Emotion-Text-Data-Classification-with-Natural-Language-Processing-NLP

Read the datasets

In [5]:
```
train=pd.read_csv(
    'C:/Users/zizhe/Desktop/Leo Zhao/NLP/Emotions dataset for NLP/train.txt',
    sep=";",
    names=["Description","Emotion"])

validate=pd.read_csv(
    'C:/Users/zizhe/Desktop/Leo Zhao/NLP/Emotions dataset for NLP/val.txt',
    sep=";",
    names=["Description","Emotion"])

test=pd.read_csv(
    'C:/Users/zizhe/Desktop/Leo Zhao/NLP/Emotions dataset for NLP/test.txt',
    sep=";",
    names=["Description","Emotion"])
```

Display head of all dataset

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

Out[6]:

	Description	Emotion
0	i didnt feel humiliated	sadness
1	i can go from feeling so hopeless to so damned...	sadness
2	im grabbing a minute to post i feel greedy wrong	anger
3	i am ever feeling nostalgic about the fireplac...	love
4	i am feeling grouchy	anger

In [7]:
```
validate.head()
```

Out[7]:

	Description	Emotion
0	im feeling quite sad and sorry for myself but ...	sadness
1	i feel like i am still looking at a blank canv...	sadness
2	i feel like a faithful servant	love
3	i am just feeling cranky and blue	anger
4	i can have for a treat or if i am feeling festive	joy

In [8]:
```
test.head()
```

Out[8]:

	Description	Emotion
0	im feeling rather rotten so im not very ambiti...	sadness
1	im updating my blog because i feel shitty	sadness
2	i never make her separate from me because i do...	sadness
3	i left with my bouquet of red and yellow tulip...	joy
4	i was feeling a little vain when i did this one	sadness

Data Understanding

describe_data function will help us understand each dataset

In [9]:
```
def describe_data(data_type,data,label):
    print(data_type," DESCRIPTION")
    print("-------------------------")
    size=data.shape
    null_values=data.isnull().sum().sum()

    label_count=data[label].value_counts()
    print(data_type," shape:",size,"\n")
    print(data_type," contains:",null_values," null values\n")
    print("Label counts:")
    print(label_count)
    print()

    print("*****",data_type," label count plot ****")
    sns.countplot(
        data=data,
        x=label
    )
```

Describe training data

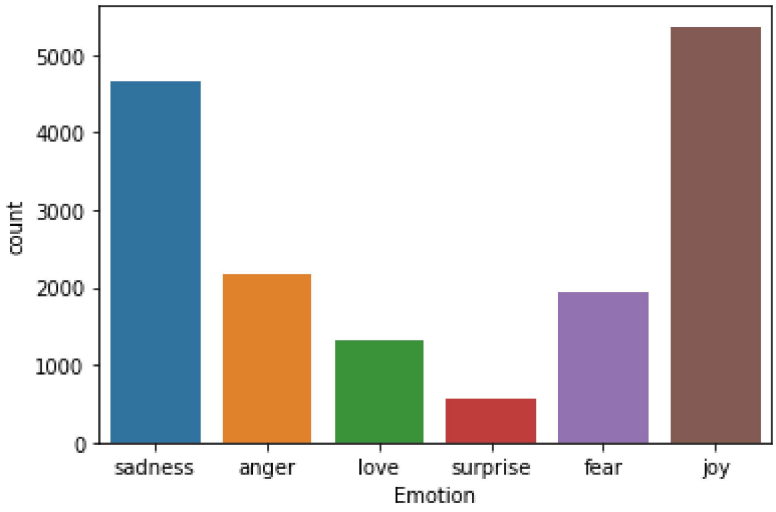
In [10]:
```
describe_data("Training Data",train,"Emotion")
```

```
Training Data  DESCRIPTION
-------------------------
Training Data  shape: (16000, 2)

Training Data  contains: 0  null values

Label counts:
joy         5362
sadness     4666
anger       2159
fear        1937
love        1304
surprise     572
Name: Emotion, dtype: int64

***** Training Data  label count plot ****
```



Describe validation data

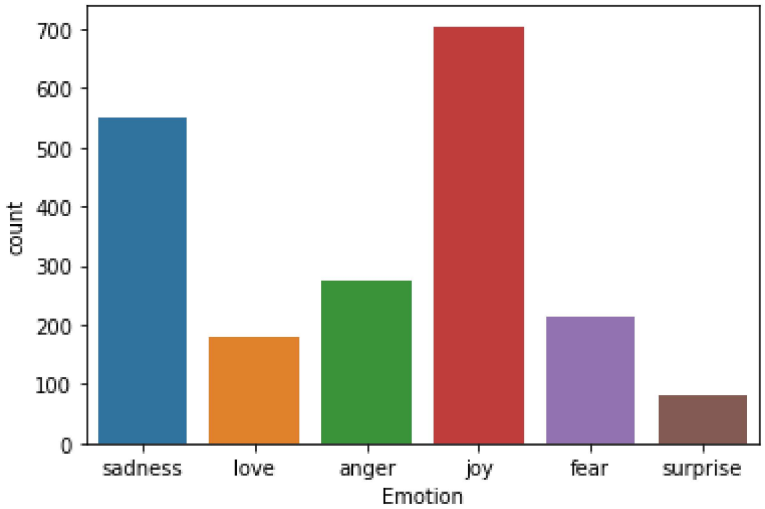
In [11]:
```
describe_data("Validation Data",validate,"Emotion")
```

```
Validation Data  DESCRIPTION
--------------------------
Validation Data  shape: (2000, 2)

Validation Data  contains: 0  null values

Label counts:
joy         704
sadness     550
anger       275
fear        212
love        178
surprise     81
Name: Emotion, dtype: int64

***** Validation Data  label count plot ****
```



Describe test data

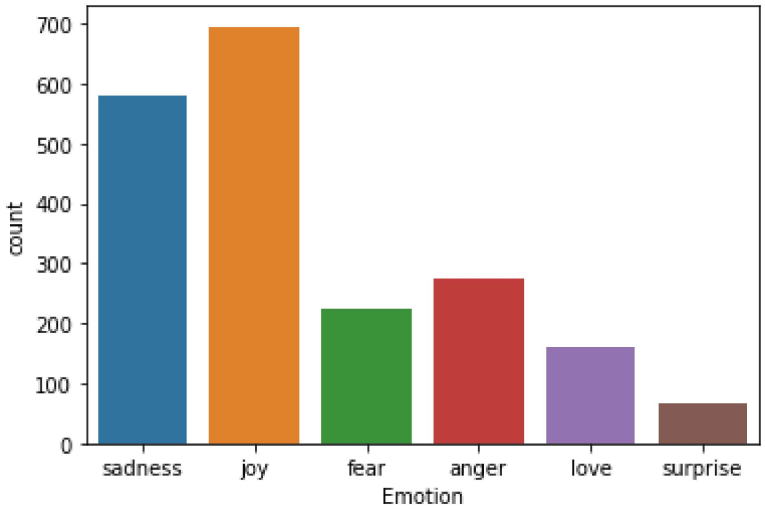
In [12]:
```
describe_data("Test Data",test,"Emotion")
```

```
Test Data  DESCRIPTION
--------------------------
Test Data  shape: (2000, 2)

Test Data  contains: 0  null values

Label counts:
joy         695
sadness     581
anger       275
fear        224
love        159
surprise     66
Name: Emotion, dtype: int64

***** Test Data  label count plot ****
```



Conclusion: All the dataset are free of null values. The test and validate data size are equal which is smaller than the traning data, which is obvious that more data is required in traning model than validating and testing model. The Distribution of joy and sadness labels are denser than any other labels in all dataset.

In NLP the distribution of text lengths plays a vital role in text preparation and model building, so lets try to extract insights of text lenghts in the dataset. I will specifically choose training data for this because traning data will be used for model building later.

The kdeplot shows that the curve is right-skewed, and its so because of few lengthy text description which have pulled the curved towards right.

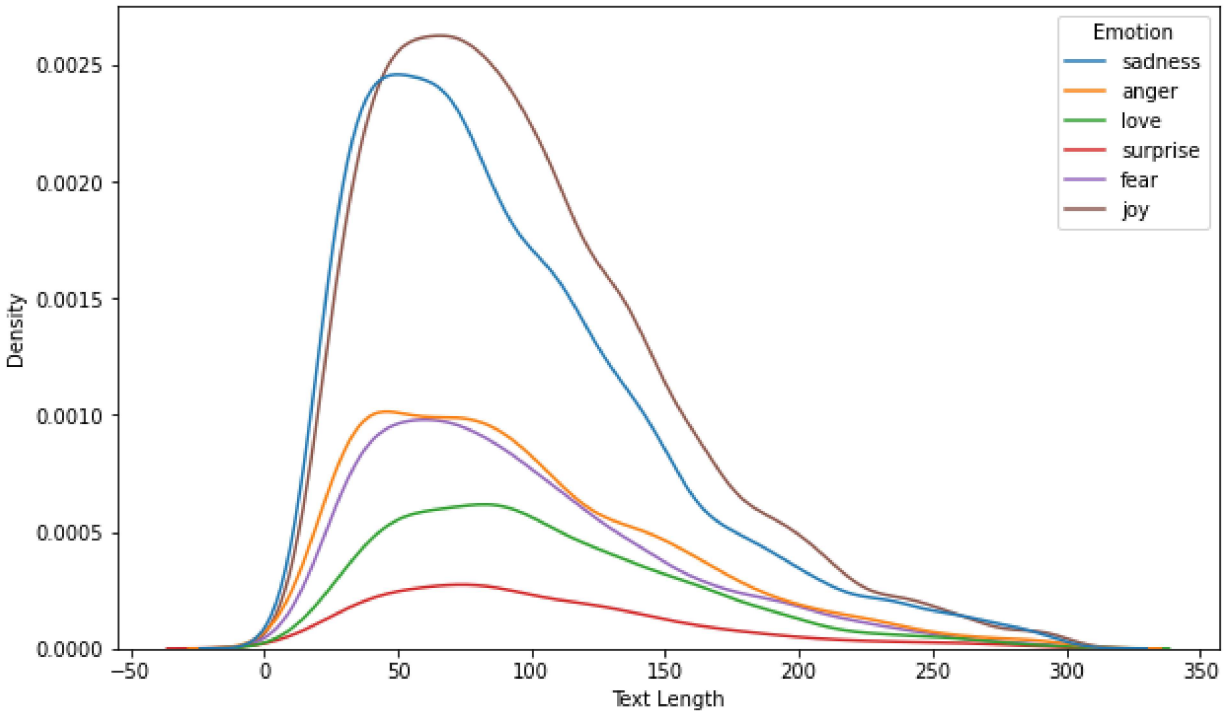
However we can see that all messages are concentrated in the range of 1-150. This plays vital role in text preparation as I have mentioned earlier

In [13]:
```
train["Text Length"]=train["Description"].apply(len)

fig=plt.figure(figsize=(10,6))

sns.kdeplot(
    x=train["Text Length"],
    hue=train["Emotion"]
```

```
)
plt.show()
```



We have mininum Description length of 7 and maximum length of 300 however most of them are concentrated in between 1-150

In [14]:
```
train[["Text Length"]].describe()
```

Out[14]:

	Text Length
count	16000.000000
mean	96.845812
std	55.904953
min	7.000000
25%	53.000000
50%	86.000000
75%	129.000000
max	300.000000

Data Preparation

Before preparing text for the model wenoticed that data labels with object datatype. So to feed our data to the model it must be converted into machine understandable form. For that reason let I have label encoded the labels using the funcion "label_encode"

In [15]:
```
def label_encode(data,label):
    labels=data[label].map(
    {
        "joy":0,
        "sadness":1,
        "anger":2,
        "fear":3,
        "love":4,
        "surprise":5
    }
    )
    return labels
```

Label encode in all the dataset

In [16]:
```
train["Label"]=label_encode(train,"Emotion")
validate["Label"]=label_encode(validate,"Emotion")
test["Label"]=label_encode(test,"Emotion")
```

Check the dataset

In [17]:
```
train.head()
```

Out[17]:

	Description	Emotion	Text Length	Label
0	i didnt feel humiliated	sadness	23	1
1	i can go from feeling so hopeless to so damned...	sadness	108	1
2	im grabbing a minute to post i feel greedy wrong	anger	48	2
3	i am ever feeling nostalgic about the fireplac...	love	92	4
4	i am feeling grouchy	anger	20	2

In [18]:
```
test.head()
```

Out[18]:

	Description	Emotion	Label
0	im feeling rather rotten so im not very ambiti...	sadness	1
1	im updating my blog because i feel shitty	sadness	1
2	i never make her separate from me because i do...	sadness	1
3	i left with my bouquet of red and yellow tulip...	joy	0
4	i was feeling a little vain when i did this one	sadness	1

In [19]:
```
validate.head()
```

Out[19]:

	Description	Emotion	Label
0	im feeling quite sad and sorry for myself but ...	sadness	1
1	i feel like i am still looking at a blank canv...	sadness	1
2	i feel like a faithful servant	love	4
3	i am just feeling cranky and blue	anger	2
4	i can have for a treat or if i am feeling festive	joy	0

Text Preparation

Import Libraries for text preparation

In [20]:
```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

I will take vocabulary size of 10,000 and a sentence length of 150 for my text preparation and Model Building later. This is the reason we have to get insights of the traning text description lengths

In [22]:
```
vocab_size=10000
sentence_len=150
```

The function takes raw text(description) then tokenize it, perform one_hot, and return the embedded_document which can be fed to the Sequential model later.

In [23]:
```
def data_preparation(data,description):
    stemmer=PorterStemmer()

    corpus=[]

    for text in data[description]:
        text=re.sub("[^a-zA-Z]"," ",text)
        text=text.lower()
        text=text.split()

        text=[stemmer.stem(words)
             for words in text
              if words not in stopwords.words("english")
             ]
        text=" ".join(text)
        corpus.append(text)

    oneHot_doc=[one_hot(input_text=words,n=vocab_size)
                for words in corpus
                ]

    embedded_doc=pad_sequences(sequences=oneHot_doc,
```

```
                                maxlen=sentence_len,
                                padding="pre")
    return embedded_doc
```

Create embedded document for training,validation and testing data which will be used for traning model, validating model and testing model respectively later.

The three below embedded document serves as the independent variable, and for convinience I have named it with familiar names, which we use in typical ML and DL dataset.

In [24]:
```
X_train=data_preparation(train,"Description")
X_validate=data_preparation(validate,"Description")
X_test=data_preparation(test,"Description")
```

Extract the target varaibles

In [25]:
```
y_train=train["Label"]
y_validate=validate["Label"]
y_test=test["Label"]
```

Model Building

Import libraries for model building

In [26]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.optimizers import Adam
```

Function build_model will return model which can be hypertuned using kerastuner later

In [27]:
```
def build_model(hp):

    model=Sequential()

    model.add(
    Embedding(
        input_dim=vocab_size,
        output_dim=hp.Int(
            "output_dim:",
            min_value=40,
            max_value=120,
            step=10
        ),
        input_length=sentence_len
    )
    )

    model.add(
        LSTM(
```

```
        units=128
        )
    )
    for j in range(hp.Int(
        "Dense Layers",
        min_value=1,
        max_value=5,
        step=1
    )):
        model.add(
        Dense(
            units=hp.Int(
                "units_"+str(j),
                min_value=32,
                max_value=256,
                step=32
            ),
            activation="relu",
            kernel_initializer=hp.Choice(
                "kernel_init"+str(j),
                values=["he_uniform","he_normal"]
            )
        )
        )
        model.add(
        Dropout(
        rate=hp.Float(
            "drop_rate"+str(j),
            min_value=0.1,
            max_value=0.5,
            step=0.1
        )
        )
        )

    model.add(
    Dense(

        units=6,
        activation="softmax"
    )
    )

    model.compile(
        optimizer=Adam(
        learning_rate=hp.Choice(
            "learnRate",
            values=[0.01,0.001,0.0001]
        )
        ),
        loss="sparse_categorical_crossentropy",
        metrics=["accuracy"]
    )

    return model
```

import library for model hypertuning

In [28]:
```
from kerastuner.tuners import RandomSearch
```

Initialize a tuner for the model.

In [29]:
```
tuner=RandomSearch(
    objective="val_accuracy",
    max_trials=2,
    executions_per_trial=2,
    directory="emotionNLP",
    project_name="hypertuningNLP"
    )
```

search through the parameters for using traning and validation data

Note: One can increase the number of trails and epochs to improve the accuracy scores

In [30]:
```
tuner.search(
    X_train,
    y_train,
    validation_data=(X_validate,
                     y_validate),
    epochs=3
    )
```

```
Trial 2 Complete [00h 00m 45s]
val_accuracy: 0.8557499945163727

Best val_accuracy So Far: 0.8557499945163727
Total elapsed time: 00h 01m 33s
```

Initialize the model with one of the best models generated by the tuner on training and validate data

In [31]:
```
model=tuner.get_best_models(num_models=1)[0]
```

Get the model summary

In [32]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 150, 50)           500000
_________________________________________________________________
lstm (LSTM)                  (None, 128)               91648
_________________________________________________________________
dense (Dense)                (None, 224)               28896
_________________________________________________________________
dropout (Dropout)            (None, 224)               0
_________________________________________________________________
dense_1 (Dense)              (None, 160)               36000
_________________________________________________________________
dropout_1 (Dropout)          (None, 160)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               20608
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                4128
_________________________________________________________________
dropout_3 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 198
=================================================================
Total params: 681,478
Trainable params: 681,478
Non-trainable params: 0
_________________________________________________________________
```

Model Evaluation

I have my model ready so lets evaluate it on the test data to see how it performs on the test data.

In [33]:
```
predict_classes=model.predict(X_test)
```

Remember we took softmax activation on our output layer, hence unlike binary classification where we use sigmoid to predict the sentiment labels,softmax activation function turns numbers aka logits into probabilities that sum to one. Softmax function outputs a vector that represents the probability distributions of a list of potential outcomes.

Thus to get the predicted sentiment class we take the vector with the highest probability and hence the output. Which can be performed by the code below

In [40]:
```
y_pred=[np.argmax(label) for label in predict_classes]
```

create a dataframe out of predicted labels and name its column as Predicted

In [42]:
```
predict=pd.DataFrame(
    y_pred,
```

```
    columns=["Predicted"]
)
```

Map the labels into original sentiment classes

In [43]:
```
predict["Predicted Label"]=predict["Predicted"].map(
    {
        0:"joy",
        1:"sadness",
        2:"anger",
        3:"fear",
        4:"love",
        5:"surprise"
    }
)
```

Concatenate the Predicted Label of predict dataframe with the original test data to compare them

In [44]:
```
predict_df=pd.concat([test["Description"],
                      test["Emotion"],
                      predict["Predicted Label"]],
                     axis=1)
```

From the 10 random samples I have printed, my model have predicted all right so we are moving in a right direction

In [45]:
```
predict_df.sample(10)
```

Out[45]:

	Description	Emotion	Predicted Label
955	i really enjoy cabernet for how aggressive the...	joy	joy
1119	i would feel so i don t know maybe a little re...	anger	anger
1610	i was trying to demonstrate that i understood ...	fear	fear
23	i also tell you in hopes that anyone who is st...	sadness	sadness
1361	i feel that this is for others to decide helli...	joy	joy
549	i pull out one of my favorite books to make my...	sadness	sadness
193	i really dont like quinn because i feel like s...	anger	sadness
476	i feel quite helpless in all of this so prayer...	sadness	sadness
653	i prepare i feel thankful that these events to...	joy	joy
1985	i wasnt feeling sociable i really wasnt	joy	joy

import classification metrics to evaluate our model

In [46]:
```
from sklearn.metrics import accuracy_score,confusion_matrix
```

In [47]:
```
score=accuracy_score(y_test,y_pred)
cm=confusion_matrix(y_test,y_pred)
```

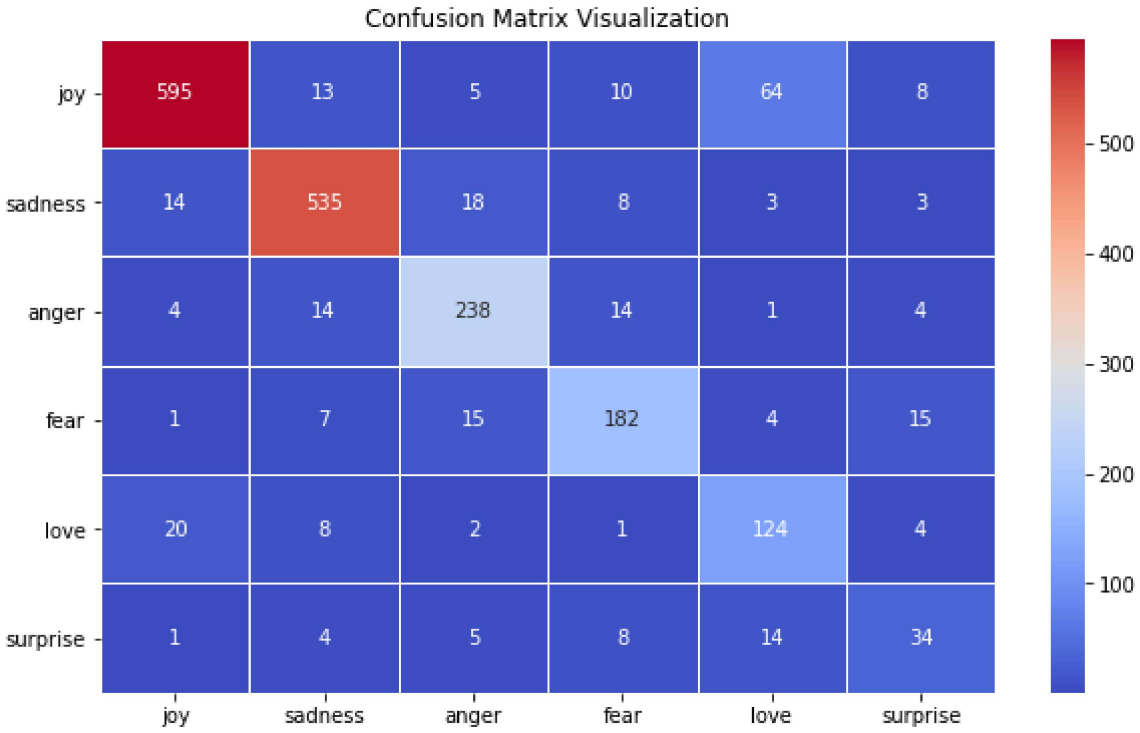
We got test score of 85.40% which isnt bad for limmited number of executions/epochs, however as mentioned earlier you can still improve the model accuracy score by increasing number of max_trials, executions_per_trial & epochs.

In [48]:
```
print("Test Score:{:.2f}%".format(score*100))
```

```
Test Score:85.40%
```

Lets visualize the confusion matrix using seaborn heatmap

In [49]:
```
labels=["joy","sadness","anger","fear","love","surprise"]
fig=plt.figure(figsize=(10,6))
sns.heatmap(cm,
            annot=True,
            xticklabels=labels,
            yticklabels=labels,
            cmap="coolwarm",
            linewidths=0.5
           )
plt.title("Confusion Matrix Visualization")
plt.yticks(rotation=0)
plt.show()
```



Our model have classified most of the text description in the right classes, we notice that model have miss-classified 39 joy text description into love which is the highest miss-classification the model have done so far, however it is logical because joy and love would can have often mean same things.

Our model is doing fine with the testing dataset now its time how our model predicts with a complete new data

I have defined a function that will take take description from and outputs the emotions expressed by the model

In [50]:
```
def classify_emotions(model,message):

    for sentences in message:
        sentences=nltk.sent_tokenize(message)

        for sentence in sentences:
            words=re.sub("[^a-zA-Z]"," ",sentence)

            if words not in set(stopwords.words('english')):
                word=nltk.word_tokenize(words)
                word=" ".join(word)

    oneHot=[one_hot(word,n=vocab_size)]

    text=pad_sequences(oneHot,maxlen=sentence_len,padding='pre')

    predict_classes=model.predict(text)

    y_pred=[np.argmax(label) for label in predict_classes]

    if y_pred==[0]:
        print("It describes joy")

    elif y_pred==[1]:
        print("It describes sadness")

    elif y_pred==[2]:
        print("It describes anger")

    elif y_pred==[3]:
        print("It describes fear")

    elif y_pred==[4]:
        print("It describes love")

    elif y_pred==[5]:
        print("It describes surprise")
```

Lets feed two text description and lets see how my model predicts

In [51]:
```
emotion1="I have tried several times to get back with you, but this aint working with
```

The model classify emotion1 as an anger and we can agree with that

In [52]:
```
classify_emotions(model,emotion1)
```

```
It describes sadness
```