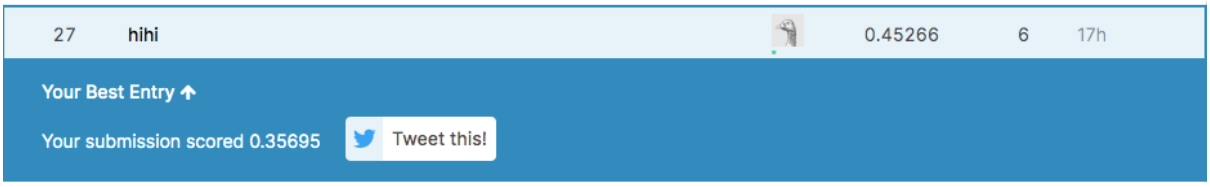
Kaggle Report

108064532

- Screenshot of leaderboard

public:



private:

- Result/Method

1. I use **(word embedding + w2v + LSTM)** and **(Bag of Word)**
2. And the result of **(word embedding + w2v + LSTM)** is better than **(Bag of Word)**

- Method and Observation

1. I found the choice of optimizer is quite important, the best suitable optimizer have strong impact on my training process(my optimizer is adam)
2. I also found if I lower the batch size, the result would be better
3. The emotions are imbalanced, but it's not suitable to use sampling

- Problems

1. I encountered underfitting at first, so my solutions are to make more epoch, make model more complex or modify the input.
2. It also means my model is to simple, so I adjust layer of my model and the units in hidden layer.
3. And it also means my input is not helping the model to distinguish different emotions.

- Code on jupyter(I will explain code below)

In [1]:

```
import pandas as pd
```

executed in 267ms, finished 14:48:06 2019-12-21

Load training data

In [2]:

```

#load the data we use
data_identification = pd.read_csv('data_identification.csv')
emotion = pd.read_csv('emotion.csv')
sampleSubmission = pd.read_csv('sampleSubmission.csv')

#test input

print(data_identification.head(5))
print('--------------------')
print(emotion.head(5))
print('--------------------')
print(sampleSubmission.head(5))
```

executed in 1.55s, finished 14:48:08 2019-12-21

```
   tweet_id identification
0  0x28cc61           test
1  0x29e452          train
2  0x2b3819          train
3  0x2db41f           test
4  0x2a2acc          train
--------------------
   tweet_id       emotion
0  0x3140b1       sadness
1  0x368b73       disgust
2  0x296183  anticipation
3  0x2bd6e1           joy
4  0x2ee1dd  anticipation
--------------------
         id   emotion
0  0x2c7743  surprise
1  0x2c1eed  surprise
2  0x2826ea  surprise
3  0x356d9a  surprise
4  0x20fd95  surprise
```

In [3]:

```
import json
from pandas.io.json import json_normalize

tweet_json = pd.read_json('tweets_DM.json', lines=True)
#used for nested format column['_source']
tweets_normalize = json_normalize(tweet_json['_source'])

tweets_normalize.head(5)
```

executed in 52.6s, finished 14:49:01 2019-12-21

Out[3]:

	tweet.hashtags	tweet.tweet_id	tweet.text
0	[Snapchat]	0x376b20	People who post "add me on #Snapchat" must be ...
1	[freepress, TrumpLegacy, CNN]	0x2d5350	@brianklaas As we see, Trump is dangerous to #...
2	[bibleverse]	0x28b412	Confident of your obedience, I write to you, k...
3	[]	0x1cd5b0	Now ISSA is stalking Tasha 😂 😂 😂 <LH>
4	[]	0x2de201	"Trust is not the same as faith. A friend is s...

In [4]:

```
tweets_normalize.rename(columns={'tweet.hashtags': 'hashtags', 'tweet.text': '
tweets_normalize.head(5)
```

executed in 136ms, finished 14:49:01 2019-12-21

Out[4]:

	hashtags	tweet_id	text
0	[Snapchat]	0x376b20	People who post "add me on #Snapchat" must be ...
1	[freepress, TrumpLegacy, CNN]	0x2d5350	@brianklaas As we see, Trump is dangerous to #...
2	[bibleverse]	0x28b412	Confident of your obedience, I write to you, k...
3	[]	0x1cd5b0	Now ISSA is stalking Tasha 😂 😂 😂 <LH>
4	[]	0x2de201	"Trust is not the same as faith. A friend is s...

Merge column of emotion and identification

In [5]:

```
#merge emotion to dataframe according tweet_id
#outer->union
merge_pd = pd.merge(tweets_normalize, data_identification, how='outer', on=['t
```

executed in 3.11s, finished 14:49:04 2019-12-21

In [6]:

```
merge_pd = pd.merge(merge_pd, emotion, how='outer', on=['tweet_id'])
```

executed in 2.93s, finished 14:49:07 2019-12-21

In [7]:

```
merge_pd.head(5)
```

executed in 8ms, finished 14:49:07 2019-12-21

Out[7]:

	hashtags	tweet_id	text	identification	emotion
0	[Snapchat]	0x376b20	People who post "add me on #Snapchat" must be ...	train	anticipation
1	[freepress, TrumpLegacy, CNN]	0x2d5350	@brianklaas As we see, Trump is dangerous to #...	train	sadness
2	[bibleverse]	0x28b412	Confident of your obedience, I write to you, k...	test	NaN
3	[]	0x1cd5b0	Now ISSA is stalking Tasha 😂😂😂 <LH>	train	fear
4	[]	0x2de201	"Trust is not the same as faith. A friend is s...	test	NaN

In [8]:

```
merge_pd.head(5).text
```

executed in 4ms, finished 14:49:07 2019-12-21

Out[8]:

```
0    People who post "add me on #Snapchat" must be ...
1    @brianklaas As we see, Trump is dangerous to #...
2    Confident of your obedience, I write to you, k...
3                  Now ISSA is stalking Tasha 😂😂😂 <LH>
4    "Trust is not the same as faith. A friend is s...
Name: text, dtype: object
```

In [9]:

```
tweets_train = merge_pd[merge_pd['identification'] == 'train']
tweets_test = merge_pd[merge_pd['identification'] == 'test']
```

executed in 866ms, finished 14:49:08 2019-12-21

In [10]:

```
print(tweets_train.count())
print('------------------')
print(tweets_test.count())
```

executed in 657ms, finished 14:49:08 2019-12-21

```
hashtags          1455563
tweet_id          1455563
text              1455563
identification    1455563
emotion           1455563
dtype: int64
------------------
hashtags          411972
tweet_id          411972
text              411972
identification    411972
emotion                0
dtype: int64
```

Save Data to Pickle

Some advantages for using pickle structure:

- Because it stores the attribute type, it's more convenient for cross-platform use.
- When your data is huge, it could use less space to store also consume less loading time.

In [11]:

```
## save to pickle file
tweets_train.to_pickle("train_df_Angel.pkl")
tweets_test.to_pickle("test_df_Angel.pkl")
```

executed in 3.81s, finished 14:49:12 2019-12-21

Read pickle

In [12]:

```
train_pickle = pd.read_pickle("train_df_Angel.pkl")
test_pickle = pd.read_pickle("test_df_Angel.pkl")
```

executed in 4.38s, finished 14:49:17 2019-12-21

In [13]:

```
#train_pickle['text'][3] = train_pickle['text'][3].replace('<LH>', str(train_p
print(train_pickle['text'][1455563])
#train_pickle.head(10)
```

executed in 6ms, finished 14:49:17 2019-12-21

```
Van Til: "God has an attitude of favor to men as men, whether they be
elect or reprobate." <LH> #CommonGrace #Election #Reprobation
```

Remove < LH >

In [14]:

```
def replaceLabel(text, emotion):
    temp = text[:]
    temp = temp.replace('<LH>', emotion)
    return temp
```

executed in 2ms, finished 14:49:17 2019-12-21

In [15]:

```
from tqdm import tqdm
tqdm.pandas(desc = "progress-bar")

train_pickle['replaceLabel'] = train_pickle.progress_apply(lambda x: replaceLa
```

executed in 23.1s, finished 14:49:40 2019-12-21

```
progress-bar: 100%|| 1455563/1455563 [00:23<00:00, 631
26.22it/s]
```

Tokenize the text column

In [17]:

```
from nltk.tokenize import TweetTokenizer
token = TweetTokenizer()
from tqdm import tqdm
tqdm.pandas(desc = "progress-bar")

```

executed in 809ms, finished 14:49:41 2019-12-21

In [18]:

```
train_pickle['Label_tokenized'] = train_pickle['text'].progress_apply(lambda x
#train_pickle.head(5)
```

executed in 50.3s, finished 14:50:31 2019-12-21

```
progress-bar: 100%|| 1455563/1455563 [00:50<00:00, 289
77.10it/s]
```

In [19]:

```
train_pickle.head(5)
```

executed in 21ms, finished 14:50:31 2019-12-21

Out[19]:

	hashtags	tweet_id	text	identification	emotion	replaceLabel	Lab
0	[Snapchat]	0x376b20	People who post "add me on #Snapchat" must be ...	train	anticipation	People who post "add me on #Snapchat" must be ...	[Pe
1	[freepress, TrumpLegacy, CNN]	0x2d5350	@brianklaas As we see, Trump is dangerous to #...	train	sadness	@brianklaas As we see, Trump is dangerous to #...	[w
3	[]	0x1cd5b0	Now ISSA is stalking Tasha 😂😂😂 <LH>	train	fear	Now ISSA is stalking Tasha 😂😂😂 fear	sta
5	[authentic, LaughOutLoud]	0x1d755c	@RISKshow @TheKevinAllison Thx for the BEST TI...	train	joy	@RISKshow @TheKevinAllison Thx for the BEST TI...	@
6	[]	0x2c91a8	Still waiting on those supplies Liscus. <LH>	train	anticipation	Still waiting on those supplies Liscus. antici...	[

In [23]:

```
print(train_corpus[0])
print(train_corpus[1])
```

executed in 2ms, finished 14:58:49 2019-12-21

```
['People', 'who', 'post', '"', 'add', 'me', 'on', '#Snapchat', '"', 'm
ust', 'be', 'dehydrated', '.', 'Cuz', 'man', '...', "that's", '<LH>']
['@brianklaas', 'As', 'we', 'see', ',', 'Trump', 'is', 'dangerous', 't
o', '#freepress', 'around', 'the', 'world', '.', 'What', 'a', '<LH>',
'<LH>', '#TrumpLegacy', '.', '#CNN']
```

Train Word2vec model

Train Word2Vec model myself

In [11]:

```
import gensim
from gensim.models import Word2Vec

##parameter setting
vector_dim = 50
window_size = 5
min_count = 1
training_iter = 25

## model
word2vec_model = Word2Vec(sentences=train_corpus,
                          size=vector_dim, window=window_size,
                          min_count=min_count, iter=training_iter)
```

executed in 54ms, finished 16:09:37 2019-12-21

...

In [ ]:

```
emoList = train_pickle['emotion'].unique().tolist()
print(emoList)
```

executed in 10m 32s, finished 14:58:38 2019-12-21

In [ ]:

```
word2vec_model.wv.most_similar('anticipation')
```

executed in 10m 32s, finished 14:58:38 2019-12-21

In [ ]:

```
word2vec_model.wv.most_similar('sadness')
```

executed in 10m 32s, finished 14:58:38 2019-12-21

In [ ]:

```
word2vec_model.wv.most_similar('fear')
```

executed in 10m 32s, finished 14:58:38 2019-12-21

In [ ]:

```
topn = 500

for i, emo in enumerate(emoList):

    similarList = [emo]
    print(similarList)
    similarList.extend([word_ for word_, sim_ in word2vec_model.wv.most_simila

    #the first emtion
    if (i == 0):
        df = pd.DataFrame({'label':similarList})
        df['emotion'] = [emo for i in range(topn+1)]
    #remaining emotions
    else:
        df2 = pd.DataFrame({'label':similarList})
        df2['emotion'] = [emo for i in range(topn+1)]
        df = df.append(df2)
    print(df)
```

executed in 10m 32s, finished 14:58:38 2019-12-21

In [ ]:

```
from sklearn.manifold import TSNE

# w2v model
model = word2vec_model
target_words = df['label'].tolist()
```

executed in 10m 32s, finished 14:58:38 2019-12-21

In [ ]:

```
target_words
```

executed in 10m 32s, finished 14:58:38 2019-12-21

In [ ]:

```
len(target_words)
```

executed in 10m 32s, finished 14:58:38 2019-12-21

In [ ]:

```
# prepare training word vectors
size = 200
target_size = len(target_words)
all_word = list(model.wv.vocab.keys())
word_train = target_words + all_word[:size]
X_train = model.wv[word_train]

# t-SNE model
tsne = TSNE(n_components=2, metric='cosine', random_state=28)

## training
X_tsne = tsne.fit_transform(X_train)
```

executed in 10m 32s, finished 14:58:38 2019-12-21

In [ ]:

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')

df['X'] = X_tsne[:target_size, 0]
df['Y'] = X_tsne[:target_size, 1]

sns.set(rc={'figure.figsize':(7.5,7.5), 'figure.dpi':150})
sns.lmplot( x="X", y="Y", data=df, fit_reg=False, hue='emotion', legend=False,
plt.legend(loc='lower right', bbox_to_anchor=(1.4, 0))
```

executed in 10m 32s, finished 14:58:38 2019-12-21

Everything just glues together, which indicate that it probably won't give a good result

In [ ]:

```
df[df['emotion'] == 'anticipation'].head(10)
```

executed in 10m 32s, finished 14:58:38 2019-12-21

Count the occurrence of words

In [ ]:

```
from sklearn.feature_extraction.text import CountVectorizer
from nltk.tokenize import TweetTokenizer
tknzr = TweetTokenizer()

# build analyzers (bag-of-words)
BOW = CountVectorizer(analyzer='word', stop_words=None, lowercase=False, token

# apply analyzer to training data
BOW.fit(train_pickle['text'])
```

executed in 10m 32s, finished 14:58:38 2019-12-21

In [ ]:

```
len(BOW.get_feature_names())
```

executed in 10m 32s, finished 14:58:38 2019-12-21

In [ ]:

```
print(BOW.get_feature_names()[:100])
```

executed in 10m 32s, finished 14:58:38 2019-12-21

Too many unique words, so it's not suitable to use BOW. Because the matrix would be too large.

Clean Data

So we need to clean data

Function of cleaning data

In [25]:

```
import nltk
import string
from nltk.tokenize import TweetTokenizer
tknzr = TweetTokenizer(preserve_case=False)

s_word = list(string.punctuation)
s_word.append('<lh>')
s_word.append('’’')
s_word = set(s_word)

def clean(text):
    result = tknzr.tokenize(text)

    for i, word in enumerate(result):
        temp = word.split('#')
        for w in temp:
            if w != '':
                result[i] = w

    for i, word in enumerate(result):
        temp = word.split('@')
        for w in temp:
            if w != '':
                result[i] = w

    result = [word for word in result if not word in s_word]
    result = ' '.join(result)
    return result
```

executed in 6ms, finished 15:02:01 2019-12-21

In [26]:

```
train_pickle['clean_text'] = train_pickle['text'].progress_apply(lambda x: cle
```

executed in 1m 8.18s, finished 15:03:11 2019-12-21

```
progress-bar: 100%|| 1455563/1455563 [01:08<00:00, 213
65.46it/s]
```

In [27]:

```
train_pickle['tokenized'] = train_pickle['clean_text'].progress_apply(lambda x
```

executed in 53.6s, finished 15:04:04 2019-12-21

```
progress-bar: 100%|| 1455563/1455563 [00:53<00:00, 271
50.03it/s]
```

In [28]:

```
1  test_pickle['clean_text'] = test_pickle['text'].progress_apply(lambda x: clean
```

executed in 22.6s, finished 15:04:27 2019-12-21

```
progress-bar: 100%|| 411972/411972 [00:22<00:00, 1821
0.70it/s]
```

In [29]:

```
1  test_pickle['tokenized'] = test_pickle['clean_text'].progress_apply(lambda x:
```

executed in 17.5s, finished 15:04:45 2019-12-21

```
progress-bar: 100%|| 411972/411972 [00:17<00:00, 2351
9.64it/s]
```

In [30]:

```
1  train_pickle.head(5)
```

executed in 40ms, finished 15:04:45 2019-12-21

Out[30]:

	hashtags	tweet_id	text	identification	emotion	replaceLabel	Lab
0	[Snapchat]	0x376b20	People who post "add me on #Snapchat" must be ...	train	anticipation	People who post "add me on #Snapchat" must be ...	[Pe
1	[freepress, TrumpLegacy, CNN]	0x2d5350	@brianklaas As we see, Trump is dangerous to #...	train	sadness	@brianklaas As we see, Trump is dangerous to #...	[w
3	[]	0x1cd5b0	Now ISSA is stalking Tasha 😂😂😂 <LH>	train	fear	Now ISSA is stalking Tasha 😂😂😂 fear	sta
5	[authentic, LaughOutLoud]	0x1d755c	@RISKshow @TheKevinAllison Thx for the BEST TI...	train	joy	@RISKshow @TheKevinAllison Thx for the BEST TI...	@
6	[]	0x2c91a8	Still waiting on those supplies Liscus. <LH>	train	anticipation	Still waiting on those supplies Liscus. antici...	[

Consider words both exist in train and test

In [31]:

```
from collections import Counter

def findToken(df):
    cnt = Counter()
    for text in tqdm(df['tokenized'].values):
        if text != '':
            cnt.update(text)
    tokenized_only_dict = cnt

    tokenized_only_df = pd.DataFrame.from_dict(tokenized_only_dict, orient='in
    tokenized_only_df.rename(columns={0: 'count'}, inplace=True)
    tokenized_only_df.sort_values('count', ascending=False, inplace=True)
    return tokenized_only_df
```

executed in 6ms, finished 15:05:08 2019-12-21

In [32]:

```
trainToken = findToken(train_pickle)
testToken = findToken(test_pickle)
print(len(trainToken))
print(len(testToken))
```

executed in 5.74s, finished 15:05:25 2019-12-21

```
100%|██████████| 1455563/1455563 [00:04<00:00, 360255.79it/s]
100%|██████████| 411972/411972 [00:01<00:00, 343268.25it/s]

832561
299232
```

In [33]:

```
# unique in train
s1 = trainToken[~trainToken.index.isin(testToken.index)].index.tolist()
# unique in test
s2 = testToken[~testToken.index.isin(trainToken.index)].index.tolist()

s_word = s1 + s2
s_word.append('')
s_word = set(s_word)

del trainToken
del testToken
```

executed in 599ms, finished 15:06:27 2019-12-21

In [34]:

```
train_pickle['clean_text'] = train_pickle['tokenized'].progress_apply(
    lambda x: ' '.join([item for item in x if item not in s_word]))

test_pickle['clean_text'] = test_pickle['tokenized'].progress_apply(
    lambda x: ' '.join([item for item in x if item not in s_word]))

train_pickle['tokenized'] = train_pickle['clean_text'].progress_apply(lambda x
test_pickle['tokenized'] = test_pickle['clean_text'].progress_apply(lambda x:
```

executed in 10.5s, finished 15:06:44 2019-12-21

```
progress-bar: 100%|| 1455563/1455563 [00:02<00:00, 548
444.02it/s]
progress-bar: 100%|| 411972/411972 [00:00<00:00, 52695
0.43it/s]
progress-bar: 100%|| 1455563/1455563 [00:05<00:00, 249
622.49it/s]
progress-bar: 100%|| 411972/411972 [00:00<00:00, 67645
5.73it/s]
```

In [35]:

```
trainToken = findToken(train_pickle)
testToken = findToken(test_pickle)
print(len(trainToken))
print(len(testToken))
```

executed in 5.13s, finished 15:06:51 2019-12-21

```
100%| 1455563/1455563 [00:03<00:00, 389004.06it/s]
100%| 411972/411972 [00:01<00:00, 333401.92it/s]

154225
154225
```

Remove conjunctions

In [36]:

```
trainToken = findToken(train_pickle)
trainToken = trainToken.drop([''])
```

executed in 3.77s, finished 15:06:59 2019-12-21

```
100%| 1455563/1455563 [00:03<00:00, 400467.65it/s]
```

In [37]:

```
import nltk

def findRedundant():
    result = []
    tags = nltk.pos_tag(trainToken.index.tolist())
    for tag in tags:
        if (tag[1] == 'DT' or
            tag[1] == 'TO' or
            tag[1] == 'PRP' or
            tag[1] == 'IN' or
            tag[1] == 'PRP$' or
            tag[1] == 'CC' or
            tag[1] == 'CD' or
            tag[1] == ':'):
            result.append(tag[0])
    return result
```

executed in 5ms, finished 15:07:22 2019-12-21

In [38]:

```
re = findRedundant()
print(len(re))
re = set(re)
```

executed in 5.87s, finished 15:07:37 2019-12-21

```
10268
```

In [39]:

```
train_pickle['clean_text'] = train_pickle['tokenized'].progress_apply(
    lambda x: ' '.join([item for item in x if item not in re]))

test_pickle['clean_text'] = test_pickle['tokenized'].progress_apply(
    lambda x: ' '.join([item for item in x if item not in re]))

train_pickle['tokenized'] = train_pickle['clean_text'].progress_apply(lambda x
test_pickle['tokenized'] = test_pickle['clean_text'].progress_apply(lambda x:
```

executed in 5.70s, finished 15:07:48 2019-12-21

```
progress-bar: 100%|| 1455563/1455563 [00:02<00:00, 649
149.07it/s]
progress-bar: 100%|| 411972/411972 [00:00<00:00, 53606
0.67it/s]
progress-bar: 100%|| 1455563/1455563 [00:01<00:00, 929
798.93it/s]
progress-bar: 100%|| 411972/411972 [00:00<00:00, 86702
5.03it/s]
```

In [49]:

```
1  train_pickle.head(5)
```

executed in 31ms, finished 15:16:25 2019-12-21

Out[49]:

	hashtags	tweet_id	text	identification	emotion	replaceLabel	Lab
0	[Snapchat]	0x376b20	People who post "add me on #Snapchat" must be ...	train	anticipation	People who post "add me on #Snapchat" must be ...	[Pe
1	[freepress, TrumpLegacy, CNN]	0x2d5350	@brianklaas As we see, Trump is dangerous to #...	train	sadness	@brianklaas As we see, Trump is dangerous to #...	[w
3	[]	0x1cd5b0	Now ISSA is stalking Tasha 😂😂😂 <LH>	train	fear	Now ISSA is stalking Tasha 😂😂😂 fear	sta
5	[authentic, LaughOutLoud]	0x1d755c	@RISKshow @TheKevinAllison Thx for the BEST TI...	train	joy	@RISKshow @TheKevinAllison Thx for the BEST TI...	@
6	[]	0x2c91a8	Still waiting on those supplies Liscus. <LH>	train	anticipation	Still waiting on those supplies Liscus. antici...	[

In [53]:

```
1  train_pickle.to_pickle("./train_df_clean.pkl")
2  test_pickle.to_pickle("./test_df_clean.pkl")
```

executed in 16.2s, finished 15:22:29 2019-12-21

Feature Engineering

In [1]:

```
1  import pandas as pd
2
3  train_df = pd.read_pickle("./train_df_clean.pkl")
4  test_df = pd.read_pickle("./test_df_clean.pkl")
```

executed in 12.1s, finished 16:22:06 2019-12-21

BOW

BOW

In [2]:

```
from sklearn.feature_extraction.text import CountVectorizer
from nltk.tokenize import TweetTokenizer
tknzr = TweetTokenizer()

# build analyzers (bag-of-words)
BOW = CountVectorizer(analyzer='word', max_features=3000, max_df=0.8, tokenize

# apply analyzer to training data
BOW.fit(train_df['clean_text'])
```

executed in 38.7s, finished 15:25:57 2019-12-21

Out[2]:

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='conten
t',
        lowercase=True, max_df=0.8, max_features=3000, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=<bound method TweetTokenizer.tokenize of <nltk.token
ize.casual.TweetTokenizer object at 0x7f9c3f4fc978>>,
        vocabulary=None)
```

In [3]:

```
BOW_train = BOW.transform(train_df['clean_text'])
```

executed in 37.9s, finished 15:26:38 2019-12-21

w2v

In [4]:

```
1  train_df.head(5)
```

executed in 18ms, finished 15:27:06 2019-12-21

Out[4]:

	hashtags	tweet_id	text	identification	emotion	replaceLabel	Lab
0	[Snapchat]	0x376b20	People who post "add me on #Snapchat" must be ...	train	anticipation	People who post "add me on #Snapchat" must be ...	[Pe
1	[freepress, TrumpLegacy, CNN]	0x2d5350	@brianklaas As we see, Trump is dangerous to #...	train	sadness	@brianklaas As we see, Trump is dangerous to #...	[w
3	[]	0x1cd5b0	Now ISSA is stalking Tasha 😂😂😂 <LH>	train	fear	Now ISSA is stalking Tasha 😂😂😂 fear	sta
5	[authentic, LaughOutLoud]	0x1d755c	@RISKshow @TheKevinAllison Thx for the BEST TI...	train	joy	@RISKshow @TheKevinAllison Thx for the BEST TI...	@
6	[]	0x2c91a8	Still waiting on those supplies Liscus. <LH>	train	anticipation	Still waiting on those supplies Liscus. antici...	[

In [2]:

```
1  training_corpus = train_df['tokenized'].values
2  training_corpus[:4]
```

executed in 12ms, finished 16:22:06 2019-12-21

Out[2]:

```
array([list(['people', 'who', 'post', 'add', 'snapchat', 'must', 'be',
'dehydrated', 'cuz', 'man', "that's"]),
       list(['brianklaas', 'see', 'trump', 'is', 'dangerous', 'freepre
ss', 'around', 'world', 'what', 'cnn']),
       list(['now', 'issa', 'is', 'stalking', 'tasha', '😂', '😂',
'😂']),
       list(['riskshow', 'thx', 'best', 'time', 'tonight', 'what', 'st
ories', 'heartbreakingly', 'authentic', 'laughoutloud', 'good'])],
      dtype=object)
```

In [3]:

```
import gensim
from gensim.models import Word2Vec

## setting
vector_dim = 50
window_size = 5
min_count = 1
training_iter = 20

## model
word2vec_model = Word2Vec(sentences=training_corpus,
                          size=vector_dim, window=window_size,
                          min_count=min_count, iter=training_iter)
word2vec_model.save("word2vec_twitter_50.model")
```

executed in 3m 19s, finished 16:27:16 2019-12-21

In [ ]:

```
import numpy as np
from tqdm import tqdm
tqdm.pandas(desc = "progress-bar")

w2v_train = np.array([np.mean([word2vec_model.wv[w] for w in words if w in wor
                  [np.zeros(vector_dim)], axis=0) for words in tqdm(train_df.c
         ])
```

executed in 1m 40.5s, finished 16:13:42 2019-12-21

glove

In [ ]:

```
embeddings_index = dict();
with open('glove.twitter.27B.50d.txt','r', encoding='utf-8') as f:
    for line in f:
        values = line.split();
        word = values[0];
        coefs = np.asarray(values[1:], dtype='float32');
        embeddings_index[word] = coefs;
print('Found %s word vectors.' % len(embeddings_index))
```

executed in 10m 32s, finished 14:58:38 2019-12-21

In [ ]:

```
glove_train = np.array([np.mean([embeddings_index[w] for w in words if w in em
                  [np.zeros(vector_dim)], axis=0) for words in tqdm(train_df.c
         ])
```

executed in 10m 32s, finished 14:58:38 2019-12-21

w2v & TFIDF

In [2]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import TweetTokenizer
tknzr = TweetTokenizer()

# build analyzers (bag-of-words)
TFID = TfidfVectorizer(analyzer='word', stop_words=None, lowercase=False, toke

# apply analyzer to training data
TFID.fit(train_df['clean_text'])
```

executed in 37.6s, finished 16:08:01 2019-12-21

Out[2]:

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='conten
t',
        lowercase=False, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=T
rue,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=<bound method TweetTokenizer.tokenize of <nltk.token
ize.casual.TweetTokenizer object at 0x7efd3828f908>>,
        use_idf=True, vocabulary=None)
```

In [ ]:

```
import numpy as np
from tqdm import tqdm
tqdm.pandas(desc = "progress-bar")

w2v_train = np.array([np.mean([word2vec_model.wv[w] * TFID.idf_[TFID.vocabular
                    [np.zeros(vector_dim)], axis=0) for words in tqdm(train_df.c
          ])
```

executed in 1m 59.5s, finished 16:17:14 2019-12-21

In [8]:

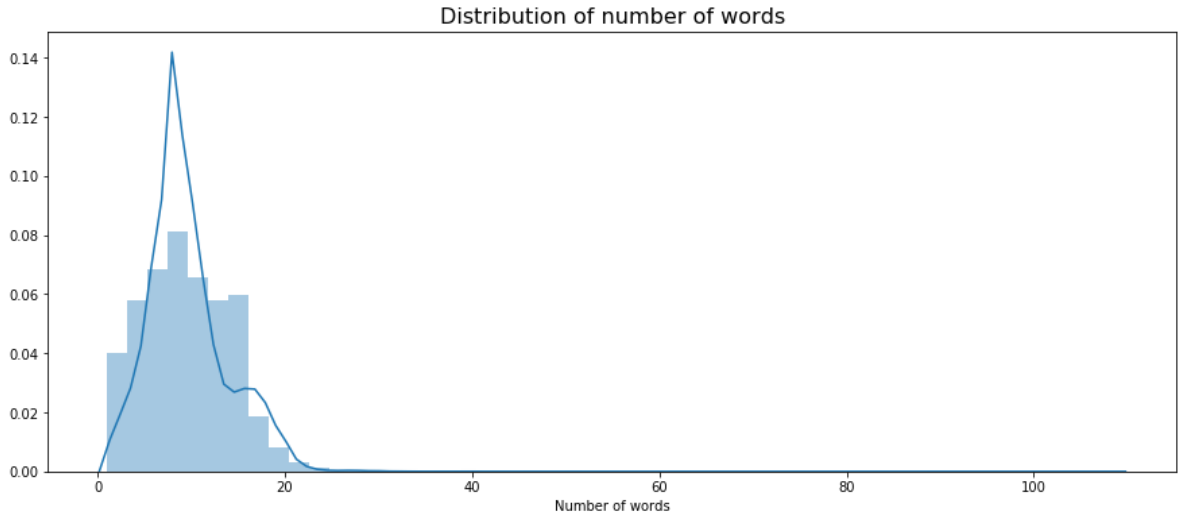
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
document_lengths = np.array(list(map(len, train_df.clean_text.str.split(' '))))

fig, ax = plt.subplots(figsize=(15,6))
ax.set_title("Distribution of number of words", fontsize=16)
ax.set_xlabel("Number of words")
sns.distplot(document_lengths, bins=50, ax=ax)
```

executed in 7.13s, finished 15:30:42 2019-12-21

Out[8]:

Plot to see the distribution of number of words

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9c174fc9b0>
```



In [2]:

```
import pandas as pd
from gensim.models import Word2Vec

word2vec_model = Word2Vec.load("word2vec_twitter_50.model")

train_df = pd.read_pickle("./train_df_clean.pkl")
test_df = pd.read_pickle("./test_df_clean.pkl")
```

executed in 12.9s, finished 16:41:26 2019-12-21

In [4]:

```
# padding for word embedding
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

MAX_SEQUENCE_LENGTH = 20

tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_df['clean_text'])
sequences = tokenizer.texts_to_sequences(train_df['clean_text'])

word_index = tokenizer.word_index

wordEmbedding_w2v_train  = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH
```

executed in 35.7s, finished 16:32:57 2019-12-21

Construct the embedding matrix

In [6]:

```
# form embedding matrix (w2v)
import numpy as np
from tqdm import tqdm
vector_dim = 50
embedding_matrix = np.zeros((len(word_index) + 1, vector_dim))
missingWord = []
for word, i in tqdm(word_index.items()):
    try:
        embedding_vector = word2vec_model.wv[word]
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
    except:
        missingWord.append(word)
```

executed in 280ms, finished 16:33:46 2019-12-21

```
100%|| 138999/138999 [00:00<00:00, 509909.10it/s]
```

In [7]:

```
len(missingWord)
```

executed in 3ms, finished 16:29:43 2019-12-21

Out[7]:

```
2347
```

In [12]:

```
print(missingWord)
```

executed in 38ms, finished 15:32:13 2019-12-21

```
s', 'content', 'instead', 'worst', 'positive', 'guess', 'past', 'watch
ed', 'story', 'gone', 'turn', 'used', 'check', 'single', 'truth', 'se
t', 'dont', 'saying', 'rest', 'woke', '🌈 ', 'move', 'july', 'decembe
r', '😱 ', 'late', 'self', 'donniewahlberg', 'pay', 'gotta', 'listen',
'inspiration', 'absolutely', 'forget', '🇨 ', 'means', 'favorite', 'fat
her', 'loves', 'send', 'feels', 'sorry', 'heard', 'post', 'strong', 'h
eart', 'half', 'eyes', 'months', 'coffee', 'boy', 'party', '💞 ', 'fema
le', 'wisdom', 'early', 'safe', 'order', 'stand', 'point', 'dad', 'sha
ll', 'won', 'hit', 'issa', 'cnn', 'vote', 'room', 'minutes', 'childre
n', "you've", 'side', 'smile', 'dead', 'share', 'tweets', 'thoughts',
'goes', 'prayers', 'knows', 'football', 'media', 'fight', 'wanted', 'l
ives', '👊 ', 'yeah', 'dear', 'pain', 'dog', 'fake', 'water', 'welcom
e', 'hair', 'summer', 'wake', 'body', 'sick', 'fan', 'x', 'least', 'al
ive', 'meet', 'nfl', 'lose', 'lawrence', 'gift', 'stuff', '3', 'spiri
t', 'usa', 'dinner', '💚 ', "she's", 'maga', 'death', 'blessing', '💗 ',
'💛 ', 'human', 'break', "they're", 'sweet', '😳 ', 'plan', 'funny', 'co
rinthians', 'parents', 'hand', "haven't", 'strength', 'lucky', 'youtub
e', 'second', 'ones', 'gives', 'happened', 'happiness', 'anymore', 'br
other', 'bought', 'important', 'gratitude', 'hello', 'voice', 'worth',
```

In [7]:

```
#modeling
# for a classificaiton problem, you need to provide both training & testing da
div = int(train_df.shape[0]*0.8)

wordEmbedding_w2v_X_train = wordEmbedding_w2v_train[:div]
wordEmbedding_w2v_y_train = train_df['emotion'][:div]

wordEmbedding_w2v_X_test = wordEmbedding_w2v_train[div:]
wordEmbedding_w2v_y_test = train_df['emotion'][div:]

## take a look at data dimension is a good habbit  :)
print('we_w2v_X_train.shape: ', wordEmbedding_w2v_X_train.shape)
print('we_w2v_y_train.shape: ', wordEmbedding_w2v_y_train.shape)
print('we_w2v_X_test.shape: ', wordEmbedding_w2v_X_test.shape)
print('we_w2v_y_test.shape: ', wordEmbedding_w2v_y_test.shape)
```

executed in 10ms, finished 16:33:51 2019-12-21

```
we_w2v_X_train.shape:  (1164450, 20)
we_w2v_y_train.shape:  (1164450,)
we_w2v_X_test.shape:  (291113, 20)
we_w2v_y_test.shape:  (291113,)
```

Deep learning - LSTM

In [8]:

```
import keras
from sklearn.preprocessing import LabelEncoder

def label_encode(le, labels):
    enc = le.transform(labels)
    return keras.utils.to_categorical(enc)

def label_decode(le, one_hot_label):
    dec = np.argmax(one_hot_label, axis=1)
    return le.inverse_transform(dec)
```

executed in 9ms, finished 16:33:53 2019-12-21

Check the shape of dataset

In [9]:

```
def encode(y_train, y_test):
    label_encoder = LabelEncoder()
    label_encoder.fit(y_train)
    print('check label: ', label_encoder.classes_)
    print('\n## Before convert')
    print('y_train[0:4]:\n', y_train[0:4])
    print('\ny_train.shape: ', y_train.shape)
    print('y_test.shape: ', y_test.shape)

    re_y_train = label_encode(label_encoder, y_train)
    re_y_test = label_encode(label_encoder, y_test)

    print('\n\n## After convert')
    print('y_train[0:4]:\n', y_train[0:4])
    print('\ny_train.shape: ', y_train.shape)
    print('y_test.shape: ', y_test.shape)

    return re_y_train, re_y_test, label_encoder
```

executed in 6ms, finished 16:33:54 2019-12-21

In [10]:

```
le_wordEmbedding_w2v_y_train, le_wordEmbedding_w2v_y_test, label_encoder = enc
```

executed in 152ms, finished 16:33:55 2019-12-21

```
check label:  ['anger' 'anticipation' 'disgust' 'fear' 'joy' 'sadness'
'surprise'
 'trust']

## Before convert
y_train[0:4]:
 0    anticipation
1         sadness
3            fear
5             joy
Name: emotion, dtype: object

y_train.shape:  (1164450,)
y_test.shape:  (291113,)


## After convert
y_train[0:4]:
 0    anticipation
1         sadness
3            fear
5             joy
Name: emotion, dtype: object

y_train.shape:  (1164450,)
y_test.shape:  (291113,)
```

In [11]:

```
# I/O check
input_shape = wordEmbedding_w2v_X_train.shape[1]
print('input_shape: ', input_shape)

output_shape = len(label_encoder.classes_)
print('output_shape: ', output_shape)
```

executed in 9ms, finished 16:33:56 2019-12-21

```
input_shape:  20
output_shape:  8
```

In [1]:

```
from keras.models import Sequential
from keras.layers import Dense, LSTM, GRU, Dropout, Activation, ActivityRegula
from keras.layers.embeddings import Embedding
from keras.regularizers import l2
from keras.initializers import Constant
from keras import optimizers

model = Sequential()
model.add(Embedding(len(word_index) + 1,
                            vector_dim,
                            embeddings_initializer=Constant(embedding_matrix),
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=False))
# model.add(SpatialDropout1D(0.7))
model.add(LSTM(512, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(8, activation='softmax'))

adam = optimizers.Adamax(lr=0.002, beta_1=0.9, beta_2=0.999)
model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['acc']

model.summary()
```

executed in 1.50s, finished 16:34:13 2019-12-21

...

In [14]:

```
wordEmbedding_w2v_X_test
```

executed in 4ms, finished 16:29:58 2019-12-21

Out[14]:

```
array([[    0,     0,     0, ...,    96,    14,    49],
       [    0,     0,     0, ...,  4209, 47041, 41258],
       [    0,     0,     0, ...,   667,  1656,   384],
       ...,
       [    0,     0,     0, ...,   767, 10562, 35402],
       [    0,     0,     0, ...,   208,  2336,  4147],
       [    0,     0,     0, ...,     3,   381, 13268]], dtype=int32)
```

In [ ]:

```
epochs = 3
batch_size = 128
history = model.fit(wordEmbedding_w2v_X_train, le_wordEmbedding_w2v_y_train,
                    epochs=epochs, batch_size=batch_size,
                    validation_data = (wordEmbedding_w2v_X_test, le_wordEmbedd
```

execution queued 16:34:01 2019-12-21

In [ ]:

```
## precision, recall, f1-score,
from sklearn.metrics import classification_report

y_pred_result = model.predict(x = wordEmbedding_w2v_X_test, batch_size=128)
y_pred_result = label_decode(label_encoder, y_pred_result)
print(classification_report(y_true=wordEmbedding_w2v_y_test, y_pred=y_pred_res
```

executed in 10.3s, finished 16:29:15 2019-12-21

In [ ]:

```

```

In [1]:

Final process to train model(I save the pickle after preprocessing, and directly load it)

```
import pandas as pd
from gensim.models import Word2Vec

# form embedding matrix (w2v)
import numpy as np
from tqdm import tqdm

# padding for word embedding
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

import keras
from sklearn.preprocessing import LabelEncoder

word2vec_model = Word2Vec.load("word2vec_twitter_50.model")

train_df = pd.read_pickle("./train_df_clean.pkl")
test_df = pd.read_pickle("./test_df_clean.pkl")



MAX_SEQUENCE_LENGTH = 20

tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_df['clean_text'])
sequences = tokenizer.texts_to_sequences(train_df['clean_text'])

word_index = tokenizer.word_index

wordEmbedding_w2v_train  = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)


vector_dim = 50
embedding_matrix = np.zeros((len(word_index) + 1, vector_dim))
missingWord = []
for word, i in tqdm(word_index.items()):
    try:
        embedding_vector = word2vec_model.wv[word]
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
    except:
        missingWord.append(word)

#modeling
# for a classificaiton problem, you need to provide both training & testing dat
div = int(train_df.shape[0]*0.8)

wordEmbedding_w2v_X_train = wordEmbedding_w2v_train[:div]
wordEmbedding_w2v_y_train = train_df['emotion'][:div]

wordEmbedding_w2v_X_test = wordEmbedding_w2v_train[div:]
wordEmbedding_w2v_y_test = train_df['emotion'][div:]

def label_encode(le, labels):
    enc = le.transform(labels)
    return keras.utils.to_categorical(enc)

def label_decode(le, one_hot_label):
    dec = np.argmax(one_hot_label, axis=1)
```

```
    return le.inverse_transform(dec)

def encode(y_train, y_test):
    label_encoder = LabelEncoder()
    label_encoder.fit(y_train)
    print('check label: ', label_encoder.classes_)
    print('\n## Before convert')
    print('y_train[0:4]:\n', y_train[0:4])
    print('\ny_train.shape: ', y_train.shape)
    print('y_test.shape: ', y_test.shape)

    re_y_train = label_encode(label_encoder, y_train)
    re_y_test = label_encode(label_encoder, y_test)

    print('\n\n## After convert')
    print('y_train[0:4]:\n', y_train[0:4])
    print('\ny_train.shape: ', y_train.shape)
    print('y_test.shape: ', y_test.shape)

    return re_y_train, re_y_test, label_encoder

le_wordEmbedding_w2v_y_train, le_wordEmbedding_w2v_y_test, label_encoder = enco

# I/O check
input_shape = wordEmbedding_w2v_X_train.shape[1]
print('input_shape: ', input_shape)

output_shape = len(label_encoder.classes_)
print('output_shape: ', output_shape)

from keras.models import Sequential
from keras.layers import Dense, LSTM, GRU, Dropout, Activation, ActivityRegular
from keras.layers.embeddings import Embedding
from keras.regularizers import l2
from keras.initializers import Constant
from keras import optimizers

model = Sequential()
model.add(Embedding(len(word_index) + 1,
                            vector_dim,
                            embeddings_initializer=Constant(embedding_matrix),
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=False))
# model.add(SpatialDropout1D(0.7))
model.add(LSTM(256, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))#0.2
model.add(Dense(8, activation='softmax'))
model.add(Dense(8, activation='softmax'))

# adam = optimizers.Adamax(lr=0.002, beta_1=0.9, beta_2=0.999)
model.compile(
    optimizer="nadam",
    loss='categorical_crossentropy',
    metrics=['acc'])

model.summary()

epochs = 5
batch_size = 32
history = model.fit(wordEmbedding_w2v_X_train, le_wordEmbedding_w2v_y_train,
```

```
                    epochs=epochs, batch_size=batch_size, verbose=1,
                    validation_data = (wordEmbedding_w2v_X_test, le_wordEmbeddi

## precision, recall, f1-score,
from sklearn.metrics import classification_report

y_pred_result = model.predict(x = wordEmbedding_w2v_X_test, batch_size=128)
y_pred_result = label_decode(label_encoder, y_pred_result)
print(classification_report(y_true=wordEmbedding_w2v_y_test, y_pred=y_pred_resu
```

```
Using TensorFlow backend.
/home/hsnl-iot/DataMining_2019/VENV/DataMining/lib/python3.5/site-pack
ages/tensorflow/python/framework/dtypes.py:523: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future v
ersion of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/hsnl-iot/DataMining_2019/VENV/DataMining/lib/python3.5/site-pack
ages/tensorflow/python/framework/dtypes.py:524: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future v
ersion of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/hsnl-iot/DataMining_2019/VENV/DataMining/lib/python3.5/site-pack
ages/tensorflow/python/framework/dtypes.py:525: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future v
ersion of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/hsnl-iot/DataMining_2019/VENV/DataMining/lib/python3.5/site-pack
ages/tensorflow/python/framework/dtypes.py:526: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future v
ersion of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/hsnl-iot/DataMining_2019/VENV/DataMining/lib/python3.5/site-pack
ages/tensorflow/python/framework/dtypes.py:527: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future v
ersion of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/hsnl-iot/DataMining_2019/VENV/DataMining/lib/python3.5/site-pack
ages/tensorflow/python/framework/dtypes.py:532: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future v
ersion of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
100%|██████████| 138999/138999 [00:00<00:00, 241942.83it/s]

check label:  ['anger' 'anticipation' 'disgust' 'fear' 'joy' 'sadness'
'surprise'
 'trust']

## Before convert
y_train[0:4]:
 0    anticipation
1         sadness
3            fear
5             joy
Name: emotion, dtype: object

y_train.shape:  (1164450,)
y_test.shape:  (291113,)


## After convert
y_train[0:4]:
```

```
  0    anticipation
1         sadness
3            fear
5             joy
Name: emotion, dtype: object

y_train.shape:  (1164450,)
y_test.shape:  (291113,)
input_shape:  20
output_shape:  8
```

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 20, 50)	6950000
lstm_1 (LSTM)	(None, 256)	314368
dense_1 (Dense)	(None, 512)	131584
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 8)	4104
dense_3 (Dense)	(None, 8)	72

```
Total params: 7,400,128
Trainable params: 450,128
Non-trainable params: 6,950,000

Train on 1164450 samples, validate on 291113 samples
Epoch 1/5
1164450/1164450 [==============================] - 842s 723us/step - l
oss: 1.3942 - acc: 0.4976 - val_loss: 1.3239 - val_acc: 0.5259
Epoch 2/5
1164450/1164450 [==============================] - 846s 726us/step - l
oss: 1.3554 - acc: 0.5123 - val_loss: 1.3132 - val_acc: 0.5307
Epoch 3/5
1164450/1164450 [==============================] - 847s 727us/step - l
oss: 1.3524 - acc: 0.5137 - val_loss: 1.3099 - val_acc: 0.5297
Epoch 4/5
1164450/1164450 [==============================] - 844s 724us/step - l
oss: 1.3486 - acc: 0.5150 - val_loss: 1.3099 - val_acc: 0.5322
Epoch 5/5
1164450/1164450 [==============================] - 846s 727us/step - l
oss: 1.3483 - acc: 0.5153 - val_loss: 1.3010 - val_acc: 0.5335
```

	precision	recall	f1-score	support
anger	0.68	0.16	0.26	8003
anticipation	0.66	0.49	0.56	49901
disgust	0.44	0.38	0.41	27449
fear	0.70	0.33	0.45	12844
joy	0.52	0.83	0.64	103127
sadness	0.45	0.47	0.46	38638
surprise	0.72	0.14	0.24	9816
trust	0.61	0.23	0.34	41335
accuracy			0.53	291113
macro avg	0.60	0.38	0.42	291113
weighted avg	0.56	0.53	0.51	291113

In [2]:

Save final result to prediction.csv

```
## predict
# tests = pd.read_pickle("test_df_clean.pkl")

test_sequences = tokenizer.texts_to_sequences(test_df["clean_text"])
test_sequences  = pad_sequences(test_sequences, maxlen=MAX_SEQUENCE_LENGTH)

pred_result = model.predict(test_sequences, batch_size=128)
pred_result = np.array(label_decode(label_encoder, pred_result))
print(pred_result.shape)
test_df['emotion'] = pred_result
test_df = test_df.drop('hashtags', axis=1)
test_df = test_df.drop('text', axis=1)
test_df = test_df.drop('identification', axis=1)
test_df = test_df.drop('clean_text', axis=1)
test_df = test_df.drop('tokenized', axis=1)
test_df.rename(columns={'tweet_id':'id'}, inplace=True)

test_df.to_csv('prediction.csv', index=False)
```

```
(411972,)
```