BTC

June 9, 2024

[1]:
```
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from wordcloud import WordCloud,STOPWORDS
from bs4 import BeautifulSoup
import re,string,unicodedata
import os
from IPython.display import Image

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score,␣
↪confusion_matrix, RocCurveDisplay, PrecisionRecallDisplay,␣
↪ConfusionMatrixDisplay
#from xgboost.sklearn import XGBClassifier

import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Dense,Input, Embedding,LSTM,Dropout,Conv1D,␣
↪MaxPooling1D,␣
↪GlobalMaxPooling1D,Dropout,Bidirectional,Flatten,BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model
#import transformers
```

```
#import tokenizers
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score
from sklearn.utils.class_weight import compute_class_weight
```

```
2024-06-09 07:34:30.175016: I tensorflow/core/util/port.cc:113] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-06-09 07:34:30.184501: I external/local_tsl/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2024-06-09 07:34:30.608371: I external/local_tsl/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2024-06-09 07:34:32.226716: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
2024-06-09 07:34:34.773379: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

[2]:
```
#loading data
data = pd.read_excel('final_data1.xlsx')
data1= data.copy()
```

[3]:
```
data.head()
```

[3]:
```
   item       date                                               text  number  \
0   1.0 2022-10-03  DMG Blockchain Solutions goes live on the Boso…      24
1   2.0 2022-10-03  NYDIG Promotes Leaders Amidst Record Bitcoin B…      24
2   3.0 2022-10-03  Bitcoin Mining as Bad for Planet as Oil Drilli…      24
3   4.0 2022-10-03  Bitcoin climate impact greater than gold minin…      24
4   5.0 2022-10-03  Bitcoin Is â€˜Comforting' And â€˜Can't Be Stop…      24

          class        price
0  Extreme Fear  19623.58008
1  Extreme Fear  19623.58008
2  Extreme Fear  19623.58008
3  Extreme Fear  19623.58008
4  Extreme Fear  19623.58008
```

[4]:
```
#Fill missing values
data['text'] = data['text'].astype(str).fillna('')
```

[5]:
```
#map the class to numerical value add column sentiment (numerical class) change
↪the number of classes to 3 classes
mapping = {
    'Extreme Fear': 0,
    'Fear': 0,
    'Neutral': 2,
    'Greed': 1,
    'Extreme Greed': 1
}
data['label'] = data['class'].map(mapping)
```

[6]:
```
#remove Neutral class
data= data[data['label'] != 2]
```

[7]:
```
data.head()
```

[7]:
```
   item       date                                               text  number  \
0   1.0 2022-10-03  DMG Blockchain Solutions goes live on the Boso…      24
1   2.0 2022-10-03  NYDIG Promotes Leaders Amidst Record Bitcoin B…      24
2   3.0 2022-10-03  Bitcoin Mining as Bad for Planet as Oil Drilli…      24
3   4.0 2022-10-03  Bitcoin climate impact greater than gold minin…      24
4   5.0 2022-10-03  Bitcoin Is â€˜Comforting' And â€˜Can't Be Stop…      24

          class        price  label
0  Extreme Fear  19623.58008      0
1  Extreme Fear  19623.58008      0
2  Extreme Fear  19623.58008      0
3  Extreme Fear  19623.58008      0
4  Extreme Fear  19623.58008      0
```

[8]:
```
data.shape
```

[8]:
```
(2360, 7)
```

[9]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2360 entries, 0 to 2379
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   item    2281 non-null   float64
 1   date    2359 non-null   datetime64[ns]
 2   text    2360 non-null   object
```

```
 3   number  2360 non-null   int64
 4   class   2360 non-null   object
 5   price   2360 non-null   float64
 6   label   2360 non-null   int64
dtypes: datetime64[ns](1), float64(2), int64(2), object(2)
memory usage: 147.5+ KB
```

[10]:
```
data.describe()
```

[10]:
```
              item                           date       number         price  \
count  2281.000000                           2359  2360.000000   2360.000000
mean   1142.360807  2022-04-28 13:29:25.663416576    34.555508  27379.297608
min       1.000000            2021-01-01 00:00:00    10.000000  19416.568360
25%     571.000000            2021-06-01 12:00:00    23.000000  19623.580080
50%    1141.000000            2022-10-04 00:00:00    24.000000  20160.716800
75%    1711.000000            2022-10-06 00:00:00    26.000000  34616.066410
max    2301.000000            2022-10-09 00:00:00    95.000000  63503.457030
std     660.803311                            NaN    22.612206  12986.845693

             label
count  2360.000000
mean      0.207627
min       0.000000
25%       0.000000
50%       0.000000
75%       0.000000
max       1.000000
std       0.405694
```

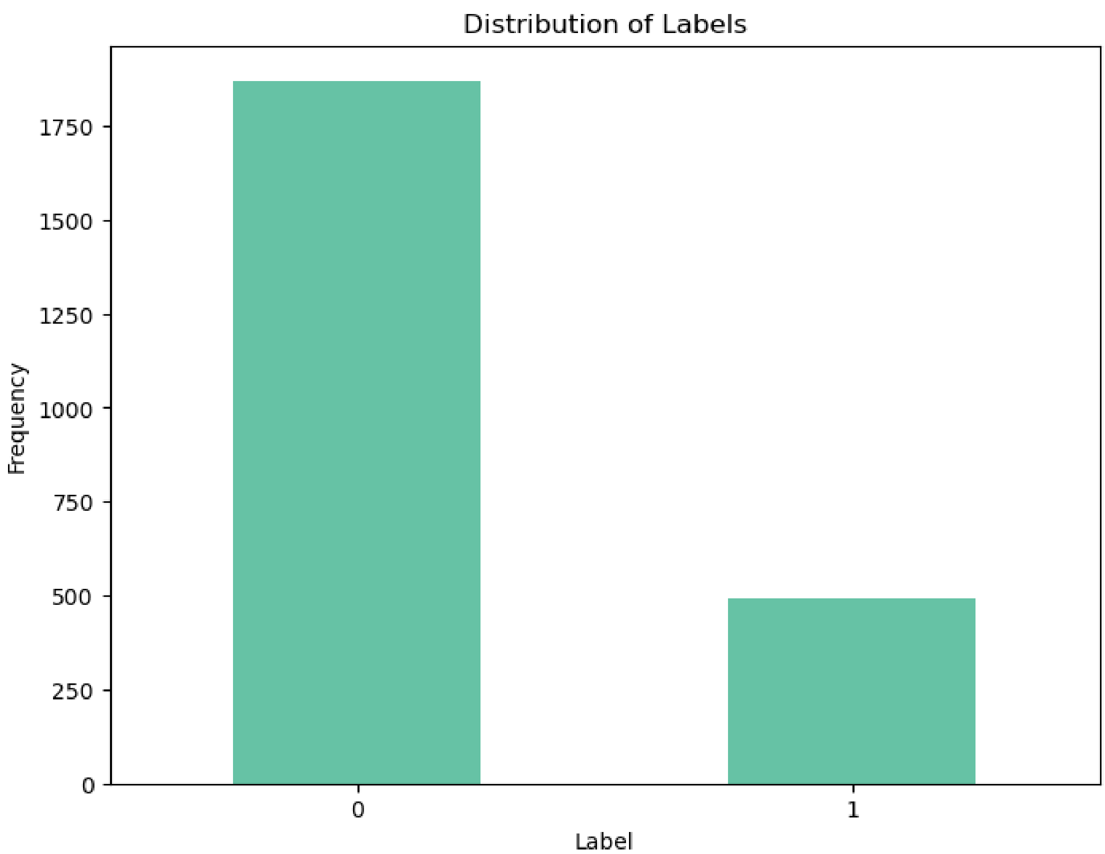
[11]:
```
data['label'].value_counts()
```

[11]:
```
label
0    1870
1     490
Name: count, dtype: int64
```

[12]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
#colors = ['#fc8d62','#66c2a5']
# Mapping labels to categories


plt.figure(figsize=(8, 6))
data['label'].value_counts().plot(kind='bar', color=['#66c2a5'])

label_mapping = {0: 'Fear', 1: 'Greed'}
data['label'] = data['label'].map(label_mapping)
```

```
plt.title('Distribution of Labels')
plt.xlabel('Label')
plt.ylabel('Frequency')
plt.xticks(rotation=0)


# Saving the plot as a PNG file
plt.savefig('label_distribution.png')
plt.show()
```

<Figure size 800x600 with 0 Axes>



[13]:
```
label_mapping = {'Fear':0, 'Greed':1}
data['label'] = data['label'].map(label_mapping)
```

[14]:
```
data.drop_duplicates(inplace = True)
```

[15]:
```
data.shape
```

[15]: (2360, 7)

[16]:
```
stop = stopwords.words('english')
wl = WordNetLemmatizer()
```

[17]:
```
mapping = {"ain't": "is not", "aren't": "are not","can't": "cannot",
           "'cause": "because", "could've": "could have", "couldn't": "could␣
 ↪not",
           "didn't": "did not",  "doesn't": "does not", "don't": "do not",␣
 ↪"hadn't": "had not",
           "hasn't": "has not", "haven't": "have not", "he'd": "he␣
 ↪would","he'll": "he will",
           "he's": "he is", "how'd": "how did", "how'd'y": "how do you",␣
 ↪"how'll": "how will",
           "how's": "how is",  "I'd": "I would", "I'd've": "I would have",␣
 ↪"I'll": "I will",
           "I'll've": "I will have","I'm": "I am", "I've": "I have", "i'd": "i␣
 ↪would",
           "i'd've": "i would have", "i'll": "i will",  "i'll've": "i will␣
 ↪have",
           "i'm": "i am", "i've": "i have", "isn't": "is not", "it'd": "it␣
 ↪would",
           "it'd've": "it would have", "it'll": "it will", "it'll've": "it will␣
 ↪have",
           "it's": "it is", "let's": "let us", "ma'am": "madam", "mayn't": "may␣
 ↪not",
           "might've": "might have","mightn't": "might not","mightn't've":␣
 ↪"might not have",
           "must've": "must have", "mustn't": "must not", "mustn't've": "must␣
 ↪not have",
           "needn't": "need not", "needn't've": "need not have","o'clock": "of␣
 ↪the clock",
           "oughtn't": "ought not", "oughtn't've": "ought not have", "shan't":␣
 ↪"shall not",
           "sha'n't": "shall not", "shan't've": "shall not have", "she'd": "she␣
 ↪would",
           "she'd've": "she would have", "she'll": "she will", "she'll've":␣
 ↪"she will have",
           "she's": "she is", "should've": "should have", "shouldn't": "should␣
 ↪not",
           "shouldn't've": "should not have", "so've": "so have","so's": "so␣
 ↪as", "this's": "this is",
           "that'd": "that would", "that'd've": "that would have", "that's":␣
 ↪"that is",
           "there'd": "there would", "there'd've": "there would have",␣
 ↪"there's": "there is",
```

```
           "here's": "here is","they'd": "they would", "they'd've": "they would␣
↪have",
           "they'll": "they will", "they'll've": "they will have", "they're":␣
↪"they are",
           "they've": "they have", "to've": "to have", "wasn't": "was not",␣
↪"we'd": "we would",
           "we'd've": "we would have", "we'll": "we will", "we'll've": "we will␣
↪have",
           "we're": "we are", "we've": "we have", "weren't": "were not",
           "what'll": "what will", "what'll've": "what will have","what're":␣
↪"what are",
           "what's": "what is", "what've": "what have", "when's": "when is",␣
↪"when've": "when have",
           "where'd": "where did", "where's": "where is", "where've": "where␣
↪have", "who'll": "who will",
           "who'll've": "who will have", "who's": "who is", "who've": "who␣
↪have", "why's": "why is",
           "why've": "why have", "will've": "will have", "won't": "will not",␣
↪"won't've": "will not have",
           "would've": "would have", "wouldn't": "would not", "wouldn't've":␣
↪"would not have",
           "y'all": "you all", "y'all'd": "you all would","y'all'd've": "you␣
↪all would have",
           "y'all're": "you all are","y'all've": "you all have","you'd": "you␣
↪would",
           "you'd've": "you would have", "you'll": "you will", "you'll've":␣
↪"you will have",
           "you're": "you are", "you've": "you have" }
```

[18]:
```
import nltk
nltk.download('wordnet')
def clean_text(text,lemmatize = True):
    soup = BeautifulSoup(text, "html.parser") #remove html tags
    text = soup.get_text()
    text = ' '.join([mapping[t] if t in mapping else t for t in text.split("␣
↪")]) #expanding chatwords and contracts clearing contractions
    emoji_clean= 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)
    text = emoji_clean.sub(r'',text)
    text = re.sub(r'\.(?=\S)', '. ',text) #add space after full stop
```

```
    text = re.sub(r'http\S+', '', text) #remove urls
    text = "".join([word.lower() for word in text if word not in string.
 ↪punctuation]) #remove punctuation
    #tokens = re.split('\W+', text) #create tokens
    if lemmatize:
        text = " ".join([wl.lemmatize(word) for word in text.split() if word␣
 ↪not in stop and word.isalpha()]) #lemmatize
    else:
        text = " ".join([word for word in text.split() if word not in stop and␣
 ↪word.isalpha()])
    return text
```

```
[nltk_data] Downloading package wordnet to /home/bitta693/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[19]:

```
data['text']=data['text'].apply(clean_text,lemmatize = True)
```

```
/tmp/ipykernel_147575/3037743114.py:4: MarkupResemblesLocatorWarning: The input
looks more like a filename than markup. You may want to open this file and pass
the filehandle into Beautiful Soup.
  soup = BeautifulSoup(text, "html.parser") #remove html tags
```

[20]:

```
#splitting into train and test
train, test= train_test_split(data, test_size=0.2, random_state=42)

#train dataset
Xtrain, ytrain = train['text'], train['label']

#test dataset
Xtest, ytest = test['text'], test['label']

print(Xtrain.shape,ytrain.shape)
print(Xtest.shape,ytest.shape)
```

```
(1888,) (1888,)
(472,) (472,)
```

[21]:

```
import numpy as np
from sklearn.utils import class_weight
classes = np.unique(ytrain)
class_weights = class_weight.compute_class_weight(
    'balanced',
    classes=classes,
    y=ytrain
)
class_weights_dict = dict(zip(classes, class_weights))
print("Class weights:", class_weights)
```

```
Class weights: [0.62891406 2.43927649]
```

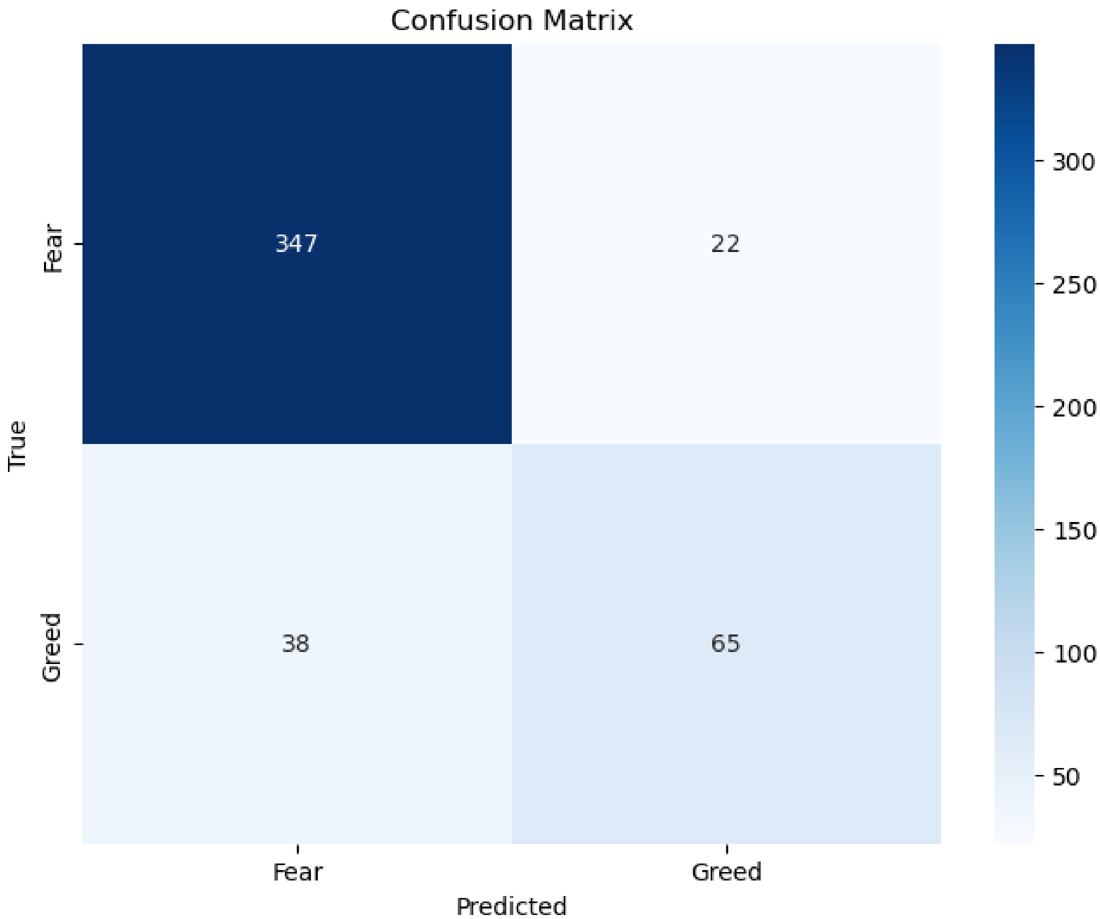
[22]:
```python
pipe = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('classifier', LogisticRegression(solver='liblinear',␣
 ↪class_weight=class_weights_dict))
])

# Train the model
pipe.fit(Xtrain, ytrain)

# Test the model
predictions = pipe.predict(Xtest)

# Evaluate the model
print("Accuracy:", accuracy_score(ytest, predictions))
print("Classification Report:")
print(classification_report(ytest, predictions))
```

```
Accuracy: 0.8728813559322034
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.94      0.92       369
           1       0.75      0.63      0.68       103

    accuracy                           0.87       472
   macro avg       0.82      0.79      0.80       472
weighted avg       0.87      0.87      0.87       472
```

[23]:
```python
# Confusion Matrix
cm = confusion_matrix(ytest, predictions)



print("Confusion Matrix:")
print(cm)

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
class_names = ['Fear', 'Greed']
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names,␣
 ↪yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.savefig('logismatrix.png')
```

```
plt.show()
```

```
Confusion Matrix:
[[347  22]
 [ 38  65]]
```



[24]:
```
vect = TfidfVectorizer()
Xtrain_vect= vect.fit_transform(Xtrain)
Xtest_vect = vect.transform(Xtest)


count_vect = CountVectorizer()
Xtrain_count = count_vect.fit_transform(Xtrain)
Xtest_count = count_vect.transform(Xtest)
```

[25]:
```
MAX_VOCAB_SIZE = 10000
tokenizer = Tokenizer(num_words = MAX_VOCAB_SIZE,oov_token="<oov>")
tokenizer.fit_on_texts(Xtrain)
```

```
word_index = tokenizer.word_index
#print(word_index)
V = len(word_index)
print("Vocabulary of the dataset is : ",V)
```

```
Vocabulary of the dataset is :  4249
```

[26]:
```
##create sequences of reviews
seq_train = tokenizer.texts_to_sequences(Xtrain)
seq_test =  tokenizer.texts_to_sequences(Xtest)
```

[27]:
```
#choice of maximum length of sequences
seq_len_list = [len(i) for i in seq_train + seq_test]

#if we take the direct maximum then
max_len=max(seq_len_list)
print('Maximum length of sequence in the list: {}'.format(max_len))
```

```
Maximum length of sequence in the list: 25
```

[28]:
```
# when setting the maximum length of sequence, variability around the average␣
 ↪is used.
max_seq_len = np.mean(seq_len_list) + 2 * np.std(seq_len_list)
max_seq_len = int(max_seq_len)
print('Maximum length of the sequence when considering data only two standard␣
 ↪deviations from average: {}'.format(max_seq_len))
```

```
Maximum length of the sequence when considering data only two standard
deviations from average: 16
```

[29]:
```
perc_covered = np.sum(np.array(seq_len_list) < max_seq_len) /␣
 ↪len(seq_len_list)*100
print('The above calculated number coveres approximately {} % of data'.
 ↪format(np.round(perc_covered,2)))
```

```
The above calculated number coveres approximately 92.33 % of data
```

[30]:
```
#create padded sequences
pad_train=pad_sequences(seq_train,truncating = 'post', padding =␣
 ↪'pre',maxlen=max_seq_len)
pad_test=pad_sequences(seq_test,truncating = 'post', padding =␣
 ↪'pre',maxlen=max_seq_len)
```

[31]:
```
def plotLearningCurve(history,epochs,name):

    epochRange = range(1,epochs+1)
    fig , ax = plt.subplots(1,2,figsize = (10,5))
```

```
    ax[0].plot(epochRange,history.history['accuracy'],label = 'Training␣
↪Accuracy')
    ax[0].plot(epochRange,history.history['val_accuracy'],label = 'Validation␣
↪Accuracy')
    ax[0].set_title('Training and Validation accuracy')
    ax[0].set_xlabel('Epoch')
    ax[0].set_ylabel('Accuracy')
    ax[0].legend()
    ax[1].plot(epochRange,history.history['loss'],label = 'Training Loss')
    ax[1].plot(epochRange,history.history['val_loss'],label = 'Validation Loss')
    ax[1].set_title('Training and Validation loss')
    ax[1].set_xlabel('Epoch')
    ax[1].set_ylabel('Loss')
    ax[1].legend()
    fig.tight_layout()
    plt.savefig(name)
    plt.show()
```

[32]:
```
#Splitting training set for validation purposes
Xtrain,Xval,ytrain,yval=train_test_split(pad_train,ytrain,
                                         test_size=0.2,random_state=10)
```

[33]:
```
from tensorflow.keras.layers import Input, Embedding, BatchNormalization,␣
↪Dropout, Conv1D, MaxPooling1D, Bidirectional, LSTM, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

def lstm_model(Xtrain, Xval, ytrain, yval, V, D, maxlen, epochs):
    print("----Building the model----")
    i = Input(shape=(maxlen,))
    x = Embedding(V + 1, D)(i)
    x = BatchNormalization()(x)
    x = Dropout(0.3)(x)
    x = Conv1D(32, 5, activation='relu')(x)
    x = Dropout(0.3)(x)
    x = MaxPooling1D(2)(x)
    x = Bidirectional(LSTM(128, return_sequences=True))(x)
    x = LSTM(64)(x)
    x = Dropout(0.5)(x)
    x = Dense(1, activation='sigmoid')(x)
    model = Model(i, x)
    model.summary()
```

```
    ytrain = np.array(ytrain).reshape(-1, 1)
    yval = np.array(yval).reshape(-1, 1)


    classes, counts = np.unique(ytrain, return_counts=True)
    print(f"Classes: {classes}")
    print(f"Counts: {counts}")


    class_weights = compute_class_weight(class_weight='balanced',␣
↪classes=classes, y=ytrain.ravel())
    class_weights = dict(zip(classes, class_weights))
    print(f"Class weights: {class_weights}")


    print("----Training the network----")
    model.compile(optimizer=Adam(0.000007),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    r = model.fit(Xtrain, ytrain,
                  validation_data=(Xval, yval),
                  epochs=epochs,
                  verbose=2,
                  batch_size=32,
                  class_weight=class_weights)



    train_score = model.evaluate(Xtrain, ytrain, verbose=0)
    val_score = model.evaluate(Xval, yval, verbose=0)

    print(f"Train score: {train_score}")
    print(f"Validation score: {val_score}")

    n_epochs = len(r.history['loss'])

    return r, model, n_epochs
```

[34]:
```
D = 64 #embedding dims
epochs = 140
r,model,n_epochs = lstm_model(Xtrain,Xval,ytrain,yval,V,D,max_seq_len,epochs)
```

```
----Building the model----

2024-06-09 07:34:39.962309: E
external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:282] failed call to
cuInit: CUDA_ERROR_NO_DEVICE: no CUDA-capable device is detected
```

```
Model: "functional_1"

 Layer (type)                         Output Shape                    Param #

 input_layer (InputLayer)             (None, 16)                            0

 embedding (Embedding)                (None, 16, 64)                  272,000

 batch_normalization                  (None, 16, 64)                      256
 (BatchNormalization)

 dropout (Dropout)                    (None, 16, 64)                        0

 conv1d (Conv1D)                      (None, 12, 32)                   10,272

 dropout_1 (Dropout)                  (None, 12, 32)                        0

 max_pooling1d (MaxPooling1D)         (None, 6, 32)                         0

 bidirectional (Bidirectional)        (None, 6, 256)                  164,864

 lstm_1 (LSTM)                        (None, 64)                       82,176

 dropout_2 (Dropout)                  (None, 64)                            0

 dense (Dense)                        (None, 1)                            65


 Total params: 529,633 (2.02 MB)

 Trainable params: 529,505 (2.02 MB)

 Non-trainable params: 128 (512.00 B)

Classes: [0 1]
Counts: [1197  313]
Class weights: {0: 0.6307435254803676, 1: 2.412140575079872}
----Training the network----
Epoch 1/140
48/48 - 5s - 114ms/step - accuracy: 0.5669 - loss: 0.6860 - val_accuracy: 0.8280
- val_loss: 0.6915
Epoch 2/140
48/48 - 1s - 14ms/step - accuracy: 0.5868 - loss: 0.6767 - val_accuracy: 0.8228
- val_loss: 0.6907
```

```
Epoch 3/140
48/48 - 1s - 13ms/step - accuracy: 0.6285 - loss: 0.6645 - val_accuracy: 0.8016
- val_loss: 0.6896
Epoch 4/140
48/48 - 1s - 13ms/step - accuracy: 0.6470 - loss: 0.6524 - val_accuracy: 0.7778
- val_loss: 0.6881
Epoch 5/140
48/48 - 1s - 13ms/step - accuracy: 0.6788 - loss: 0.6437 - val_accuracy: 0.7804
- val_loss: 0.6855
Epoch 6/140
48/48 - 1s - 13ms/step - accuracy: 0.6987 - loss: 0.6307 - val_accuracy: 0.7566
- val_loss: 0.6827
Epoch 7/140
48/48 - 1s - 12ms/step - accuracy: 0.6967 - loss: 0.6224 - val_accuracy: 0.7540
- val_loss: 0.6774
Epoch 8/140
48/48 - 1s - 12ms/step - accuracy: 0.7179 - loss: 0.6117 - val_accuracy: 0.7513
- val_loss: 0.6705
Epoch 9/140
48/48 - 1s - 13ms/step - accuracy: 0.7199 - loss: 0.5969 - val_accuracy: 0.7487
- val_loss: 0.6621
Epoch 10/140
48/48 - 1s - 13ms/step - accuracy: 0.7265 - loss: 0.5911 - val_accuracy: 0.7513
- val_loss: 0.6500
Epoch 11/140
48/48 - 1s - 13ms/step - accuracy: 0.7384 - loss: 0.5746 - val_accuracy: 0.7646
- val_loss: 0.6343
Epoch 12/140
48/48 - 1s - 12ms/step - accuracy: 0.7563 - loss: 0.5563 - val_accuracy: 0.7672
- val_loss: 0.6160
Epoch 13/140
48/48 - 1s - 13ms/step - accuracy: 0.7748 - loss: 0.5447 - val_accuracy: 0.7725
- val_loss: 0.5971
Epoch 14/140
48/48 - 1s - 13ms/step - accuracy: 0.7709 - loss: 0.5316 - val_accuracy: 0.7751
- val_loss: 0.5769
Epoch 15/140
48/48 - 1s - 12ms/step - accuracy: 0.7781 - loss: 0.5232 - val_accuracy: 0.7751
- val_loss: 0.5603
Epoch 16/140
48/48 - 1s - 12ms/step - accuracy: 0.7828 - loss: 0.5087 - val_accuracy: 0.7778
- val_loss: 0.5451
Epoch 17/140
48/48 - 1s - 12ms/step - accuracy: 0.7947 - loss: 0.4954 - val_accuracy: 0.7778
- val_loss: 0.5291
Epoch 18/140
48/48 - 1s - 13ms/step - accuracy: 0.7967 - loss: 0.4862 - val_accuracy: 0.7910
- val_loss: 0.5151
```

```
Epoch 19/140
48/48 - 1s - 13ms/step - accuracy: 0.7987 - loss: 0.4866 - val_accuracy: 0.7937
- val_loss: 0.5070
Epoch 20/140
48/48 - 1s - 13ms/step - accuracy: 0.7934 - loss: 0.4763 - val_accuracy: 0.8069
- val_loss: 0.4977
Epoch 21/140
48/48 - 1s - 13ms/step - accuracy: 0.8040 - loss: 0.4703 - val_accuracy: 0.8042
- val_loss: 0.4929
Epoch 22/140
48/48 - 1s - 12ms/step - accuracy: 0.7974 - loss: 0.4621 - val_accuracy: 0.8069
- val_loss: 0.4868
Epoch 23/140
48/48 - 1s - 12ms/step - accuracy: 0.7980 - loss: 0.4567 - val_accuracy: 0.8069
- val_loss: 0.4835
Epoch 24/140
48/48 - 1s - 12ms/step - accuracy: 0.8033 - loss: 0.4595 - val_accuracy: 0.8069
- val_loss: 0.4802
Epoch 25/140
48/48 - 1s - 13ms/step - accuracy: 0.8046 - loss: 0.4492 - val_accuracy: 0.8122
- val_loss: 0.4766
Epoch 26/140
48/48 - 1s - 12ms/step - accuracy: 0.8073 - loss: 0.4420 - val_accuracy: 0.8122
- val_loss: 0.4718
Epoch 27/140
48/48 - 1s - 13ms/step - accuracy: 0.8113 - loss: 0.4386 - val_accuracy: 0.8148
- val_loss: 0.4677
Epoch 28/140
48/48 - 1s - 13ms/step - accuracy: 0.8079 - loss: 0.4378 - val_accuracy: 0.8175
- val_loss: 0.4631
Epoch 29/140
48/48 - 1s - 13ms/step - accuracy: 0.8079 - loss: 0.4369 - val_accuracy: 0.8175
- val_loss: 0.4619
Epoch 30/140
48/48 - 1s - 13ms/step - accuracy: 0.8119 - loss: 0.4311 - val_accuracy: 0.8148
- val_loss: 0.4642
Epoch 31/140
48/48 - 1s - 12ms/step - accuracy: 0.8119 - loss: 0.4261 - val_accuracy: 0.8122
- val_loss: 0.4636
Epoch 32/140
48/48 - 1s - 13ms/step - accuracy: 0.8073 - loss: 0.4317 - val_accuracy: 0.8122
- val_loss: 0.4623
Epoch 33/140
48/48 - 1s - 13ms/step - accuracy: 0.8172 - loss: 0.4221 - val_accuracy: 0.8122
- val_loss: 0.4586
Epoch 34/140
48/48 - 1s - 13ms/step - accuracy: 0.8139 - loss: 0.4253 - val_accuracy: 0.8175
- val_loss: 0.4525
```

```
Epoch 35/140
48/48 - 1s - 13ms/step - accuracy: 0.8093 - loss: 0.4248 - val_accuracy: 0.8148
- val_loss: 0.4557
Epoch 36/140
48/48 - 1s - 13ms/step - accuracy: 0.8172 - loss: 0.4259 - val_accuracy: 0.8148
- val_loss: 0.4538
Epoch 37/140
48/48 - 1s - 13ms/step - accuracy: 0.8113 - loss: 0.4131 - val_accuracy: 0.8095
- val_loss: 0.4553
Epoch 38/140
48/48 - 1s - 12ms/step - accuracy: 0.8132 - loss: 0.4252 - val_accuracy: 0.8095
- val_loss: 0.4529
Epoch 39/140
48/48 - 1s - 13ms/step - accuracy: 0.8205 - loss: 0.4126 - val_accuracy: 0.8122
- val_loss: 0.4495
Epoch 40/140
48/48 - 1s - 12ms/step - accuracy: 0.8172 - loss: 0.4092 - val_accuracy: 0.8095
- val_loss: 0.4520
Epoch 41/140
48/48 - 1s - 13ms/step - accuracy: 0.8232 - loss: 0.4122 - val_accuracy: 0.8122
- val_loss: 0.4478
Epoch 42/140
48/48 - 1s - 12ms/step - accuracy: 0.8212 - loss: 0.4101 - val_accuracy: 0.8122
- val_loss: 0.4453
Epoch 43/140
48/48 - 1s - 12ms/step - accuracy: 0.8152 - loss: 0.4167 - val_accuracy: 0.8122
- val_loss: 0.4439
Epoch 44/140
48/48 - 1s - 12ms/step - accuracy: 0.8185 - loss: 0.4006 - val_accuracy: 0.8122
- val_loss: 0.4438
Epoch 45/140
48/48 - 1s - 12ms/step - accuracy: 0.8179 - loss: 0.4073 - val_accuracy: 0.8095
- val_loss: 0.4458
Epoch 46/140
48/48 - 1s - 12ms/step - accuracy: 0.8199 - loss: 0.4023 - val_accuracy: 0.8042
- val_loss: 0.4473
Epoch 47/140
48/48 - 1s - 12ms/step - accuracy: 0.8205 - loss: 0.3994 - val_accuracy: 0.8069
- val_loss: 0.4440
Epoch 48/140
48/48 - 1s - 13ms/step - accuracy: 0.8272 - loss: 0.4033 - val_accuracy: 0.8069
- val_loss: 0.4434
Epoch 49/140
48/48 - 1s - 12ms/step - accuracy: 0.8291 - loss: 0.4034 - val_accuracy: 0.8042
- val_loss: 0.4456
Epoch 50/140
48/48 - 1s - 12ms/step - accuracy: 0.8245 - loss: 0.3883 - val_accuracy: 0.8122
- val_loss: 0.4427
```

```
Epoch 51/140
48/48 - 1s - 12ms/step - accuracy: 0.8265 - loss: 0.3929 - val_accuracy: 0.8122
- val_loss: 0.4410
Epoch 52/140
48/48 - 1s - 12ms/step - accuracy: 0.8185 - loss: 0.3931 - val_accuracy: 0.8069
- val_loss: 0.4405
Epoch 53/140
48/48 - 1s - 12ms/step - accuracy: 0.8291 - loss: 0.3951 - val_accuracy: 0.8069
- val_loss: 0.4391
Epoch 54/140
48/48 - 1s - 12ms/step - accuracy: 0.8272 - loss: 0.3873 - val_accuracy: 0.8095
- val_loss: 0.4384
Epoch 55/140
48/48 - 1s - 13ms/step - accuracy: 0.8305 - loss: 0.3947 - val_accuracy: 0.8148
- val_loss: 0.4356
Epoch 56/140
48/48 - 1s - 12ms/step - accuracy: 0.8298 - loss: 0.3915 - val_accuracy: 0.8175
- val_loss: 0.4325
Epoch 57/140
48/48 - 1s - 12ms/step - accuracy: 0.8358 - loss: 0.3872 - val_accuracy: 0.8175
- val_loss: 0.4323
Epoch 58/140
48/48 - 1s - 13ms/step - accuracy: 0.8344 - loss: 0.3862 - val_accuracy: 0.8148
- val_loss: 0.4322
Epoch 59/140
48/48 - 1s - 13ms/step - accuracy: 0.8430 - loss: 0.3795 - val_accuracy: 0.8175
- val_loss: 0.4263
Epoch 60/140
48/48 - 1s - 13ms/step - accuracy: 0.8364 - loss: 0.3728 - val_accuracy: 0.8201
- val_loss: 0.4268
Epoch 61/140
48/48 - 1s - 12ms/step - accuracy: 0.8411 - loss: 0.3795 - val_accuracy: 0.8095
- val_loss: 0.4309
Epoch 62/140
48/48 - 1s - 12ms/step - accuracy: 0.8404 - loss: 0.3621 - val_accuracy: 0.8175
- val_loss: 0.4281
Epoch 63/140
48/48 - 1s - 12ms/step - accuracy: 0.8377 - loss: 0.3675 - val_accuracy: 0.8201
- val_loss: 0.4261
Epoch 64/140
48/48 - 1s - 12ms/step - accuracy: 0.8497 - loss: 0.3501 - val_accuracy: 0.8333
- val_loss: 0.4189
Epoch 65/140
48/48 - 1s - 12ms/step - accuracy: 0.8503 - loss: 0.3539 - val_accuracy: 0.8307
- val_loss: 0.4193
Epoch 66/140
48/48 - 1s - 12ms/step - accuracy: 0.8589 - loss: 0.3511 - val_accuracy: 0.8360
- val_loss: 0.4172
```

```
Epoch 67/140
48/48 - 1s - 12ms/step - accuracy: 0.8470 - loss: 0.3547 - val_accuracy: 0.8360
- val_loss: 0.4181
Epoch 68/140
48/48 - 1s - 12ms/step - accuracy: 0.8563 - loss: 0.3509 - val_accuracy: 0.8360
- val_loss: 0.4165
Epoch 69/140
48/48 - 1s - 12ms/step - accuracy: 0.8543 - loss: 0.3479 - val_accuracy: 0.8307
- val_loss: 0.4209
Epoch 70/140
48/48 - 1s - 12ms/step - accuracy: 0.8563 - loss: 0.3435 - val_accuracy: 0.8280
- val_loss: 0.4213
Epoch 71/140
48/48 - 1s - 12ms/step - accuracy: 0.8616 - loss: 0.3382 - val_accuracy: 0.8360
- val_loss: 0.4167
Epoch 72/140
48/48 - 1s - 13ms/step - accuracy: 0.8570 - loss: 0.3395 - val_accuracy: 0.8333
- val_loss: 0.4166
Epoch 73/140
48/48 - 1s - 12ms/step - accuracy: 0.8583 - loss: 0.3315 - val_accuracy: 0.8386
- val_loss: 0.4121
Epoch 74/140
48/48 - 1s - 13ms/step - accuracy: 0.8629 - loss: 0.3297 - val_accuracy: 0.8386
- val_loss: 0.4094
Epoch 75/140
48/48 - 1s - 13ms/step - accuracy: 0.8623 - loss: 0.3326 - val_accuracy: 0.8386
- val_loss: 0.4119
Epoch 76/140
48/48 - 1s - 12ms/step - accuracy: 0.8695 - loss: 0.3191 - val_accuracy: 0.8386
- val_loss: 0.4095
Epoch 77/140
48/48 - 1s - 13ms/step - accuracy: 0.8702 - loss: 0.3196 - val_accuracy: 0.8386
- val_loss: 0.4094
Epoch 78/140
48/48 - 1s - 13ms/step - accuracy: 0.8728 - loss: 0.3158 - val_accuracy: 0.8413
- val_loss: 0.4074
Epoch 79/140
48/48 - 1s - 12ms/step - accuracy: 0.8762 - loss: 0.3190 - val_accuracy: 0.8439
- val_loss: 0.4070
Epoch 80/140
48/48 - 1s - 13ms/step - accuracy: 0.8656 - loss: 0.3156 - val_accuracy: 0.8439
- val_loss: 0.4065
Epoch 81/140
48/48 - 1s - 13ms/step - accuracy: 0.8834 - loss: 0.3038 - val_accuracy: 0.8439
- val_loss: 0.4019
Epoch 82/140
48/48 - 1s - 13ms/step - accuracy: 0.8781 - loss: 0.3043 - val_accuracy: 0.8413
- val_loss: 0.4009
```

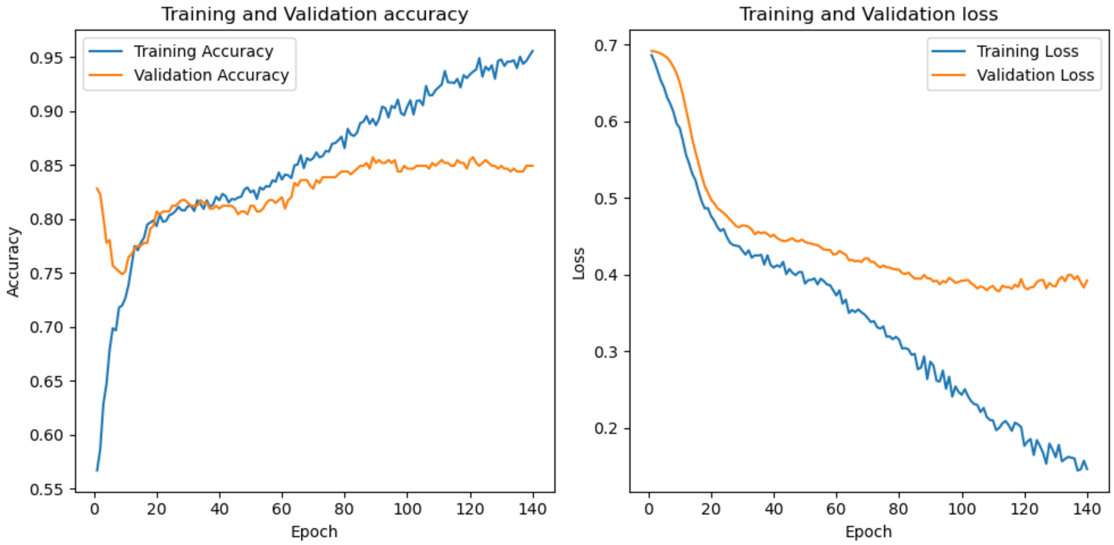
```
Epoch 83/140
48/48 - 1s - 12ms/step - accuracy: 0.8768 - loss: 0.3027 - val_accuracy: 0.8439
- val_loss: 0.4026
Epoch 84/140
48/48 - 1s - 12ms/step - accuracy: 0.8801 - loss: 0.2955 - val_accuracy: 0.8466
- val_loss: 0.3984
Epoch 85/140
48/48 - 1s - 13ms/step - accuracy: 0.8887 - loss: 0.2967 - val_accuracy: 0.8492
- val_loss: 0.3950
Epoch 86/140
48/48 - 1s - 12ms/step - accuracy: 0.8901 - loss: 0.2768 - val_accuracy: 0.8492
- val_loss: 0.3949
Epoch 87/140
48/48 - 1s - 12ms/step - accuracy: 0.8954 - loss: 0.2788 - val_accuracy: 0.8519
- val_loss: 0.3947
Epoch 88/140
48/48 - 1s - 13ms/step - accuracy: 0.8881 - loss: 0.2935 - val_accuracy: 0.8466
- val_loss: 0.3996
Epoch 89/140
48/48 - 1s - 12ms/step - accuracy: 0.8927 - loss: 0.2640 - val_accuracy: 0.8571
- val_loss: 0.3953
Epoch 90/140
48/48 - 1s - 13ms/step - accuracy: 0.8868 - loss: 0.2866 - val_accuracy: 0.8519
- val_loss: 0.3950
Epoch 91/140
48/48 - 1s - 13ms/step - accuracy: 0.8927 - loss: 0.2813 - val_accuracy: 0.8545
- val_loss: 0.3909
Epoch 92/140
48/48 - 1s - 13ms/step - accuracy: 0.9040 - loss: 0.2617 - val_accuracy: 0.8519
- val_loss: 0.3923
Epoch 93/140
48/48 - 1s - 12ms/step - accuracy: 0.9026 - loss: 0.2608 - val_accuracy: 0.8519
- val_loss: 0.3874
Epoch 94/140
48/48 - 1s - 12ms/step - accuracy: 0.8940 - loss: 0.2750 - val_accuracy: 0.8545
- val_loss: 0.3916
Epoch 95/140
48/48 - 1s - 12ms/step - accuracy: 0.9046 - loss: 0.2513 - val_accuracy: 0.8519
- val_loss: 0.3898
Epoch 96/140
48/48 - 1s - 12ms/step - accuracy: 0.9026 - loss: 0.2670 - val_accuracy: 0.8545
- val_loss: 0.3958
Epoch 97/140
48/48 - 1s - 13ms/step - accuracy: 0.9106 - loss: 0.2410 - val_accuracy: 0.8439
- val_loss: 0.3929
Epoch 98/140
48/48 - 1s - 12ms/step - accuracy: 0.8980 - loss: 0.2543 - val_accuracy: 0.8439
- val_loss: 0.3893
```

```
Epoch 99/140
48/48 - 1s - 13ms/step - accuracy: 0.8960 - loss: 0.2473 - val_accuracy: 0.8492
- val_loss: 0.3904
Epoch 100/140
48/48 - 1s - 12ms/step - accuracy: 0.9040 - loss: 0.2434 - val_accuracy: 0.8466
- val_loss: 0.3922
Epoch 101/140
48/48 - 1s - 12ms/step - accuracy: 0.9099 - loss: 0.2507 - val_accuracy: 0.8466
- val_loss: 0.3922
Epoch 102/140
48/48 - 1s - 12ms/step - accuracy: 0.8967 - loss: 0.2418 - val_accuracy: 0.8466
- val_loss: 0.3932
Epoch 103/140
48/48 - 1s - 12ms/step - accuracy: 0.9093 - loss: 0.2352 - val_accuracy: 0.8492
- val_loss: 0.3887
Epoch 104/140
48/48 - 1s - 14ms/step - accuracy: 0.9099 - loss: 0.2313 - val_accuracy: 0.8492
- val_loss: 0.3860
Epoch 105/140
48/48 - 1s - 12ms/step - accuracy: 0.9053 - loss: 0.2302 - val_accuracy: 0.8492
- val_loss: 0.3820
Epoch 106/140
48/48 - 1s - 13ms/step - accuracy: 0.9232 - loss: 0.2209 - val_accuracy: 0.8492
- val_loss: 0.3847
Epoch 107/140
48/48 - 1s - 12ms/step - accuracy: 0.9146 - loss: 0.2263 - val_accuracy: 0.8466
- val_loss: 0.3832
Epoch 108/140
48/48 - 1s - 12ms/step - accuracy: 0.9146 - loss: 0.2146 - val_accuracy: 0.8519
- val_loss: 0.3797
Epoch 109/140
48/48 - 1s - 12ms/step - accuracy: 0.9192 - loss: 0.2107 - val_accuracy: 0.8492
- val_loss: 0.3834
Epoch 110/140
48/48 - 1s - 12ms/step - accuracy: 0.9219 - loss: 0.2100 - val_accuracy: 0.8519
- val_loss: 0.3851
Epoch 111/140
48/48 - 1s - 12ms/step - accuracy: 0.9245 - loss: 0.1974 - val_accuracy: 0.8545
- val_loss: 0.3798
Epoch 112/140
48/48 - 1s - 12ms/step - accuracy: 0.9371 - loss: 0.2003 - val_accuracy: 0.8519
- val_loss: 0.3784
Epoch 113/140
48/48 - 1s - 12ms/step - accuracy: 0.9265 - loss: 0.2063 - val_accuracy: 0.8519
- val_loss: 0.3855
Epoch 114/140
48/48 - 1s - 14ms/step - accuracy: 0.9265 - loss: 0.2094 - val_accuracy: 0.8492
- val_loss: 0.3835
```

```
Epoch 115/140
48/48 - 1s - 13ms/step - accuracy: 0.9258 - loss: 0.2039 - val_accuracy: 0.8492
- val_loss: 0.3840
Epoch 116/140
48/48 - 1s - 12ms/step - accuracy: 0.9298 - loss: 0.1965 - val_accuracy: 0.8545
- val_loss: 0.3814
Epoch 117/140
48/48 - 1s - 13ms/step - accuracy: 0.9219 - loss: 0.2072 - val_accuracy: 0.8519
- val_loss: 0.3870
Epoch 118/140
48/48 - 1s - 13ms/step - accuracy: 0.9331 - loss: 0.2048 - val_accuracy: 0.8519
- val_loss: 0.3839
Epoch 119/140
48/48 - 1s - 12ms/step - accuracy: 0.9305 - loss: 0.2013 - val_accuracy: 0.8466
- val_loss: 0.3943
Epoch 120/140
48/48 - 1s - 12ms/step - accuracy: 0.9338 - loss: 0.1770 - val_accuracy: 0.8545
- val_loss: 0.3836
Epoch 121/140
48/48 - 1s - 12ms/step - accuracy: 0.9364 - loss: 0.1826 - val_accuracy: 0.8571
- val_loss: 0.3808
Epoch 122/140
48/48 - 1s - 14ms/step - accuracy: 0.9384 - loss: 0.1858 - val_accuracy: 0.8519
- val_loss: 0.3834
Epoch 123/140
48/48 - 1s - 12ms/step - accuracy: 0.9490 - loss: 0.1650 - val_accuracy: 0.8492
- val_loss: 0.3841
Epoch 124/140
48/48 - 1s - 12ms/step - accuracy: 0.9318 - loss: 0.1844 - val_accuracy: 0.8519
- val_loss: 0.3902
Epoch 125/140
48/48 - 1s - 13ms/step - accuracy: 0.9411 - loss: 0.1767 - val_accuracy: 0.8545
- val_loss: 0.3928
Epoch 126/140
48/48 - 1s - 13ms/step - accuracy: 0.9377 - loss: 0.1676 - val_accuracy: 0.8519
- val_loss: 0.3931
Epoch 127/140
48/48 - 1s - 13ms/step - accuracy: 0.9424 - loss: 0.1536 - val_accuracy: 0.8492
- val_loss: 0.3823
Epoch 128/140
48/48 - 1s - 13ms/step - accuracy: 0.9298 - loss: 0.1798 - val_accuracy: 0.8492
- val_loss: 0.3891
Epoch 129/140
48/48 - 1s - 13ms/step - accuracy: 0.9464 - loss: 0.1720 - val_accuracy: 0.8466
- val_loss: 0.3854
Epoch 130/140
48/48 - 1s - 13ms/step - accuracy: 0.9477 - loss: 0.1621 - val_accuracy: 0.8492
- val_loss: 0.3847
```

```
Epoch 131/140
48/48 - 1s - 13ms/step - accuracy: 0.9417 - loss: 0.1782 - val_accuracy: 0.8466
- val_loss: 0.3934
Epoch 132/140
48/48 - 1s - 12ms/step - accuracy: 0.9457 - loss: 0.1568 - val_accuracy: 0.8466
- val_loss: 0.3971
Epoch 133/140
48/48 - 1s - 13ms/step - accuracy: 0.9457 - loss: 0.1599 - val_accuracy: 0.8439
- val_loss: 0.3914
Epoch 134/140
48/48 - 1s - 13ms/step - accuracy: 0.9470 - loss: 0.1626 - val_accuracy: 0.8466
- val_loss: 0.3997
Epoch 135/140
48/48 - 1s - 13ms/step - accuracy: 0.9397 - loss: 0.1616 - val_accuracy: 0.8439
- val_loss: 0.3993
Epoch 136/140
48/48 - 1s - 13ms/step - accuracy: 0.9503 - loss: 0.1605 - val_accuracy: 0.8439
- val_loss: 0.3939
Epoch 137/140
48/48 - 1s - 13ms/step - accuracy: 0.9437 - loss: 0.1448 - val_accuracy: 0.8439
- val_loss: 0.3980
Epoch 138/140
48/48 - 1s - 13ms/step - accuracy: 0.9464 - loss: 0.1463 - val_accuracy: 0.8492
- val_loss: 0.3900
Epoch 139/140
48/48 - 1s - 12ms/step - accuracy: 0.9510 - loss: 0.1575 - val_accuracy: 0.8492
- val_loss: 0.3833
Epoch 140/140
48/48 - 1s - 13ms/step - accuracy: 0.9556 - loss: 0.1470 - val_accuracy: 0.8492
- val_loss: 0.3923
Train score: [0.09877098351716995, 0.9794701933860779]
Validation score: [0.392261803150177, 0.8492063283920288]
```

[35]:
```
plotLearningCurve(r,n_epochs,'1')
```

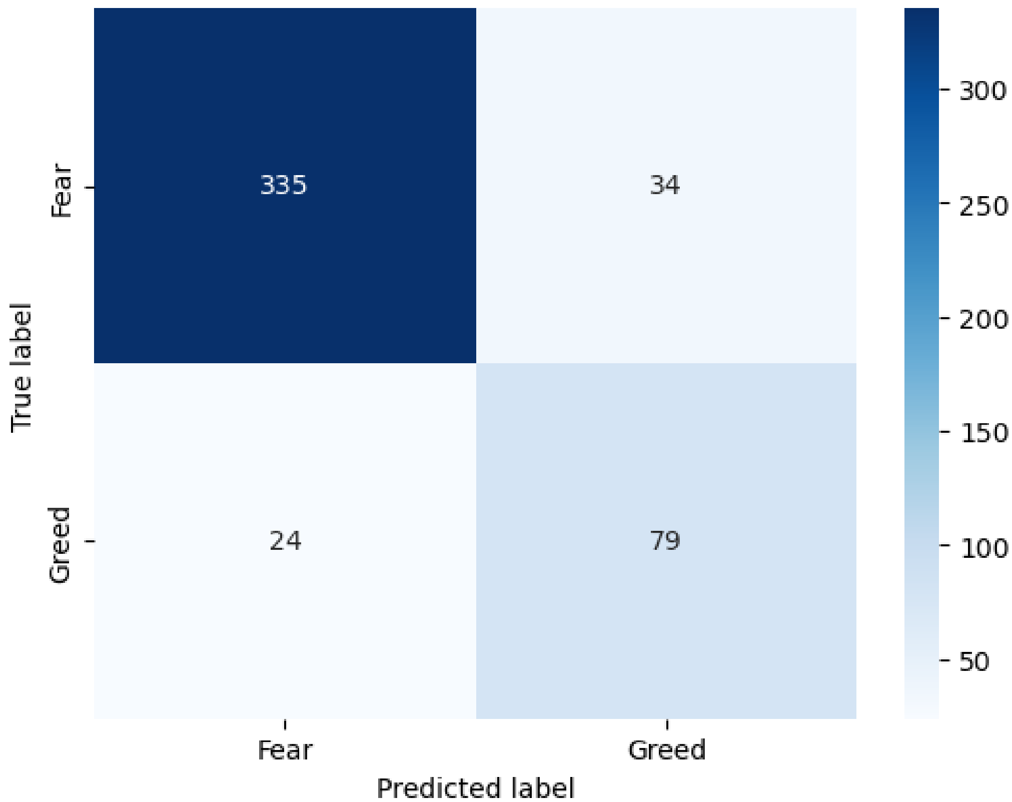


[36]:
```
print("Evaluate Model Performance on Test set")
result = model.evaluate(pad_test,ytest)
print(dict(zip(model.metrics_names, result)))
```

```
Evaluate Model Performance on Test set
15/15                 0s 5ms/step -
accuracy: 0.8902 - loss: 0.2720
{'loss': 0.2992241382598877, 'compile_metrics': 0.8771186470985413}
```

[37]:
```
ypred = model.predict(pad_test)
ypred = ypred>0.5
#Get the confusion matrix
cf_matrix = confusion_matrix(ytest, ypred)
class_names = ['Fear', 'Greed']
sns.heatmap(cf_matrix , annot=True, fmt='d', cmap='Blues',␣
↪xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.savefig('LSTMmat.png')
plt.show()
print(classification_report(ytest, ypred))
```

```
15/15                 1s 30ms/step
```




```
              precision    recall  f1-score   support

           0       0.93      0.91      0.92       369
           1       0.70      0.77      0.73       103

    accuracy                           0.88       472
   macro avg       0.82      0.84      0.83       472
weighted avg       0.88      0.88      0.88       472
```

[38]:

```
from tensorflow.keras.layers import Input, Embedding, BatchNormalization,␣
 ↪Dropout, Conv1D, MaxPooling1D, Bidirectional, SimpleRNN, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

def rnn_model(Xtrain, Xval, ytrain, yval, V, D, maxlen, epochs):
    print("----Building the model----")
    i = Input(shape=(maxlen,))
    x = Embedding(V + 1, D)(i)
    x = BatchNormalization()(x)
    x = Dropout(0.3)(x)
```

```
    x = Conv1D(32, 5, activation='relu')(x)
    x = Dropout(0.3)(x)
    x = MaxPooling1D(2)(x)
    x = Bidirectional(SimpleRNN(128, return_sequences=True))(x)
    x = SimpleRNN(64)(x)
    x = Dropout(0.5)(x)
    x = Dense(1, activation='sigmoid')(x)
    model = Model(i, x)
    model.summary()
    ytrain = np.array(ytrain).reshape(-1, 1)
    yval = np.array(yval).reshape(-1, 1)
    classes, counts = np.unique(ytrain, return_counts=True)
    print(f"Classes: {classes}")
    print(f"Counts: {counts}")
    class_weights = compute_class_weight(class_weight='balanced',␣
↪classes=classes, y=ytrain.ravel())
    class_weights = dict(zip(classes, class_weights))
    print(f"Class weights: {class_weights}")

    # Training the LSTM
    print("----Training the network----")
    model.compile(optimizer=Adam(0.000007),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    r = model.fit(Xtrain, ytrain,
                  validation_data=(Xval, yval),
                  epochs=epochs,
                  verbose=2,
                  batch_size=32,
                  class_weight=class_weights)
                  #callbacks = callbacks

    # Evaluate the model
    train_score = model.evaluate(Xtrain, ytrain, verbose=0)
    val_score = model.evaluate(Xval, yval, verbose=0)

    print(f"Train score: {train_score}")
    print(f"Validation score: {val_score}")

    n_epochs = len(r.history['loss'])

    return r, model, n_epochs
```

[39]:

```
D = 64 #embedding
epochs = 120
r,model,n_epochs = rnn_model(Xtrain,Xval,ytrain,yval,V,D,max_seq_len,epochs)
```

```
----Building the model----
Model: "functional_3"

 Layer (type)                    Output Shape                Param #

 input_layer_1 (InputLayer)      (None, 16)                        0

 embedding_1 (Embedding)         (None, 16, 64)              272,000

 batch_normalization_1           (None, 16, 64)                  256
 (BatchNormalization)

 dropout_3 (Dropout)             (None, 16, 64)                    0

 conv1d_1 (Conv1D)               (None, 12, 32)               10,272

 dropout_4 (Dropout)             (None, 12, 32)                    0

 max_pooling1d_1 (MaxPooling1D)  (None, 6, 32)                     0

 bidirectional_1 (Bidirectional) (None, 6, 256)               41,216

 simple_rnn_1 (SimpleRNN)        (None, 64)                   20,544

 dropout_5 (Dropout)             (None, 64)                        0

 dense_1 (Dense)                 (None, 1)                        65


 Total params: 344,353 (1.31 MB)

 Trainable params: 344,225 (1.31 MB)

 Non-trainable params: 128 (512.00 B)

Classes: [0 1]
Counts: [1197  313]
Class weights: {0: 0.6307435254803676, 1: 2.412140575079872}
----Training the network----
Epoch 1/120
48/48 - 4s - 91ms/step - accuracy: 0.4788 - loss: 0.8388 - val_accuracy: 0.5238
- val_loss: 0.6912
Epoch 2/120
48/48 - 0s - 10ms/step - accuracy: 0.5046 - loss: 0.8140 - val_accuracy: 0.6296
```

```
- val_loss: 0.6716
Epoch 3/120
48/48 - 0s - 9ms/step - accuracy: 0.5126 - loss: 0.8053 - val_accuracy: 0.7169 -
val_loss: 0.6509
Epoch 4/120
48/48 - 0s - 9ms/step - accuracy: 0.5563 - loss: 0.7703 - val_accuracy: 0.7751 -
val_loss: 0.6272
Epoch 5/120
48/48 - 0s - 9ms/step - accuracy: 0.5576 - loss: 0.7560 - val_accuracy: 0.7937 -
val_loss: 0.6059
Epoch 6/120
48/48 - 0s - 9ms/step - accuracy: 0.5715 - loss: 0.7350 - val_accuracy: 0.8095 -
val_loss: 0.5866
Epoch 7/120
48/48 - 0s - 10ms/step - accuracy: 0.5887 - loss: 0.7331 - val_accuracy: 0.8201
- val_loss: 0.5659
Epoch 8/120
48/48 - 0s - 10ms/step - accuracy: 0.5934 - loss: 0.7082 - val_accuracy: 0.8228
- val_loss: 0.5459
Epoch 9/120
48/48 - 0s - 9ms/step - accuracy: 0.6252 - loss: 0.7165 - val_accuracy: 0.8228 -
val_loss: 0.5315
Epoch 10/120
48/48 - 0s - 9ms/step - accuracy: 0.6377 - loss: 0.6876 - val_accuracy: 0.8228 -
val_loss: 0.5204
Epoch 11/120
48/48 - 0s - 9ms/step - accuracy: 0.6358 - loss: 0.6813 - val_accuracy: 0.8148 -
val_loss: 0.5134
Epoch 12/120
48/48 - 0s - 9ms/step - accuracy: 0.6430 - loss: 0.6562 - val_accuracy: 0.8069 -
val_loss: 0.5093
Epoch 13/120
48/48 - 0s - 9ms/step - accuracy: 0.6430 - loss: 0.6861 - val_accuracy: 0.8042 -
val_loss: 0.4977
Epoch 14/120
48/48 - 0s - 9ms/step - accuracy: 0.6530 - loss: 0.6469 - val_accuracy: 0.8016 -
val_loss: 0.4947
Epoch 15/120
48/48 - 0s - 9ms/step - accuracy: 0.6464 - loss: 0.6587 - val_accuracy: 0.7989 -
val_loss: 0.4904
Epoch 16/120
48/48 - 0s - 9ms/step - accuracy: 0.6742 - loss: 0.6241 - val_accuracy: 0.7910 -
val_loss: 0.4926
Epoch 17/120
48/48 - 0s - 9ms/step - accuracy: 0.6636 - loss: 0.6404 - val_accuracy: 0.7910 -
val_loss: 0.4823
Epoch 18/120
48/48 - 0s - 9ms/step - accuracy: 0.6815 - loss: 0.6137 - val_accuracy: 0.7937 -
```

```
val_loss: 0.4719
Epoch 19/120
48/48 - 0s - 9ms/step - accuracy: 0.7060 - loss: 0.5837 - val_accuracy: 0.7884 -
val_loss: 0.4744
Epoch 20/120
48/48 - 0s - 9ms/step - accuracy: 0.7066 - loss: 0.6005 - val_accuracy: 0.7857 -
val_loss: 0.4653
Epoch 21/120
48/48 - 0s - 9ms/step - accuracy: 0.6887 - loss: 0.5921 - val_accuracy: 0.7937 -
val_loss: 0.4564
Epoch 22/120
48/48 - 0s - 9ms/step - accuracy: 0.6874 - loss: 0.6170 - val_accuracy: 0.7937 -
val_loss: 0.4551
Epoch 23/120
48/48 - 0s - 9ms/step - accuracy: 0.7007 - loss: 0.5821 - val_accuracy: 0.7963 -
val_loss: 0.4501
Epoch 24/120
48/48 - 0s - 9ms/step - accuracy: 0.7252 - loss: 0.5565 - val_accuracy: 0.7937 -
val_loss: 0.4442
Epoch 25/120
48/48 - 0s - 9ms/step - accuracy: 0.7325 - loss: 0.5547 - val_accuracy: 0.7937 -
val_loss: 0.4413
Epoch 26/120
48/48 - 0s - 9ms/step - accuracy: 0.7219 - loss: 0.5478 - val_accuracy: 0.7937 -
val_loss: 0.4390
Epoch 27/120
48/48 - 0s - 9ms/step - accuracy: 0.7344 - loss: 0.5477 - val_accuracy: 0.7963 -
val_loss: 0.4358
Epoch 28/120
48/48 - 0s - 9ms/step - accuracy: 0.7285 - loss: 0.5531 - val_accuracy: 0.7937 -
val_loss: 0.4392
Epoch 29/120
48/48 - 1s - 13ms/step - accuracy: 0.7483 - loss: 0.5335 - val_accuracy: 0.7963
- val_loss: 0.4338
Epoch 30/120
48/48 - 0s - 10ms/step - accuracy: 0.7483 - loss: 0.5147 - val_accuracy: 0.7989
- val_loss: 0.4332
Epoch 31/120
48/48 - 0s - 10ms/step - accuracy: 0.7417 - loss: 0.5383 - val_accuracy: 0.7963
- val_loss: 0.4303
Epoch 32/120
48/48 - 0s - 9ms/step - accuracy: 0.7583 - loss: 0.5232 - val_accuracy: 0.8016 -
val_loss: 0.4296
Epoch 33/120
48/48 - 0s - 9ms/step - accuracy: 0.7477 - loss: 0.5170 - val_accuracy: 0.8016 -
val_loss: 0.4347
Epoch 34/120
48/48 - 0s - 9ms/step - accuracy: 0.7437 - loss: 0.5398 - val_accuracy: 0.7963 -
```

```
val_loss: 0.4376
Epoch 35/120
48/48 - 0s - 9ms/step - accuracy: 0.7364 - loss: 0.5237 - val_accuracy: 0.7963 -
val_loss: 0.4346
Epoch 36/120
48/48 - 0s - 10ms/step - accuracy: 0.7457 - loss: 0.5187 - val_accuracy: 0.8042
- val_loss: 0.4294
Epoch 37/120
48/48 - 0s - 9ms/step - accuracy: 0.7636 - loss: 0.5077 - val_accuracy: 0.8042 -
val_loss: 0.4291
Epoch 38/120
48/48 - 1s - 13ms/step - accuracy: 0.7576 - loss: 0.5255 - val_accuracy: 0.8042
- val_loss: 0.4313
Epoch 39/120
48/48 - 1s - 12ms/step - accuracy: 0.7616 - loss: 0.5091 - val_accuracy: 0.8095
- val_loss: 0.4240
Epoch 40/120
48/48 - 0s - 9ms/step - accuracy: 0.7576 - loss: 0.4959 - val_accuracy: 0.8042 -
val_loss: 0.4271
Epoch 41/120
48/48 - 0s - 8ms/step - accuracy: 0.7550 - loss: 0.4945 - val_accuracy: 0.8042 -
val_loss: 0.4296
Epoch 42/120
48/48 - 0s - 8ms/step - accuracy: 0.7642 - loss: 0.5007 - val_accuracy: 0.8069 -
val_loss: 0.4256
Epoch 43/120
48/48 - 0s - 9ms/step - accuracy: 0.7636 - loss: 0.5074 - val_accuracy: 0.8095 -
val_loss: 0.4237
Epoch 44/120
48/48 - 0s - 8ms/step - accuracy: 0.7715 - loss: 0.4913 - val_accuracy: 0.8148 -
val_loss: 0.4210
Epoch 45/120
48/48 - 0s - 9ms/step - accuracy: 0.7854 - loss: 0.4591 - val_accuracy: 0.8148 -
val_loss: 0.4233
Epoch 46/120
48/48 - 0s - 8ms/step - accuracy: 0.7623 - loss: 0.5013 - val_accuracy: 0.8175 -
val_loss: 0.4203
Epoch 47/120
48/48 - 0s - 9ms/step - accuracy: 0.7715 - loss: 0.4957 - val_accuracy: 0.8148 -
val_loss: 0.4202
Epoch 48/120
48/48 - 0s - 9ms/step - accuracy: 0.7662 - loss: 0.5268 - val_accuracy: 0.8122 -
val_loss: 0.4232
Epoch 49/120
48/48 - 0s - 9ms/step - accuracy: 0.7781 - loss: 0.4852 - val_accuracy: 0.8148 -
val_loss: 0.4216
Epoch 50/120
48/48 - 0s - 9ms/step - accuracy: 0.7874 - loss: 0.4709 - val_accuracy: 0.8148 -
```

```
val_loss: 0.4231
Epoch 51/120
48/48 - 0s - 9ms/step - accuracy: 0.7656 - loss: 0.4946 - val_accuracy: 0.8148 -
val_loss: 0.4249
Epoch 52/120
48/48 - 0s - 9ms/step - accuracy: 0.7775 - loss: 0.4946 - val_accuracy: 0.8148 -
val_loss: 0.4250
Epoch 53/120
48/48 - 0s - 9ms/step - accuracy: 0.7848 - loss: 0.4581 - val_accuracy: 0.8201 -
val_loss: 0.4235
Epoch 54/120
48/48 - 0s - 9ms/step - accuracy: 0.7834 - loss: 0.4732 - val_accuracy: 0.8201 -
val_loss: 0.4238
Epoch 55/120
48/48 - 0s - 9ms/step - accuracy: 0.7874 - loss: 0.4613 - val_accuracy: 0.8201 -
val_loss: 0.4263
Epoch 56/120
48/48 - 0s - 9ms/step - accuracy: 0.7874 - loss: 0.4910 - val_accuracy: 0.8201 -
val_loss: 0.4236
Epoch 57/120
48/48 - 0s - 9ms/step - accuracy: 0.7815 - loss: 0.4806 - val_accuracy: 0.8228 -
val_loss: 0.4228
Epoch 58/120
48/48 - 0s - 9ms/step - accuracy: 0.7815 - loss: 0.4585 - val_accuracy: 0.8254 -
val_loss: 0.4194
Epoch 59/120
48/48 - 0s - 9ms/step - accuracy: 0.7874 - loss: 0.4662 - val_accuracy: 0.8254 -
val_loss: 0.4217
Epoch 60/120
48/48 - 0s - 8ms/step - accuracy: 0.7834 - loss: 0.4735 - val_accuracy: 0.8333 -
val_loss: 0.4169
Epoch 61/120
48/48 - 1s - 14ms/step - accuracy: 0.7808 - loss: 0.4781 - val_accuracy: 0.8333
- val_loss: 0.4194
Epoch 62/120
48/48 - 0s - 9ms/step - accuracy: 0.7940 - loss: 0.4593 - val_accuracy: 0.8333 -
val_loss: 0.4172
Epoch 63/120
48/48 - 0s - 10ms/step - accuracy: 0.7907 - loss: 0.4486 - val_accuracy: 0.8333
- val_loss: 0.4182
Epoch 64/120
48/48 - 0s - 9ms/step - accuracy: 0.7881 - loss: 0.4704 - val_accuracy: 0.8333 -
val_loss: 0.4219
Epoch 65/120
48/48 - 0s - 9ms/step - accuracy: 0.7887 - loss: 0.4481 - val_accuracy: 0.8333 -
val_loss: 0.4183
Epoch 66/120
48/48 - 0s - 9ms/step - accuracy: 0.7940 - loss: 0.4655 - val_accuracy: 0.8333 -
```

```
val_loss: 0.4243
Epoch 67/120
48/48 - 0s - 9ms/step - accuracy: 0.7987 - loss: 0.4349 - val_accuracy: 0.8333 -
val_loss: 0.4212
Epoch 68/120
48/48 - 0s - 9ms/step - accuracy: 0.7993 - loss: 0.4608 - val_accuracy: 0.8333 -
val_loss: 0.4210
Epoch 69/120
48/48 - 0s - 8ms/step - accuracy: 0.7901 - loss: 0.4447 - val_accuracy: 0.8333 -
val_loss: 0.4169
Epoch 70/120
48/48 - 0s - 9ms/step - accuracy: 0.7887 - loss: 0.4669 - val_accuracy: 0.8333 -
val_loss: 0.4150
Epoch 71/120
48/48 - 0s - 8ms/step - accuracy: 0.7940 - loss: 0.4651 - val_accuracy: 0.8333 -
val_loss: 0.4183
Epoch 72/120
48/48 - 0s - 8ms/step - accuracy: 0.8119 - loss: 0.4338 - val_accuracy: 0.8333 -
val_loss: 0.4144
Epoch 73/120
48/48 - 0s - 9ms/step - accuracy: 0.8026 - loss: 0.4546 - val_accuracy: 0.8360 -
val_loss: 0.4143
Epoch 74/120
48/48 - 0s - 8ms/step - accuracy: 0.8079 - loss: 0.4567 - val_accuracy: 0.8333 -
val_loss: 0.4179
Epoch 75/120
48/48 - 0s - 9ms/step - accuracy: 0.7940 - loss: 0.4647 - val_accuracy: 0.8360 -
val_loss: 0.4169
Epoch 76/120
48/48 - 0s - 9ms/step - accuracy: 0.8106 - loss: 0.4290 - val_accuracy: 0.8360 -
val_loss: 0.4137
Epoch 77/120
48/48 - 0s - 9ms/step - accuracy: 0.8146 - loss: 0.4258 - val_accuracy: 0.8360 -
val_loss: 0.4130
Epoch 78/120
48/48 - 0s - 9ms/step - accuracy: 0.8172 - loss: 0.4248 - val_accuracy: 0.8360 -
val_loss: 0.4175
Epoch 79/120
48/48 - 0s - 9ms/step - accuracy: 0.8033 - loss: 0.4414 - val_accuracy: 0.8360 -
val_loss: 0.4144
Epoch 80/120
48/48 - 0s - 9ms/step - accuracy: 0.8139 - loss: 0.4226 - val_accuracy: 0.8360 -
val_loss: 0.4126
Epoch 81/120
48/48 - 0s - 9ms/step - accuracy: 0.8020 - loss: 0.4420 - val_accuracy: 0.8360 -
val_loss: 0.4123
Epoch 82/120
48/48 - 0s - 10ms/step - accuracy: 0.8046 - loss: 0.4196 - val_accuracy: 0.8360
```

```
- val_loss: 0.4148
Epoch 83/120
48/48 - 0s - 9ms/step - accuracy: 0.8053 - loss: 0.4204 - val_accuracy: 0.8360 -
val_loss: 0.4083
Epoch 84/120
48/48 - 0s - 9ms/step - accuracy: 0.8126 - loss: 0.4062 - val_accuracy: 0.8413 -
val_loss: 0.4039
Epoch 85/120
48/48 - 0s - 9ms/step - accuracy: 0.8053 - loss: 0.4306 - val_accuracy: 0.8413 -
val_loss: 0.4053
Epoch 86/120
48/48 - 0s - 9ms/step - accuracy: 0.7987 - loss: 0.4431 - val_accuracy: 0.8360 -
val_loss: 0.4083
Epoch 87/120
48/48 - 0s - 9ms/step - accuracy: 0.7980 - loss: 0.4381 - val_accuracy: 0.8413 -
val_loss: 0.4040
Epoch 88/120
48/48 - 0s - 9ms/step - accuracy: 0.8073 - loss: 0.4182 - val_accuracy: 0.8439 -
val_loss: 0.4015
Epoch 89/120
48/48 - 0s - 9ms/step - accuracy: 0.8106 - loss: 0.4282 - val_accuracy: 0.8466 -
val_loss: 0.4020
Epoch 90/120
48/48 - 0s - 9ms/step - accuracy: 0.8146 - loss: 0.4113 - val_accuracy: 0.8466 -
val_loss: 0.4013
Epoch 91/120
48/48 - 0s - 9ms/step - accuracy: 0.8119 - loss: 0.4281 - val_accuracy: 0.8386 -
val_loss: 0.4102
Epoch 92/120
48/48 - 0s - 9ms/step - accuracy: 0.8172 - loss: 0.4140 - val_accuracy: 0.8413 -
val_loss: 0.4109
Epoch 93/120
48/48 - 0s - 9ms/step - accuracy: 0.8119 - loss: 0.4190 - val_accuracy: 0.8466 -
val_loss: 0.4069
Epoch 94/120
48/48 - 0s - 9ms/step - accuracy: 0.8106 - loss: 0.4168 - val_accuracy: 0.8386 -
val_loss: 0.4133
Epoch 95/120
48/48 - 0s - 9ms/step - accuracy: 0.8192 - loss: 0.3969 - val_accuracy: 0.8466 -
val_loss: 0.4057
Epoch 96/120
48/48 - 0s - 9ms/step - accuracy: 0.8265 - loss: 0.4008 - val_accuracy: 0.8466 -
val_loss: 0.4005
Epoch 97/120
48/48 - 0s - 9ms/step - accuracy: 0.8278 - loss: 0.4014 - val_accuracy: 0.8466 -
val_loss: 0.4053
Epoch 98/120
48/48 - 0s - 9ms/step - accuracy: 0.8417 - loss: 0.3760 - val_accuracy: 0.8466 -
```

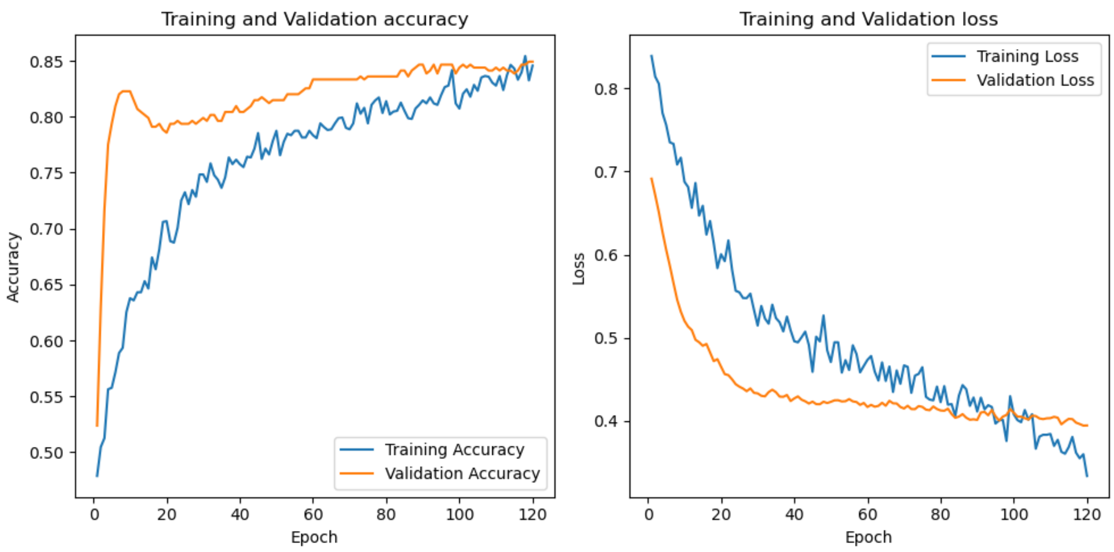
```
val_loss: 0.4073
Epoch 99/120
48/48 - 0s - 9ms/step - accuracy: 0.8119 - loss: 0.4298 - val_accuracy: 0.8386 -
val_loss: 0.4143
Epoch 100/120
48/48 - 0s - 9ms/step - accuracy: 0.8073 - loss: 0.4080 - val_accuracy: 0.8439 -
val_loss: 0.4100
Epoch 101/120
48/48 - 0s - 9ms/step - accuracy: 0.8205 - loss: 0.4011 - val_accuracy: 0.8466 -
val_loss: 0.4052
Epoch 102/120
48/48 - 0s - 10ms/step - accuracy: 0.8245 - loss: 0.3986 - val_accuracy: 0.8439
- val_loss: 0.4052
Epoch 103/120
48/48 - 0s - 9ms/step - accuracy: 0.8179 - loss: 0.4132 - val_accuracy: 0.8466 -
val_loss: 0.4033
Epoch 104/120
48/48 - 0s - 10ms/step - accuracy: 0.8285 - loss: 0.4016 - val_accuracy: 0.8439
- val_loss: 0.4018
Epoch 105/120
48/48 - 0s - 9ms/step - accuracy: 0.8232 - loss: 0.4082 - val_accuracy: 0.8439 -
val_loss: 0.4064
Epoch 106/120
48/48 - 0s - 9ms/step - accuracy: 0.8351 - loss: 0.3667 - val_accuracy: 0.8439 -
val_loss: 0.4059
Epoch 107/120
48/48 - 0s - 9ms/step - accuracy: 0.8364 - loss: 0.3812 - val_accuracy: 0.8439 -
val_loss: 0.4031
Epoch 108/120
48/48 - 0s - 9ms/step - accuracy: 0.8358 - loss: 0.3835 - val_accuracy: 0.8413 -
val_loss: 0.4023
Epoch 109/120
48/48 - 0s - 9ms/step - accuracy: 0.8305 - loss: 0.3834 - val_accuracy: 0.8413 -
val_loss: 0.4032
Epoch 110/120
48/48 - 0s - 9ms/step - accuracy: 0.8278 - loss: 0.3844 - val_accuracy: 0.8439 -
val_loss: 0.4034
Epoch 111/120
48/48 - 0s - 9ms/step - accuracy: 0.8364 - loss: 0.3701 - val_accuracy: 0.8413 -
val_loss: 0.4050
Epoch 112/120
48/48 - 0s - 9ms/step - accuracy: 0.8238 - loss: 0.3774 - val_accuracy: 0.8439 -
val_loss: 0.4040
Epoch 113/120
48/48 - 0s - 9ms/step - accuracy: 0.8371 - loss: 0.3630 - val_accuracy: 0.8413 -
val_loss: 0.3960
Epoch 114/120
48/48 - 0s - 9ms/step - accuracy: 0.8464 - loss: 0.3607 - val_accuracy: 0.8413 -
```

```
val_loss: 0.3996
Epoch 115/120
48/48 - 0s - 9ms/step - accuracy: 0.8430 - loss: 0.3686 - val_accuracy: 0.8386 -
val_loss: 0.4026
Epoch 116/120
48/48 - 0s - 9ms/step - accuracy: 0.8331 - loss: 0.3808 - val_accuracy: 0.8413 -
val_loss: 0.4023
Epoch 117/120
48/48 - 0s - 9ms/step - accuracy: 0.8397 - loss: 0.3619 - val_accuracy: 0.8466 -
val_loss: 0.3981
Epoch 118/120
48/48 - 0s - 9ms/step - accuracy: 0.8543 - loss: 0.3552 - val_accuracy: 0.8466 -
val_loss: 0.3964
Epoch 119/120
48/48 - 0s - 9ms/step - accuracy: 0.8325 - loss: 0.3600 - val_accuracy: 0.8492 -
val_loss: 0.3945
Epoch 120/120
48/48 - 0s - 9ms/step - accuracy: 0.8457 - loss: 0.3340 - val_accuracy: 0.8492 -
val_loss: 0.3946
Train score: [0.24222174286842346, 0.8960264921188354]
Validation score: [0.39464399218559265, 0.8492063283920288]
```

[40]:
```
print("Evaluate Model Performance on Test set")
result = model.evaluate(pad_test,ytest)
print(dict(zip(model.metrics_names, result)))
```

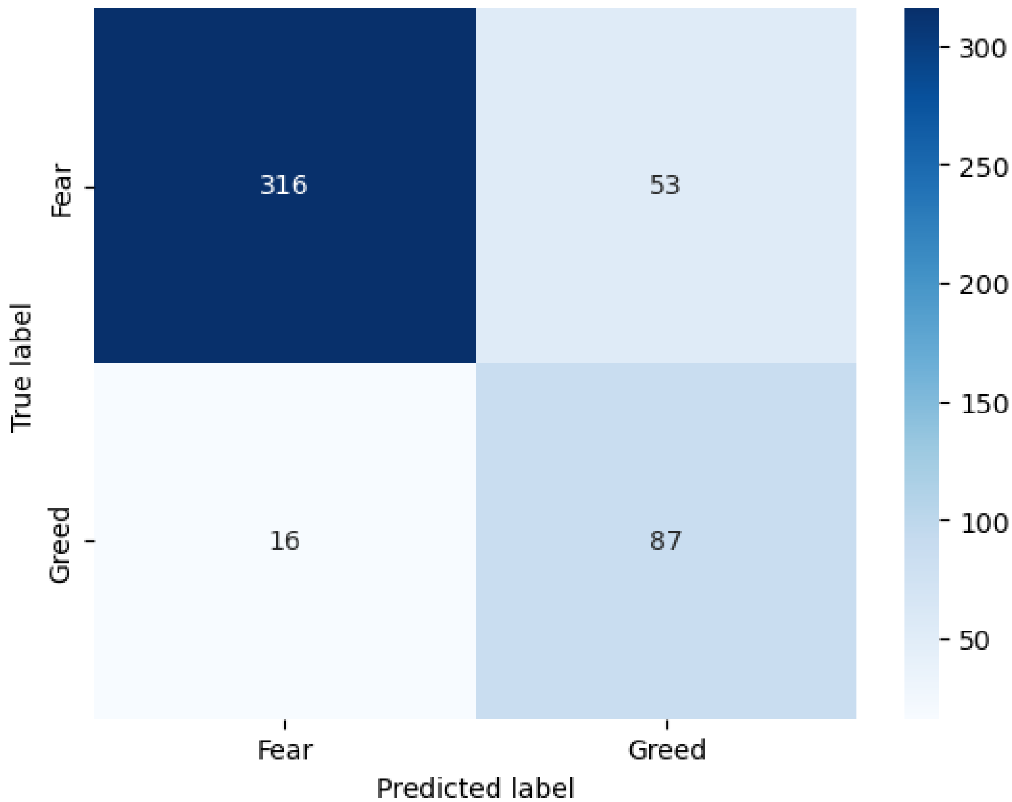
```
Evaluate Model Performance on Test set
15/15            0s 3ms/step -
accuracy: 0.8642 - loss: 0.3555
{'loss': 0.38578107953071594, 'compile_metrics': 0.8538135886192322}
```

[41]:
```
plotLearningCurve(r,n_epochs,'2')
```



[42]:
```
#Generate predictions for the test dataset
ypred = model.predict(pad_test)
ypred = ypred>0.5
#Get the confusion matrix
cf_matrix = confusion_matrix(ytest, ypred)
class_names = ['Fear', 'Greed']
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.savefig('RNNmat.png')
plt.show()
print(classification_report(ytest, ypred))
```

```
15/15                 1s 26ms/step
```



	precision	recall	f1-score	support
0	0.95	0.86	0.90	369
1	0.62	0.84	0.72	103
accuracy			0.85	472
macro avg	0.79	0.85	0.81	472
weighted avg	0.88	0.85	0.86	472