1 Natural Language Processing with Disaster Tweets

Jiajie Xu, 11Mar2022

2 Overview

2.1 Business Problem

Twitter has become an important communication channel in times of emergency. The ubiquitousness of smartphones enables people to announce an emergency they're observing in real-time. Because of this, more agencies are interested in programatically monitoring Twitter (i.e. disaster relief organizations and news agencies).

3 Data Loading and cleaning

In [1]:

```
from distutils.version import LooseVersion
import warnings
import tensorflow as tf

# Check TensorFlow Version
assert LooseVersion(tf.__version__) >= LooseVersion('1.0'), 'Please use TensorF
print('TensorFlow Version: {}'.format(tf.__version__))

# Check for a GPU
if not tf.test.gpu_device_name():
    warnings.warn('No GPU found. Please ensure you have installed TensorFlow cc
else:
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
```

```
TensorFlow Version: 2.7.0
Default GPU Device: /device:GPU:0
```

In [2]:
```
#need to install worlcloud on local pc by:
#conda install -c conda-forge wordcloud=1.6.0
#need to install: pip install transformers
import warnings
warnings.filterwarnings('ignore')
%config Completer.use_jedi = False

import os
import numpy as np
import pandas as pd
# !pip install text_hammer
import text_hammer as th
import seaborn as sns
import matplotlib.pyplot as plt
import re
import en_core_web_sm
import pydot
from wordcloud import WordCloud
from wordcloud import STOPWORDS
from nltk.corpus import stopwords
from collections import defaultdict
#%%time
from tqdm._tqdm_notebook import tqdm_notebook
tqdm_notebook.pandas()
from transformers import AutoTokenizer,TFBertModel
from sklearn.model_selection import train_test_split, KFold

max_len = 36

import tensorflow as tf
tf.config.experimental.list_physical_devices('GPU')

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.initializers import TruncatedNormal
from tensorflow.keras.losses import CategoricalCrossentropy,BinaryCrossentropy
from tensorflow.keras.metrics import CategoricalAccuracy,BinaryAccuracy
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import Input, Dense
```

3.1 Import data

In [3]:
```
train_data = pd.read_csv('train.csv',usecols=['id','text','target'])
test_data = pd.read_csv('test.csv',usecols=['id','text'])
sample_data = pd.read_csv('sample_submission.csv')
```

In [4]:
```
import sys
print(sys.executable)
```

```
C:\Users\xu663\python.exe
```

In [5]:
```
test_data.head()
```

Out[5]:

	id	text
0	0	Just happened a terrible car crash
1	2	Heard about #earthquake is different cities, s...
2	3	there is a forest fire at spot pond, geese are...
3	9	Apocalypse lighting. #Spokane #wildfires
4	11	Typhoon Soudelor kills 28 in China and Taiwan

In [6]:
```
train_data.head()
```

Out[6]:

	id	text	target
0	1	Our Deeds are the Reason of this #earthquake M...	1
1	4	Forest fire near La Ronge Sask. Canada	1
2	5	All residents asked to 'shelter in place' are ...	1
3	6	13,000 people receive #wildfires evacuation or...	1
4	7	Just got sent this photo from Ruby #Alaska as ...	1

In [7]:
```
train_data.shape
```

Out[7]:
```
(7613, 3)
```

In [8]:
```
test_data.shape
```

Out[8]:
```
(3263, 2)
```

In [9]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7613 entries, 0 to 7612
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   id      7613 non-null   int64 
 1   text    7613 non-null   object
 2   target  7613 non-null   int64 
dtypes: int64(2), object(1)
memory usage: 178.6+ KB
```

In [10]:
```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3263 entries, 0 to 3262
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   id      3263 non-null   int64 
 1   text    3263 non-null   object
dtypes: int64(1), object(1)
memory usage: 51.1+ KB
```

In [11]:
```
train_data.isnull().sum()
```

Out[11]:
```
id        0
text      0
target    0
dtype: int64
```

In [12]:
```
test_data.isnull().sum()
```

Out[12]:
```
id      0
text    0
dtype: int64
```

3.2 data cleaning

In [13]:
```
#data cleaning
def text_preprocessing(df,col_name):
    column = col_name
    df[column] = df[column].progress_apply(lambda x:str(x).lower())
#     df[column] = df[column].progress_apply(lambda x: th.cont_exp(x)) #you're
    df[column] = df[column].progress_apply(lambda x: th.remove_emails(x))
    df[column] = df[column].progress_apply(lambda x: th.remove_html_tags(x))
#     df[column] = df[column].progress_apply(lambda x: ps.remove_stopwords(x))
    df[column] = df[column].progress_apply(lambda x: th.remove_special_chars(x)
    df[column] = df[column].progress_apply(lambda x: th.remove_accented_chars(x
#     df[column] = df[column].progress_apply(lambda x: th.make_base(x)) #ran ->
    return(df)
```

In [14]:
```
train_cleaned_data = text_preprocessing(train_data,'text')
```

```
0%|          | 0/7613 [00:00<?, ?it/s]
0%|          | 0/7613 [00:00<?, ?it/s]
0%|          | 0/7613 [00:00<?, ?it/s]
0%|          | 0/7613 [00:00<?, ?it/s]
0%|          | 0/7613 [00:00<?, ?it/s]
```

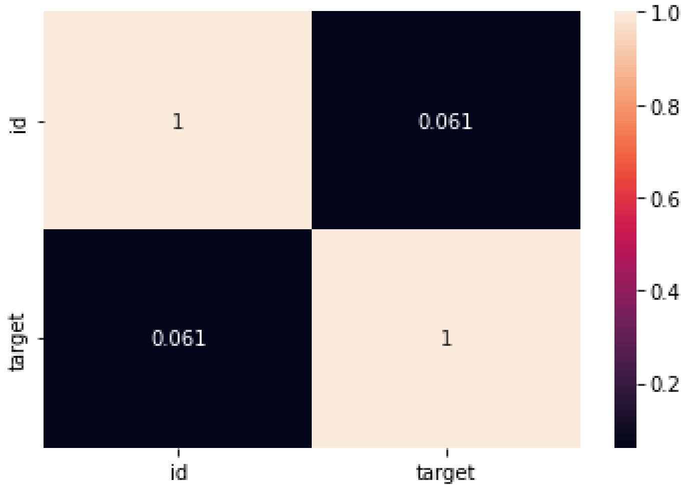
In [15]:
```
train_cleaned_data.head(30)
```

Out[15]:

	id	text	target
0	1	our deeds are the reason of this earthquake ma...	1
1	4	forest fire near la ronge sask canada	1
2	5	all residents asked to shelter in place are be...	1
3	6	13000 people receive wildfires evacuation orde...	1
4	7	just got sent this photo from ruby alaska as s...	1
5	8	rockyfire update california hwy 20 closed in b...	1
6	10	flood disaster heavy rain causes flash floodin...	1
7	13	im on top of the hill and i can see a fire in ...	1
8	14	theres an emergency evacuation happening now i...	1
9	15	im afraid that the tornado is coming to our area	1
10	16	three people died from the heat wave so far	1
11	17	haha south tampa is getting flooded hah wait a...	1
12	18	raining flooding florida tampabay tampa 18 or ...	1
13	19	flood in bago myanmar we arrived bago	1
14	20	damage to school bus on 80 in multi car crash ...	1
15	23	whats up man	0
16	24	i love fruits	0
17	25	summer is lovely	0
18	26	my car is so fast	0
19	28	what a goooooooaaaaaal	0
20	31	this is ridiculous	0
21	32	london is cool	0
22	33	love skiing	0
23	34	what a wonderful day	0
24	36	looooool	0
25	37	no wayi cant eat that shit	0
26	38	was in nyc last week	0
27	39	love my girlfriend	0
28	40	cooool	0
29	41	do you like pasta	0

In [16]:
```
import seaborn as sns
sns.heatmap(train_cleaned_data.corr(), annot = True)
```

Out[16]: <AxesSubplot:>



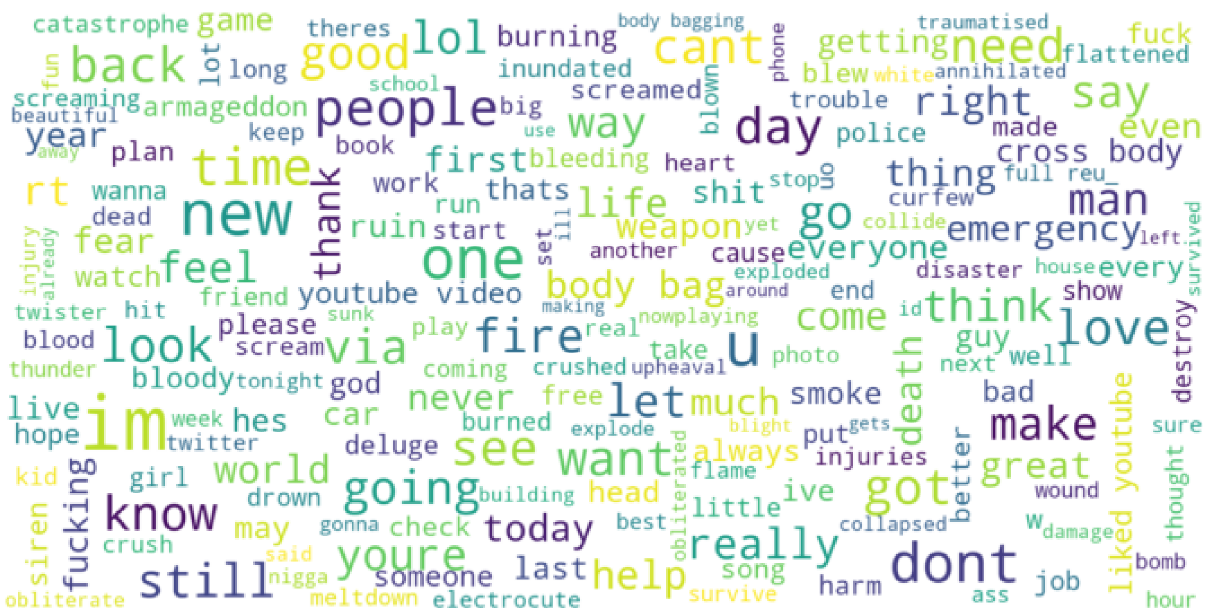
In [17]:
```
stop_words = set(stopwords.words('english'))
train_data['text'] = train_data['text'].apply(lambda x: ' '.join([word for worc
```

3.3 Disaster Tweets wordcloud

In [18]:
```
disaster_tweets = train_data[train_data.target == 1]
disaster_string = []
for t in disaster_tweets.text:
    disaster_string.append(t)
disaster_string = pd.Series(disaster_string).str.cat(sep=' ')
wordcloud = WordCloud(width=1600, height=800,max_font_size=100, background_colo
plt.figure(figsize=(12,10))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

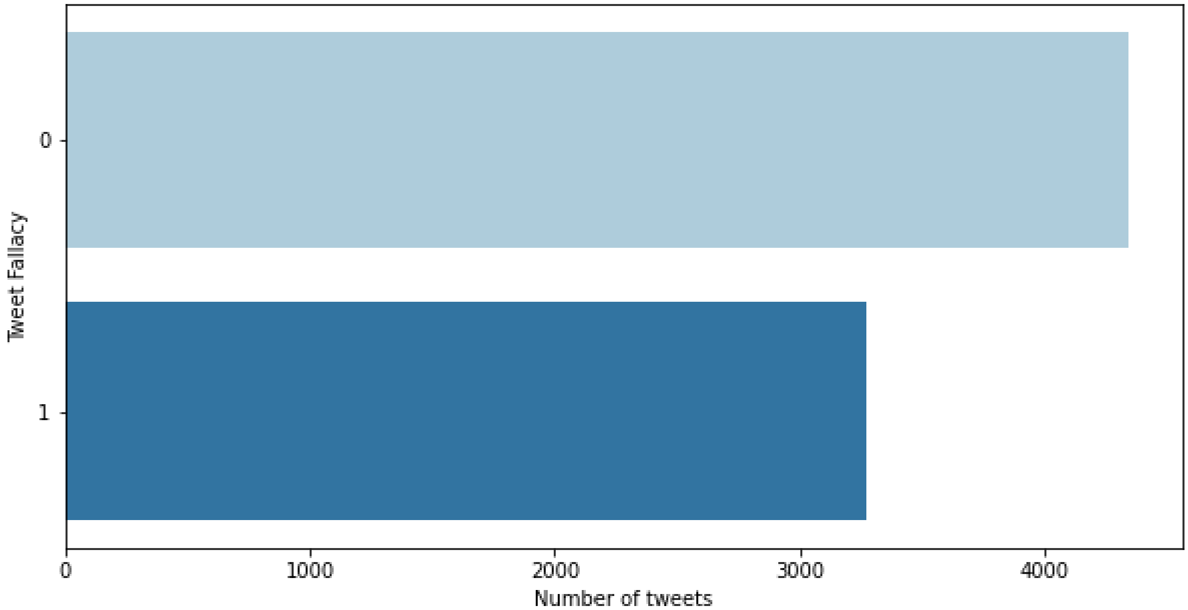
3.4 Positive tweets wordcloud

In [19]:
```
# Positive tweets wordcloud
formal_tweets = train_data[train_data.target == 0]
formal_string = []
for t in formal_tweets.text:
    formal_string.append(t)
formal_string = pd.Series(formal_string).str.cat(sep=' ')
wordcloud = WordCloud(width=1600, height=800,max_font_size=100, background_colc
plt.figure(figsize=(12,10))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

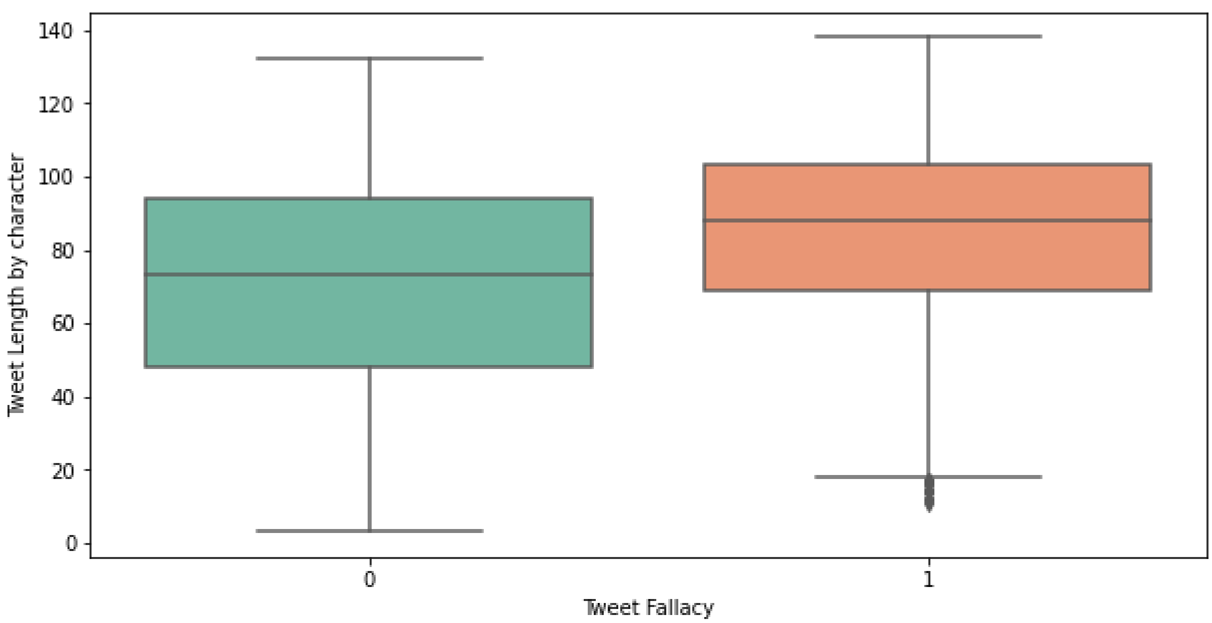


3.5 Visualizing data distribution

In [20]:
```
#Visualizing class distribution
plt.figure(figsize=(10,5))
sns.countplot(y='target',data = train_data,palette="Paired")
plt.ylabel("Tweet Fallacy")
plt.xlabel("Number of tweets")
plt.show()
```

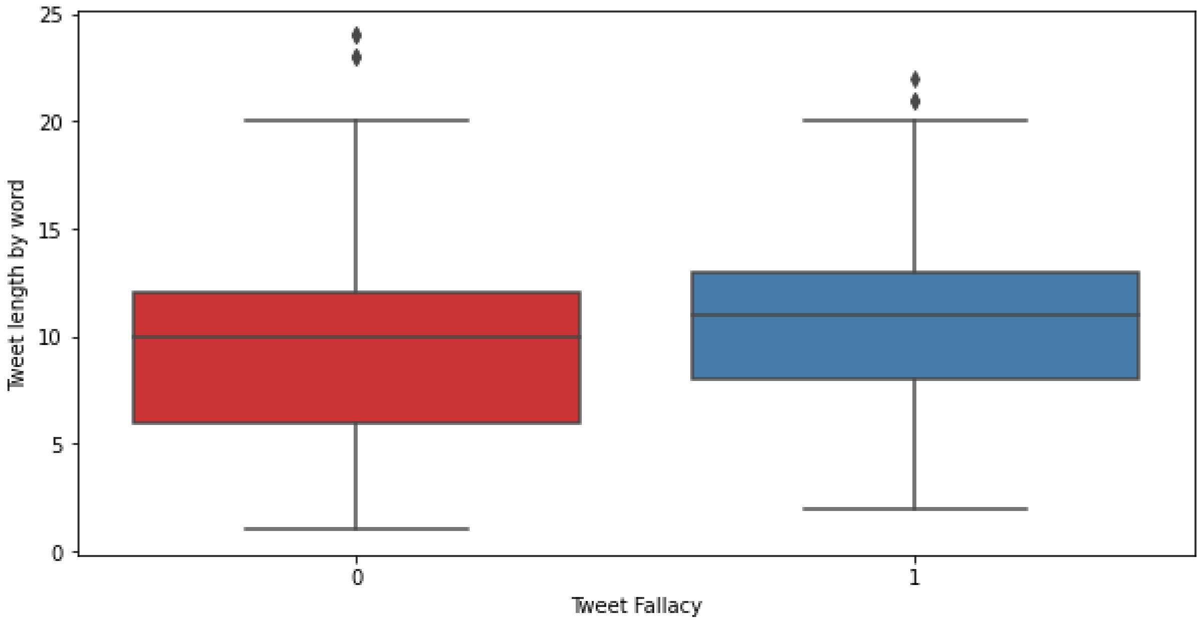


In [21]:
```
#Visualizing tweet length by characaters
plt.figure(figsize=(10,5))
train_sent = train_data['text'].str.len()
sns.boxplot(x="target",y=train_sent,data=train_data,palette="Set2")
plt.xlabel("Tweet Fallacy")
plt.ylabel("Tweet Length by character")
plt.show()
```



In [22]:

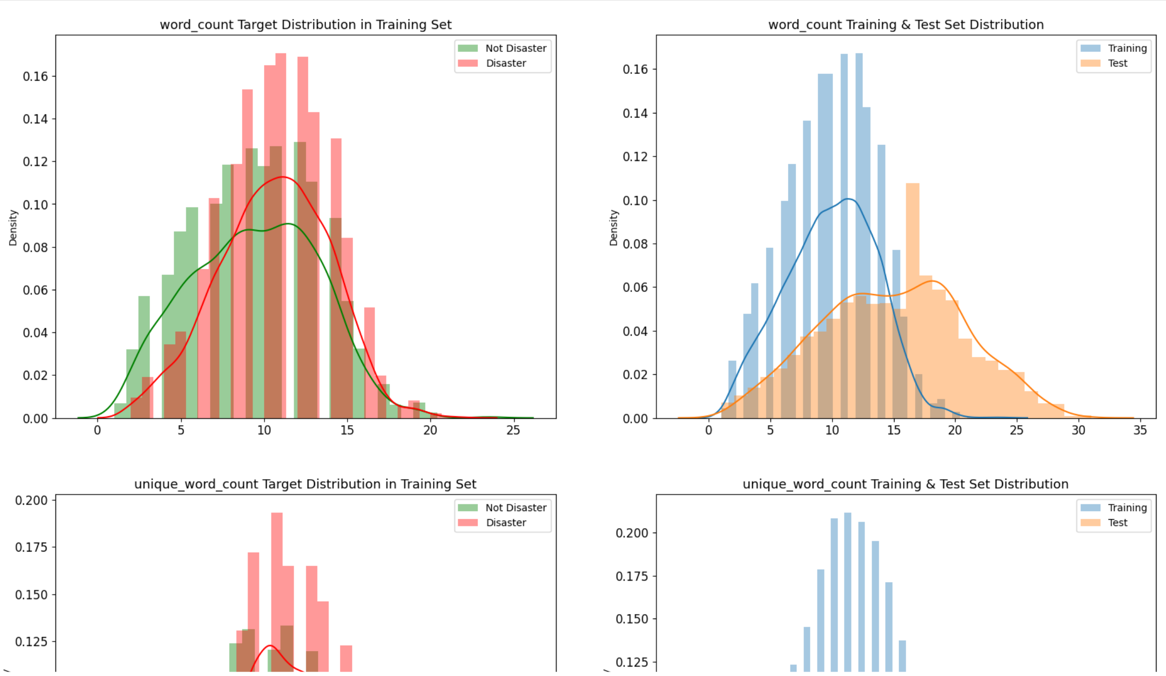
```
#Visualizing tweet length by words
plt.figure(figsize=(10,5))
train_sent = train_data['text'].str.split().map(lambda x : len(x))
sns.boxplot(x="target",y=train_sent,data=train_data,palette="Set1")
plt.xlabel("Tweet Fallacy")
plt.ylabel("Tweet length by word")
plt.show()
```



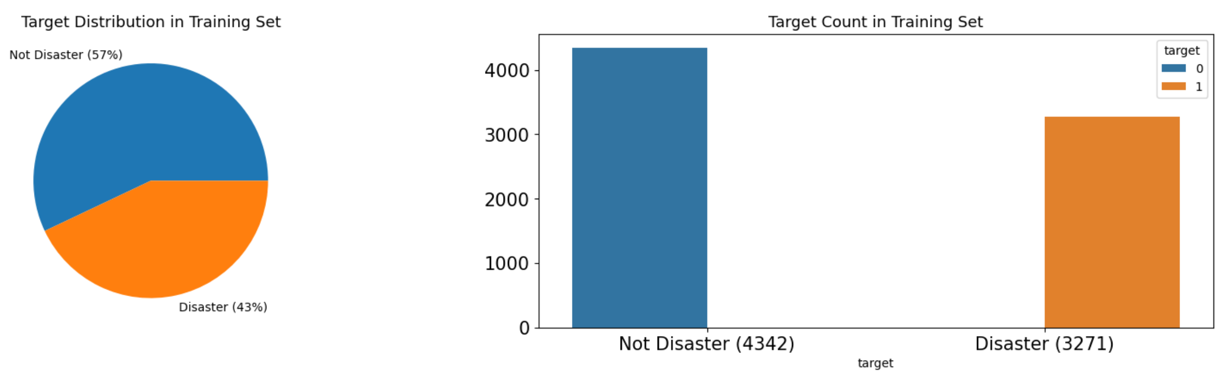
In [23]:

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

# unique_word_count
train_data['unique_word_count'] = train_data['text'].apply(lambda x: len(set(st
test_data['unique_word_count'] = test_data['text'].apply(lambda x: len(set(str(

# stop_word_count
#Stopwords are the English words which does not add much meaning to a sentence.

train_data['stop_word_count'] = train_data['text'].apply(lambda x: len([w for w
test_data['stop_word_count'] = test_data['text'].apply(lambda x: len([w for w i

# url_count
train_data['url_count'] = train_data['text'].apply(lambda x: len([w for w in st
test_data['url_count'] = test_data['text'].apply(lambda x: len([w for w in str(

# mean_word_length
train_data['mean_word_length'] = train_data['text'].apply(lambda x: np.mean([le
test_data['mean_word_length'] = test_data['text'].apply(lambda x: np.mean([len(

# char_count
train_data['char_count'] = train_data['text'].apply(lambda x: len(str(x)))
test_data['char_count'] = test_data['text'].apply(lambda x: len(str(x)))
```

In [24]:
```
METAFEATURES = ['word_count', 'unique_word_count', 'stop_word_count', 'url_cour
                'char_count']
DISASTER_TWEETS = train_data['target'] == 1

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

for i, feature in enumerate(METAFEATURES):
    sns.distplot(train_data.loc[~DISASTER_TWEETS][feature], label='Not Disaster
    sns.distplot(train_data.loc[DISASTER_TWEETS][feature], label='Disaster', ax

    sns.distplot(train_data[feature], label='Training', ax=axes[i][1])
    sns.distplot(test_data[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', font
    axes[i][1].set_title(f'{feature} Training & Test Set Distribution', fontsiz

plt.show()
```



In [25]:
```
fig, axes = plt.subplots(ncols=2, figsize=(17, 4), dpi=100)
plt.tight_layout()

train_data.groupby('target').count()['id'].plot(kind='pie', ax=axes[0], labels=
sns.countplot(x=train_data['target'], hue=train_data['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()
```



4 BERT MODEL

BERT - Bidirectional Encoder Representations from Transformers

LOADING BERT MODEL

In [26]:
```
tokenizer = AutoTokenizer.from_pretrained('bert-large-uncased')
bert = TFBertModel.from_pretrained('bert-large-uncased')
```

```
Some layers from the model checkpoint at bert-large-uncased were not used when
initializing TFBertModel: ['nsp___cls', 'mlm___cls']
- This IS expected if you are initializing TFBertModel from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing TFBertModel from the checkpoint
of a model that you expect to be exactly identical (initializing a BertForSeque
nceClassification model from a BertForSequenceClassification model).
All the layers of TFBertModel were initialized from the model checkpoint at ber
t-large-uncased.
If your task is similar to the task the model of the checkpoint was trained on,
you can already use TFBertModel for predictions without further training.
```

In [27]:
```
tokenizer('onl01-dtsc-pt-062821')
```

Out[27]:
```
{'input_ids': [101, 2006, 2140, 24096, 1011, 26718, 11020, 1011, 13866, 1011, 5
757, 22407, 17465, 102], 'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

In [28]:
```
print("max len of tweets",max([len(x.split()) for x in train_data.text]))
max_length = 36
```

```
max len of tweets 24
```

In [29]:
```
x_train = tokenizer(
    text=train_data.text.tolist(),
    add_special_tokens=True,
    max_length=36,
    truncation=True,
    padding=True,
    return_tensors='tf',
    return_token_type_ids = False,
    return_attention_mask = True,
    verbose = True)
```

In [30]:
```
x_train['input_ids'].shape
```

Out[30]:
```
TensorShape([7613, 36])
```

In [31]:
```
x_train['attention_mask'].shape
```

Out[31]:
```
TensorShape([7613, 36])
```

In [32]:
```
y_train = train_data.target.values
y_train
```

Out[32]:
```
array([1, 1, 1, ..., 1, 1, 1], dtype=int64)
```

In [33]:
```
train_data.target.value_counts()
```

Out[33]:
```
0    4342
1    3271
Name: target, dtype: int64
```

Import data in model

In [34]:
```
input_ids = Input(shape=(max_len,), dtype=tf.int32, name="input_ids")
input_mask = Input(shape=(max_len,), dtype=tf.int32, name="attention_mask")
# embeddings = dbert_model(input_ids,attention_mask = input_mask)[0]


embeddings = bert(input_ids,attention_mask = input_mask)[1] #(0 is the last hid
# out = tf.keras.layers.GlobalMaxPool1D()(embeddings)
out = tf.keras.layers.Dropout(0.1)(embeddings)

out = Dense(128, activation='relu')(out)
out = tf.keras.layers.Dropout(0.1)(out)
out = Dense(32,activation = 'relu')(out)

y = Dense(1,activation = 'sigmoid')(out)

model = tf.keras.Model(inputs=[input_ids, input_mask], outputs=y)
model.layers[2].trainable = True

# for training bert our lr must be so small
```

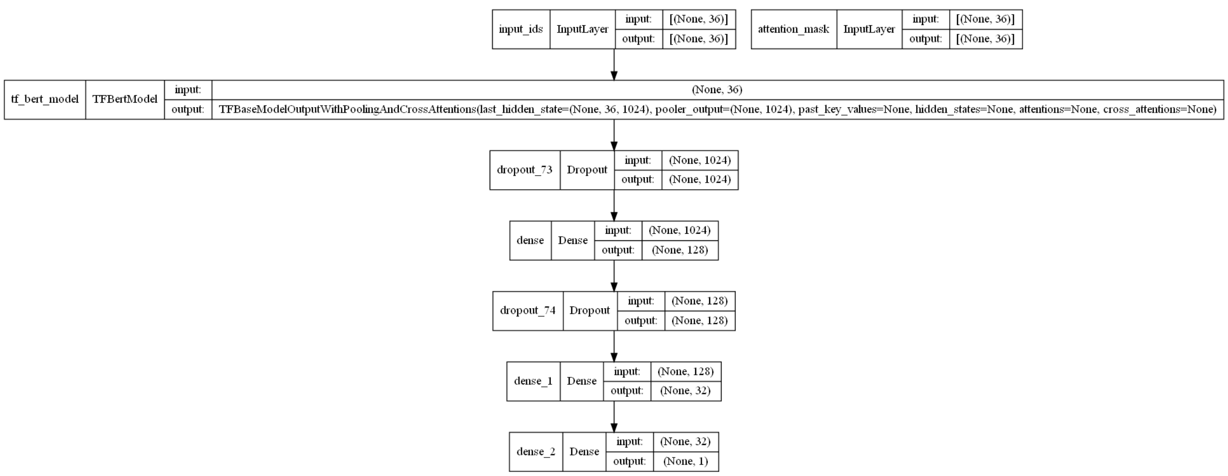
In [35]:
```
model.summary()
```

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
=================================================================================================
 input_ids (InputLayer)         [(None, 36)]         0           []

 attention_mask (InputLayer)    [(None, 36)]         0           []

 tf_bert_model (TFBertModel)    TFBaseModelOutputWi  335141888   ['input_ids[0][0]',
                                thPoolingAndCrossAt               'attention_mask[0][0]']
                                tentions(last_hidde
                                n_state=(None, 36,
                                1024),
                                 pooler_output=(Non
                                e, 1024),
                                 past_key_values=No
                                ne, hidden_states=N
                                one, attentions=Non
                                e, cross_attentions
                                =None)

 dropout_73 (Dropout)           (None, 1024)         0           ['tf_bert_model[0][1]']

 dense (Dense)                  (None, 128)          131200      ['dropout_73[0][0]']

 dropout_74 (Dropout)           (None, 128)          0           ['dense[0][0]']

 dense_1 (Dense)                (None, 32)           4128        ['dropout_74[0][0]']

 dense_2 (Dense)                (None, 1)            33          ['dense_1[0][0]']

=================================================================================================
Total params: 335,277,249
Trainable params: 335,277,249
Non-trainable params: 0
_________________________________________________________________________________________________
```

In [36]:
```
optimizer = Adam(
    learning_rate=6e-06, # this learning rate is for bert model.
    epsilon=1e-08,
    decay=0.01,
    clipnorm=1.0)

# Set loss and metrics
loss = BinaryCrossentropy(from_logits = True)
metric = BinaryAccuracy('accuracy'),
# Compile the model
model.compile(
    optimizer = optimizer,
    loss = loss,
    metrics = metric)
```

In [37]:
```
plot_model(model, show_shapes = True)
```

Out[37]:



4.1 Fit the model

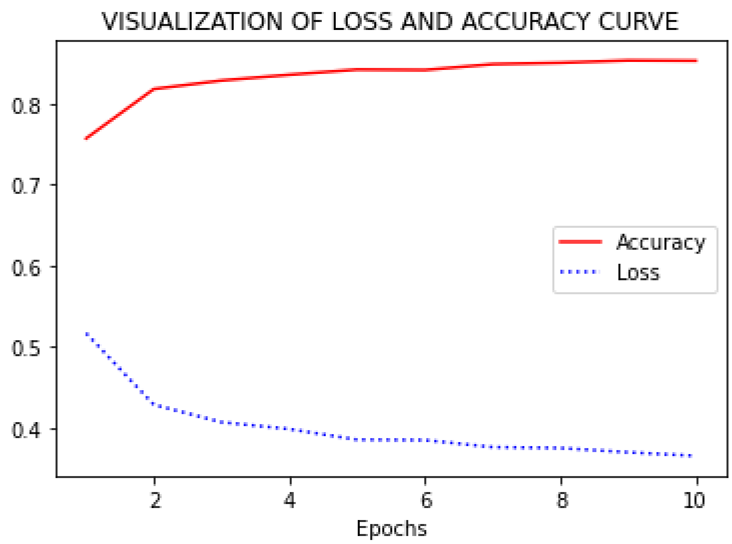
In [38]:
```
# Fit the model
final = model.fit(
    x ={'input_ids':x_train['input_ids'],'attention_mask':x_train['attention_ma
    y = y_train,
#   validation_split = 0.1,
  epochs=10,
#   epochs=10,
    batch_size=10
)
```

```
Epoch 1/10
762/762 [==============================] - 127s 142ms/step - loss: 0.5172 - acc
uracy: 0.7573
Epoch 2/10
762/762 [==============================] - 108s 142ms/step - loss: 0.4288 - acc
uracy: 0.8183
Epoch 3/10
762/762 [==============================] - 108s 142ms/step - loss: 0.4072 - acc
uracy: 0.8288
Epoch 4/10
762/762 [==============================] - 108s 142ms/step - loss: 0.3987 - acc
uracy: 0.8359
Epoch 5/10
762/762 [==============================] - 108s 142ms/step - loss: 0.3856 - acc
uracy: 0.8424
Epoch 6/10
762/762 [==============================] - 108s 142ms/step - loss: 0.3851 - acc
uracy: 0.8418
Epoch 7/10
762/762 [==============================] - 108s 142ms/step - loss: 0.3763 - acc
uracy: 0.8491
Epoch 8/10
762/762 [==============================] - 108s 141ms/step - loss: 0.3753 - acc
uracy: 0.8508
Epoch 9/10
762/762 [==============================] - 108s 142ms/step - loss: 0.3700 - acc
uracy: 0.8537
Epoch 10/10
762/762 [==============================] - 108s 142ms/step - loss: 0.3655 - acc
uracy: 0.8533
```

This is running results showing below: Epoch 1/9 762/762 [==============================] - 127s 139ms/step - loss: 0.5276 - accuracy: 0.7609 Epoch 2/9 762/762 [==============================] - 106s 139ms/step - loss: 0.4376 - accuracy: 0.8223 Epoch 3/9 762/762 [==============================] - 107s 141ms/step - loss: 0.4161 - accuracy: 0.8290 Epoch 4/9 762/762 [==============================] - 107s 141ms/step - loss: 0.4069 - accuracy: 0.8337 Epoch 5/9 762/762 [==============================] - 109s 144ms/step - loss: 0.3987 - accuracy: 0.8373 Epoch 6/9 762/762 [==============================] - 106s 140ms/step - loss: 0.3891 - accuracy: 0.8430 Epoch 7/9 762/762 [==============================] - 107s 140ms/step - loss: 0.3884 - accuracy: 0.8403 Epoch 8/9 762/762 [==============================] - 107s 140ms/step - loss: 0.3898 - accuracy: 0.8463 Epoch 9/9 762/762 [==============================] - 108s 142ms/step - loss: 0.3789 - accuracy: 0.8518

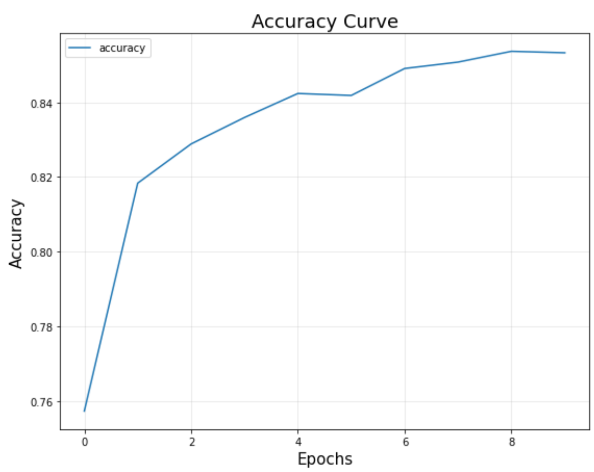
In [39]:
```
def visual_accuracy_and_loss(final):
    acc = final.history['accuracy']
    loss = final.history['loss']
    epochs_plot = np.arange(1, len(loss) + 1)
    plt.clf()
    plt.plot(epochs_plot, acc, 'r', label='Accuracy')
    plt.plot(epochs_plot, loss, 'b:', label='Loss')
    plt.title('VISUALIZATION OF LOSS AND ACCURACY CURVE')
    plt.xlabel('Epochs')
    plt.legend()
    plt.show()
```

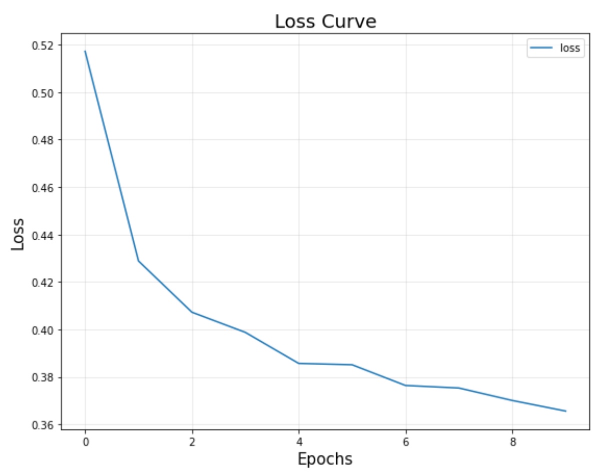
In [40]:
```
visual_accuracy_and_loss(final)
```



4.2 Plot the loss and accuracy curves

In [41]:
```
# Plot the loss and accuracy curves

#Diffining Figure
f = plt.figure(figsize=(20,7))

#Adding Subplot 1 (For Accuracy)
f.add_subplot(121)

plt.plot(final.epoch,final.history['accuracy'],label = "accuracy") # Accuracy c


plt.title("Accuracy Curve",fontsize=18)
plt.xlabel("Epochs",fontsize=15)
plt.ylabel("Accuracy",fontsize=15)
plt.grid(alpha=0.3)
plt.legend()

#Adding Subplot 1 (For Loss)
f.add_subplot(122)

plt.plot(final.epoch,final.history['loss'],label="loss") # Loss curve


plt.title("Loss Curve",fontsize=18)
plt.xlabel("Epochs",fontsize=15)
plt.ylabel("Loss",fontsize=15)
plt.grid(alpha=0.3)
plt.legend()

plt.show()
```





In [42]:
```
test_data
```

Out[42]:

	id	text	word_count	unique_word_count	stop_word_count	url_count	mean_w
0	0	Just happened a terrible car crash	6	6	2	0	
1	2	Heard about #earthquake is different cities, s...	9	9	2	0	
2	3	there is a forest fire at spot pond, geese are...	19	19	10	0	
3	9	Apocalypse lighting. #Spokane #wildfires	4	4	0	0	
4	11	Typhoon Soudelor kills 28 in China and Taiwan	8	8	2	0	
...	...	...	...	...	...	...	
3258	10861	EARTHQUAKE SAFETY LOS ANGELES ÛÒ SAFETY FASTE...	8	7	0	0	
3259	10865	Storm in RI worse than last hurricane. My city...	23	22	7	0	
3260	10868	Green Line derailment in Chicago http://t.co/U...	6	6	1	1	
3261	10874	MEG issues Hazardous Weather Outlook (HWO) htt...	7	7	0	1	
3262	10875	#CityofCalgary has activated its Municipal Eme...	8	8	2	0	

3263 rows × 8 columns

In [43]:
```
x_test = tokenizer(
    text=test_data.text.tolist(),
    add_special_tokens=True,
    max_length=36,
    truncation=True,
    padding=True,
    return_tensors='tf',
    return_token_type_ids = False,
    return_attention_mask = True,
    verbose = True)
```

In [44]:
```
predicted = model.predict({'input_ids':x_test['input_ids'],'attention_mask':x_t
```

In [45]:
```
y_predicted = np.where(predicted>0.5,1,0)
```

In [46]:
```
y_predicted = y_predicted.reshape((1,3263))[0]
```

In [47]:
```
sample_data['id'] = test_data.id
sample_data['target'] = y_predicted
```

In [48]:
```
sample_data.head()
```

Out[48]:

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

In [49]:
```
sample_data.to_csv('submission.csv',index = False)
print(" Successfully completed! ")
```

```
Successfully completed!
```

5 MultinomialNB Model

The multinomial Naive Bayes classifier

In [50]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, StratifiedKFold,GridSearch
X_train, X_test, y_train, y_test  = train_test_split(train_cleaned_data.text,
                                                       train_cleaned_data.ta
                                                       stratify=train_cleane
                                                       random_state = 1314)
```

In [51]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(analyzer='word', stop_words='english', token_pattern=r'
train_tfidf = tfidf.fit_transform(X_train)
test_tfidf = tfidf.transform(X_test)
test = tfidf.transform(test_data.text)
```

In [52]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import f1_score
```

In [53]:
```
clf = MultinomialNB(alpha=1)
scores = cross_val_score(clf, train_tfidf, y_train, cv=5, scoring="f1")
#scores
```

5.1 Train the Model

In [59]:
```
clf.fit(train_tfidf, y_train)
```

Out[59]: MultinomialNB(alpha=1)

5.2 Predictions and Evaluation

In [60]:
```
f1_score(y_test, clf.predict(test_tfidf))
```

Out[60]: 0.7234525837592276

In [55]:
```
clf.predict(test_tfidf)
```

Out[55]: array([1, 0, 0, ..., 0, 0, 0], dtype=int64)

6 Conclusion

After analysis, my team believe this BERT is suitable for NLP on Twitter. And accuracy can be improved to 0.85 through iteration.

We would suggest:

Large amount of dataset for training

Better GPU learning environment

Adding more features like time, location, and etc.

7 Future work

1. With more time, I would like to dig into relationship between NLP models and make a comparison.
2. For other media content, we can try to fit the model and find results as well.
3. I want to see if we can add image processing to capture the emergency.

```
In [ ]:
```