hottest", tweet)
    tweet = re.sub(r"TrapMusic", "Trap Music", tweet)
    tweet = re.sub(r"ProphetMuhammad", "Prophet Muhammad", tweet)
    tweet = re.sub(r"PantherAttack", "Panther Attack", tweet)
    tweet = re.sub(r"StrategicPatience", "Strategic Patience", tweet)
    tweet = re.sub(r"socialnews", "social news", tweet)
    tweet = re.sub(r"NASAHurricane", "NASA Hurricane", tweet)
    tweet = re.sub(r"onlinecommunities", "online communities", tweet)
    tweet = re.sub(r"humanconsumption", "human consumption", tweet)
    tweet = re.sub(r"Typhoon-Devastated", "Typhoon Devastated", tweet)
    tweet = re.sub(r"Meat-Loving", "Meat Loving", tweet)
    tweet = re.sub(r"facialabuse", "facial abuse", tweet)
    tweet = re.sub(r"LakeCounty", "Lake County", tweet)
    tweet = re.sub(r"BeingAuthor", "Being Author", tweet)
    tweet = re.sub(r"withheavenly", "with heavenly", tweet)
    tweet = re.sub(r"thankU", "thank you", tweet)
    tweet = re.sub(r"iTunesMusic", "iTunes Music", tweet)
    tweet = re.sub(r"OffensiveContent", "Offensive Content", tweet)
    tweet = re.sub(r"WorstSummerJob", "Worst Summer Job", tweet)
    tweet = re.sub(r"HarryBeCareful", "Harry Be Careful", tweet)
    tweet = re.sub(r"NASASolarSystem", "NASA Solar System", tweet)
    tweet = re.sub(r"animalrescue", "animal rescue", tweet)
    tweet = re.sub(r"KurtSchlichter", "Kurt Schlichter", tweet)
    tweet = re.sub(r"aRmageddon", "armageddon", tweet)
    tweet = re.sub(r"Throwingknifes", "Throwing knives", tweet)
    tweet = re.sub(r"GodsLove", "God's Love", tweet)
    tweet = re.sub(r"bookboost", "book boost", tweet)
    tweet = re.sub(r"ibooklove", "I book love", tweet)
    tweet = re.sub(r"NestleIndia", "Nestle India", tweet)
    tweet = re.sub(r"realDonaldTrump", "Donald Trump", tweet)
    tweet = re.sub(r"DavidVonderhaar", "David Vonderhaar", tweet)
    tweet = re.sub(r"CecilTheLion", "Cecil The Lion", tweet)
    tweet = re.sub(r"weathernetwork", "weather network", tweet)
    tweet = re.sub(r"withBioterrorism&use", "with Bioterrorism & use", tweet)
    tweet = re.sub(r"Hostage&2", "Hostage & 2", tweet)
    tweet = re.sub(r"GOPDebate", "GOP Debate", tweet)
    tweet = re.sub(r"RickPerry", "Rick Perry", tweet)
    tweet = re.sub(r"frontpage", "front page", tweet)
    tweet = re.sub(r"NewsInTweets", "News In Tweets", tweet)
    tweet = re.sub(r"ViralSpell", "Viral Spell", tweet)
    tweet = re.sub(r"til_now", "until now", tweet)
    tweet = re.sub(r"volcanoinRussia", "volcano in Russia", tweet)
```

```
    tweet = re.sub(r"ZippedNews", "Zipped News", tweet)
    tweet = re.sub(r"MicheleBachman", "Michele Bachman", tweet)
    tweet = re.sub(r"53inch", "53 inch", tweet)
    tweet = re.sub(r"KerrickTrial", "Kerrick Trial", tweet)
    tweet = re.sub(r"abstorm", "Alberta Storm", tweet)
    tweet = re.sub(r"Beyhive", "Beyonce hive", tweet)
    tweet = re.sub(r"IDFire", "Idaho Fire", tweet)
    tweet = re.sub(r"DETECTADO", "Detected", tweet)
    tweet = re.sub(r"RockyFire", "Rocky Fire", tweet)
    tweet = re.sub(r"Listen/Buy", "Listen / Buy", tweet)
    tweet = re.sub(r"NickCannon", "Nick Cannon", tweet)
    tweet = re.sub(r"FaroeIslands", "Faroe Islands", tweet)
    tweet = re.sub(r"yycstorm", "Calgary Storm", tweet)
    tweet = re.sub(r"IDPs:", "Internally Displaced People :", tweet)
    tweet = re.sub(r"ArtistsUnited", "Artists United", tweet)
    tweet = re.sub(r"ClaytonBryant", "Clayton Bryant", tweet)
    tweet = re.sub(r"jimmyfallon", "jimmy fallon", tweet)
    tweet = re.sub(r"justinbieber", "justin bieber", tweet)
    tweet = re.sub(r"UTC2015", "UTC 2015", tweet)
    tweet = re.sub(r"Time2015", "Time 2015", tweet)
    tweet = re.sub(r"djicemoon", "dj icemoon", tweet)
    tweet = re.sub(r"LivingSafely", "Living Safely", tweet)
    tweet = re.sub(r"FIFA16", "Fifa 2016", tweet)
    tweet = re.sub(r"thisiswhywecanthavenicethings", "this is why we cannot have nice things", twe
    tweet = re.sub(r"bbcnews", "bbc news", tweet)
    tweet = re.sub(r"UndergroundRailraod", "Underground Railraod", tweet)
    tweet = re.sub(r"c4news", "c4 news", tweet)
    tweet = re.sub(r"OBLITERATION", "obliteration", tweet)
    tweet = re.sub(r"MUDSLIDE", "mudslide", tweet)
    tweet = re.sub(r"NoSurrender", "No Surrender", tweet)
    tweet = re.sub(r"NotExplained", "Not Explained", tweet)
    tweet = re.sub(r"greatbritishbakeoff", "great british bake off", tweet)
    tweet = re.sub(r"LondonFire", "London Fire", tweet)
    tweet = re.sub(r"KOTAWeather", "KOTA Weather", tweet)
    tweet = re.sub(r"LuchaUnderground", "Lucha Underground", tweet)
    tweet = re.sub(r"KOIN6News", "KOIN 6 News", tweet)
    tweet = re.sub(r"LiveOnK2", "Live On K2", tweet)
    tweet = re.sub(r"9NewsGoldCoast", "9 News Gold Coast", tweet)
    tweet = re.sub(r"nikeplus", "nike plus", tweet)
    tweet = re.sub(r"david_cameron", "David Cameron", tweet)
    tweet = re.sub(r"peterjukes", "Peter Jukes", tweet)
    tweet = re.sub(r"JamesMelville", "James Melville", tweet)
    tweet = re.sub(r"megynkelly", "Megyn Kelly", tweet)
    tweet = re.sub(r"cnewslive", "C News Live", tweet)
    tweet = re.sub(r"JamaicaObserver", "Jamaica Observer", tweet)
    tweet = re.sub(r"TweetLikeItsSeptember11th2001", "Tweet like it is september 11th 2001", tweet
    tweet = re.sub(r"cbplawyers", "cbp lawyers", tweet)
    tweet = re.sub(r"fewmoretweets", "few more tweets", tweet)
    tweet = re.sub(r"BlackLivesMatter", "Black Lives Matter", tweet)
    tweet = re.sub(r"cjoyner", "Chris Joyner", tweet)
    tweet = re.sub(r"ENGvAUS", "England vs Australia", tweet)
    tweet = re.sub(r"ScottWalker", "Scott Walker", tweet)
    tweet = re.sub(r"MikeParrActor", "Michael Parr", tweet)
    tweet = re.sub(r"4PlayThursdays", "Foreplay Thursdays", tweet)
    tweet = re.sub(r"TGF2015", "Tontitown Grape Festival", tweet)
    tweet = re.sub(r"realmandyrain", "Mandy Rain", tweet)
    tweet = re.sub(r"GraysonDolan", "Grayson Dolan", tweet)
    tweet = re.sub(r"ApolloBrown", "Apollo Brown", tweet)
```