1. Title : Natural Language Processing with Disaster Tweets

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).

But, it's not always clear whether a person's words are actually announcing a disaster.

The author explicitly uses the word "ABLAZE" but means it metaphorically. This is clear to a human right away, especially with the visual aid. But it's less clear to a machine.

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

2. Importing required Libraries

Using TensorFlow backend.

In [1]:

```
import re
import numpy as np
import pandas as pd
from tqdm import tqdm
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.corpus import stopwords

import warnings
warnings.filterwarnings('ignore')

import keras
from keras.initializers import Constant
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras import layers, Sequential
from tensorflow.keras import optimizers
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.layers import LSTM
```

3. Loading the data

In [2]:

```
train_data = pd.read_csv('input/train.csv', dtype={'id': np.int16, 'target': np.int16
test_data = pd.read_csv('input/test.csv', dtype={'id': np.int16})
```

4. EDA

1) Check missing values

In [3]:

```
train_data.isnull().sum()
```

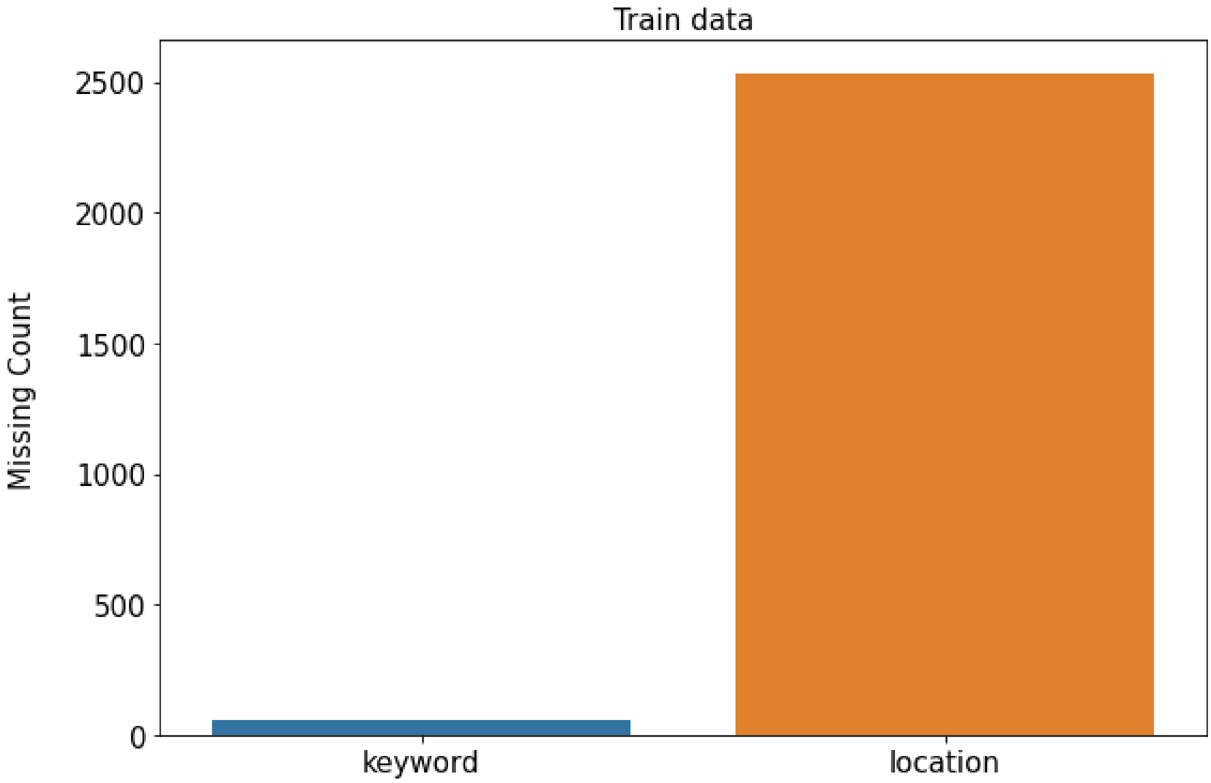
Out[3]:

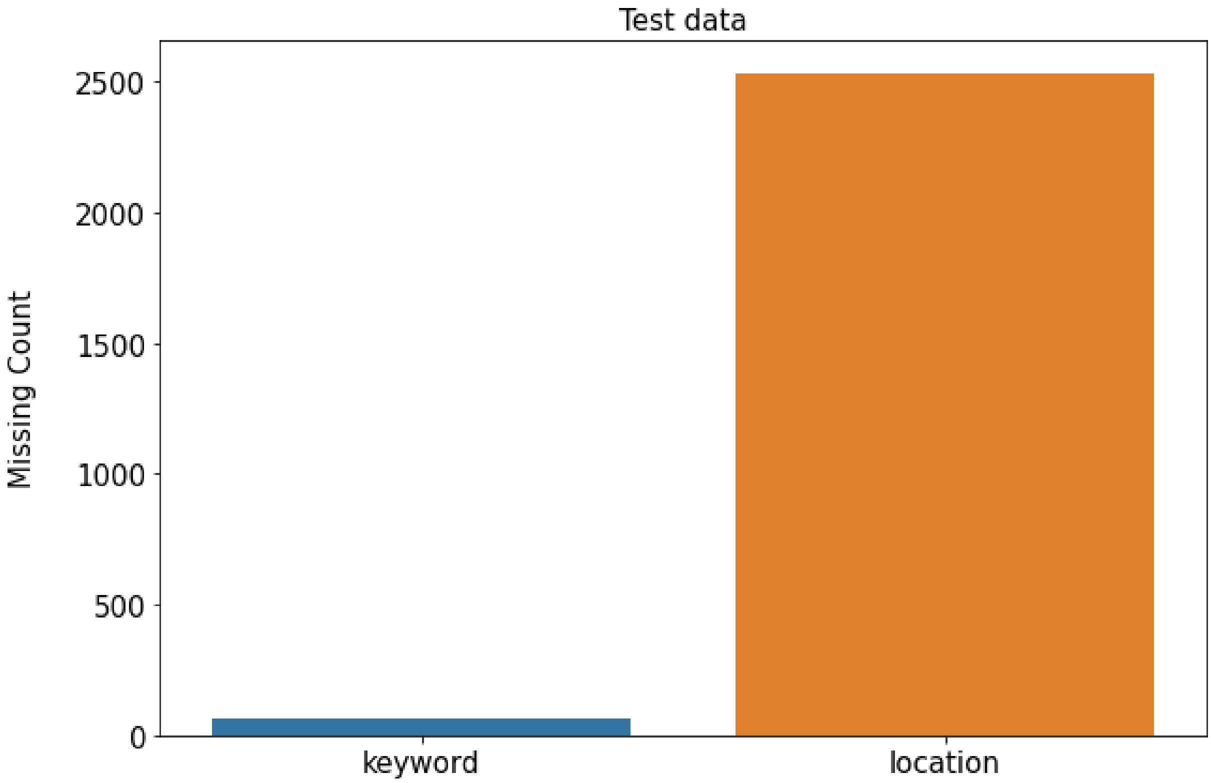
```
id             0
keyword       61
location    2533
text           0
target         0
dtype: int64
```

Missing values exist in the keyword, location variables.

In [4]:

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

fig, axes = plt.subplots(2,figsize=(10, 15))

sns.barplot(x=train_data[miss_cols].isnull().sum().index, y=train_data[miss_cols].isr
sns.barplot(x=train_data[miss_cols].isnull().sum().index, y=train_data[miss_cols].isr

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

axes[0].set_title('Train data', fontsize=15)
axes[1].set_title('Test data', fontsize=15)

plt.show()
```





LOCATION variable has many missing values.

In [5]:
```
train_data.groupby('target').count()['id']
```

Out[5]:
```
target
0    4342
1    3271
Name: id, dtype: int64
```

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

In [6]:
```
train_x = train_data['text'].copy()
```

```
train_y = train_data['target'].copy()
```

0 is more than 1, but it is not much different.

2) Data Cleaning

Missing values must be processed before data analysis can be performed.

In [7]:

```
stop = stopwords.words('english')
def clean(text):

    text = re.sub(r'http₩S+', ' ', text)

    text = re.sub(r'<.*?>', ' ', text)

    text = re.sub(r'#₩w+', ' ', text)

    text = re.sub(r'@₩w+', ' ', text)

    text = re.sub(r'₩d+', ' ', text)

    text = text.split()

    text = ' '.join([word for word in text if word not in stop])

    return text
```

Removing Stop words

In [8]:

```
train_x_cleaned = train_x.apply(clean)
train_x_cleaned.head()
```

Out[8]:

```
0                Our Deeds Reason May ALLAH Forgive us
1                 Forest fire near La Ronge Sask. Canada
2      All residents asked 'shelter place' notified o...
3          , people receive evacuation orders California
4           Just got sent photo Ruby smoke pours school
Name: text, dtype: object
```

Max Length

In [9]:

```
max_len = max(train_x_cleaned.apply(len))
print('max length: {}'.format(max_len))
```

```
max length: 141
```

3) Tokenize

the text must be vectorized by generating a sequence of specified lengths for each tweet in the dataset. Using the Keras Tokenizer

In [10]:

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_x_cleaned)
vocab_size = len(tokenizer.word_index) + 1
x = tokenizer.texts_to_sequences(train_x_cleaned)
x = pad_sequences(x, max_len, padding='post')
y = train_y
print('train_x_clean:', train_x_cleaned[4])
print('*'*50)
print('x:',x[5])
print('vocabulary size:{}'.format(vocab_size))
```

```
train_x_clean: Just got sent photo Ruby smoke pours school
**************************************************
x: [ 318   45 1367  772 6083  478 1112  319    6    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0]
vocabulary size:13947
```

In [11]:
```
x.shape
```

Out[11]:
```
(7613, 141)
```

In [12]:
```
y.shape
```

Out[12]:
```
(7613,)
```

5. Create Models

model 1

GRU implementation with basic embedding layer

In [13]:
```
epoch_size =10
batch_size = 32
embedding_dim = 16
optimizer = optimizers.Adam(lr=3e-4)

model = Sequential([
    layers.Embedding(vocab_size, embedding_dim, input_length=max_len),
    layers.Bidirectional(layers.GRU(256, return_sequences=True)),
    layers.GlobalMaxPool1D(),
#     layers.Dense(128, activation='relu'),
#     layers.Dropout(0.4),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.4),
    layers.Dense(2, activation='sigmoid')
])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 141, 16)           223152
_________________________________________________________________
bidirectional (Bidirectional (None, 141, 512)          420864
_________________________________________________________________
global_max_pooling1d (Global (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 64)                32832
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 130
```

```
=================================================================
Total params: 676,978
Trainable params: 676,978
Non-trainable params: 0
_________________________________________________________________
```

training the model

In [14]:

```
model.compile(loss='sparse_categorical_crossentropy', optimizer = 'adam', metrics=['a
history = model.fit(x, y, epochs=epoch_size, validation_split=0.1)
```

```
Epoch 1/10
215/215 [==============================] - 47s 208ms/step - loss: 0.5872 - accuracy:
0.6926 - val_loss: 0.5210 - val_accuracy: 0.7572
Epoch 2/10
215/215 [==============================] - 46s 214ms/step - loss: 0.3567 - accuracy:
0.8581 - val_loss: 0.5055 - val_accuracy: 0.7598
Epoch 3/10
215/215 [==============================] - 52s 243ms/step - loss: 0.2310 - accuracy:
0.9149 - val_loss: 0.5791 - val_accuracy: 0.7480
Epoch 4/10
215/215 [==============================] - 60s 278ms/step - loss: 0.1572 - accuracy:
0.9466 - val_loss: 0.6171 - val_accuracy: 0.7441
Epoch 5/10
215/215 [==============================] - 61s 285ms/step - loss: 0.1179 - accuracy:
0.9597 - val_loss: 0.7881 - val_accuracy: 0.7507
Epoch 6/10
215/215 [==============================] - 62s 288ms/step - loss: 0.0967 - accuracy:
0.9666 - val_loss: 0.8090 - val_accuracy: 0.7231
Epoch 7/10
215/215 [==============================] - 62s 287ms/step - loss: 0.0826 - accuracy:
0.9695 - val_loss: 0.8915 - val_accuracy: 0.7257
Epoch 8/10
215/215 [==============================] - 62s 289ms/step - loss: 0.0719 - accuracy:
0.9712 - val_loss: 0.9324 - val_accuracy: 0.7257
Epoch 9/10
215/215 [==============================] - 63s 293ms/step - loss: 0.0655 - accuracy:
0.9736 - val_loss: 1.0139 - val_accuracy: 0.7415
Epoch 10/10
215/215 [==============================] - 64s 299ms/step - loss: 0.0594 - accuracy:
0.9747 - val_loss: 1.3628 - val_accuracy: 0.7323
```

In [15]:

```
test_x = test_data['text'].copy()
test_x = test_x.apply(clean)
test_x = tokenizer.texts_to_sequences(test_x)
test_x = pad_sequences(test_x, max_len, padding='post')
```

prediction

In [16]:

```
test_pred = np.argmax(model.predict(test_x), axis=1)
print(test_pred)
```

```
[1 0 1 ... 1 1 1]
```

Check the loss ans accuracy

In [17]:

```
history1 = history.history

trg_loss = history1['loss']
val_loss = history1['val_loss']

trg_acc = history1['accuracy']
val_acc = history1['val_accuracy']
```

```
epochs = range(1, len(trg_acc) + 1)

# plot losses and accuracies for training and validation
fig = plt.figure(figsize=(12,6))
ax = fig.add_subplot(1, 2, 1)
plt.plot(epochs, trg_loss, marker='o', label='Training Loss')
plt.plot(epochs, val_loss, marker='x', label='Validation Loss')
plt.title("Training / Validation Loss")
ax.set_ylabel("Loss")
ax.set_xlabel("Epochs")
plt.legend(loc='best')

ax = fig.add_subplot(1, 2, 2)
plt.plot(epochs, trg_acc, marker='o', label='Training Accuracy')
plt.plot(epochs, val_acc, marker='^', label='Validation Accuracy')
plt.title("Training / Validation Accuracy")
ax.set_ylabel("Accuracy")
ax.set_xlabel("Epochs")
plt.legend(loc='best')
plt.show()
```

In train data, it is a desirable state in which the loss is small and the accuracy value is high. However, in validation data, it is not convergent and the value is not good.

model 2

Add Dropout layer

In [18]:

```
epoch_size =10
batch_size = 32
embedding_dim = 16
optimizer = optimizers.Adam(lr=3e-4)

model2 = Sequential([
    layers.Embedding(vocab_size, embedding_dim, input_length=max_len),
    layers.Bidirectional(layers.GRU(256, return_sequences=True)),
    layers.GlobalMaxPool1D(),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.4),
    layers.Dense(2, activation='sigmoid')
])
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 141, 16)           223152
_________________________________________________________________
bidirectional_1 (Bidirection (None, 141, 512)          420864
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                32832
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 130
=================================================================
Total params: 676,978
Trainable params: 676,978
Non-trainable params: 0
_________________________________________________________________
```

In [19]:

```
model2.compile(loss='sparse_categorical_crossentropy', optimizer = 'adam', metrics=['
history = model2.fit(x, y, epochs=epoch_size, validation_split=0.1)
```

```
Epoch 1/10
215/215 [==============================] - 64s 289ms/step - loss: 0.5809 - accuracy:
0.6887 - val_loss: 0.4593 - val_accuracy: 0.7874
Epoch 2/10
215/215 [==============================] - 64s 299ms/step - loss: 0.3520 - accuracy:
0.8622 - val_loss: 0.4848 - val_accuracy: 0.7979
Epoch 3/10
215/215 [==============================] - 65s 303ms/step - loss: 0.2303 - accuracy:
0.9174 - val_loss: 0.5995 - val_accuracy: 0.7572
Epoch 4/10
215/215 [==============================] - 64s 298ms/step - loss: 0.1565 - accuracy:
0.9470 - val_loss: 0.6014 - val_accuracy: 0.7887
Epoch 5/10
215/215 [==============================] - 64s 298ms/step - loss: 0.1189 - accuracy:
0.9599 - val_loss: 0.6471 - val_accuracy: 0.7730
Epoch 6/10
215/215 [==============================] - 65s 302ms/step - loss: 0.0921 - accuracy:
0.9677 - val_loss: 0.7340 - val_accuracy: 0.7480
Epoch 7/10
215/215 [==============================] - 65s 303ms/step - loss: 0.0860 - accuracy:
0.9704 - val_loss: 0.8525 - val_accuracy: 0.7546
Epoch 8/10
215/215 [==============================] - 65s 302ms/step - loss: 0.0731 - accuracy:
0.9714 - val_loss: 0.9728 - val_accuracy: 0.7533
Epoch 9/10
215/215 [==============================] - 65s 303ms/step - loss: 0.0601 - accuracy:
0.9752 - val_loss: 1.0421 - val_accuracy: 0.7572
Epoch 10/10
215/215 [==============================] - 65s 301ms/step - loss: 0.0540 - accuracy:
0.9759 - val_loss: 0.9407 - val_accuracy: 0.7559
```

In [20]:

```
test_pred = np.argmax(model2.predict(test_x), axis=1)
print(test_pred)
```

```
[1 0 1 ... 1 1 1]
```

In [21]:

```
history2 = history.history

trg_loss = history2['loss']
val_loss = history2['val_loss']
```

```
trg_acc = history2['accuracy']
val_acc = history2['val_accuracy']

epochs = range(1, len(trg_acc) + 1)

# plot losses and accuracies for training and validation
fig = plt.figure(figsize=(12,6))
ax = fig.add_subplot(1, 2, 1)
plt.plot(epochs, trg_loss, marker='o', label='Training Loss')
plt.plot(epochs, val_loss, marker='x', label='Validation Loss')
plt.title("Training / Validation Loss")
ax.set_ylabel("Loss")
ax.set_xlabel("Epochs")
plt.legend(loc='best')

ax = fig.add_subplot(1, 2, 2)
plt.plot(epochs, trg_acc, marker='o', label='Training Accuracy')
plt.plot(epochs, val_acc, marker='^', label='Validation Accuracy')
plt.title("Training / Validation Accuracy")
ax.set_ylabel("Accuracy")
ax.set_xlabel("Epochs")
plt.legend(loc='best')
plt.show()
```

It is improved over model 1, but it is still not convergent and has a good value in validation data.

model 3

LSTM implementation with basic embedding layer

In [22]:

```
epoch_size =10
batch_size = 32
embedding_dim = 16
optimizer = optimizers.Adam(lr=3e-4)

model3 = Sequential([
    layers.Embedding(vocab_size, embedding_dim, input_length=max_len, trainable=False
    layers.SpatialDropout1D(0.2),

    layers.Bidirectional(layers.LSTM(64, recurrent_dropout=0.5, dropout=0.5, return_s
    layers.Bidirectional(layers.LSTM(64, recurrent_dropout=0.5, dropout=0.5)),
```

```
    layers.Dense(64, activation='relu'),
    layers.Dense(2, activation='sigmoid')
])
model3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 141, 16)           223152
_________________________________________________________________
spatial_dropout1d (SpatialDr (None, 141, 16)           0
_________________________________________________________________
bidirectional_2 (Bidirection (None, 141, 128)          41472
_________________________________________________________________
bidirectional_3 (Bidirection (None, 128)               98816
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 130
=================================================================
Total params: 371,826
Trainable params: 148,674
Non-trainable params: 223,152
_________________________________________________________________
```

In [23]:

```
model3.compile(loss='sparse_categorical_crossentropy', optimizer = optimizer, metrics
history = model3.fit(x, y, epochs=epoch_size, validation_split=0.1)
```

```
Epoch 1/10
215/215 [==============================] - 60s 250ms/step - loss: 0.6830 - accuracy:
0.5736 - val_loss: 0.6955 - val_accuracy: 0.5341
Epoch 2/10
215/215 [==============================] - 54s 252ms/step - loss: 0.6789 - accuracy:
0.5744 - val_loss: 0.6871 - val_accuracy: 0.5341
Epoch 3/10
215/215 [==============================] - 54s 252ms/step - loss: 0.6768 - accuracy:
0.5709 - val_loss: 0.6813 - val_accuracy: 0.5381
Epoch 4/10
215/215 [==============================] - 54s 253ms/step - loss: 0.6747 - accuracy:
0.5771 - val_loss: 0.6806 - val_accuracy: 0.5315
Epoch 5/10
215/215 [==============================] - 54s 253ms/step - loss: 0.6734 - accuracy:
0.5809 - val_loss: 0.6791 - val_accuracy: 0.5328
Epoch 6/10
215/215 [==============================] - 55s 255ms/step - loss: 0.6722 - accuracy:
0.5780 - val_loss: 0.6776 - val_accuracy: 0.5459
Epoch 7/10
215/215 [==============================] - 55s 256ms/step - loss: 0.6721 - accuracy:
0.5754 - val_loss: 0.6750 - val_accuracy: 0.5538
Epoch 8/10
215/215 [==============================] - 55s 258ms/step - loss: 0.6698 - accuracy:
0.5849 - val_loss: 0.6762 - val_accuracy: 0.5486
Epoch 9/10
215/215 [==============================] - 56s 260ms/step - loss: 0.6692 - accuracy:
0.5839 - val_loss: 0.6830 - val_accuracy: 0.5302
Epoch 10/10
215/215 [==============================] - 57s 265ms/step - loss: 0.6692 - accuracy:
0.5812 - val_loss: 0.6751 - val_accuracy: 0.5525
```

In [24]:

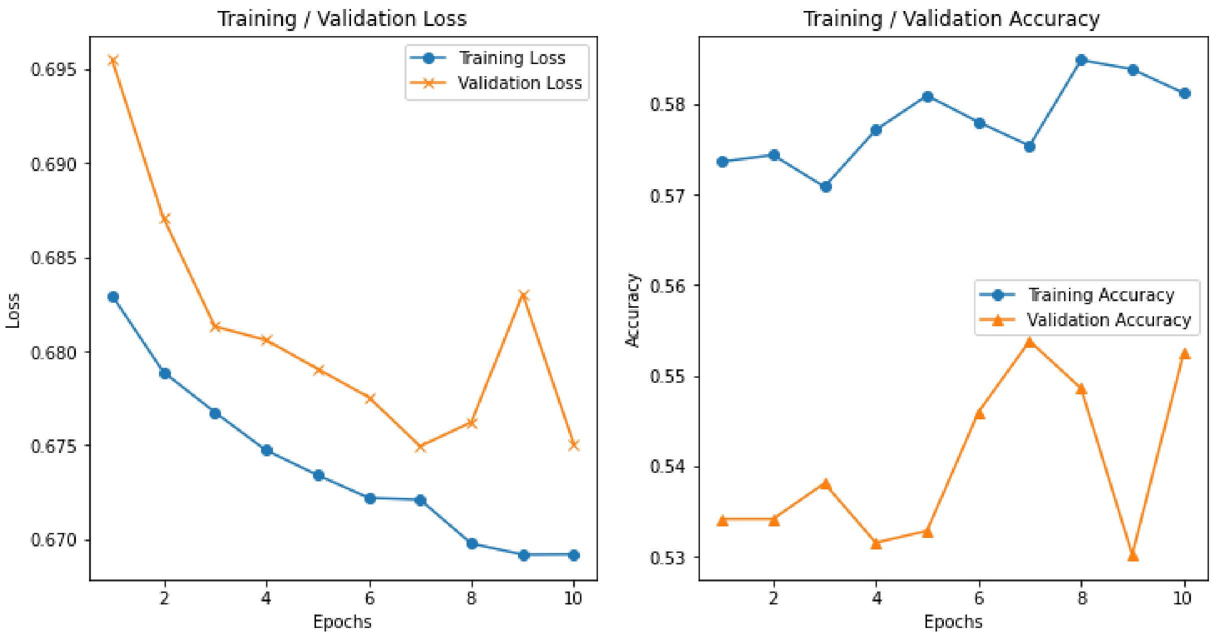
```
test_pred = np.argmax(model3.predict(test_x), axis=1)
print(test_pred)
```

```
[0 0 0 ... 0 0 0]
```

In [25]:

```
history3 = history.history
```

```
trg_loss = history3['loss']
val_loss = history3['val_loss']

trg_acc = history3['accuracy']
val_acc = history3['val_accuracy']

epochs = range(1, len(trg_acc) + 1)

# plot losses and accuracies for training and validation
fig = plt.figure(figsize=(12,6))
ax = fig.add_subplot(1, 2, 1)
plt.plot(epochs, trg_loss, marker='o', label='Training Loss')
plt.plot(epochs, val_loss, marker='x', label='Validation Loss')
plt.title("Training / Validation Loss")
ax.set_ylabel("Loss")
ax.set_xlabel("Epochs")
plt.legend(loc='best')

ax = fig.add_subplot(1, 2, 2)
plt.plot(epochs, trg_acc, marker='o', label='Training Accuracy')
plt.plot(epochs, val_acc, marker='^', label='Validation Accuracy')
plt.title("Training / Validation Accuracy")
ax.set_ylabel("Accuracy")
ax.set_xlabel("Epochs")
plt.legend(loc='best')
plt.show()
```



It is improved over model 2, but it is still not convergent and has a good value in validation data.

model 4

GRU implementation with basic embedding layer and 3 Dense layers

In [26]:

```
epoch_size =10
batch_size = 32
embedding_dim = 16
optimizer = optimizers.Adam(lr=3e-4)

model4 = Sequential([
    layers.Embedding(vocab_size, embedding_dim, input_length=max_len),
    layers.Bidirectional(layers.GRU(256, return_sequences=True)),
    layers.GlobalMaxPool1D(),
```

```
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.4),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.4),
    layers.Dense(2, activation='sigmoid')
])
model4.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 141, 16)           223152
_________________________________________________________________
bidirectional_4 (Bidirection (None, 141, 512)          420864
_________________________________________________________________
global_max_pooling1d_2 (Glob (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               65664
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_7 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_8 (Dense)              (None, 2)                 130
=================================================================
Total params: 718,066
Trainable params: 718,066
Non-trainable params: 0
_________________________________________________________________
```

In [27]:

```
model4.compile(loss='sparse_categorical_crossentropy', optimizer = optimizer, metrics
history = model4.fit(x, y, epochs=epoch_size, validation_split=0.2)
```

```
Epoch 1/10
191/191 [==============================] - 108s 558ms/step - loss: 0.6758 - accuracy:
0.5796 - val_loss: 0.6510 - val_accuracy: 0.5542
Epoch 2/10
191/191 [==============================] - 108s 566ms/step - loss: 0.4853 - accuracy:
0.7851 - val_loss: 0.5271 - val_accuracy: 0.7485
Epoch 3/10
191/191 [==============================] - 107s 563ms/step - loss: 0.3294 - accuracy:
0.8765 - val_loss: 0.5050 - val_accuracy: 0.7551
Epoch 4/10
191/191 [==============================] - 107s 563ms/step - loss: 0.2313 - accuracy:
0.9243 - val_loss: 0.5755 - val_accuracy: 0.7571
Epoch 5/10
191/191 [==============================] - 107s 560ms/step - loss: 0.1709 - accuracy:
0.9425 - val_loss: 0.6335 - val_accuracy: 0.7387
Epoch 6/10
191/191 [==============================] - 107s 559ms/step - loss: 0.1401 - accuracy:
0.9548 - val_loss: 0.7627 - val_accuracy: 0.7249
Epoch 7/10
191/191 [==============================] - 107s 559ms/step - loss: 0.8050 - accuracy:
0.7066 - val_loss: 0.6906 - val_accuracy: 0.5345
Epoch 8/10
191/191 [==============================] - 108s 565ms/step - loss: 0.6802 - accuracy:
0.5757 - val_loss: 0.6886 - val_accuracy: 0.5345
Epoch 9/10
191/191 [==============================] - 108s 566ms/step - loss: 0.6437 - accuracy:
0.6115 - val_loss: 0.6392 - val_accuracy: 0.6658
Epoch 10/10
191/191 [==============================] - 107s 559ms/step - loss: 0.3609 - accuracy:
0.8578 - val_loss: 0.7054 - val_accuracy: 0.6875
```

In [28]:

```
test_pred = np.argmax(model2.predict(test_x), axis=1)
print(test_pred)
```

```
[1 0 1 ... 1 1 1]
```

In [29]:

```
history4 = history.history

trg_loss = history4['loss']
val_loss = history4['val_loss']

trg_acc = history4['accuracy']
val_acc = history4['val_accuracy']

epochs = range(1, len(trg_acc) + 1)

# plot losses and accuracies for training and validation
fig = plt.figure(figsize=(12,6))
ax = fig.add_subplot(1, 2, 1)
plt.plot(epochs, trg_loss, marker='o', label='Training Loss')
plt.plot(epochs, val_loss, marker='x', label='Validation Loss')
plt.title("Training / Validation Loss")
ax.set_ylabel("Loss")
ax.set_xlabel("Epochs")
plt.legend(loc='best')

ax = fig.add_subplot(1, 2, 2)
plt.plot(epochs, trg_acc, marker='o', label='Training Accuracy')
plt.plot(epochs, val_acc, marker='^', label='Validation Accuracy')
plt.title("Training / Validation Accuracy")
ax.set_ylabel("Accuracy")
ax.set_xlabel("Epochs")
plt.legend(loc='best')
plt.show()
```

The loss value is high and the accuracy is significantly different from the train result.

model 5

GloVe embedded LSTM, RNN and add BatchNormalization

In [30]:

```
embeddings_dictionary = dict()
embedding_dim = 100
```

```python
glove_file = open('glove.6B/glove.6B.100d.txt', encoding='UTF8')
for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = np.asarray(records[1:], dtype='float32')
    embeddings_dictionary [word] = vector_dimensions
glove_file.close()
```

In [31]:

```python
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, index in tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector
```

In [32]:

```python
epoch_size =10
batch_size = 32

model5 = Sequential([
    layers.Embedding(input_dim=embedding_matrix.shape[0],
                     output_dim=embedding_matrix.shape[1],
                     weights = [embedding_matrix]
                     ),
    layers.Bidirectional(LSTM(64, return_sequences = True, recurrent_dropout=0.2)),
    layers.GlobalMaxPool1D(),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])
model5.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, None, 100)         1394700
_________________________________________________________________
bidirectional_5 (Bidirection (None, None, 128)         84480
_________________________________________________________________
global_max_pooling1d_3 (Glob (None, 128)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 128)               512
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 65
=================================================================
Total params: 1,504,525
Trainable params: 1,504,269
Non-trainable params: 256
_________________________________________________________________
```

In [33]:
```
model5.compile(loss='binary_crossentropy', optimizer = RMSprop(learning_rate=0.0001),
history5 = model5.fit(x, y, epochs=epoch_size, validation_split=0.2)
```

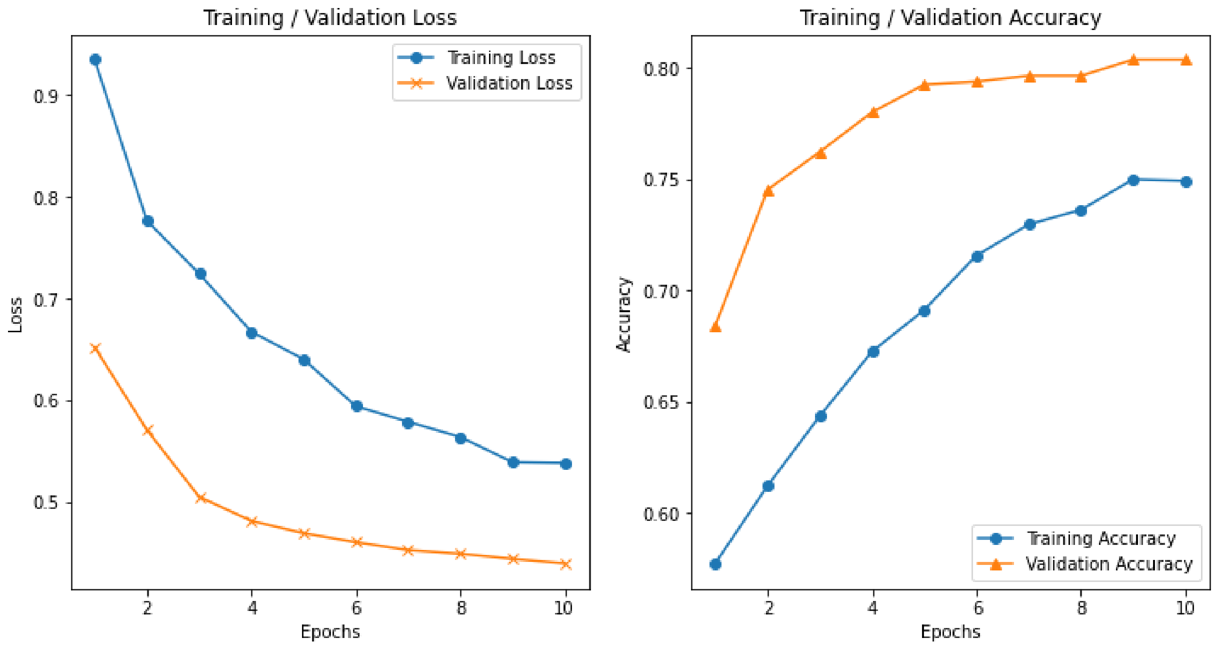
```
Epoch 1/10
191/191 [==============================] - 23s 105ms/step - loss: 0.9342 - accuracy:
0.5775 - val_loss: 0.6520 - val_accuracy: 0.6842
Epoch 2/10
191/191 [==============================] - 21s 108ms/step - loss: 0.7763 - accuracy:
0.6123 - val_loss: 0.5710 - val_accuracy: 0.7452
Epoch 3/10
191/191 [==============================] - 20s 107ms/step - loss: 0.7242 - accuracy:
0.6437 - val_loss: 0.5052 - val_accuracy: 0.7623
Epoch 4/10
191/191 [==============================] - 21s 107ms/step - loss: 0.6672 - accuracy:
0.6726 - val_loss: 0.4816 - val_accuracy: 0.7800
Epoch 5/10
191/191 [==============================] - 20s 107ms/step - loss: 0.6402 - accuracy:
0.6913 - val_loss: 0.4695 - val_accuracy: 0.7925
Epoch 6/10
191/191 [==============================] - 21s 107ms/step - loss: 0.5941 - accuracy:
0.7158 - val_loss: 0.4607 - val_accuracy: 0.7938
Epoch 7/10
191/191 [==============================] - 20s 106ms/step - loss: 0.5789 - accuracy:
0.7297 - val_loss: 0.4529 - val_accuracy: 0.7965
Epoch 8/10
191/191 [==============================] - 20s 107ms/step - loss: 0.5639 - accuracy:
0.7363 - val_loss: 0.4493 - val_accuracy: 0.7965
Epoch 9/10
191/191 [==============================] - 20s 107ms/step - loss: 0.5392 - accuracy:
0.7499 - val_loss: 0.4444 - val_accuracy: 0.8037
Epoch 10/10
191/191 [==============================] - 20s 106ms/step - loss: 0.5385 - accuracy:
0.7493 - val_loss: 0.4398 - val_accuracy: 0.8037
```

In [34]:
```
test_pred = np.argmax(model5.predict(test_x), axis=1)
print(test_pred)
```

```
[0 0 0 ... 0 0 0]
```

In [35]:
```
history = history5.history

trg_loss = history['loss']
val_loss = history['val_loss']

trg_acc = history['accuracy']
val_acc = history['val_accuracy']

epochs = range(1, len(trg_acc) + 1)

# plot losses and accuracies for training and validation
fig = plt.figure(figsize=(12,6))
ax = fig.add_subplot(1, 2, 1)
plt.plot(epochs, trg_loss, marker='o', label='Training Loss')
plt.plot(epochs, val_loss, marker='x', label='Validation Loss')
plt.title("Training / Validation Loss")
ax.set_ylabel("Loss")
ax.set_xlabel("Epochs")
plt.legend(loc='best')

ax = fig.add_subplot(1, 2, 2)
plt.plot(epochs, trg_acc, marker='o', label='Training Accuracy')
plt.plot(epochs, val_acc, marker='^', label='Validation Accuracy')
plt.title("Training / Validation Accuracy")
ax.set_ylabel("Accuracy")
ax.set_xlabel("Epochs")
```

```
plt.legend(loc='best')
plt.show()
```



Both loss and accuracy values converge and the difference from the train data results tends to decrease. However, the loss value is still large and the accumulation value seems to need further improvement.

model 6

GloVe embedded GRU

In [36]:

```
epoch_size =10
batch_size = 32

model6 = Sequential([
    layers.Embedding(input_dim=embedding_matrix.shape[0],
                     output_dim=embedding_matrix.shape[1],
                     weights = [embedding_matrix]
                     ),
    layers.Bidirectional(layers.GRU(256, return_sequences=True)),
    layers.GlobalMaxPool1D(),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.4),
    layers.Dense(2, activation='sigmoid')
])
model6.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, None, 100)         1394700
_________________________________________________________________
bidirectional_6 (Bidirection (None, None, 512)         549888
_________________________________________________________________
global_max_pooling1d_4 (Glob (None, 512)               0
_________________________________________________________________
dense_12 (Dense)             (None, 64)                32832
_________________________________________________________________
dropout_7 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_13 (Dense)             (None, 2)                 130
=================================================================
```

```
Total params: 1,977,550
Trainable params: 1,977,550
Non-trainable params: 0
_________________________________________________________________
```

In [37]:

```python
model6.compile(loss='sparse_categorical_crossentropy', optimizer = RMSprop(learning_r
history6 = model6.fit(x, y, epochs=epoch_size, validation_split=0.1)
```

```
Epoch 1/10
215/215 [==============================] - 142s 651ms/step - loss: 0.5484 - accuracy:
0.7254 - val_loss: 0.4592 - val_accuracy: 0.7940
Epoch 2/10
215/215 [==============================] - 144s 667ms/step - loss: 0.4653 - accuracy:
0.7942 - val_loss: 0.4424 - val_accuracy: 0.7966
Epoch 3/10
215/215 [==============================] - 144s 671ms/step - loss: 0.4405 - accuracy:
0.8098 - val_loss: 0.4555 - val_accuracy: 0.8005
Epoch 4/10
215/215 [==============================] - 144s 669ms/step - loss: 0.4211 - accuracy:
0.8165 - val_loss: 0.4485 - val_accuracy: 0.8045
Epoch 5/10
215/215 [==============================] - 144s 670ms/step - loss: 0.4124 - accuracy:
0.8219 - val_loss: 0.4305 - val_accuracy: 0.8097
Epoch 6/10
215/215 [==============================] - 145s 677ms/step - loss: 0.4002 - accuracy:
0.8323 - val_loss: 0.4333 - val_accuracy: 0.8150
Epoch 7/10
215/215 [==============================] - 146s 677ms/step - loss: 0.3874 - accuracy:
0.8384 - val_loss: 0.4287 - val_accuracy: 0.8163
Epoch 8/10
215/215 [==============================] - 146s 679ms/step - loss: 0.3763 - accuracy:
0.8415 - val_loss: 0.4253 - val_accuracy: 0.8176
Epoch 9/10
215/215 [==============================] - 148s 687ms/step - loss: 0.3612 - accuracy:
0.8489 - val_loss: 0.4335 - val_accuracy: 0.8071
Epoch 10/10
215/215 [==============================] - 146s 681ms/step - loss: 0.3501 - accuracy:
0.8548 - val_loss: 0.4339 - val_accuracy: 0.8071
```

In [38]:

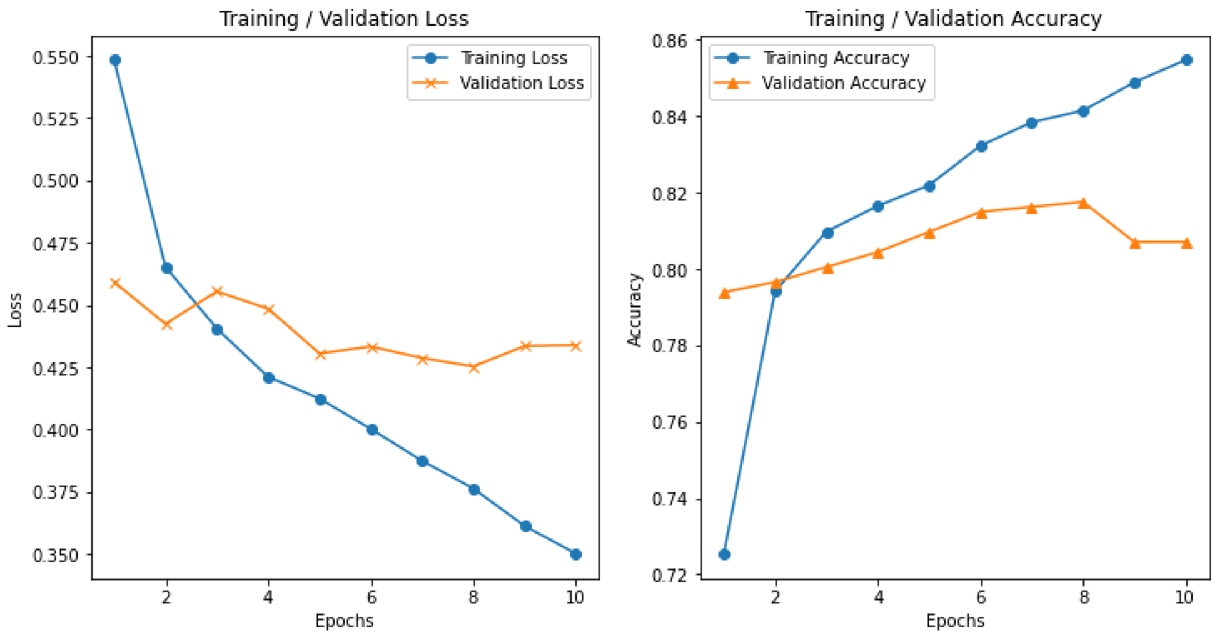
```python
test_pred = np.argmax(model6.predict(test_x), axis=1)
print(test_pred)
```

```
[1 0 1 ... 1 1 0]
```

In [39]:

```python
history = history6.history

trg_loss = history['loss']
val_loss = history['val_loss']

trg_acc = history['accuracy']
val_acc = history['val_accuracy']

epochs = range(1, len(trg_acc) + 1)

# plot losses and accuracies for training and validation
fig = plt.figure(figsize=(12,6))
ax = fig.add_subplot(1, 2, 1)
plt.plot(epochs, trg_loss, marker='o', label='Training Loss')
plt.plot(epochs, val_loss, marker='x', label='Validation Loss')
plt.title("Training / Validation Loss")
ax.set_ylabel("Loss")
ax.set_xlabel("Epochs")
plt.legend(loc='best')

ax = fig.add_subplot(1, 2, 2)
```

```
plt.plot(epochs, trg_acc, marker='o', label='Training Accuracy')
plt.plot(epochs, val_acc, marker='^', label='Validation Accuracy')
plt.title("Training / Validation Accuracy")
ax.set_ylabel("Accuracy")
ax.set_xlabel("Epochs")
plt.legend(loc='best')
plt.show()
```



Both loss and accuracy values were improved in the desired direction.

6. Create a submission

Generate the results as a csv file.

In [40]:

```
submission = pd.DataFrame({'id':test_data['id'], 'target':test_pred})
submission.to_csv('submission.csv', index=False)
```

In [ ]:

```

```