In [1]:

```
!pip install keras_self_attention
```

```
Requirement already satisfied: keras_self_attention in /usr/local/lib/python3.6/dist-packages
(0.47.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from
keras_self_attention) (1.18.5)
Requirement already satisfied: Keras in /usr/local/lib/python3.6/dist-packages (from
keras_self_attention) (2.4.3)
Requirement already satisfied: scipy>=0.14 in /usr/local/lib/python3.6/dist-packages (from Keras->
keras_self_attention) (1.4.1)
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packages (from Keras-
>keras_self_attention) (2.10.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.6/dist-packages (from Keras-
>keras_self_attention) (3.13)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from h5py->Keras-
>keras_self_attention) (1.15.0)
```

In [2]:

```
pip install keras_metrics
```

```
Requirement already satisfied: keras_metrics in /usr/local/lib/python3.6/dist-packages (1.1.0)
Requirement already satisfied: Keras>=2.1.5 in /usr/local/lib/python3.6/dist-packages (from
keras_metrics) (2.4.3)
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.6/dist-packages (from
Keras>=2.1.5->keras_metrics) (1.18.5)
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packages (from Keras>=2.1.5-
>keras_metrics) (2.10.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.6/dist-packages (from
Keras>=2.1.5->keras_metrics) (3.13)
Requirement already satisfied: scipy>=0.14 in /usr/local/lib/python3.6/dist-packages (from
Keras>=2.1.5->keras_metrics) (1.4.1)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from h5py-
>Keras>=2.1.5->keras_metrics) (1.15.0)
```

In [3]:

```
import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from sklearn.ensemble import RandomForestClassifier
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.pipeline import Pipeline
import operator
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.preprocessing.sequence import pad_sequences
from keras.callbacks import EarlyStopping
from keras.preprocessing.text import Tokenizer
from keras.initializers import Constant
from keras.layers import TimeDistributed
from keras_self_attention import SeqSelfAttention
from numpy.random import seed
import h5py
seed(1)
import codecs
import datetime
import logging
import plotly.graph_objs as go
```

```
import matplotlib.pyplot as plt
import pydot
import keras
from keras import backend as k
k.set_learning_phase(1)
from keras import initializers
from keras.optimizers import RMSprop
from keras.models import Sequential,Model
from keras.layers import Dense,LSTM,GRU,Dropout,Input,Activation,Add,concatenate, Embedding, Repeat
Vector,SpatialDropout1D
from keras.layers.advanced_activations import LeakyReLU,PReLU
from keras.callbacks import ModelCheckpoint
from keras.models import load_model
from keras.optimizers import Adam
from keras.preprocessing.sequence import pad_sequences
import tensorflow as tf
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import re
import sys
import pickle
from tqdm import tqdm
import os
from keras.layers import BatchNormalization
from keras.layers import Bidirectional
import keras_metrics as km
```

```
WARNING:tensorflow:From <ipython-input-3-101e6e9988cc>:37: set_learning_phase (from
tensorflow.python.keras.backend) is deprecated and will be removed after 2020-10-11.
Instructions for updating:
Simply pass a True/False value to the `training` argument of the `__call__` method of your layer o
r model.
```

In [4]:

```
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).
```

https://towardsdatascience.com/how-to-quickly-build-a-tensorflow-training-pipeline-15e9ae4d78a0

https://www.thepythoncode.com/article/text-classification-using-tensorflow-2-and-keras-in-python

In [5]:

```
df1 = pd.read_csv('/content/drive/My Drive/train_CS2.csv')
#df1 = pd.read_csv('train.csv')
print(df1.columns)
```

```
Index(['textID', 'text', 'selected_text', 'sentiment'], dtype='object')
```

In [6]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27481 entries, 0 to 27480
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   textID         27481 non-null  object
 1   text           27480 non-null  object
 2   selected_text  27480 non-null  object
 3   sentiment      27481 non-null  object
dtypes: object(4)
memory usage: 858.9+ KB
```

In [7]:

```
positive_text_y = df1[df1['sentiment'] == 'positive']['selected_text']
negative_text_y = df1[df1['sentiment'] == 'negative']['selected_text']
neutral_text_y = df1[df1['sentiment'] == 'neutral']['selected_text']
```

In [8]:

```
# data cleaning
def cleantext(text):
    text = str(text)
    text = text.split()
    words= []
    for t in text:
        words.append(t)
    text=" ".join(words)
    text=text.lower()
    text=re.sub(r"9\\/11"," 911 ",text)
    text=re.sub(r"\n"," ",text)
    text=re.sub(r":"," ",text)
    text=re.sub(r"-"," ",text)
    text=re.sub(r"\_"," ",text)
    text=re.sub(r"\d+"," ",text)
    text=re.sub(r"\."","",text)
    #text=re.sub(r"\,"," ",text)
    text=re.sub(r"[$#@%&!~?%{}()]"," ",text)

    return text
```

In [9]:

```
# for setting the selected_text to 1 in y_pad
def set_to_one(X_train_pad,y_train_pad):
    y_train_pad1 = []
    for k in range(len(X_train_pad)):
        X1 = X_train_pad[k].copy()
        y = y_train_pad[k].copy()
        for i in range(len(X1)):
            if(X1[i] == 0):
                X1[i] = 2
            else:
                flag = 0
                for j in range(len(y)):
                    if(X1[i] == y[j]):
                        X1[i] = 1
                        flag=1
                        break
                if(flag == 0):
                    X1[i] = 0
        y_train_pad1.append(X1)
    return(y_train_pad1)
```

In [10]:

```
def maskedLoss(y_true, y_pred):

    y_true = tf.expand_dims(y_true,2)

    #getting mask value
    mask = tf.math.logical_not(tf.math.equal(y_true, 2))


    #converting mask dtype to loss_ dtype
    mask = tf.cast(mask, dtype=y_true.dtype)

    y_true = y_true * mask

    loss_function = tf.keras.losses.BinaryCrossentropy(from_logits=False, reduction='none')

    #calculating the loss
    loss_ = loss_function(y_true, y_pred)

    #converting mask dtype to loss_ dtype
```

```
    mask = tf.cast(mask, dtype=loss_.dtype)

    #applying the mask to loss
    loss_ = loss_ * mask

    #getting mean over all the values
    loss_ = tf.reduce_sum(loss_) / tf.reduce_sum(mask)

    return loss_
```

In [11]:

```
def get_embedding_vectors(vocab_size, EMBEDDING_DIM, tokenizer):

  embeddings_index = {}
  f = codecs.open('/content/drive/My Drive/wiki-news-300d-1M.vec', encoding='utf-8')
  for line in tqdm(f):
    values = line.rstrip().rsplit(' ')
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
  f.close()

  words_not_found = []
  nb_words = min(vocab_size , len(tokenizer.word_index)+1)
  embedding_matrix = np.zeros((nb_words, EMBEDDING_DIM))
  for word, i in tokenizer.word_index.items():
    if i >= nb_words:
      continue
    embedding_vector = embeddings_index.get(word)

    if (embedding_vector is not None) and len(embedding_vector) > 0:
        embedding_matrix[i] = embedding_vector
    else:
        words_not_found.append(word)

  return embedding_matrix,nb_words
```

In [12]:

```
def build_model(nb_words,EMBEDDING_DIM, max_length, weights):
  model = Sequential()
  model.add(Embedding(nb_words,EMBEDDING_DIM, input_length=max_length,
weights=weights,trainable=False))
  model.add(Bidirectional(LSTM(50,return_sequences=True)))
  model.add(SeqSelfAttention(attention_width=10,attention_type=SeqSelfAttention.ATTENTION_TYPE_MUL,
                        kernel_regularizer=keras.regularizers.l2(1e-4),
                        bias_regularizer=keras.regularizers.l1(1e-4),
                        attention_regularizer_weight=1e-4,attention_activation='softmax'))
  model.add(TimeDistributed(Dense(1, activation ='sigmoid')))

  return model
```

In [13]:

```
def arr1_train_temp(ndarr_train_temp):
    #arr1_train_pos = np.empty(ndarr_train_temp.shape)
    for i in range(ndarr_train_temp.shape[0]):
    #for i in range(2):

        list1 = np.dstack(ndarr_train_temp[i])
        list2 = list1.copy()

        for k in range(len(list2[0][0])):
            list2[0][0][k] = np.max(list1[0][0]) - list1[0][0][k]

        th = 0.045
        for j in range(len(list2[0][0])):
            if (list2[0][0][j] < th):
                list2[0][0][j] = 1
            else:
                list2[0][0][j] = 0

        ndarr_train_temp[i] = (list2[0].astype(int).T).copy()
```

```
        #ndarr_train_temp.append(ndarr_train_temp[i])

    return( ndarr_train_temp)
```

In [14]:

```
def rev(arr1_train,X_train_pad):
    y_train_pad_rev = []
    for k in range(len(arr1_train)):
        X1 = arr1_train[k].copy()
        y = X_train_pad[k].copy()
        for i in range(len(X1)):
            if(X1[i] == 1):
                X1[i] = y[i]
        y_train_pad_rev.append(X1)
    return y_train_pad_rev
```

In [15]:

```
# Function takes a tokenized sentence and returns the words
def sequence_to_text(list_of_indices,reverse_word_map):
  # Looking up words in dictionary
  words = [reverse_word_map.get(letter) for letter in list_of_indices]
  return(words)
```

In [16]:

```
def pred_txt(my_texts_train):
    predicted_txts_train = []
    for row in my_texts_train:
        words= []
        text = ''
        for word in row:
            if(word != None):
                words.append(word)
                text=" ".join(words)
        predicted_txts_train.append(text)
    return predicted_txts_train
#predicted_txts
```

In [17]:

```
def jaccard(str1, str2):
    a = set(str1.lower().split())
    b = set(str2.lower().split())
    c = a.intersection(b)
    if (len(a) == 0):
        return 0
    return float(len(c)) / (len(a) + len(b) - len(c))
```

In [18]:

```
def jaccard_score(predicted_txts_train_pos,y_train_pos):
  jaccard_vals_train_pos = []
  for i in range(len(predicted_txts_train_pos)):
    jac = jaccard(predicted_txts_train_pos[i],y_train_pos.iloc[i].to_string())
    jaccard_vals_train_pos.append(jac)
  return np.mean(jaccard_vals_train_pos)
```

In [19]:

```
def final_fun_1(df1):
  y = pd.DataFrame(df1['selected_text'])
  X = pd.DataFrame(df1['text'])

  positive_text_X = df1[df1['sentiment'] == 'positive']['text']
  negative_text_X = df1[df1['sentiment'] == 'negative']['text']
  neutral_text_X = df1[df1['sentiment'] == 'neutral']['text']

  positive_text_y = df1[df1['sentiment'] == 'positive']['selected_text']
  negative_text_y = df1[df1['sentiment'] == 'negative']['selected_text']
```

```
  negative_text_y = df1[df1['sentiment'] == 'negative']['selected_text']
  neutral_text_y = df1[df1['sentiment'] == 'neutral']['selected_text']

  positive_text_X = positive_text_X.apply(lambda x: cleantext(x))
  negative_text_X = negative_text_X.apply(lambda x: cleantext(x))
  neutral_text_X = neutral_text_X.apply(lambda x: cleantext(x))

  positive_text_y = positive_text_y.apply(lambda x: cleantext(x))
  negative_text_y = negative_text_y.apply(lambda x: cleantext(x))
  neutral_text_y = neutral_text_y.apply(lambda x: cleantext(x))

  positive_text_X = pd.DataFrame(positive_text_X)
  negative_text_X = pd.DataFrame(negative_text_X)
  neutral_text_X = pd.DataFrame(neutral_text_X)

  positive_text_y = pd.DataFrame(positive_text_y)
  negative_text_y = pd.DataFrame(negative_text_y)
  neutral_text_y = pd.DataFrame(neutral_text_y)

  # This is fixed.
  EMBEDDING_DIM = 300

  #before tokenizing the data divided whole data into train and test sets
  X_train_pos,X_test_pos,y_train_pos,y_test_pos = train_test_split(positive_text_X,positive_text_y,
test_size=0.2,random_state=42)
  X_train_neg,X_test_neg,y_train_neg,y_test_neg = train_test_split(negative_text_X,negative_text_y,
test_size=0.2,random_state=42)
  X_train_neu,X_test_neu,y_train_neu,y_test_neu = train_test_split(neutral_text_X,neutral_text_y,te
st_size=0.2,random_state=42)

  X_temp = X_train_pos.append(X_train_neg)
  X_train = X_temp.append(X_train_neu)

  # Tokenize: text-to-word-sequence

  # tokenizing and computing no of unique words
  tokenizer = Tokenizer()
  tokenizer.fit_on_texts(X_train['text'].values)

  # computed total vocabualry from 'text' column as 'selected_text' is subset of 'text'
  max_length = max([len(s.split()) for s in X_train['text']])
  vocab_size = len(tokenizer.word_index) +1

  #converting texts to sequences
  X_train_pos_tokens = tokenizer.texts_to_sequences(X_train_pos['text'].values)
  X_train_neg_tokens = tokenizer.texts_to_sequences(X_train_neg['text'].values)
  X_train_neu_tokens = tokenizer.texts_to_sequences(X_train_neu['text'].values)

  X_test_pos_tokens = tokenizer.texts_to_sequences(X_test_pos['text'].values)
  X_test_neg_tokens = tokenizer.texts_to_sequences(X_test_neg['text'].values)
  X_test_neu_tokens = tokenizer.texts_to_sequences(X_test_neu['text'].values)

  y_train_pos_tokens = tokenizer.texts_to_sequences(y_train_pos['selected_text'].values)
  y_train_neg_tokens = tokenizer.texts_to_sequences(y_train_neg['selected_text'].values)
  y_train_neu_tokens = tokenizer.texts_to_sequences(y_train_neu['selected_text'].values)


  y_test_pos_tokens = tokenizer.texts_to_sequences(y_test_pos['selected_text'].values)
  y_test_neg_tokens = tokenizer.texts_to_sequences(y_test_neg['selected_text'].values)
  y_test_neu_tokens = tokenizer.texts_to_sequences(y_test_neu['selected_text'].values)

  #padding zeros at the end
  X_train_pos_pad = pad_sequences(X_train_pos_tokens, maxlen=max_length, padding='post', truncating
='post')
  X_train_neg_pad = pad_sequences(X_train_neg_tokens, maxlen=max_length, padding='post', truncating
='post')
  X_train_neu_pad = pad_sequences(X_train_neu_tokens, maxlen=max_length, padding='post', truncating
='post')

  X_test_pos_pad = pad_sequences(X_test_pos_tokens, maxlen=max_length, padding='post', truncating='
post')
  X_test_neg_pad = pad_sequences(X_test_neg_tokens, maxlen=max_length, padding='post', truncating='
post')
  X_test_neu_pad = pad_sequences(X_test_neu_tokens, maxlen=max_length, padding='post', truncating='
post')

  y_train_pos_pad = pad_sequences(y_train_pos_tokens, maxlen=max_length, padding='post', truncating
='post')
```

```
- post')
  y_train_neg_pad = pad_sequences(y_train_neg_tokens, maxlen=max_length, padding='post', truncating
='post')
  y_train_neu_pad = pad_sequences(y_train_neu_tokens, maxlen=max_length, padding='post', truncating
='post')

  y_test_pos_pad = pad_sequences(y_test_pos_tokens, maxlen=max_length, padding='post', truncating='
post')
  y_test_neg_pad = pad_sequences(y_test_neg_tokens, maxlen=max_length, padding='post', truncating='
post')
  y_test_neu_pad = pad_sequences(y_test_neu_tokens, maxlen=max_length, padding='post', truncating='
post')

  y_train_pos_pad = set_to_one(X_train_pos_pad,y_train_pos_pad)
  y_train_neg_pad = set_to_one(X_train_neg_pad,y_train_neg_pad)
  y_train_neu_pad = set_to_one(X_train_neu_pad,y_train_neu_pad)

  y_test_pos_pad = set_to_one(X_test_pos_pad,y_test_pos_pad)
  y_test_neg_pad = set_to_one(X_test_neg_pad,y_test_neg_pad)
  y_test_neu_pad = set_to_one(X_test_neu_pad,y_test_neu_pad)

  y_train_pos_pad = pd.DataFrame(y_train_pos_pad)
  y_train_neg_pad = pd.DataFrame(y_train_neg_pad)
  y_train_neu_pad = pd.DataFrame(y_train_neu_pad)

  y_test_pos_pad = pd.DataFrame(y_test_pos_pad)
  y_test_neg_pad = pd.DataFrame(y_test_neg_pad)
  y_test_neu_pad = pd.DataFrame(y_test_neu_pad)

  reverse_word_map = dict(map(reversed, tokenizer.word_index.items()))

  embedding_matrix,nb_words = get_embedding_vectors(vocab_size, EMBEDDING_DIM,tokenizer)

  loss_function = tf.keras.losses.BinaryCrossentropy(from_logits=False, reduction='none')

  model = build_model(nb_words,EMBEDDING_DIM, max_length, weights=[embedding_matrix])

  #for positive tweets
  pos_filepath = '/content/drive/My Drive/CS2_pos_final_model/weights_pos_best.h5'
  f1 = h5py.File(pos_filepath, 'r')

  model.load_weights(pos_filepath)

  model.compile(loss = maskedLoss, optimizer='adam', metrics=[km.binary_precision(), km.binary_reca
ll()])
  pos_metrics = model.evaluate(X_test_pos_pad,y_test_pos_pad)

  #for negative tweets
  neg_filepath = '/content/drive/My Drive/CS2_pos_final_model/weights_neg_best.h5'
  f2 = h5py.File(neg_filepath, 'r')

  model.load_weights(neg_filepath)

  model.compile(loss = maskedLoss, optimizer='adam', metrics=[km.binary_precision(), km.binary_reca
ll()])
  neg_metrics = model.evaluate(X_test_neg_pad,y_test_neg_pad)

  #For neutral tweets
  neu_filepath = '/content/drive/My Drive/CS2_pos_final_model/weights_neu_best.h5'
  f3 = h5py.File(neu_filepath, 'r')

  model.load_weights(neu_filepath)

  model.compile(loss = maskedLoss, optimizer='adam', metrics=[km.binary_precision(), km.binary_reca
ll()])
  neu_metrics = model.evaluate(X_test_neu_pad,y_test_neu_pad)

  ndarr_train_pos = model.predict(X_train_pos_pad)
  ndarr_test_pos = model.predict(X_test_pos_pad)

  ndarr_train_neg = model.predict(X_train_neg_pad)
  ndarr_test_neg = model.predict(X_test_neg_pad)

  ndarr_train_neu = model.predict(X_train_neu_pad)
  ndarr_test_neu = model.predict(X_test_neu_pad)

  arr1_train_pos = arr1_train_temp(ndarr_train_pos)
  arr1_test_pos = arr1_train_temp(ndarr_test_pos)
```

```
  arr1_test_pos = arr1_train_temp(ndarr_test_pos)

  arr1_train_neg = arr1_train_temp(ndarr_train_neg)
  arr1_test_neg = arr1_train_temp(ndarr_test_neg)

  arr1_train_neu = arr1_train_temp(ndarr_train_neu)
  arr1_test_neu = arr1_train_temp(ndarr_test_neu)

  arr1_train_pos = arr1_train_pos.reshape(-1,33)
  arr1_test_pos = arr1_test_pos.reshape(-1,33)

  arr1_train_neg = arr1_train_neg.reshape(-1,33)
  arr1_test_neg = arr1_test_neg.reshape(-1,33)

  arr1_train_neu = arr1_train_neu.reshape(-1,33)
  arr1_test_neu = arr1_test_neu.reshape(-1,33)

  y_train_pos_pad_rev = rev(arr1_train_pos,X_train_pos_pad)
  y_test_pos_pad_rev = rev(arr1_test_pos,X_test_pos_pad)

  y_train_neg_pad_rev = rev(arr1_train_neg,X_train_neg_pad)
  y_test_neg_pad_rev = rev(arr1_test_neg,X_test_neg_pad)

  y_train_neu_pad_rev = rev(arr1_train_neu,X_train_neu_pad)
  y_test_neu_pad_rev = rev(arr1_test_neu,X_test_neu_pad)

  my_texts_train_pos = []
  for i in range(len(y_train_pos_pad_rev)):
    words = sequence_to_text(y_train_pos_pad_rev[i],reverse_word_map)
    my_texts_train_pos.append(words)

  my_texts_test_pos = []
  for i in range(len(y_test_pos_pad_rev)):
    words = sequence_to_text(y_test_pos_pad_rev[i],reverse_word_map)
    my_texts_test_pos.append(words)

  my_texts_train_neg = []
  for i in range(len(y_train_neg_pad_rev)):
    words = sequence_to_text(y_train_neg_pad_rev[i],reverse_word_map)
    my_texts_train_neg.append(words)

  my_texts_test_neg = []
  for i in range(len(y_test_neg_pad_rev)):
    words = sequence_to_text(y_test_neg_pad_rev[i],reverse_word_map)
    my_texts_test_neg.append(words)

  my_texts_train_neu = []
  for i in range(len(y_train_neu_pad_rev)):
    words = sequence_to_text(y_train_neu_pad_rev[i],reverse_word_map)
    my_texts_train_neu.append(words)

  my_texts_test_neu = []
  for i in range(len(y_test_neu_pad_rev)):
    words = sequence_to_text(y_test_neu_pad_rev[i],reverse_word_map)
    my_texts_test_neu.append(words)

  predicted_txts_train_pos = pred_txt(my_texts_train_pos)
  predicted_txts_test_pos = pred_txt(my_texts_test_pos)

  predicted_txts_train_neg = pred_txt(my_texts_train_neg)
  predicted_txts_test_neg = pred_txt(my_texts_test_neg)

  predicted_txts_train_neu = pred_txt(my_texts_train_neu)
  predicted_txts_test_neu = pred_txt(my_texts_test_neu)

  jaccard_vals_train_pos = jaccard_score(predicted_txts_train_pos,y_train_pos)
  print('The jaccard score of train positive tweets is:', np.mean(jaccard_vals_train_pos))

  jaccard_vals_train_neg = jaccard_score(predicted_txts_train_neg,y_train_neg)
  print('The jaccard score of train positive tweets is:', np.mean(jaccard_vals_train_neg))

  jaccard_vals_train_neu = jaccard_score(predicted_txts_train_neu,y_train_neu)
  print('The jaccard score of train positive tweets is:', np.mean(jaccard_vals_train_neu))

  return
predicted_txts_train_pos,predicted_txts_test_pos,predicted_txts_train_neg,predicted_txts_test_neg,
predicted_txts_train_neu,predicted_txts_test_neu
```

In [20]:

```
predicted_txts_train_pos,predicted_txts_test_pos,predicted_txts_train_neg,predicted_txts_test_neg,
predicted_txts_train_neu,predicted_txts_test_neu = final_fun_1(df1)
```

```
999995it [01:42, 9712.90it/s]
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras_metrics/metrics.py:51:
calling Layer.add_update (from tensorflow.python.keras.engine.base_layer) with inputs is
deprecated and will be removed in a future version.
Instructions for updating:
`inputs` is now automatically inferred
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras_metrics/metrics.py:26:
Layer.updates (from tensorflow.python.keras.engine.base_layer) is deprecated and will be removed i
n a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatically.
54/54 [==============================] - 1s 18ms/step - loss: 6.9584 - precision: 0.5958 - recall:
0.6404
49/49 [==============================] - 1s 16ms/step - loss: 7.5788 - precision: 0.5699 - recall:
0.4591
70/70 [==============================] - 1s 17ms/step - loss: 1.3836 - precision: 0.9715 - recall:
0.9942
The jaccard score of train positive tweets is: 0.22315528296274018
The jaccard score of train positive tweets is: 0.2348763945020372
The jaccard score of train positive tweets is: 0.5271872832847861
```