Sentiment Analysis

Patrick Herlihy

ID: 010440477

Date: 03/06/23

Research Question.

Using customer review data can we train a neural network model to accurately detect positive and negative sentiments in text?

Data Preparation Steps

- Read the .txt data into python and parse the strings into an array of reviews
- Concatenate and load the data into a Pandas Dataframe
- Explore the data, check for unique characters
- Remove punctuation from the reviews
- Convert all characters to lowercase
- Remove stopwords from the reviews using the list of stopwords from nltk.corpus
- Lemmatize the reviews using the nltk.stem WordNetLemmatizer
- Explore the cleaned data: generate wordclouds to review the most common words for each sentiment, find the length of the longest review, the distribution of review lengths of reviews using a histogram
- Tokenize the data, parsing each of the reviews into individual words
- Encode/Vectorize the tokenized vocabulary, converting each word to an individual integer
- Create a padded sequence for each of the reviews.
- Use SciKit Learns Train test split to split the padded sequences and associated sentiment values 20% Test, 80% train. When fitting the model to the RNN we used a 20% for validation.
- The full cleaned, train, and test datasets were exported to csv

In [1]:
```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#PreProcessing
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator # to create a Word Cloud
from PIL import Image # Pillow with WordCloud to image manipulation
from nltk.tokenize import word_tokenize

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import datetime

#Model Construction
import tensorflow.keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import LSTM,Dense, Dropout, SpatialDropout1D
from tensorflow.keras.layers import Embedding
from tensorflow.keras.callbacks import EarlyStopping

#Evaluation
from sklearn.datasets import make_classification
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

Load the data:

In [3]:
```
#Load the data into a pandas dataframe
path='/Users/herlihpj/Desktop/Data Analytics/D213 - Advanced Data Analytics/Task 2/Sentiment Labeled Sentences/'

#Function to load and parse the data from a .txt file
def load_the_data(file):
    with open(path+file) as raw_text:
        reviews = raw_text.readlines()
    for nums, data in enumerate(reviews):
        data=data.replace('\n','')
        splitlist=data.split('\t')
        splitlist[1]=int(splitlist[1])
```

```
        reviews[nums]=splitlist
    return(reviews)
#Load in the amazon data
amazon_reviews_df = pd.DataFrame(load_the_data('amazon_cells_labelled.txt'), columns=['Review', 'Sentiment'])
#Load in the IMBD data
imbd_reviews_df = pd.DataFrame(load_the_data('imdb_labelled.txt'), columns=['Review', 'Sentiment'])
#Load in the Yelp data
yelp_reviews_df = pd.DataFrame(load_the_data('yelp_labelled.txt'), columns=['Review', 'Sentiment'])

#Combine all 3 dataframes into one dataframe
all_reviews = pd.concat([amazon_reviews_df,imbd_reviews_df, yelp_reviews_df], ignore_index=True)
print('Shape: ',all_reviews.shape)
#Check for null values
print('Summary of Null: ')
print(all_reviews.isna().sum())

all_reviews.head()
```

```
Shape:  (3000, 2)
Summary of Null:
Review       0
Sentiment    0
dtype: int64
```

Out[3]:

	Review	Sentiment
0	So there is no way for me to plug it in here i...	0
1	Good case, Excellent value.	1
2	Great for the jawbone.	1
3	Tied to charger for conversations lasting more...	0
4	The mic is great.	1

Explore the data for unique characters:

In [4]:

```
def find_unique_chars(strings):
    # join all strings into one
    all_strings = ''.join(strings)
    # find unique characters using set()
    unique_chars = set(all_strings)
    return(unique_chars)
review_strings = all_reviews.Review.values
unique_review_chars=find_unique_chars(review_strings)
```

```
print('Unique Chars: ', unique_review_chars)
print('Total Unique Chars: ',len(unique_review_chars))
```

```
Unique Chars:  {'t', '.', 'e', 'C', '+', 'H', 'V', '-', 'l', 'S', ']', 'X', '—', 'E', 'J', '&', '5', '?', 'K', '4', 'v',
'j', 'o', ' ', '[', 'n', 'p', '/', '$', 'u', 'O', 'r', '#', ')', 'm', 'q', 'Y', 'N', '…', 'P', 'z', 'a', ',', '0', "'",
'L', 'i', ';', 's', 'W', '2', 'G', 'f', 'g', 'c', '6', 'ª', '%', '7', 'Â', '9', '(', 'D', 'U', 'B', 'A', '3', 'w', '*',
'Ã', 'Q', '8', 'y', '1', '©', 'T', '–', 'I', 'b', 'M', 'd', '"', 'F', 'h', '!', 'Z', 'R', 'x', '¥', 'k', ':'}
Total Unique Chars:  91
```

Remove Punctuation:

In [5]:
```
##Data Cleaning
def remove_punctuation(text):
    clean_text = "".join(letter for letter in text if letter not in ("?", ".", ";", ":",  "!",'"'))
    return clean_text

#remove punctuation
all_reviews['Review_Updated'] = all_reviews['Review'].apply(remove_punctuation)
```

Convert all Characters to lowercase:

In [6]:
```
#convert all characters to lowercase, helps reduce input vector
all_reviews['Review_Updated']=all_reviews['Review_Updated'].str.lower()
```

Remove Stopwords:

In [7]:
```
#Remove Stopwords
words_to_remove = stopwords.words('english')
all_reviews['Review_Updated'] = all_reviews['Review_Updated'].apply(lambda x: " ".join(x for x in x.split() if x not in
```

Lemmitize the reviews:

In [8]:
```
#LEmmitization
lemmatizer = WordNetLemmatizer()
# define a function to perform lemmatization on a string
def lemmatize_text(text):
    return ' '.join([lemmatizer.lemmatize(w, pos='v') for w in text.split()])
# apply the lemmatize_text function to the 'text' column
all_reviews['Review_Lemmatized'] = all_reviews['Review_Updated'].apply(lemmatize_text)
all_reviews.head(25)
```

Out[8]:

	Review	Sentiment	Review_Updated	Review_Lemmatized
0	So there is no way for me to plug it in here i...	0	way plug us unless go converter	way plug us unless go converter
1	Good case, Excellent value.	1	good case, excellent value	good case, excellent value
2	Great for the jawbone.	1	great jawbone	great jawbone
3	Tied to charger for conversations lasting more...	0	tied charger conversations lasting 45 minutesm...	tie charger conversations last 45 minutesmajor...
4	The mic is great.	1	mic great	mic great
5	I have to jiggle the plug to get it to line up...	0	jiggle plug get line right get decent volume	jiggle plug get line right get decent volume
6	If you have several dozen or several hundred c...	0	several dozen several hundred contacts, imagin...	several dozen several hundred contacts, imagin...
7	If you are Razr owner...you must have this!	1	razr owneryou must	razr owneryou must
8	Needless to say, I wasted my money.	0	needless say, wasted money	needle say, waste money
9	What a waste of money and time!.	0	waste money time	waste money time
10	And the sound quality is great.	1	sound quality great	sound quality great
11	He was very impressed when going from the orig...	1	impressed going original battery extended battery	impress go original battery extend battery
12	If the two were seperated by a mere 5+ ft I st...	0	two seperated mere 5+ ft started notice excess...	two seperated mere 5+ ft start notice excessiv...
13	Very good quality though	1	good quality though	good quality though
14	The design is very odd, as the ear "clip" is n...	0	design odd, ear clip comfortable	design odd, ear clip comfortable
15	Highly recommend for any one who has a blue to...	1	highly recommend one blue tooth phone	highly recommend one blue tooth phone
16	I advise EVERYONE DO NOT BE FOOLED!	0	advise everyone fooled	advise everyone fool
17	So Far So Good!.	1	far good	far good
18	Works great!.	1	works great	work great
19	It clicks into place in a way that makes you w...	0	clicks place way makes wonder long mechanism w...	click place way make wonder long mechanism wou...

	Review	Sentiment	Review_Updated	Review_Lemmatized
20	I went on Motorola's website and followed all ...	0	went motorola's website followed directions, c...	go motorola's website follow directions, could...
21	I bought this to use with my Kindle Fire and a...	1	bought use kindle fire absolutely loved	buy use kindle fire absolutely love
22	The commercials are the most misleading.	0	commercials misleading	commercials mislead
23	I have yet to run this new battery below two b...	1	yet run new battery two bars that's three days...	yet run new battery two bar that's three days ...
24	I bought it for my mother and she had a proble...	0	bought mother problem battery	buy mother problem battery

Explore the Cleaned dataset

In [9]:
```
#Check value counts of data
print(all_reviews.Sentiment.value_counts())

#Word Clouds
print('Positive Wordcloud')
positive_words = " ".join(review for review in all_reviews[all_reviews.Sentiment == 1]['Review_Lemmatized'])
wordcloud = WordCloud().generate(positive_words)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()

print('Negative Wordcloud')
negative_words = " ".join(review for review in all_reviews[all_reviews.Sentiment == 0]['Review_Lemmatized'])
wordcloud = WordCloud().generate(negative_words)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()

#Histogram of Sentence Lengths?
review_lengths = all_reviews['Review_Lemmatized'].apply(lambda x: len(x))
max_review_length=max(review_lengths)
print('Longest Review: ',max_review_length)
# Plot a histogram of the string lengths
plt.hist(review_lengths, bins=range(1, max(review_lengths) + 2))
plt.title('Histogram of Review Lengths After Cleaning')
plt.xlabel('Length')
```

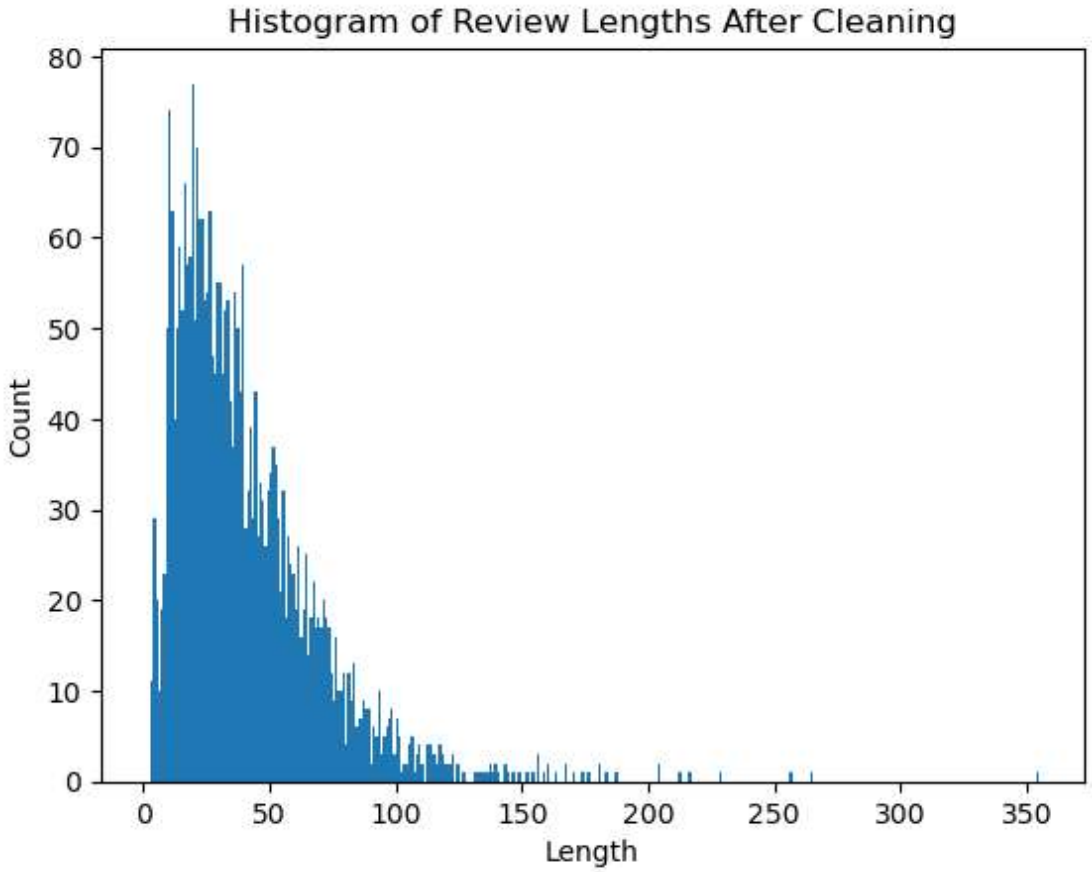
```
plt.ylabel('Count')
plt.show()
```

```
0    1500
1    1500
Name: Sentiment, dtype: int64
```

Positive Wordcloud

Negative Wordcloud


```
Longest Review:  354
```



In [11]:
```
#Store the sentiment Values
sentiment_label = all_reviews.Sentiment.values
sentiment_label
```

Out[11]:
```
array([0, 1, 1, ..., 0, 0, 0], dtype=int64)
```

Tokenize the data: Parse sentences into individual words

In [12]:
```
#Tokenize: Parse sentences into individual words
#Apply the tokenizer and pad to a max length
reviews_final = all_reviews.Review_Lemmatized.values
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(reviews_final)
#Size of the vocabulary
```

```
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary size: ',vocab_size) #Vocab size must be same size as tokenizer
```

```
Vocabulary size:  4684
```

Vectorize the data

In [13]:
```
#Vectorize: Encode words as integers
encoded_docs = tokenizer.texts_to_sequences(reviews_final)
```

Create padded sequences

In [14]:
```
#Add Padding: ensure equal length for tensorflow (array of zeros for word locations and integer)
padded_sequence = pad_sequences(encoded_docs, maxlen=max_review_length) #change to max sentence length?
#print(tokenizer.word_index)

#Review what a review looks like encoded and padded
print(reviews_final[0])
print(encoded_docs[0])
print(padded_sequence[0])
```

```
way plug us unless go converter
[54, 198, 102, 530, 10, 1944]
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0   54  198
  102  530   10 1944]
```

Split the data into Training and Test data

In [16]:

```
#Train/Test split the datasets:
X = padded_sequence
y = np.array(all_reviews.Sentiment)
X_train, y_train, X_label, y_label = train_test_split(X, y, test_size = 0.20, random_state = 10)

print('Reviews:')
print('Training data reviews shape = ', X_train.shape)
print('Test data reviews shape = ', y_train.shape)
print('Labels:')
print('Training data sentiments shape = ', X_label.shape)
print('Testing data setiments shape = ', y_label.shape)
```

```
Reviews:
Training data reviews shape =  (2400, 354)
Test data reviews shape =  (600, 354)
Labels:
Training data sentiments shape =  (2400,)
Testing data setiments shape =  (600,)
```

Export train, test, and full cleaned dataset to csv

In [19]:
```
#Save the datasets to csv?
all_reviews.to_csv(path+'Full Cleaned Dataset.csv')

# Convert X_train and y_train into DataFrames
X_train_df = pd.DataFrame(X_train)
y_train_df = pd.DataFrame(y_train)
X_test_df = pd.DataFrame(X_label)
y_test_df = pd.DataFrame(y_label)

#Export to CSV's
X_train_df.to_csv(path+'Training_data_reviews.csv')
X_test_df.to_csv(path+'Training_data_sentiments.csv')
y_train_df.to_csv(path+'Test_data_reviews.csv')
y_test_df.to_csv(path+'Test_data_sentiments.csv')

print('Data has been exported to csv.')
```

```
Data has been exported to csv.
```

Data has been prepared!

Begin Model Construction

In [25]:
```
print(X_train.shape[1]) #200
#embed_size = 128 #always us a factor of 2, 128 is a good start
#Build the model using LSTM
embedding_vector_length = 64
model = Sequential()
model.add(Embedding(vocab_size, embedding_vector_length, input_length=X_train.shape[1]) )
model.add(SpatialDropout1D(0.3))
model.add(LSTM(50, dropout=0.5, recurrent_dropout=0.5))
model.add(Dropout(0.25))
model.add(Dense(1, activation='sigmoid'))
#compile the model
```

```
model.compile(loss='binary_crossentropy',optimizer='adam', metrics=['accuracy'])
#summarize the model
print(model.summary())
```

```
354
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 354, 64)           299776

 spatial_dropout1d_1 (Spatia  (None, 354, 64)          0
 lDropout1D)

 lstm_1 (LSTM)               (None, 50)                23000

 dropout_1 (Dropout)         (None, 50)                0

 dense_1 (Dense)             (None, 1)                 51

=================================================================
Total params: 322,827
Trainable params: 322,827
Non-trainable params: 0
_________________________________________________________________
None
```

Fit the data to the Model

In [26]:
```
#Fit the model and try 7 epochs for model accuracy scores
early_stopping_monitor = EarlyStopping(patience=2) #helps prevent overfitting (2-3 common)
history = model.fit(X_train,X_label,validation_split=0.2, epochs=10, batch_size=32)
```

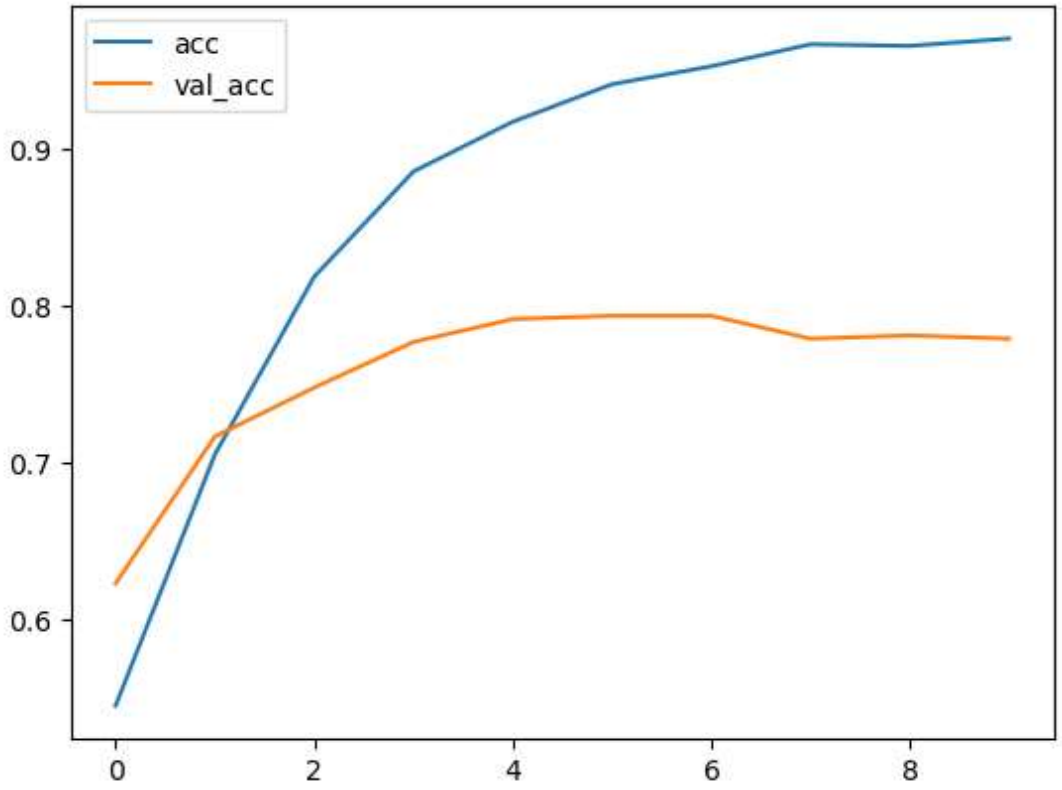
```
Epoch 1/10
60/60 [==============================] - 24s 351ms/step - loss: 0.6913 - accuracy: 0.5448 - val_loss: 0.6867 - val_accur
acy: 0.6229
Epoch 2/10
60/60 [==============================] - 22s 362ms/step - loss: 0.6548 - accuracy: 0.7052 - val_loss: 0.6054 - val_accur
acy: 0.7167
Epoch 3/10
60/60 [==============================] - 22s 367ms/step - loss: 0.4673 - accuracy: 0.8188 - val_loss: 0.5127 - val_accur
acy: 0.7479
Epoch 4/10
60/60 [==============================] - 23s 381ms/step - loss: 0.3021 - accuracy: 0.8859 - val_loss: 0.4742 - val_accur
acy: 0.7771
Epoch 5/10
60/60 [==============================] - 27s 453ms/step - loss: 0.2102 - accuracy: 0.9177 - val_loss: 0.4953 - val_accur
acy: 0.7917
Epoch 6/10
60/60 [==============================] - 28s 472ms/step - loss: 0.1524 - accuracy: 0.9417 - val_loss: 0.5178 - val_accur
acy: 0.7937
Epoch 7/10
60/60 [==============================] - 24s 393ms/step - loss: 0.1261 - accuracy: 0.9531 - val_loss: 0.5567 - val_accur
acy: 0.7937
Epoch 8/10
60/60 [==============================] - 23s 384ms/step - loss: 0.0936 - accuracy: 0.9672 - val_loss: 0.6141 - val_accur
acy: 0.7792
Epoch 9/10
60/60 [==============================] - 23s 387ms/step - loss: 0.0883 - accuracy: 0.9661 - val_loss: 0.6361 - val_accur
acy: 0.7812
Epoch 10/10
60/60 [==============================] - 21s 353ms/step - loss: 0.0869 - accuracy: 0.9708 - val_loss: 0.6370 - val_accur
acy: 0.7792
```

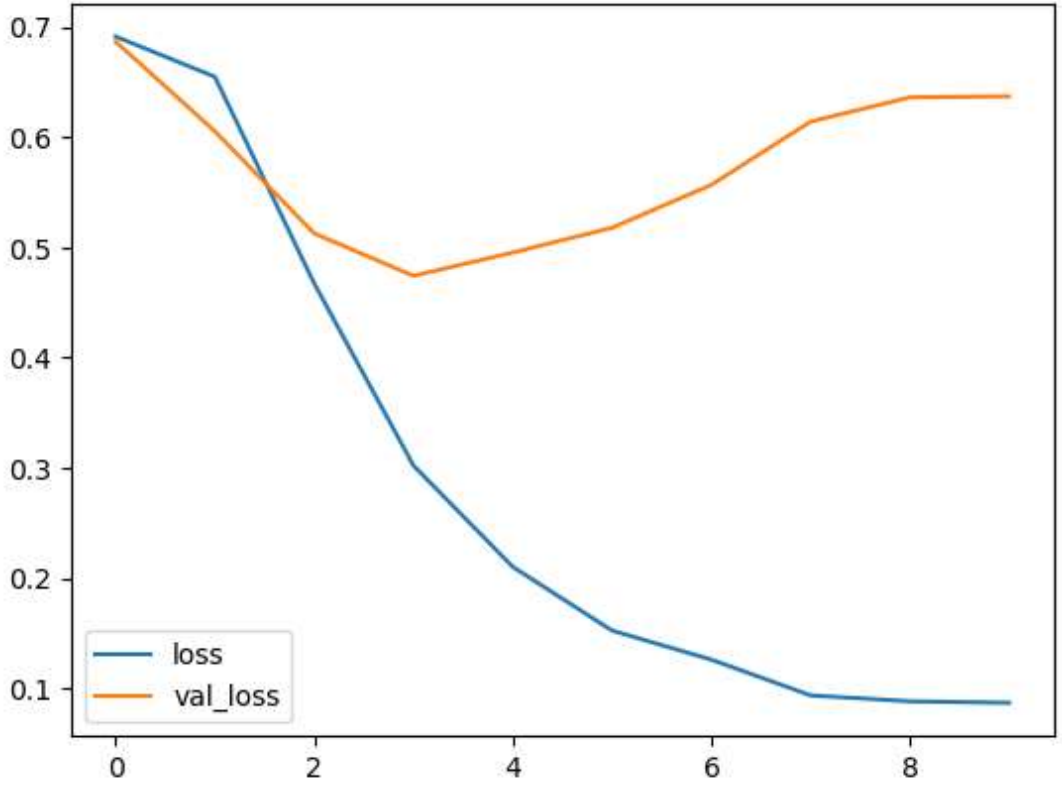
Generate graphs showing model accuracy and loss

In [27]:

```
#Plot the Model Accuracy and Loss
#Accuracy
plt.plot(history.history['accuracy'], label='acc')
plt.plot(history.history['val_accuracy'], label='val_acc')
plt.legend()
plt.show()

#Loss
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.legend()
plt.show()
```


acc
val_acc
0.9
0.8
0.7
0.6
0
2
4
6
8



Evaluate the model on the testing data

In [28]:
```
model.evaluate(y_train, y_label)
#78.7% accurate
```

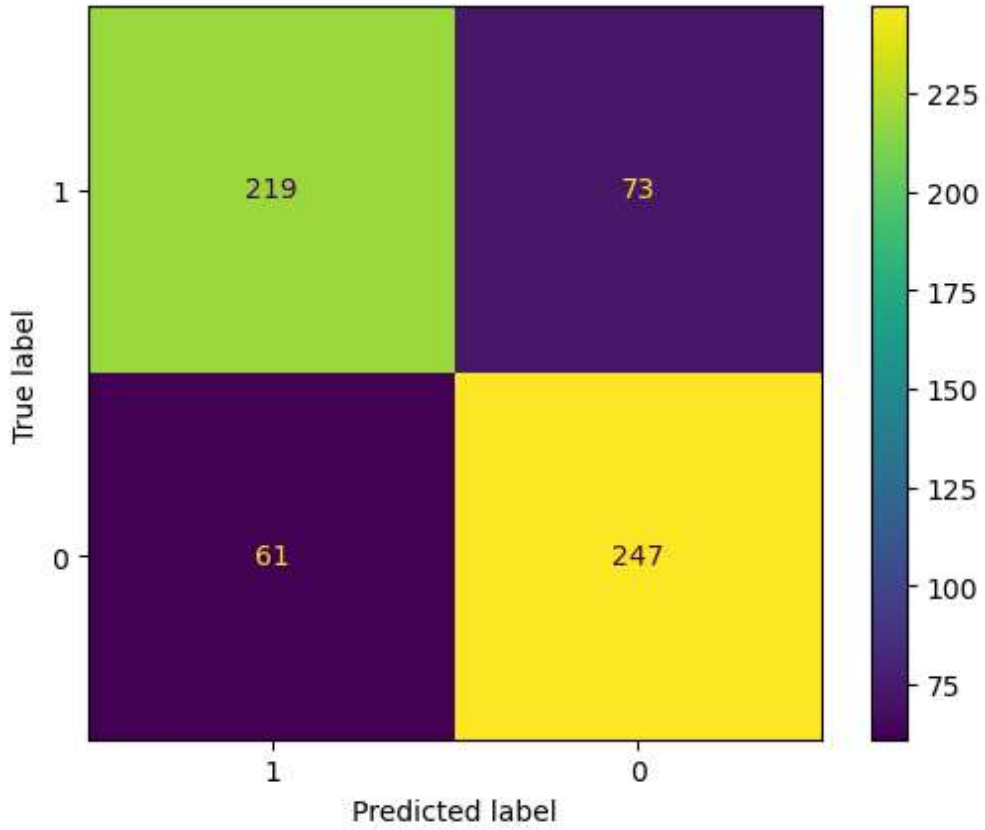
```
19/19 [==============================] - 2s 55ms/step - loss: 0.6855 - accuracy: 0.7767
```

Out[28]:
```
[0.6854796409606934, 0.7766666412353516]
```

Confusion Matrix of the Test Results

In [29]:
```
predictions = model.predict(y_train)
predictions = np.round(predictions,0).astype(int)
cm = confusion_matrix(y_label, predictions, labels=[1, 0])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[1, 0])
disp.plot()
plt.show()
```

```
19/19 [==============================] - 1s 49ms/step
```



Utilize the model to predict sentiment in future phrases

In [30]:
```
#Function to use the model to predict sentiment
def predict_sentiment(text):
    tw = tokenizer.texts_to_sequences([text])
    tw = pad_sequences(tw,maxlen=200)
    prediction = int(model.predict(tw).round().item())
    print(text)
    if prediction>0:
        print('POSITIVE SENTIMENT DETECTED! -', prediction)
    else:
        print('NEGATIVE SENTIMENT DETECTED! -', prediction)

test_sentence1 = "Lola (my dog) is a good girl!"
predict_sentiment(test_sentence1)
```

```
test_sentence1 = "Sedona is a movie rockstar!"
predict_sentiment(test_sentence1)
```

```
1/1 [==============================] - 0s 302ms/step
Lola (my dog) is a good girl!
POSITIVE SENTIMENT DETECTED! - 1
1/1 [==============================] - 0s 45ms/step
Sedona is a movie rockstar!
NEGATIVE SENTIMENT DETECTED! - 0
```

In [31]:
```
#Saving the Model
model.save(path+"Trained_Sentiment_Model.keras")
```

References

Data:

'From Group to Individual Labels using Deep Features', Kotzias et. al,. KDD 2015

Agrawal, S. (2021, April 8). Sentiment analysis using LSTM step-by-step. Medium. Retrieved March 6, 2023, from https://towardsdatascience.com/sentiment-analysis-using-lstm-step-by-step-50d074f09948