1. **Data preparation, exploration, visualization**

In this assignment, the data was in the form of tweets and therefore it needed to be cleaned and preprocessed. Cleaning the data involved removing the stop words, or the most commonly used words, as well as punctuation and special characters. In addition, I converted verbs to their lemma form which converted participle verb forms to their root form. I used the WordNet database to do this. I tried this both ways and converting verbs to their lemma form was better. Next, I removed URLs as well as words that appeared only once in the vocabulary. In general, preprocessing is extremely important to get good results in NLP problems. Even minor errors in spelling could cause words to be interpreted differently. Lastly, I applied preprocessing functions on both the training and test sets, created a function to remove any emojis in the tweets, and again applied preprocessing functions on the training and test sets. Once the data was cleaned, I then proceeded to extract the words across all of the tweets. To do this, I created a function called create_vocab, concatenated the training and test datasets as a single data frame, and then called the create_vocab function on the full data frame in order to generate the number of words across all of the tweets. Next, I looked at the words that appeared multiple times across all of the tweets and created a new list to store those words. I also created a function called filter to filter the dataset based on which words are already present in the vocab dictionary that was created earlier and applied the filter function on both the training and test dataset. The last stage of preparing the data involved preprocessing the data. This stage primarily involved getting the tokens which are basically the different units that can be obtained from the text such as words, characters, or n-grams. In this stage, I used Keras to perform tokenization. First, I imported the Tokenizer library from Keras and created a function called create_tokenizer to perform the tokenization of the tweets. Then I applied create_tokenizer on the training dataset and found the number of unique tokens. Finally, I converted the text to sequences, a list of integers, and converted the sequences into a 2D integer tensor which has a shape of samples by maxlen.

2. **Review research design and modeling methods**

This assignment entailed using Recurrent Neural Networks (RNNs) to predict which tweets are about real disasters and which ones aren't. In this assignment, I created a LSTM neural network and a GRU neural network. There are a lot of similarities in creating neural networks using RNNs and creating neural networks in Assignments 6 and 7. However, there are also a lot of differences in terms of how the models were evaluated and a RNN and CNN are designed. In this assignment, the F1 score was the metric that was used for evaluating the performance of the neural networks whereas in assignments 6 and 7, the log loss and the accuracy were used to evaluate the performance of the neural networks that were used. In addition, the design of RNNs is different than that of CNNs and the neural networks that were used in prior assignments. A CNN is composed of multiple building blocks, such as convolution layers, pooling layers, and fully connected layers, and is designed to automatically and adaptively learn spatial hierarchies of features through a

backpropagation algorithm whereas RNNs, specifically the LSTM and GRU, are designed so that a RNN doesn't suffer from short term memory. Short term memory is a significant issue that plagues RNNs mainly because during back propagation, RNNs suffer from the vanishing gradient problem in which a gradient shrinks as it backpropagates through time. LSTM and GRU have internal mechanisms called gates that are able to regulate the flow of information and these gates can learn which data in a sequence should be kept and which data can be ignored. As a result, relevant information can be passed through the sequential that are being processed by the RNN in order to make predictions. Once the LSTM and GRU neural networks were prepared, I then prepared the test data for submission and submitted my results to Kaggle for evaluation

3. **Review results, evaluate models**

I felt that the results generated by the RNNs were acceptable. The GRU model seemed to perform slightly better than the LSTM. I also generated a classification report of the test data to better understand the overall performance of both the LSTM and GRU models. When analyzing the classification report of the LSTM model, I noticed that the LSTM model was willing to sacrifice a lower precision score in favor of a higher recall. By having a higher recall and a lower precision the LSTM model was much more successful in remembering the tweets that are in fact disaster tweets, while not being so accurate. Conversely, the LSTM model was willing to sacrifice a lower recall score in favor of a higher precision score when identifying the non-disaster tweets. This means that while the LSTM model did a better job accurately identifying the non-disaster tweets, the LSTM model did not do a great job in remembering the non-disaster tweets. When analyzing the classification report of the GRU model, I noticed that when compared to the LSTM model, the GRU model had a slightly higher precision score while the recall score was slightly less when identifying the disaster tweets and when identifying the non-disaster tweets, the GRU model had a slightly lower precision score while having a higher recall score. Therefore, the GRU model was willing to sacrifice recall in favor of a higher precision score when identifying the disaster tweets and was willing to sacrifice precision in favor of a higher recall score when identifying the non-disaster tweets. The results for the LSTM and GRU models are the following:

LSTM Model

	precision	recall	f1-score	support
0	0.75	0.90	0.82	874
1	0.81	0.59	0.69	649
accuracy			0.77	1523
macro avg	0.78	0.75	0.75	1523
weighted avg	0.78	0.77	0.76	1523

GRU Model

	precision	recall	f1-score	support
0	0.77	0.88	0.82	874
1	0.80	0.65	0.72	649
accuracy			0.78	1523
macro avg	0.79	0.76	0.77	1523
weighted avg	0.78	0.78	0.78	1523

4. Kaggle Submission relative to Peers

Username - anaswarj

Submission and Description	Public Score
GRU.csv a day ago by Anaswar Jayakumar MSDS 422 - GRU NLP	0.77198
LSTM.csv a day ago by Anaswar Jayakumar MSDS 422 - NLP LSTM	0.77015

5. Exposition, problem description and management recommendations

For problems involving natural language processing (NLP), neural networks such as RNNs outperform all other algorithms. Specifically, RNNs produce state of the art performance with respect to NLP while generalizing well (i.e. not overfitting the data). RNNs are well suited for NLP because NLP can be thought of as an extension of time series analysis. In time series analysis, data is being analyzed and over time and therefore it is important to remember both past and present data. Likewise, the goal of NLP is to ideally be able to fully understand the entire contents of the documents, including the contextual nuances of the text. RNNs come into play because of their ability to remember information over time. In NLP, large amounts of natural language data are often being processed at any given time and the ability to remember information is critical in being able to accurately extract information and insights that are contained in the text. One important technique that I could have used is Bidirectional Encoder Representations from Transformers (BERT), a state-of-the-art language model for NLP. BERT is a Transformer-based machine learning technique for NLP pretraining similar to that of word2vec, GloVe (global vectors), and fastText. Some of the Kagglers used BERT and achieved excellent scores. In addition, I could have spent more time on preprocessing the tweets. For example, one such Kaggler preprocessed the tweets by correcting the extraneous features such as URLs, hashtags, and special characters as well as mislabeled samples in the tweets. In addition, I could have also done cross validation as well to get better results.

LSTM

May 23, 2021

You are to compete in the Real or Not: NLP with Disaster Tweets competition on Kaggle.com. https://www.kaggle.com/c/nlp-getting-started/notebooks (Links to an external site.)

In this competition you will compete at least two varieties of RNN and submit your predictions to Kaggle. RNNs are well suited to the analysis of sequences, as needed for natural language processing (NLP). Initial background reading for this assignment comes from Chapter 14 (pp. 379–411) of the Géron textbook:

- Géron, A. (2017). Hands-on machine learning with Scikit-Learn & TensorFlow: Concepts, tools, and techniques to build intelligent systems. Sebastopol, CA: O'Reilly. [ISBN-13 978-1-491-96229-9]. Source code available via Github (Links to an external site.)

Specialized RNN models have been developed to accommodate the needs of many language processing tasks. For example LSTM and GRU models help address the problem of vanishing gradients and generally improve performance.

Larger relevant vocabularies are usually associated with more accurate models, but training with larger vocabularies requires more memory and longer processing times. We can speed up the training process by using pretrained word vectors and subsets of pretrained word vectors.

Technologies such as word2vec, GloVe (global vectors), and fastText provide ways of representing words as numeric vectors. These numeric vectors or neural network embeddings capture the meaning of words as well as their common usage as parts of speech. Word embeddings have extensive applications in natural language processing. For this competition, you should use the techniques promulgated on the Kaggle discussion board (e.g., https://www.kaggle.com/mashiat/nlp-rnn (Links to an external site.))

GloVe, a method for estimating pretrained word vectors, was developed at Stanford (Links to an external site.). A tutorial and code for using GloVe embeddings is available here. (Links to an external site.)

A third set of pretrained vectors is fastText, described here (Links to an external site.). Word embeddings are an active area of research, as shown in recent developments in probabilistic fastText (Links to an external site.).

We can also test vocabulary sizes associated with pretrained word vectors, defined as part of the language model. For example, we might compare a vocabulary of the top 10,000 words in English versus a vocabulary of the top 30,000 words.

For all Kaggle competitions, you must submit a screen snapshot that identifies you along with your scores on the submissions. Submit your work as a single .pdf file that is legible. Include your code as an appendix. Look at the rubric to see how you will be graded. Your work will be compared against your peers on the performance metric(s).

[3]:
```
# This Python 3 environment comes with many helpful analytics libraries
# installed
# It is defined by the kaggle/python Docker image:
# https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter)
# will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that
# gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved
# outside of the current session
```

[4]:
```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.pyplot import xticks
from nltk.corpus import stopwords
import nltk
import re
from nltk.stem import WordNetLemmatizer
import string
from nltk.tokenize import word_tokenize
from nltk.util import ngrams
from collections import defaultdict
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from keras.utils.vis_utils import plot_model
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint, EarlyStopping, Callback
import tensorflow as tf
from sklearn.metrics import f1_score
from wordcloud import WordCloud,STOPWORDS
```

```
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from keras.preprocessing.sequence import pad_sequences
from numpy import array
from numpy import asarray
from numpy import zeros
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense,Flatten,Embedding,Activation,Dropout
from keras.layers import Conv1D,MaxPooling1D,GlobalMaxPooling1D,LSTM
from keras.layers import Bidirectional
```

[6]:
```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call
drive.mount("/content/drive/", force_remount=True).
```

[7]:
```
# load train and test datasets
train= pd.read_csv('/content/drive/MyDrive/NLP train (4).csv')
test=pd.read_csv('/content/drive/MyDrive/NLP test (3).csv')
```

[8]:
```
# check the no. of rows and columns in the dataset
train.shape, test.shape
```

[8]:
```
((7613, 5), (3263, 4))
```

[9]:
```
train.head()
```

[9]:

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

[5 rows x 5 columns]

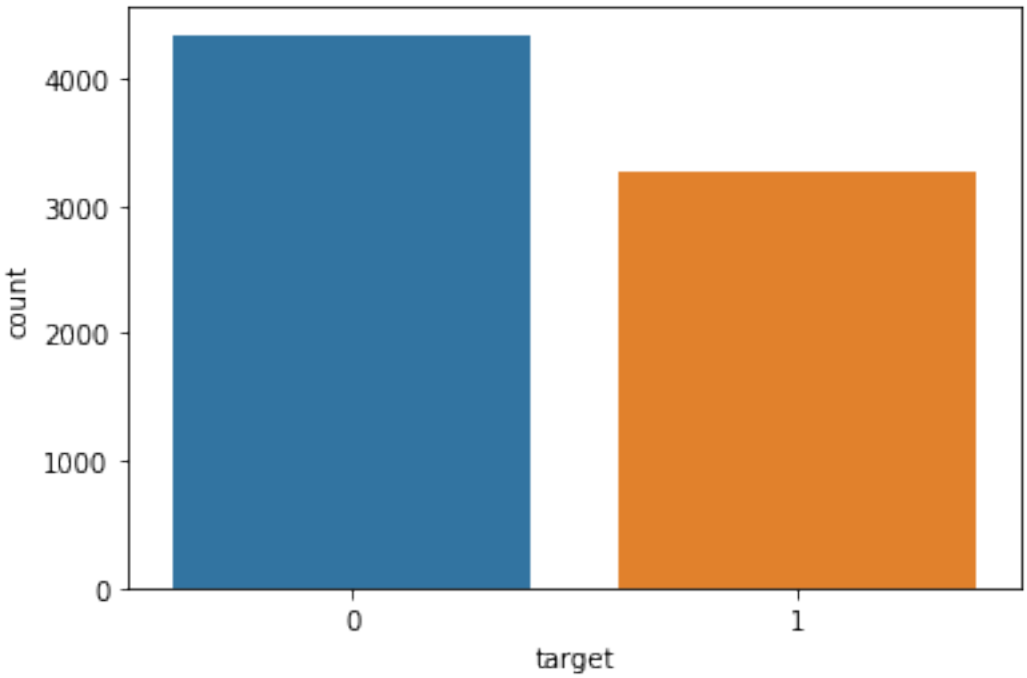
[10]:
```
train.isnull().sum().sort_values(ascending = False)
```

[10]:
```
location    2533
keyword       61
target         0
text           0
id             0
```

```
dtype: int64
```

[11]:
```
sns.countplot(x=train.target)
```

[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fce63b41b90>
```



1 Data Cleaning

[12]:
```
#lets save stopwords in a variable
import nltk
nltk.download('stopwords')

stop = list(stopwords.words("english"))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[13]:
```
# save list of punctuation/special characters in a variable
punctuation = list(string.punctuation)
```

[14]:
```
# create an object to convert the words to its lemma form
lemma = WordNetLemmatizer()
```

[15]:
```
# lets make a combine list of stopwords and punctuations
sw_pun = stop + punctuation
```

[16]:
```
# function to preprocess the messages
def preprocess(tweet):
    tweet = re.sub(r"https?:\/\/t.co\/[A-Za-z0-9]+", "", tweet) # removing urls
    tweet = re.sub('[^\w]',' ',tweet) # remove embedded special characters in
    # words (for example #earthquake)

    tweet = re.sub('[\d]','',tweet) # this will remove numeric characters
    tweet = tweet.lower()
    words = tweet.split()
    sentence = ""
    for word in words:
        if word not in (sw_pun):  # removing stopwords & punctuations
            word = lemma.lemmatize(word,pos = 'v')  # converting to lemma
            if len(word) > 3: # we will consider words with length  greater
            # than 3 only
                sentence = sentence + word + ' '
    return(sentence)
```

[17]:
```
import nltk
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Unzipping corpora/wordnet.zip.
```

[17]: True

[18]:
```
# apply preprocessing functions on the train and test datasets
train['text'] = train['text'].apply(lambda s : preprocess(s))
test['text'] = test['text'].apply(lambda s : preprocess(s))
```

[19]:
```
# function to remove emojis
def remove_emoji(text):
    emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)
```

[20]:
```
# applying the function on the train and the test datasets
train['text'] = train['text'].apply(lambda s : remove_emoji(s))
```

```
test['text'] = test['text'].apply(lambda s : remove_emoji(s))
```

2 Vocabulary Creation

[21]:
```
# function to create vocab
from collections import Counter
def create_vocab(df):
    vocab = Counter()
    for i in range(df.shape[0]):
        vocab.update(df.text[i].split())
    return(vocab)
```

[ ]:
```
# concatenate training and testing datasets
master=pd.concat((train,test)).reset_index(drop=True)

# call vocabulary creation function on master dataset
vocab = create_vocab(master)

# lets check the no. of words in the vocabulary
print(vocab)
print(len(vocab))
```

[23]:
```
# lets check the most common 50 words in the vocabulary
vocab.most_common(50)
```

[23]:
```
[('like', 560),
 ('fire', 534),
 ('bomb', 338),
 ('news', 299),
 ('people', 286),
 ('burn', 262),
 ('time', 254),
 ('kill', 253),
 ('make', 248),
 ('attack', 240),
 ('flood', 233),
 ('crash', 232),
 ('build', 231),
 ('emergency', 229),
 ('video', 228),
 ('come', 223),
 ('disaster', 220),
 ('take', 217),
 ('would', 214),
 ('body', 209),
 ('think', 204),
```

```
 ('police', 199),
 ('look', 193),
 ('know', 192),
 ('love', 190),
 ('watch', 188),
 ('home', 187),
 ('storm', 187),
 ('still', 181),
 ('train', 177),
 ('suicide', 177),
 ('live', 172),
 ('first', 170),
 ('collapse', 169),
 ('back', 164),
 ('scream', 164),
 ('california', 159),
 ('want', 156),
 ('drown', 152),
 ('cause', 151),
 ('need', 150),
 ('work', 149),
 ('today', 149),
 ('world', 148),
 ('nuclear', 148),
 ('hiroshima', 147),
 ('year', 143),
 ('full', 143),
 ('service', 142),
 ('destroy', 138)]
```

[24]:
```
# create the final vocab by considering words with more than one occurence
final_vocab = []
min_occur = 2
for k,v in vocab.items():
    if v >= min_occur:
        final_vocab.append(k)
```

[ ]:
```
print(final_vocab)
```

[25]:
```
# lets check the no. of the words in the final vocabulary
vocab_size = len(final_vocab)
vocab_size
```

[25]: 6025

[26]:
```
# function to filter the dataset, keep only words which are present in the
# vocab
```

```
def filter(tweet):
    sentence = ""
    for word in tweet.split():
        if word in final_vocab:
            sentence = sentence + word + ' '
    return(sentence)
```

[27]:
```
# apply filter function on the train and test datasets
train['text'] = train['text'].apply(lambda s : filter(s))
test['text'] = test['text'].apply(lambda s : filter(s))
```

[53]:
```
train['text']
```

[53]:
```
0                  deeds reason earthquake allah forgive
1                               forest fire near canada
2       residents shelter place officer evacuation she…
3       people receive wildfires evacuation order cali…
4       send photo ruby alaska smoke wildfires pour sc…
                              …
7608        giant crane hold bridge collapse nearby home
7609    aria_ahrary thetawniest control wild fire cali…
7610                                      volcano hawaii
7611    police investigate bike collide little portuga…
7612    latest home raze northern california wildfire …
Name: text, Length: 7613, dtype: object
```

[54]:
```
test['text']
```

[54]:
```
0                                 happen terrible crash
1       hear earthquake different cities stay safe eve…
2       forest fire spot pond flee across street canno…
3                     apocalypse light spokane wildfires
4                     typhoon soudelor kill china taiwan
                              …
3258                    earthquake safety angeles safety
3259    storm worse last hurricane city others hardest…
3260                       green line derailment chicago
3261                     issue hazardous weather outlook
3262    cityofcalgary activate municipal emergency pla…
Name: text, Length: 3263, dtype: object
```

[28]:
```
# lets take a look at the update training dataset
train.text.head()
```

[28]:
```
0              deeds reason earthquake allah forgive
1                           forest fire near canada
2    residents shelter place officer evacuation she…
```

```
3    people receive wildfires evacuation order cali…
4    send photo ruby alaska smoke wildfires pour sc…
Name: text, dtype: object
```

[31]:
```
from keras.layers import Embedding
```

[32]:
```
# Number of words to consider as features
max_features = vocab_size

# Cuts off the text after this number of words (among the max_features most
# common words)
maxlen = 100
```

3 Data Preprocessing

[33]:
```
# the different units into which you can break down text (words, characters,
# or n-grams) are called tokens, and breaking text into such tokens is called
# tokenization. This can be achieved using Tokenizer in Keras

from keras.preprocessing.text import Tokenizer
# fit a tokenizer
def create_tokenizer(lines):
    # num_words = vocab_size will create a tokenizer,configured to only take
    # into account the vocab_size(6025)
    tokenizer = Tokenizer(num_words=vocab_size)
    # Build the word index, Turns strings into lists of integer indices
    tokenizer.fit_on_texts(lines)
    return tokenizer
```

[34]:
```
# create and apply tokenizer on the training dataset
tokenizer = create_tokenizer(train.text)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

```
Found 5792 unique tokens.
```

[ ]:
```
from keras import preprocessing
# convert text to sequences
sequences = tokenizer.texts_to_sequences(train.text)
print(sequences)
```

[56]:
```
# Turns the lists of integers into a 2D integer tensor of shape
# (samples, maxlen), padding shorter sequences with 0s
train_text = preprocessing.sequence.pad_sequences(sequences, maxlen=maxlen)
train_text
```

[56]:
```
array([[   0,    0,    0, …,  175, 1156, 1552],
       [   0,    0,    0, …,    2,  146,  881],
       [   0,    0,    0, …,  329,  239,  365],
       …,
       [   0,    0,    0, …,    0,  427, 1149],
       [   0,    0,    0, …,   55,  774,  215],
       [   0,    0,    0, …,   31,  100,    4]], dtype=int32)
```

4 Model Building and Evaluation

[37]:
```
# Test train split
X_train, X_test, y_train, y_test = train_test_split(train_text, train.target,
                                                    test_size = 0.2,
                                                    random_state = 42)
```

[38]:
```
import keras.backend as K
def get_f1(y_true, y_pred): #taken from old keras source code
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val
```

[39]:
```
from keras.layers import LSTM

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))
```

[40]:
```
model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=[get_f1])
```

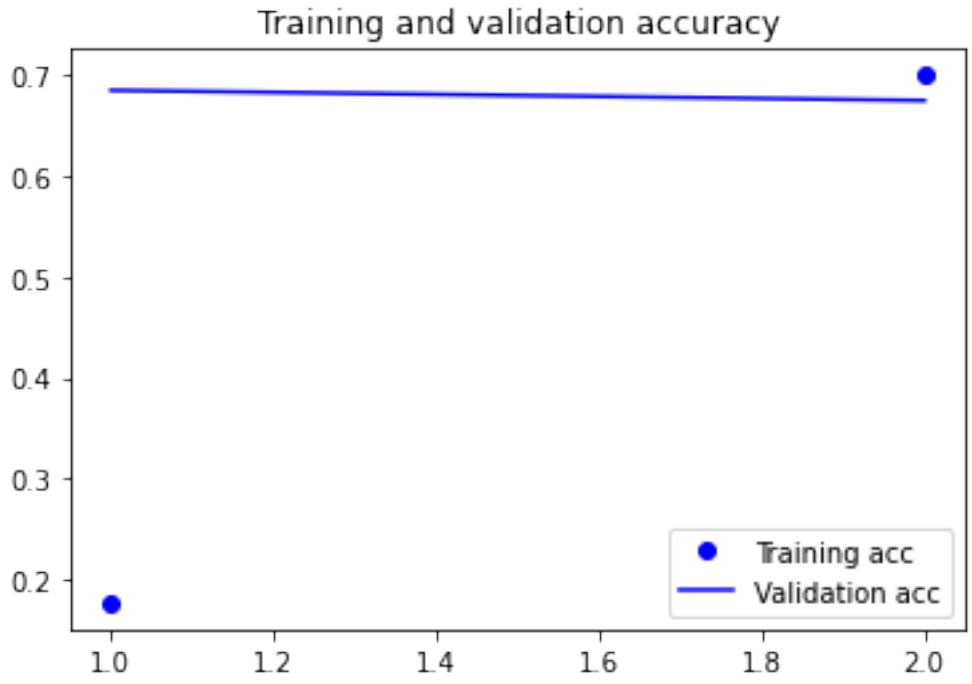
[41]:
```
callbacks_list = [EarlyStopping(monitor='get_f1',patience=1),
                  ModelCheckpoint(filepath='./LSTM.h5',monitor='val_loss',
                                  save_best_only=True)]
```

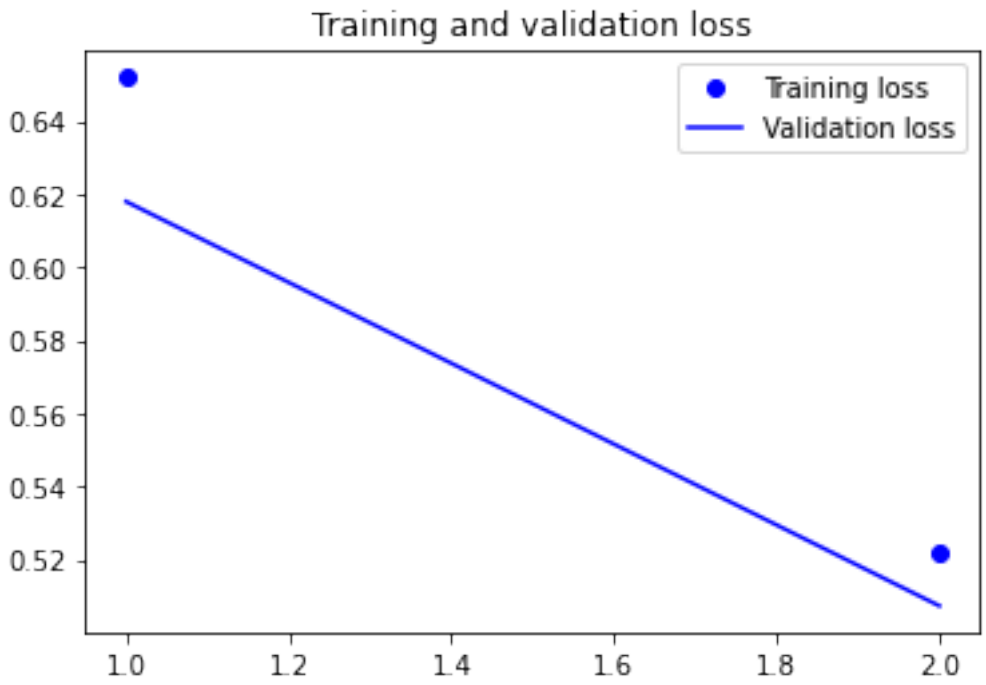
[42]:
```
history = model.fit(X_train, y_train, epochs=100, batch_size=128,
                    callbacks=callbacks_list, validation_split=0.2)
```

```
Epoch 1/100
39/39 [==============================] - 6s 92ms/step - loss: 0.6695 - get_f1:
0.0604 - val_loss: 0.6180 - val_get_f1: 0.6839
Epoch 2/100
```

```
39/39 [==============================] - 3s 78ms/step - loss: 0.5530 - get_f1:
0.6895 - val_loss: 0.5073 - val_get_f1: 0.6738
```

[43]:
```
# check model performance
acc = history.history['get_f1']
val_acc = history.history['val_get_f1']

loss = history.history['loss']
val_loss = history.history['val_loss']

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

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





[44]:
```
dependencies = {
    'get_f1': get_f1
}

# load the model from disk
loaded_model_LSTM = tf.keras.models.load_model('./LSTM.h5',
                                               custom_objects=dependencies)
```

[45]:
```
# prediction on the test dataset
#X_test_Set = tokenizer.texts_to_matrix(X_test, mode = 'freq')
y_pred = loaded_model_LSTM.predict_classes(X_test)

# important metrices
print(classification_report(y_test, y_pred))
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning:
`model.predict_classes()` is deprecated and will be removed after 2021-01-01.
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model
does multi-class classification   (e.g. if it uses a `softmax` last-layer
activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does
binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '

              precision    recall  f1-score   support
```

```
              0       0.75      0.90      0.82       874
              1       0.81      0.59      0.69       649

       accuracy                           0.77      1523
      macro avg       0.78      0.75      0.75      1523
   weighted avg       0.78      0.77      0.76      1523
```

[46]:
```
# convert text to sequences
sequences = tokenizer.texts_to_sequences(test.text)
# Turns the lists of integers into a 2D integer tensor of shape (samples,␣
 ↪maxlen)
test_text = preprocessing.sequence.pad_sequences(sequences, maxlen=maxlen)
```

[47]:
```
test_id = test.id
test.drop(["id","location","keyword"],1,inplace = True)

# apply tokenizer on the test dataset
test_set = tokenizer.texts_to_matrix(test.text, mode = 'freq')
```

[48]:
```
# make predictions on the test dataset
y_test_pred = loaded_model_LSTM.predict_classes(test_text)

# lets prepare for the prediction submission
sub = pd.DataFrame()
sub['id'] = test_id
sub['target'] = y_test_pred
sub.head()
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning:
`model.predict_classes()` is deprecated and will be removed after 2021-01-01.
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model
does multi-class classification   (e.g. if it uses a `softmax` last-layer
activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does
binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

[48]:

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

[49]:
```
sub.to_csv('/content/drive/MyDrive/LSTM.csv',index=False)
```

[ ]:

GRU

May 23, 2021

You are to compete in the Real or Not: NLP with Disaster Tweets competition on Kaggle.com. https://www.kaggle.com/c/nlp-getting-started/notebooks (Links to an external site.)

In this competition you will compete at least two varieties of RNN and submit your predictions to Kaggle. RNNs are well suited to the analysis of sequences, as needed for natural language processing (NLP). Initial background reading for this assignment comes from Chapter 14 (pp. 379–411) of the Géron textbook:

- Géron, A. (2017). Hands-on machine learning with Scikit-Learn & TensorFlow: Concepts, tools, and techniques to build intelligent systems. Sebastopol, CA: O'Reilly. [ISBN-13 978-1-491-96229-9]. Source code available via Github (Links to an external site.)

Specialized RNN models have been developed to accommodate the needs of many language processing tasks. For example LSTM and GRU models help address the problem of vanishing gradients and generally improve performance.

Larger relevant vocabularies are usually associated with more accurate models, but training with larger vocabularies requires more memory and longer processing times. We can speed up the training process by using pretrained word vectors and subsets of pretrained word vectors.

Technologies such as word2vec, GloVe (global vectors), and fastText provide ways of representing words as numeric vectors. These numeric vectors or neural network embeddings capture the meaning of words as well as their common usage as parts of speech. Word embeddings have extensive applications in natural language processing. For this competition, you should use the techniques promulgated on the Kaggle discussion board (e.g., https://www.kaggle.com/mashiat/nlp-rnn (Links to an external site.))

GloVe, a method for estimating pretrained word vectors, was developed at Stanford (Links to an external site.). A tutorial and code for using GloVe embeddings is available here. (Links to an external site.)

A third set of pretrained vectors is fastText, described here (Links to an external site.). Word embeddings are an active area of research, as shown in recent developments in probabilistic fastText (Links to an external site.).

We can also test vocabulary sizes associated with pretrained word vectors, defined as part of the language model. For example, we might compare a vocabulary of the top 10,000 words in English versus a vocabulary of the top 30,000 words.

For all Kaggle competitions, you must submit a screen snapshot that identifies you along with your scores on the submissions. Submit your work as a single .pdf file that is legible. Include your code as an appendix. Look at the rubric to see how you will be graded. Your work will be compared against your peers on the performance metric(s).

[56]:
```
# This Python 3 environment comes with many helpful analytics libraries
# installed
# It is defined by the kaggle/python Docker image:
# https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter)
# will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that
# gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved
# outside of the current session
```

[57]:
```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.pyplot import xticks
from nltk.corpus import stopwords
import nltk
import re
from nltk.stem import WordNetLemmatizer
import string
from nltk.tokenize import word_tokenize
from nltk.util import ngrams
from collections import defaultdict
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from keras.utils.vis_utils import plot_model
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint, EarlyStopping, Callback
import tensorflow as tf
from sklearn.metrics import f1_score
from wordcloud import WordCloud,STOPWORDS
```

```
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from keras.preprocessing.sequence import pad_sequences
from numpy import array
from numpy import asarray
from numpy import zeros
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense,Flatten,Embedding,Activation,Dropout
from keras.layers import Conv1D,MaxPooling1D,GlobalMaxPooling1D,LSTM
from keras.layers import Bidirectional
```

[58]:
```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call
drive.mount("/content/drive/", force_remount=True).
```

[59]:
```
# load train and test datasets
train= pd.read_csv('/content/drive/MyDrive/NLP train (4).csv')
test=pd.read_csv('/content/drive/MyDrive/NLP test (3).csv')
```

[60]:
```
# check the no. of rows and columns in the dataset
train.shape, test.shape
```

[60]:
```
((7613, 5), (3263, 4))
```

[61]:
```
train.head()
```

[61]:

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

[5 rows x 5 columns]

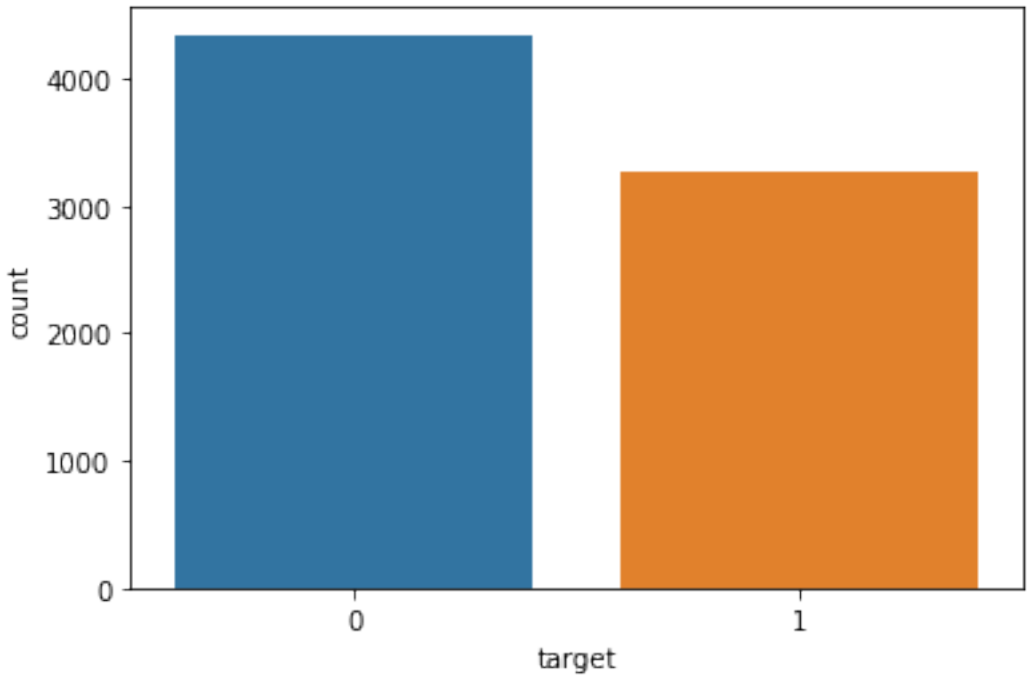
[62]:
```
train.isnull().sum().sort_values(ascending = False)
```

[62]:
```
location    2533
keyword       61
target         0
text           0
id             0
```

```
dtype: int64
```

[63]:
```
sns.countplot(x=train.target)
```

[63]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8af339d610>
```



1 Data Cleaning

[64]:
```
#lets save stopwords in a variable
import nltk
nltk.download('stopwords')

stop = list(stopwords.words("english"))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[65]:
```
# save list of punctuation/special characters in a variable
punctuation = list(string.punctuation)
```

[66]:
```
# create an object to convert the words to its lemma form
lemma = WordNetLemmatizer()
```

[67]:
```
# lets make a combine list of stopwords and punctuations
sw_pun = stop + punctuation
```

[68]:
```
# function to preprocess the messages
def preprocess(tweet):
    tweet = re.sub(r"https?:\/\/t.co\/[A-Za-z0-9]+", "", tweet) # removing urls
    tweet = re.sub('[^\w]',' ',tweet) # remove embedded special characters in
    # words (for example #earthquake)

    tweet = re.sub('[\d]','',tweet) # this will remove numeric characters
    tweet = tweet.lower()
    words = tweet.split()
    sentence = ""
    for word in words:
        if word not in (sw_pun):  # removing stopwords & punctuations
            word = lemma.lemmatize(word,pos = 'v')  # converting to lemma
            if len(word) > 3: # we will consider words with length  greater
            # than 3 only
                sentence = sentence + word + ' '
    return(sentence)
```

[69]:
```
import nltk
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[69]: True

[70]:
```
# apply preprocessing functions on the train and test datasets
train['text'] = train['text'].apply(lambda s : preprocess(s))
test['text'] = test['text'].apply(lambda s : preprocess(s))
```

[71]:
```
# function to remove emojis
def remove_emoji(text):
    emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)
```

[72]:
```
# applying the function on the train and the test datasets
train['text'] = train['text'].apply(lambda s : remove_emoji(s))
```

```
test['text'] = test['text'].apply(lambda s : remove_emoji(s))
```

2 Vocabulary Creation

[73]:
```
# function to create vocab
from collections import Counter
def create_vocab(df):
    vocab = Counter()
    for i in range(df.shape[0]):
        vocab.update(df.text[i].split())
    return(vocab)
```

[74]:
```
# concatenate training and testing datasets
master=pd.concat((train,test)).reset_index(drop=True)

# call vocabulary creation function on master dataset
vocab = create_vocab(master)

# lets check the no. of words in the vocabulary
print(vocab)
print(len(vocab))
```

[74]:
```
16442
```

[75]:
```
# lets check the most common 50 words in the vocabulary
vocab.most_common(50)
```

[75]:
```
[('like', 560),
 ('fire', 534),
 ('bomb', 338),
 ('news', 299),
 ('people', 286),
 ('burn', 262),
 ('time', 254),
 ('kill', 253),
 ('make', 248),
 ('attack', 240),
 ('flood', 233),
 ('crash', 232),
 ('build', 231),
 ('emergency', 229),
 ('video', 228),
 ('come', 223),
 ('disaster', 220),
 ('take', 217),
 ('would', 214),
```

```
 ('body', 209),
 ('think', 204),
 ('police', 199),
 ('look', 193),
 ('know', 192),
 ('love', 190),
 ('watch', 188),
 ('home', 187),
 ('storm', 187),
 ('still', 181),
 ('train', 177),
 ('suicide', 177),
 ('live', 172),
 ('first', 170),
 ('collapse', 169),
 ('back', 164),
 ('scream', 164),
 ('california', 159),
 ('want', 156),
 ('drown', 152),
 ('cause', 151),
 ('need', 150),
 ('work', 149),
 ('today', 149),
 ('world', 148),
 ('nuclear', 148),
 ('hiroshima', 147),
 ('year', 143),
 ('full', 143),
 ('service', 142),
 ('destroy', 138)]
```

[76]:
```
# create the final vocab by considering words with more than one occurence
final_vocab = []
min_occur = 2
for k,v in vocab.items():
    if v >= min_occur:
        final_vocab.append(k)
```

[ ]:
```
print(final_vocab)
```

[77]:
```
# lets check the no. of the words in the final vocabulary
vocab_size = len(final_vocab)
vocab_size
```

[77]: 6025

[78]:
```
# function to filter the dataset, keep only words which are present in the vocab
def filter(tweet):
    sentence = ""
    for word in tweet.split():
        if word in final_vocab:
            sentence = sentence + word + ' '
    return(sentence)
```

[79]:
```
# apply filter function on the train and test datasets
train['text'] = train['text'].apply(lambda s : filter(s))
test['text'] = test['text'].apply(lambda s : filter(s))
```

[80]:
```
# lets take a look at the update training dataset
train.text.head()
```

[80]:
```
0               deeds reason earthquake allah forgive
1                            forest fire near canada
2    residents shelter place officer evacuation she…
3    people receive wildfires evacuation order cali…
4    send photo ruby alaska smoke wildfires pour sc…
Name: text, dtype: object
```

[81]:
```
from keras.layers import Embedding
```

[82]:
```
# Number of words to consider as features
max_features = vocab_size

# Cuts off the text after this number of words (among the max_features most␣
 ↪common words)
maxlen = 100
```

3 Data Preprocessing

[83]:
```
# the different units into which you can break down text (words, characters,
# or n-grams) are called tokens, and breaking text into such tokens is called
# tokenization, this can be achieved using Tokenizer in Keras

from keras.preprocessing.text import Tokenizer
# fit a tokenizer
def create_tokenizer(lines):
    # num_words = vocab_size will create a tokenizer,configured to only take␣
 ↪into account the vocab_size(6025)
    tokenizer = Tokenizer(num_words=vocab_size)
    # Build th word index, Turns strings into lists of integer indices
    tokenizer.fit_on_texts(lines)
    return tokenizer
```

[84]:
```
# create and apply tokenizer on the training dataset
tokenizer = create_tokenizer(train.text)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

[85]:
```
from keras import preprocessing
# convert text to sequences
sequences = tokenizer.texts_to_sequences(train.text)
print(sequences)
```

[86]:
```
# Turns the lists of integers into a 2D integer tensor of shape
# (samples, maxlen), padding shorter sequences with 0s
train_text = preprocessing.sequence.pad_sequences(sequences, maxlen=maxlen)
```

4 Model Building and Evaluation

[87]:
```
# Test train split
X_train, X_test, y_train, y_test = train_test_split(train_text, train.target,
                                                    test_size = 0.2,
                                                    random_state = 42)
```

[88]:
```
import keras.backend as K
def get_f1(y_true, y_pred): #taken from old keras source code
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val
```

[89]:
```
from keras.layers import GRU

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(GRU(32))
model.add(Dense(1, activation='sigmoid'))
```

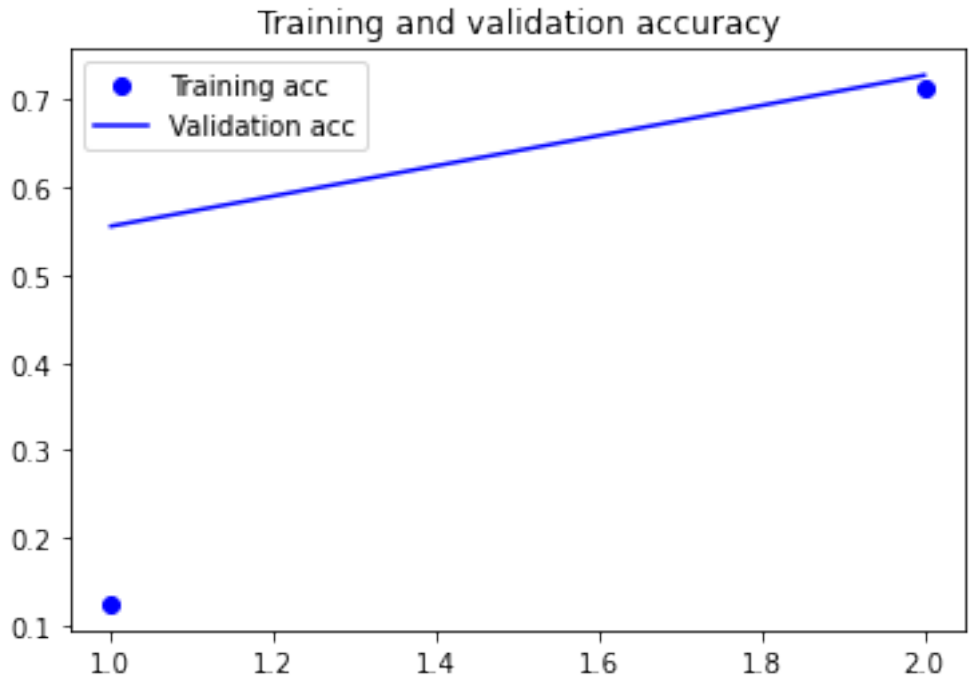
[90]:
```
model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=[get_f1])
```

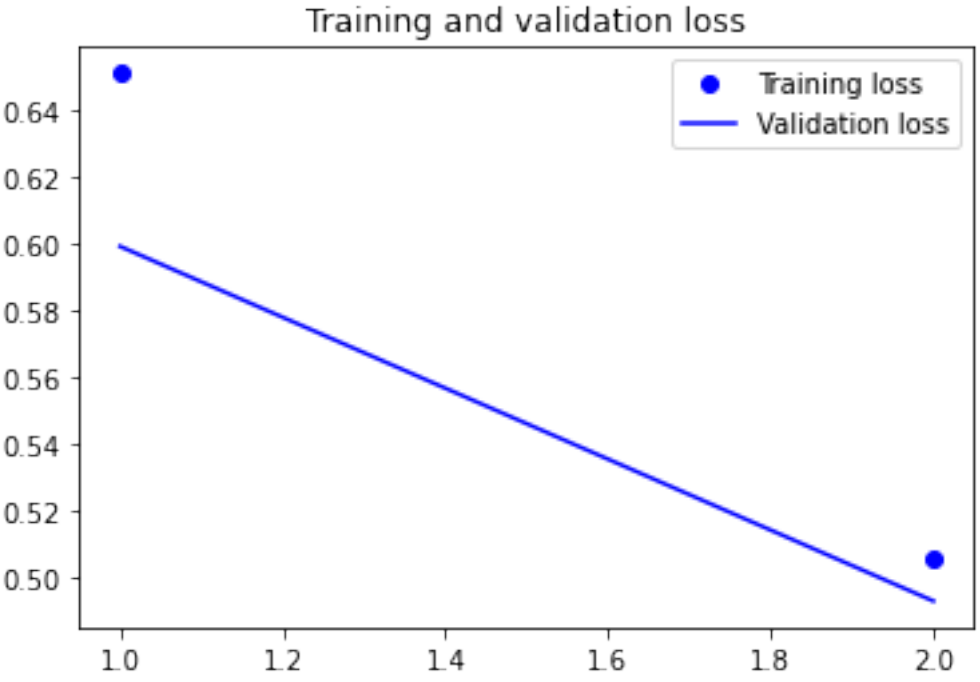
[91]:
```
callbacks_list = [EarlyStopping(monitor='get_f1',patience=1),
                  ModelCheckpoint(filepath='./GRU.h5',monitor='val_loss',
                                  save_best_only=True)]
```

[92]:
```
history = model.fit(X_train, y_train, epochs=100, batch_size=128,
                    callbacks=callbacks_list, validation_split=0.2)
```

```
Epoch 1/100
39/39 [==============================] - 5s 80ms/step - loss: 0.6732 - get_f1:
0.0842 - val_loss: 0.5993 - val_get_f1: 0.5558
Epoch 2/100
39/39 [==============================] - 3s 73ms/step - loss: 0.5398 - get_f1:
0.6497 - val_loss: 0.4928 - val_get_f1: 0.7281
```

[93]:
```
# check model performance
acc = history.history['get_f1']
val_acc = history.history['val_get_f1']

loss = history.history['loss']
val_loss = history.history['val_loss']

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

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1.0
1.2
1.4
1.6
1.8
2.0


Training and validation loss
Training loss
Validation loss
0.64
0.62
0.60
0.58
0.56
0.54
0.52
0.50
1.0
1.2
1.4
1.6
1.8
2.0

[94]:
```
dependencies = {
    'get_f1': get_f1
}

# load the model from disk
loaded_model_GRU = tf.keras.models.load_model('./GRU.h5',
                                              custom_objects=dependencies)
```

[95]:
```
# prediction on the test dataset
#X_test_Set = tokenizer.texts_to_matrix(X_test, mode = 'freq')
y_pred = loaded_model_GRU.predict_classes(X_test)

# important metrices
print(classification_report(y_test, y_pred))
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning:
`model.predict_classes()` is deprecated and will be removed after 2021-01-01.
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model
does multi-class classification   (e.g. if it uses a `softmax` last-layer
activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does
binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

	precision	recall	f1-score	support
0	0.77	0.88	0.82	874
1	0.80	0.65	0.72	649
accuracy			0.78	1523
macro avg	0.79	0.76	0.77	1523
weighted avg	0.78	0.78	0.78	1523

[96]:
```
# convert text to sequences
sequences = tokenizer.texts_to_sequences(test.text)
# Turns the lists of integers into a 2D integer tensor of shape (samples,␣
 ↪maxlen)
test_text = preprocessing.sequence.pad_sequences(sequences, maxlen=maxlen)
```

[97]:
```
test_id = test.id
test.drop(["id","location","keyword"],1,inplace = True)

# apply tokenizer on the test dataset
test_set = tokenizer.texts_to_matrix(test.text, mode = 'freq')
```

[98]:
```
# make predictions on the test dataset
y_test_pred = loaded_model_GRU.predict_classes(test_text)
```

```
# lets prepare for the prediction submission
sub = pd.DataFrame()
sub['id'] = test_id
sub['target'] = y_test_pred
sub.head()
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning:
`model.predict_classes()` is deprecated and will be removed after 2021-01-01.
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model
does multi-class classification   (e.g. if it uses a `softmax` last-layer
activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does
binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

[98]:

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

[99]:
```
sub.to_csv('/content/drive/MyDrive/GRU.csv',index=False)
```

[ ]:
```

```