Natural Language Processing - Sentiment Analysis of Restaurant Reviews using Deep Learning

Team:
Sri Vathsava Bathini (1001948962)
Chandhana krishna vardhana Chinthakindi (1001990550)
Siva Keerthik Reddy Tagira (1001986529)

CSE-6363-002
Jesus Gonzalez

Project Description

- Sentiment analysis (or opinion mining) is a natural language processing (NLP) technique used to analyze the product sentiment in customer feedback and to understand customer needs.
- Sentiment analysis is commonly performed on text data which includes reviews, feedbacks, blogs etc. and this determines whether the data is positive, negative or neutral.
- The aim of our project is to perform sentiment analysis of reviews on restaurants
- In this project we are performing sentiment analysis using two deep learning methods CNN and LSTM

Data Set

- We have performed web scraping on tripadvisor website and obtained the dataset
- The dataset is primarily based on the reviews written by different customers
- The dataset contains a total of 907 reviews.
- The raw dataset contains redundant and unwanted information which have no significance in the analysis thus we perform data cleaning and preprocessing

Data Cleaning and Preprocessing

To achieve better results from the deep learning models the format of the data has to be in a proper manner. The process of converting data to something a computer can understand is referred to as pre-processing. Our data cleaning and preprocessing contains three mains steps.

1. Remove everything(special characters) except alphabets
2. Convert the data to lowercase
3. Stemming and Removing stop words from data

In [ ]:
```
pip install nltk
```

Load dataset

In [2]:
```
import pandas as pd
df = pd.read_csv("Team7_Restaurant_Reviews_Dataset.csv")
df.head()
```

Out[2]:

	Review	Rating
0	Just had a meal here -started badly when my dr...	0
1	There's no need for social distancing at Level...	0
2	We had lunch here because it was an easy walk ...	0
3	This restaurant doesnt really stand out for an...	0
4	Lured by the street menu, we visited this plac...	0

Dataset includes 500 positive and 407 negative reviews.

In [3]:
```
df["Rating"].value_counts()
```

Out[3]:
```
1    500
0    407
Name: Rating, dtype: int64
```

Data Cleaning and Preprocessing

Step :1

For removing everything except the alphabets regular expression is used and is defined such that lowercase alphabets, upper case alphabets are part of of the pattern and all the words have single spacing. This basically remove symbols and non alphabet expressions.

Step 2:

The dataset contains a mixture of upper case and lower case. All the data is converted to lowercase and is splitted

Data Cleaning and Preprocessing

Step 3: Stemming and Removing stop words from data.

- Stemming is the process of producing morphological variants of a root/base word.
- For example, stemming algorithm reduces the words “retrieval”, “retrieved”, “retrieves” to the root word “retrieve”.
- PorterStemmer Package is used from NLTK Library for stemming data.
- A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that are useless and are filtered in preprocessing.
- Stopwords Package is used from NLTK Library for removing stopwords.

```python
import nltk
nltk.download('stopwords')

import re
import numpy as np

from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()
stopwords_english = stopwords.words('english')
preprocessed_data = []

for x in range(0, df.shape[0]):
    #Step 1: Remove everything except letters
    reviews = re.sub('[^a-zA-Z]', ' ', df["Review"][x])

    #Step 2: Convert everything to lowercase
    reviews = reviews.lower()

    #Step 3: Stemming words with NLTK
    reviews = reviews.split()
    reviews = [stemmer.stem(i) for i in reviews if i not in set(stopwords_english)]
    reviews = ' '.join(reviews)
    preprocessed_data.append(reviews)
```

Screenshot

Below are the preprocessed tweets

In [6]:
```
preprocessed_data[:5]
```

Out[6]:
```
['meal start badli driver came collect nice touch criticis footwear car set tone well seafood chowder came tini bowl smatter fi
sh chunk probabl could pass seafood label accompani soup also serv half dozen oyster order came lemon dish balsam vinegar oyste
r could scallop tast virtual star attract scallop swim helplessli pool sauc ostens hide bland tast summis along chip salad offe
r ask want dessert whilst collect order bungl drink left order anoth tabl two collect girl possibl get order wrong left horribl
underwhelm waiter told prior eat restaur among best auckland case get plane go back home wasnt ask enjoy pay offer come fall de
af ear come auckland u better sort roadsid cafe masquerad restaur',
 'need social distanc level serv quickli waiter pleasant meal came quickli mani peopl dine room take away order steak main garl
ic bread recommend us ok drink expens',
 'lunch easi walk lodg menu look better two adjac place crowd aka theori must better sinc peopl well look deceiv food seafood c
howder garlic bread squash soup chicken okay special also servic slooooowwwww',
 'restaur doesnt realli stand anyth except fact order full meal menu pack takeaway pick pretti conveni steak meal takeaway alth
ough amaz conveni made',
 'lure street menu visit place found differ menu everi item higher price ask told post price street lower honor wonder price wo
uld charg']
```

Splitting Train and Test data

- The data which is preprocessed is split into test and training set which is helpful in prediction and find the accuracy of test data.
- We have split the data into 80% training and 20% testing sets.
- We used train_test_split package from sklearn library to split the data.
- Our dataset includes 907 total reviews out of which length of training set is 725 and length of testing set is 182 respectively.

Split train and test set: %80 Train, %20 Test

```
# labels
y = df.iloc[:,1].values
# preprocessed data
X = preprocessed_data
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0, stratify=y)
```

```
[ ] print("X_train len: ", len(X_train))
    print("y_train len: ", len(y_train))
    print("X_test len: ", len(X_test))
    print("y_test len: ", len(y_test))
```

```
X_train len:  725
y_train len:  725
X_test len:  182
y_test len:  182
```

Vectorization

- Tokenization is the process in which the sentence/text is split into array of words called tokens.
- We used Tokenizer to vectorize the text and convert it into sequence of integers after restricting the tokenizer to use only top most common 2500 words.
- We used pad_sequences to convert the sequences into 2D numpy array
- We performed tokenization on test and trained data set separately using keras

```python
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

# maximum words in the vocabulary
max_words = 2500
# maximum sequence length
max_len = 200

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X_train)
# create sequences of tokens representing each sentence
training_sequences = tokenizer.texts_to_sequences(X_train)
training_padded = pad_sequences(training_sequences, maxlen=max_len, truncating="post")
print("Training sequences:\n",training_padded)
# create sequences of tokens representing each sentence
testing_sequences = tokenizer.texts_to_sequences(X_test)
testing_padded = pad_sequences(testing_sequences, maxlen=max_len, truncating="post")
print("Testing sequences:\n",testing_padded)

X_train = training_padded
X_test = testing_padded
```

Output after Vectorization

```
training sequences:
 [[  0   0   0 ...  40  31  82]
 [  0   0   0 ...  24  11 142]
 [  0   0   0 ...  20  50 211]
 ...
 [  0   0   0 ... 244 280 272]
 [  0   0   0 ...  80  19 102]
 [  0   0   0 ...  30  99 271]]
testing sequences:
 [[  0   0   0 ...  24  18   7]
 [  0   0   0 ... 148  20 120]
 [  0   0   0 ... 150  28  57]
 ...
 [  0   0   0 ...  80  19 102]
 [  0   0   0 ... 379  11   7]
 [  0   0   0 ... 150  28  57]]
```

LSTM

- LSTM stands for long short term memory. It is an artificial recurrent neural network architecture.
- LSTM has a special architecture which enables it to forget the unnecessary information. This is achieved because the recurring module of the model has a combination of four layers
- Relu and Sigmoid activation functions are being used and for optimizer RMS Prop is being used

LSTM

- Binary cross entropy loss is being used for loss criterion.
- We have used Embedded Layer, 3 Lstm layers and 2 Dense layers
- Structure - embedded layer -> LSTM layers -> dense layer
- We will train the model for 100 epochs with the callback. We stored the accuracy, loss, precision, recall at each epoch in the history

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 200, 45)           112500

 lstm_9 (LSTM)               (None, 200, 256)          309248

 lstm_10 (LSTM)              (None, 200, 216)          408672

 lstm_11 (LSTM)              (None, 152)               224352

 dense_7 (Dense)             (None, 488)               74664

 dense_8 (Dense)             (None, 1)                 489

=================================================================
Total params: 1,129,925
Trainable params: 1,129,925
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
10/10 [==============================] - ETA: 0s - loss: 0.7454 - binary_accuracy: 0.6741 - precision: 0.7029 - recall: 0.6962
Epoch 1: val_binary_accuracy improved from -inf to 0.64138, saving model to best_model1.hdf5
10/10 [==============================] - 40s 3s/step - loss: 0.7454 - binary_accuracy: 0.6741 - precision: 0.7029 - recall: 0.6962 - val_loss: 1.2625 - val_binary_accuracy: 0.6414 - val_precision: 0.6176 - val_recall: 1.0000
Epoch 2/100
10/10 [==============================] - ETA: 0s - loss: 0.3903 - binary_accuracy: 0.8638 - precision: 0.8031 - recall: 0.9937
Epoch 2: val_binary_accuracy improved from 0.64138 to 0.90345, saving model to best_model1.hdf5
10/10 [==============================] - 32s 3s/step - loss: 0.3903 - binary_accuracy: 0.8638 - precision: 0.8031 - recall: 0.9937 - val_loss: 0.2874 - val_binary_accuracy: 0.9034 - val_precision: 0.8571 - val_recall: 1.0000
Epoch 3/100
10/10 [==============================] - ETA: 0s - loss: 0.2274 - binary_accuracy: 0.9172 - precision: 0.8941 - recall: 0.9620
Epoch 3: val_binary_accuracy improved from 0.90345 to 0.91724, saving model to best_model1.hdf5
10/10 [==============================] - 31s 3s/step - loss: 0.2274 - binary_accuracy: 0.9172 - precision: 0.8941 - recall: 0.9620 - val_loss: 0.1945 - val_binary_accuracy: 0.9172 - val_precision: 0.8830 - val_recall: 0.9881
Epoch 4/100
10/10 [==============================] - ETA: 0s - loss: 0.1240 - binary_accuracy: 0.9483 - precision: 0.9360 - recall: 0.9715
Epoch 4: val_binary_accuracy improved from 0.91724 to 0.93103, saving model to best_model1.hdf5
10/10 [==============================] - 32s 3s/step - loss: 0.1240 - binary_accuracy: 0.9483 - precision: 0.9360 - recall: 0.9715 - val_loss: 0.1529 - val_binary_accuracy: 0.9310 - val_precision: 0.9111 - val_recall: 0.9762
Epoch 5/100
10/10 [==============================] - ETA: 0s - loss: 0.0717 - binary_accuracy: 0.9741 - precision: 0.9688 - recall: 0.9842
Epoch 5: val_binary_accuracy did not improve from 0.93103
10/10 [==============================] - 32s 3s/step - loss: 0.0717 - binary_accuracy: 0.9741 - precision: 0.9688 - recall: 0.9842 - val_loss: 0.1545 - val_binary_accuracy: 0.9310 - val_precision: 0.9111 - val_recall: 0.9762
Epoch 6/100
10/10 [==============================] - ETA: 0s - loss: 0.0360 - binary_accuracy: 0.9862 - precision: 0.9783 - recall: 0.9968
Epoch 6: val_binary_accuracy did not improve from 0.93103
10/10 [==============================] - 32s 3s/step - loss: 0.0360 - binary_accuracy: 0.9862 - precision: 0.9783 - recall: 0.9968 - val_loss: 0.2044 - val_binary_accuracy: 0.9310 - val_precision: 0.9744 - val_recall: 0.9048
Epoch 7/100
10/10 [==============================] - ETA: 0s - loss: 0.0170 - binary_accuracy: 0.9948 - precision: 0.9937 - recall: 0.9968
Epoch 7: val_binary_accuracy improved from 0.93103 to 0.93793, saving model to best_model1.hdf5
10/10 [==============================] - 32s 3s/step - loss: 0.0170 - binary_accuracy: 0.9948 - precision: 0.9937 - recall: 0.9968 - val_loss: 0.2212 - val_binary_accuracy: 0.9379 - val_precision: 0.9310 - val_recall: 0.9643
Epoch 8/100
10/10 [==============================] - ETA: 0s - loss: 0.0168 - binary_accuracy: 0.9948 - precision: 0.9937 - recall: 0.9968
Epoch 8: val_binary_accuracy did not improve from 0.93793
10/10 [==============================] - 32s 3s/step - loss: 0.0168 - binary_accuracy: 0.9948 - precision: 0.9937 - recall: 0.9968 - val_loss: 0.2120 - val_binary_accuracy: 0.9172 - val_precision: 0.9091 - val_recall: 0.9524
Epoch 9/100
10/10 [==============================] - ETA: 0s - loss: 0.0072 - binary_acc[illegible]ecision: 1.0000 - recall: 0.9968
Epoch 9: val_binary_accuracy did not improve from 0.93793
```

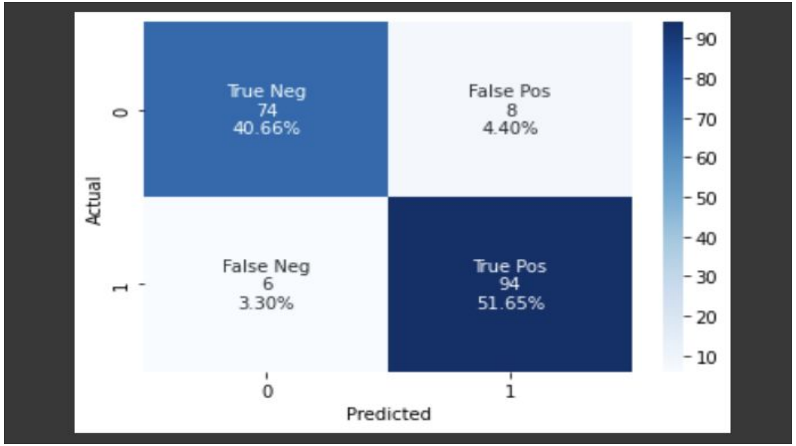
LSTM

Accuracy of test data set:

We achieved a validation accuracy of 92%

```
Evaluate on test data
3/3 [==============================] - 3s 829ms/step - loss: 0.4045 - binary_accuracy: 0.9231 - precision: 0.9216 - recall: 0.9400
test loss, test acc, test precion, test recall: [0.40448421239852905, 0.9230769276618958, 0.9215686321258545, 0.9399999976158142]
```

Confusion matrix for predictions:



CNN

- CNN stands for convolution neural network
- A CNN, in general, can be thought of as an artificial neural network with some type of specialization for being able to pick out or detect patterns
- Convolutional layers, which are hidden layers in CNNs, are what make a CNN, well, a CNN
- We will train the model for 70 epochs with the callback. We stored the accuracy, loss, precision, recall at each epoch in the history

CNN

- We are using 1 embedding layer, 2 conv1D layers, flatten and dense layers
- Relu activation and sigmoid activations are being used
- For loss criterion we are using binary entropy loss
- We are using model checkpoint to save and reuse the model later for higher validation accuracies
- Using early stopping, we strop training when a monitored metric has stopped improving

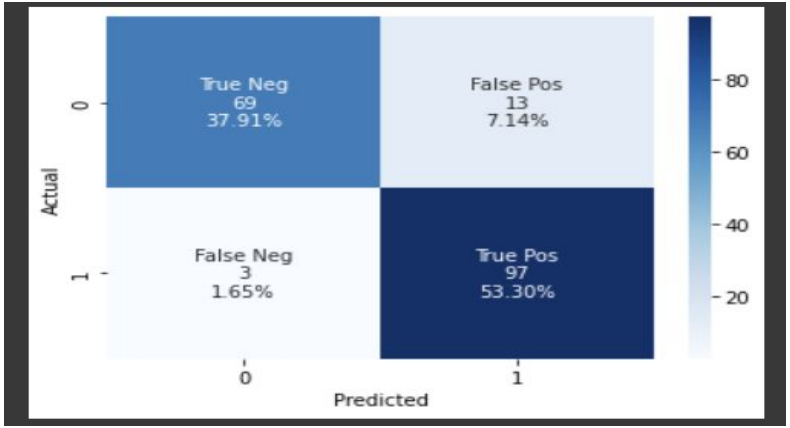
```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 200, 35)           87500

 conv1d_2 (Conv1D)           (None, 199, 38)           2698

 conv1d_3 (Conv1D)           (None, 192, 26)           7930

 flatten_1 (Flatten)         (None, 4992)              0

 dense_9 (Dense)             (None, 1)                 4993

=================================================================
Total params: 103,121
Trainable params: 103,121
Non-trainable params: 0
_________________________________________________________________
Epoch 1/70
19/19 [==============================] - ETA: 0s - loss: 0.6363 - binary_accuracy: 0.6942 - precision: 0.6455 - recall: 0.9760
Epoch 1: val_binary_accuracy improved from -inf to 0.69655, saving model to best_model3.hdf5
19/19 [==============================] - 2s 58ms/step - loss: 0.6363 - binary_accuracy: 0.6942 - precision: 0.6455 - recall: 0.9760 - val_loss: 0.5458 - val_binary_accuracy: 0.6966 - val_precision: 0.6562 - val_recall: 1.0000
Epoch 2/70
17/19 [=========================>....] - ETA: 0s - loss: 0.4667 - binary_accuracy: 0.7868 - precision: 0.7289 - recall: 0.9767
Epoch 2: val_binary_accuracy improved from 0.69655 to 0.90345, saving model to best_model3.hdf5
19/19 [==============================] - 1s 34ms/step - loss: 0.4556 - binary_accuracy: 0.7983 - precision: 0.7375 - recall: 0.9778 - val_loss: 0.3024 - val_binary_accuracy: 0.9034 - val_precision: 0.8571 - val_recall: 1.0000
Epoch 3/70
17/19 [=========================>....] - ETA: 0s - loss: 0.2308 - binary_accuracy: 0.8934 - precision: 0.8367 - recall: 0.9966
Epoch 3: val_binary_accuracy did not improve from 0.90345
19/19 [==============================] - 1s 32ms/step - loss: 0.2288 - binary_accuracy: 0.8948 - precision: 0.8418 - recall: 0.9937 - val_loss: 0.1823 - val_binary_accuracy: 0.8621 - val_precision: 0.8478 - val_recall: 0.9286
Epoch 4/70
17/19 [=========================>....] - ETA: 0s - loss: 0.1294 - binary_accuracy: 0.9706 - precision: 0.9484 - recall: 1.0000
Epoch 4: val_binary_accuracy did not improve from 0.90345
19/19 [==============================] - 1s 31ms/step - loss: 0.1264 - binary_accuracy: 0.9724 - precision: 0.9518 - recall: 1.0000 - val_loss: 0.1751 - val_binary_accuracy: 0.8759 - val_precision: 0.8511 - val_recall: 0.9524
Epoch 5/70
17/19 [=========================>....] - ETA: 0s - loss: 0.0790 - binary_accuracy: 0.9871 - precision: 0.9767 - recall: 1.0000
Epoch 5: val_binary_accuracy did not improve from 0.90345
19/19 [==============================] - 1s 31ms/step - loss: 0.0773 - binary_accuracy: 0.9862 - precision: 0.9753 - recall: 1.0000 - val_loss: 0.1834 - val_binary_accuracy: 0.8621 - val_precision: 0.8478 - val_recall: 0.9286
```

CNN

Accuracy of test data set:
We got a validation accuracy of 89%

```
Evaluate on test data
3/3 [==============================] - 0s 14ms/step - loss: 0.1780 - binary_accuracy: 0.8901 - precision: 0.8571 - recall: 0.9600
test loss, test acc, test precion, test recall: [0.17800481617450714, 0.8901098966598511, 0.8571428656578064, 0.9599999785423279]
```

Confusion matrix for predictions:



References

1. https://ieeexplore.ieee.org/document/8970976
2. https://ieeexplore.ieee.org/document/8850982
3. https://ieeexplore.ieee.org/document/9300098
4. https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47
5. https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
6. https://medium.com/@BishnoiAmit/convolution-nets-for-sentiment-analysis-dfda50768dfc

Thank you