Sentiment Analysis of UCI Data

Mike Mattinson

Western Governors University

Advanced Data Analytics – D213

Task 2: Sentiment Analysis

Dr. Festus Elleh

September 7, 2022

Revision 3

[Abstract 2](#_Toc113448340)

[Part I. Research Question 3](#_Toc113448341)

[Part II. Data Preparation 5](#_Toc113448342)

[Part III. Network Architecture 11](#_Toc113448343)

[(4) Parameters 12](#_Toc113448344)

[Part IV. Model Evaluation 14](#_Toc113448345)

[Part V. Summary and Recommendations 16](#_Toc113448346)

[Part VI. Reporting 18](#_Toc113448347)

[References 19](#_Toc113448348)

[Tables 22](#_Toc113448349)

[Figures 39](#_Toc113448350)

[Appendix A Python Code 52](#_Toc113448351)

Abstract

UCI sentiment will be analyzed and modeled using NLTK/NLP methods to classify positive or negative sentiment from combined Amazon, IMDB, Yelp dataset. Total number of records is 2,700, somewhat small for a robust model, but adequate for a simple model. Keras sequential neural network using embedding, dropout(0.5), flatten and dense(1) layers. Vocabulary (corpus) size approx. 5,000 words.

Keywords: Sentiment Analysis. NLTK. NLP. TensorFlow. Binary Classification. Logistic Classification. Keras Sequential. Embedding.

# Research Question

## Describe the purpose of this data analysis by doing the following:

### Summarize one research question that you will answer using neural network models and NLP techniques. Be sure the research question is relevant to a real-world organizational situation and sentiment analysis captured in your chosen dataset.

Can customer unstructured data reviews be used to model positive or negative sentiment?

### Define the objectives or goals of the data analysis. Be sure the objectives or goals are reasonable within the scope of the research question and are represented in the available data.

Use Keras neural network to model and predict positive and negative sentiment using a combined Amazon, IMDB and Yelp customer unstructured review dataset.

### Identify a type of neural network capable of performing a text classification task that can be trained to produce useful predictions on text sequences on the selected data set.

Keras sequential neural network using the following layers:

* Embedding layer
* Flattening layer
* Dense (1) layer

Using the following Python packages:

* import tensorflow as tf
* from tensorflow import keras
* from sklearn.model\_selection import train\_test\_split as tts
* from numpy import array
* from keras import models
* from keras import layers
* from keras import regularizers
* from sklearn.model\_selection import train\_test\_split
* from keras.preprocessing.text import Tokenizer
* from keras\_preprocessing.sequence import pad\_sequences
* import wordcloud
* from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
* import nltk
* from nltk.corpus import stopwords
* from nltk.tokenize import word\_tokenize
* from nltk.probability import FreqDist
* from nltk.stem import WordNetLemmatizer

Here are the versions of some of the important packages:

Text, letter

Description automatically generated

# Data Preparation

## Summarize data cleaning process by doing the following:

### Perform exploratory data analysis on the chosen dataset, and include an explanation of each of the following elements:

#### presence of unusual characters (e.g., emojis, non-English characters, etc.). Using the **remove\_punctuation** functiton to remove sentence punctuation :

* "?"
* "."
* ";"
* ":"
* "!"
* '"'
* ','

#### vocabulary size. Used the length of the lokenizer.word\_index to determine that the total vocabulary size is 4,425. This is identified in the model as the variable “**vocab\_size**”

#### proposed word embedding length. The optimum word embedding length is identified in the model as the variable “**input\_dim**” and is set to be the vocabulary size described above, input\_dim = 4,425.

#### statistical justification for the chosen maximum sequence length. The optimum maximum sequence length was determined through observations to be 256 and is identified in the model as the variable “**maxlen**”.

### Describe the goals of the tokenization process, including any code generated and packages that are used to normalize text during the tokenization process.

#### Tokenize sentences “First Tokenization”. In order for the stopwords, lemmazation, and infrequent words analysis to be effective, the initial data of sentence/text structure is tokenized into an array of words.

#### Tokenize words to numbers “Second Tokenization”. Then, just before the model is defined, the second tokenization is used to transform the word structure into an array of numbers. The model requires numbers to be able to model data.

### Explain the padding process used to standardize the length of sequences, including the following in your explanation:

#### Padding. if the padding occurs before or after the text sequence. The padding is applied after the second tokenization based on the “maxlen” variable.

#### a screenshot of a single padded sequence. Here is a screenshot of a single padded sequence:

A picture containing table

Description automatically generated

### Identify how many categories of sentiment will be used and an activation function for the final dense layer of the network.

#### sentiment categories. There are two (2) sentiment categories, positive sentiment indicated by an integer value of 1 and negative sentiment indicated by an integer value of 0.

#### activation function. The activation function used in the model is ‘sigmoid’. The sigmoid function is a simple case of the softmax multi-classification function. The sigmoid function takes any real number as input and outputs a value in the range of 0 to 1, the larger the input value, the closer the output will be to 1.

### Explain the steps used to prepare the data for analysis, including the size of the training, validation, and test set split.

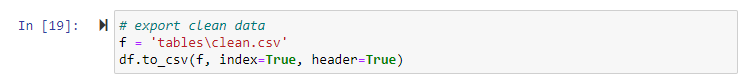
* + - * 1. Combine Data. Used pandas concat() function to combined the three (3) datasets.
        2. Punctuation. Used python join() function to remove a given list of punctuation characters.
        3. Lowercase. Used python’s lower() function to change the case of all text to lowercase.
        4. Word Tokenizer. Used nltk’s RegexpTokenizer() function to change the sentence structure to array of individual words.
        5. Stopwords. Used nltk’s stopwords.words(“english”) appended with a short list of my own stopwords, to go through and remove those words from the data.
        6. Remove Infrequent Words. Used nltk’s FreqDist() to compile and count all of words in the data. Then I removed all words based on a cutoff value. In this case, the cutoff value was set at 1, I did not want to remove any words from the already small dataset.
        7. Lemmatize. Used nltk’s WordNetLemmatizer() function to standardize the word tense.

### Provide a copy of the prepared dataset.

See Table 10. A copy of prepared dataset is attached to submission and is located in the “tables” folder.

A picture containing table

Description automatically generated



# Network Architecture

## Describe the type of network used by doing the following:

### Provide the output of the model summary of the function from TensorFlow.

See Figure 4. The final “best” model summary is as follows:

Chart

Description automatically generated

### Discuss the number of layers, the type of layers, and total number of parameters.

#### Embedding layer. The embedding layer is used to convert each word to a vector of defined size. “Embedding layer enables us to convert each word into a fixed length vector of defined size. The resultant vector is a dense one with having real values instead of just 0’s and 1’s. The fixed length of word vectors helps us to represent words in a better way along with reduced dimensions.” (Saxena, 2020)

#### Flattening layer. The flattening layer is used to reduce the dimension and shape of the input layer.

#### Dense layer. The dense layer is used to combine all of the available neurons in the model and shape the final output of the model.

#### Parameters. The parameters are sub-divided into trainable and non-trainable parameters. Depending on the vocabulary size, the tokenized word lengths and batch size, each model can have between thousands to millions of parameters. In this analysis, all of the parameters of all the candidate models were “trainable”. See Figure 5 for the number of parameters for each candidate model and Figure 6 for the number of parameters in the final “best” model.

### Justify the choice of hyperparameters, including the following elements:

#### activation functions. The sigmod activation function is commonly used for binary classification models. The activation function is specified in the dense layer.

#### number of nodes per layer. The model summary shows the number of parameters associated with each layer.

#### loss function. The loss function for the model is “binary crossentropy”. The loss function is specified in the model.compile() code as follows:

Graphical user interface, text

Description automatically generated

#### optimizer. The optimizer is “adam” and is also specified in the compile portion of the code as seen above.

#### stopping criteria. Not used. I ran multiple models of varying number of epochs. Then, looked at the outcome and made updates to the subsequent model based on those observations.

#### evaluation metric. The primary evaluation metric is the Accuracy percentage. The metric is obtained when running the test data through the prediction model.

# Model Evaluation

## Evaluate the model training process and its relevant outcomes by doing the following:

### Discuss the impact of using stopping criteria instead of defining the number of epochs, including a screenshot showing the final training epoch.

All of the candidate and final models were run using a specific number of epochs. All of the models, include some of the more advanced models, all ran in just around 1-5 minutes. Because of the size of the data and simplicity of the model, no stopping criteria as used.

### Provide visualizations of the model’s training process, including a line graph of the loss and chosen evaluation metric.

There are a number of model visualizations included with Figure 3. There is also an “Excel” table that summarizes some of the critical parameters. The “best model” and final model is included in Figure 4.

### Assess the fitness of the model and any measures taken to address overfitting.

Most all of the candidate models show some overfitting. Many different parameters and models were attempted in order to minimize the overfitting. Some models where attempted where the number of layers and neurons were increased, but in the end the simplest model with minimum number of neurons yielded the best results. In the best model, Figure 6, the number of epochs were reduced to approximately 10 epochs and this was sufficient to yield approximately 75% accuracy on test data.

After some time, I determined that the only way to get better metrics and reduce overfitting was to add a drop layer. With the dropout added, the overall accuracy increased, and the overfitting was reduced. The model summary showing the dropout layer:

Table

Description automatically generated

### Discuss the predictive accuracy of the trained network.

The final model yielded 82.6% accuracy on the testing data.

# Summary and Recommendations

## Provide the code used to save the trained network within the neural network.

“There are two formats you can use to save an entire model to disk: the TensorFlow SavedModel format, and the older Keras H5 format. The recommended format is SavedModel. It is the default when you use model.save(). (TensorFlow.org, 2022)“ I used the default format to save the model as follows:

Text, letter

Description automatically generated

## Discuss the functionality of your neural network, including the impact of the network architecture.

A lot of models were executed with and without dropout and in the end, the performance of the neural network (accuracy 82.6%) was really not that much better than the very simple logistic regression model (accuracy 81%).

The neural network models performed better with dropout layer added (accuracy 82.6%) than without dropout (accuracy 80%)

The combined dataset was relatively small, so the impact of this model design was negligible.

## Recommend a course of action based on your results.

Model training and performance accuracy should be able to achieve better results using a larger training dataset.

# Reporting

## Create your neural network using an industry-relevant interactive development environment (e.g., a Jupyter Notebook). Include a PDF or HTML document of your executed notebook presentation.

Attached Jupyter notebook. Also, copy of Python code from the notebook is included in Appendix A.

## List the web sources used to acquire data or segments of third-party code to support the application.

See references.

## Acknowledge sources, using in-text citations and references, for content that is quoted, paraphrased, or summarized.

See references.

## Demonstrate professional communication in the content and presentation of your submission.

References

Baka, B. (2017). *Python Data Structures and Algorithms: Improve application performance with graphs, stacks, and queues.* Packt Publishing - ebooks Account.

Bruce, P. C., Gedeck, P., Shmueli, G., & Patel, N. R. (2019). *Data Mining for Business Analytics Concepts, Techniques and Applications in Python.* Wiley & Sons, Incorporated, John.

Bruce, P., Bruce, A., & Gedeck, P. (2020). *Practical Statistics for Data Scientists: 50+ Essential Concepts Using R and Python.* O’Reilly Media Inc.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms.* The MIT Press.

Daityari, S. (2019). How To Perform Sentiment Analysis in Python 3 Using the Natural Language Toolkit (NLTK) | DigitalOcean. *How To Perform Sentiment Analysis in Python 3 Using the Natural Language Toolkit (NLTK) | DigitalOcean*. Retrieved July 24, 2022, from https://www.digitalocean.com/community/tutorials/how-to-perform-sentiment-analysis-in-python-3-using-the-natural-language-toolkit-nltk

Fenner, M. (2018). *Machine Learning with Python for Everyone.* Addison Wesley.

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media.

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2013). *Data Structures and Algorithms in Python.* Wiley.

Griffiths, D. (2009). *Head First Statistics.* O’Reilly Media Inc.

Griffiths, D. (2009). *Head First Statistics.* O’Reilly Media Inc.

Kumari, K. (2021, August). Text Preprocessing techniques for Performing Sentiment Analysis! *Text Preprocessing techniques for Performing Sentiment Analysis!* Retrieved August 12, 2022, from https://www.analyticsvidhya.com/blog/2021/08/text-preprocessing-techniques-for-performing-sentiment-analysis/

Larose, C. D., & Larose, D. T. (2019). *Data Science Using Python and R.* Wiley.

Li, S. (2018, June). A Beginner’s Guide on Sentiment Analysis with RNN. *A Beginner’s Guide on Sentiment Analysis with RNN*. Retrieved July 27, 2022, from https://towardsdatascience.com/a-beginners-guide-on-sentiment-analysis-with-rnn-9e100627c02e

Massaron, L., & Boschetti, A. (n.d.). *Regression Analysis with Python.* Packt Publishing - ebooks Account.

Mogyorosi, M. (n.d.). Sentiment Analysis: First Steps With Python's NLTK Library – Real Python. *Sentiment Analysis: First Steps With Python's NLTK Library – Real Python*. Retrieved July 26, 2022, from https://realpython.com/python-nltk-sentiment-analysis/

Selvaraj, N. (2020, September). A Beginner’s Guide to Sentiment Analysis with Python. *A Beginner’s Guide to Sentiment Analysis with Python*. Retrieved July 26, 2022, from https://towardsdatascience.com/a-beginners-guide-to-sentiment-analysis-in-python-95e354ea84f6

Sriniketh, J. (2021, June). Sentiment Analysis using NLTK - A Practical Approach. *Sentiment Analysis using NLTK - A Practical Approach*. Retrieved July 27, 2022, from https://www.analyticsvidhya.com/blog/2021/06/sentiment-analysis-using-nltk-a-practical-approach/

VanderPlas, J. (2016). *In Depth: Principal Component Analysis.* O'Reilly. Retrieved from https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html

Virahonda, S. (2020, October). An easy tutorial about Sentiment Analysis with Deep Learning and Keras. *An easy tutorial about Sentiment Analysis with Deep Learning and Keras*. Retrieved July 27, 2022, from https://towardsdatascience.com/an-easy-tutorial-about-sentiment-analysis-with-deep-learning-and-keras-2bf52b9cba91

Tables

[Table 1 *Raw data* 23](#_Toc113448352)

[Table 2 *Descriptive Statistics* 24](#_Toc113448353)

[Table 3 *Dataset after removing Punctuation* 25](#_Toc113448354)

[Table 4 *Dataset after converting to Lowercase* 26](#_Toc113448355)

[Table 5 *Dataset after first Tokenization* 27](#_Toc113448356)

[Table 6 *Dataset after removing Stopwords* 28](#_Toc113448357)

[Table 7 *Dataset after removing infrequent words* 29](#_Toc113448358)

[Table 8 *Dataset after applying Lemmatizer* 30](#_Toc113448359)

[Table 9 *Finding Most Common* 31](#_Toc113448360)

[Table 10 *Final Cleaned and Prepared Dataset* 32](#_Toc113448361)

[Table 11 *Finding “good” in negative sentiment* 33](#_Toc113448362)

[Table 12 *Finding “great” in negative sentiment* 34](#_Toc113448363)

[Table 13 *Finding “bad” in positive sentiment* 35](#_Toc113448364)

[Table 14 *Train Test Split* 36](#_Toc113448365)

[Table 15 *Second Tokenizer – Words (Series) -> Numbers (list of list)* 37](#_Toc113448366)

[Table 16 *Confusion Matrix from LogisticsRegression Model (Amazon dataset)* 38](#_Toc113448367)

Table   
Raw data

Text

Description automatically generated

Notes. Raw data showing 1,000 data values. The sentiment column ‘score’ is integer value of 1 for positive sentiment and 0 otherwise.

Table   
Descriptive Statistics

Text

Description automatically generated

Notes. The target variable is ‘label’. There are a total of 2,748 records. No missing data.

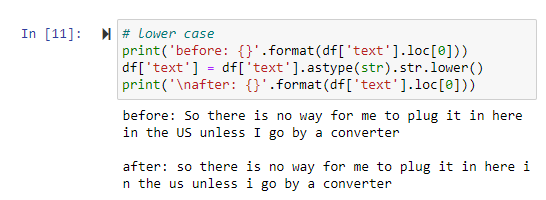
Table   
Dataset after removing Punctuation

Text, letter

Description automatically generated

Notes. Notice the sample showing before and after removing punctuation.

Table   
Dataset after converting to Lowercase



Notes. Sample showing before and after converting to lowercase.

Table   
Dataset after first Tokenization

Text, letter

Description automatically generated

Notes. df[‘text\_token’] now created as a pandas series of tokenized words that make up the original text.

Table   
Dataset after removing Stopwords

Text

Description automatically generated

Notes. Stopwords like [‘i’, ‘the’, ‘in’] etc., are removed and the sample text is displayed showing before and after.

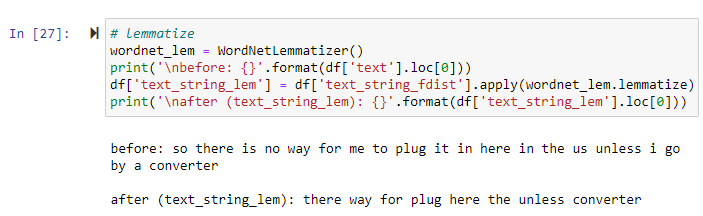
Table   
Dataset after removing infrequent words

Text

Description automatically generated

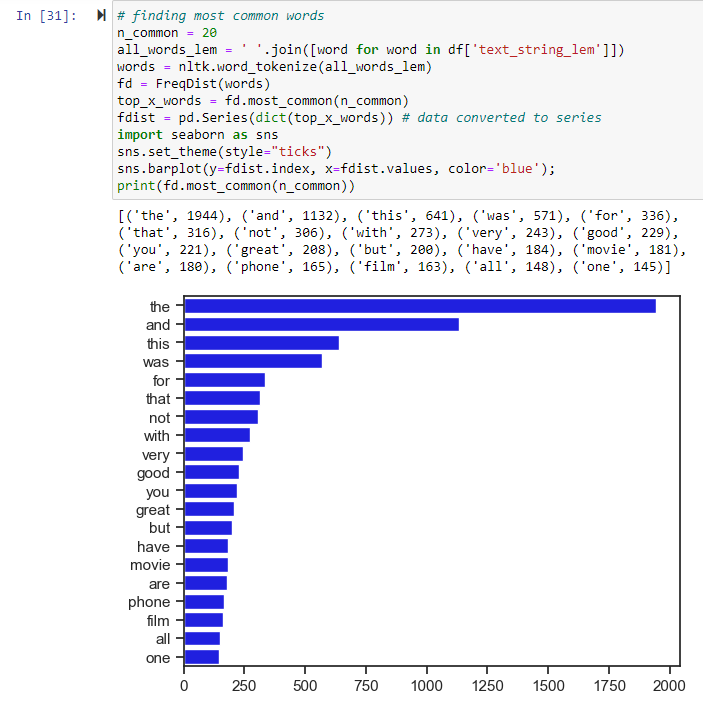
Notes. Text updated to remove infrequent words. In this case, the cutoff is set at 1, so there are no words meeting the criteria. The dataset is small and the analysis is limited based on the small size, so I did not want to remove any words from the analysis. But, this code might be helpful in the future when working with much larger datasets.

Table   
Dataset after applying Lemmatizer



Notes.

Table   
Finding Most Common



Notes. Most common word plotted as histogram.

Table   
Final Cleaned and Prepared Dataset

Text

Description automatically generated with low confidence

Notes. Final clean data is ready to model and saved in ‘tables’ folder.

Table   
Finding “good” in negative sentiment

Graphical user interface, application

Description automatically generated

Notes. The word “Good” was found in 13 rows where the sentiment was negative. It should be removed from the analysis because it will skew the results.

Table   
Finding “great” in negative sentiment

Text

Description automatically generated with medium confidence

Notes.

Table   
Finding “bad” in positive sentiment

Text, email

Description automatically generated

Notes.

Table   
Train Test Split

Text, letter

Description automatically generated

Notes. The model ready data is split into training and testing datasets. Notice the model will be using lemmatized text from the data cleaning and processing steps. Also note, all four (4) of the datasets are ‘Series’.

Table   
Second Tokenizer – Words (Series) -> Numbers (list of list)

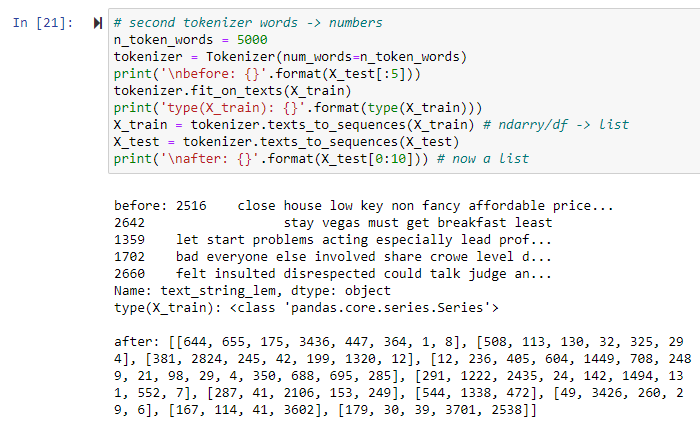
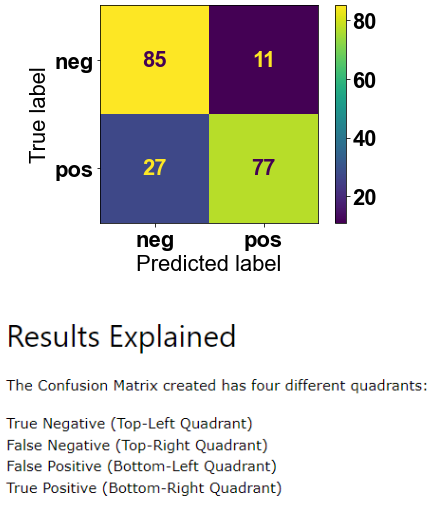


Table   
Confusion Matrix from LogisticsRegression Model (Amazon dataset)



Table

Description automatically generated

Notes. Overall accuracy is 81%. Early in the assignment, I performed a logistic regression on the Amazon dataset and this was the result.

Figures

[Figure 1 *Histogram of Scores* 40](#_Toc113448368)

[Figure 2 *Wordcloud* 41](#_Toc113448369)

[Figure 3 *Candidate Models* 42](#_Toc113448370)

[Figure 4 *Best Model* 51](#_Toc113448371)

Figure   
Histogram of Scores

Chart, bar chart

Description automatically generated

Notes. Histogram plot of raw data. Equally balanced between positive (label=1) and negative (label=0) sentiment.

Figure   
Wordcloud

Text

Description automatically generated

Notes. A wordcloud of all words regardless. Notice the words “good” and “great” appear in this wordcloud.

Figure   
Candidate Models



Notes. Table summarizes some of the best model runs. Cell is highlighted if value is different from the row above it.

Chart

Description automatically generated

Chart

Description automatically generated

Chart, line chart

Description automatically generated

Best Accuracy w/o Dropout

Chart, line chart

Description automatically generated

Chart

Description automatically generated with low confidence

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart

Description automatically generated

Figure   
Best Model

Best Model

Chart

Description automatically generated

Notes. Best model. The best model found included a dropout layer.

Appendix A Python Code

#!/usr/bin/env python

# coding: utf-8

# # D213 Task 2 Rev 3 - Mattinson

# ## imports

# In[1]:

# import required libraries

**import** tensorflow **as** tf

**from** tensorflow **import** keras

**import** numpy **as** np

**import** pandas **as** pd

**from** sklearn**.**model\_selection **import** train\_test\_split **as** tts

**from** numpy **import** array

**from** keras **import** models

**from** keras **import** layers

**from** keras **import** regularizers

**from** sklearn**.**model\_selection **import** train\_test\_split

**from** keras**.**preprocessing**.**text **import** Tokenizer

**from** keras\_preprocessing**.**sequence **import** pad\_sequences

**import** wordcloud

**from** wordcloud **import** WordCloud**,** STOPWORDS**,** ImageColorGenerator

**import** nltk

**from** nltk**.**corpus **import** stopwords

**from** nltk**.**tokenize **import** word\_tokenize

**from** nltk**.**probability **import** FreqDist

**from** nltk**.**stem **import** WordNetLemmatizer

get\_ipython**().**run\_line\_magic**(**'matplotlib'**,** 'inline'**)**

import matplotlib.pyplot as plt

print('tensorflow ver: {}'.format(tf.\_\_version\_\_))

print('nltk ver: {}'.format(nltk.\_\_version\_\_))

print('wordcloud ver: {}'.format(wordcloud.\_\_version\_\_))

print('numpy ver: {}'.format(np.\_\_version\_\_))

print('pandas ver: {}'.format(pd.\_\_version\_\_))

#print('matplotlib ver: {}'.format(plt.\_\_version\_\_))

# ## get data

# In[2]:

# read csv data

amazon = 'data/amazon\_cells\_labelled.txt'

imdb = 'data/imdb\_labelled.txt'

yelp = 'data/yelp\_labelled.txt'

colnames=['text', 'label']

amazon\_df = pd.read\_csv(amazon, sep='\t', names=colnames, header=None)

imdb\_df = pd.read\_csv(imdb, sep='\t', names=colnames, header=None)

yelp\_df = pd.read\_csv(yelp, sep='\t', names=colnames, header=None)

df = pd.concat([amazon\_df, imdb\_df, yelp\_df])

df = df.reset\_index(drop=True)

print('{}\n{}'.format(df.info(), df.shape))

df.sample(5, random\_state=0) # 5 random (0) rows of data

# In[3]:

# plot scores as bar plot

print(df['label'].value\_counts()) # output to notebook

pd.value\_counts(df['label']).plot.bar() # create plot

# In[4]:

# look at 'good' in a negative context

df[(df['text'].str.contains('good') >= 1) & (df['label'] == 0 )]

# In[5]:

# look at 'great' in a negative context

df[(df['text'].str.contains('great') >= 1) & (df['label'] == 0 )]

# In[6]:

# look at 'bad' in a positive context

df[(df['text'].str.contains('bad') >= 1) & (df['label'] == 1 )]

# ## explore data

# In[7]:

# descriptive stattics

print(type(df['label']))

print(df['label'].info())

df.describe()

# ## clean data

# In[8]:

# retype label data

# In[9]:

# remove punctuation

def remove\_punctuation(text: str) -> str:

'''remove punctuation from text'''

final = "".join(u for u in text if u not in (

"?", ".", ";", ":", "!", '"', ','))

return final # updated string

print('before: {}'.format(df['text'].loc[0]))

df['text'] = df['text'].apply(remove\_punctuation)

print('\nafter: {}'.format(df['text'].loc[0]))

# In[10]:

# lower case

print('before: {}'.format(df['text'].loc[0]))

df['text'] = df['text'].astype(str).str.lower()

print('\nafter: {}'.format(df['text'].loc[0]))

# In[11]:

# first tokenization

from nltk.tokenize import RegexpTokenizer

regexp = RegexpTokenizer('\w+')

print('before: {}'.format(df['text'].loc[0]))

df['text\_token']=df['text'].apply(regexp.tokenize)

print('\nafter: {}'.format(df['text\_token'].loc[0]))

# what is type of the new field

print('\ntext\_token type: {}'.format(type(df['text\_token'])))

# In[12]:

# remove stopwords

stopwords = nltk.corpus.stopwords.words("english")

print(stopwords[0:20]) # just first 20 stopwords...

#my\_stopwords = ['https', 'good', 'great', 'bad']

my\_stopwords = ['https']

stopwords.extend(my\_stopwords)

print('\nbefore: {}'.format(df['text'].loc[0]))

df['text\_token'] = df['text\_token'].apply(

lambda x: [item for item in x if item not in stopwords])

print('\nafter: {}'.format(df['text\_token'].loc[0]))

# In[13]:

# remove infrequent words

df['text\_string'] = df['text\_token'].apply(

lambda x: ' '.join([item for item in x if len(item)>2]))

all\_words = ' '.join([word for word in df['text\_string']])

tokenized\_words = nltk.tokenize.word\_tokenize(all\_words)

from nltk.probability import FreqDist

fdist = FreqDist(tokenized\_words)

print(fdist)

cutoff = 1 # drop words occurring less than certain amount

print('\nbefore: {}'.format(df['text'].loc[0]))

df['text\_string\_fdist'] = df['text\_token'].apply(

lambda x: ' '.join([item for item in x if fdist[item] >= cutoff ]))

print('\nafter (text\_string): {}'.format(df['text\_string'].loc[0]))

print('\nafter (text\_string\_fdist): {}'.format(df['text\_string\_fdist'].loc[0]))

# In[14]:

# lemmatize

wordnet\_lem = WordNetLemmatizer()

print('\nbefore: {}'.format(df['text'].loc[0]))

df['text\_string\_lem'] = df['text\_string\_fdist'].apply(wordnet\_lem.lemmatize)

print('\nafter (text\_string\_lem): {}'.format(df['text\_string\_lem'].loc[0]))

# In[15]:

# Defining our word cloud drawing function

# adapted from Assaker (2022)

# https://github.com/JosephAssaker/Twitter-Sentiment-Analysis-Classical-Approach-VS-Deep-Learning/blob/master/Twitter%20Sentiment%20Analysis%20-%20Classical%20Approach%20VS%20Deep%20Learning.ipynb

def plot\_wordcloud(title: str, data, color = 'black'):

print(title) # output to notebook

wordcloud = WordCloud(stopwords = STOPWORDS,

background\_color = color,

width = 2500,

height = 2000

).generate(' '.join(data))

plt.figure(1, figsize = (13, 13))

plt.imshow(wordcloud)

plt.axis('off')

plt.show() # create output plot

# In[16]:

# finding most common words

n\_common = 20

all\_words\_lem = ' '.join([word for word in df['text\_string\_lem']])

words = nltk.word\_tokenize(all\_words\_lem)

fd = FreqDist(words)

top\_x\_words = fd.most\_common(n\_common)

fdist = pd.Series(dict(top\_x\_words)) # data converted to series

import seaborn as sns

sns.set\_theme(style="ticks")

sns.barplot(y=fdist.index, x=fdist.values, color='blue');

print(fd.most\_common(n\_common))

# https://www.kirenz.com/post/2021-12-11-text-mining-and-sentiment-analysis-with-nltk-and-pandas-in-python/text-mining-and-sentiment-analysis-with-nltk-and-pandas-in-python/

# In[17]:

# wordcloud

wordcloud = WordCloud(width=600,

height=400,

random\_state=2,

max\_font\_size=100).generate(all\_words\_lem)

plt.figure(figsize=(10, 7))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis('off');

# ## export clean data

# In[18]:

# review what the data looks like after cleaning

print('{}\n{}'.format(df.info(), df.shape))

df.sample(3, random\_state=0) # 5 random (0) rows of data

# In[19]:

# export clean data

f = 'tables\clean.csv'

df.to\_csv(f, index=True, header=True)

# ## train test split

# https://www.kaggle.com/code/arunkumarramanan/awesome-ml-and-text-classification-movie-reviews

# ### seed=

# ### test\_split=

# In[20]:

# train test split

X = df['text\_string\_lem']

y = df['label']

seed = 42 # try different seeds

test\_split = 0.05 # 0.2 best so far

X\_train, X\_test, y\_train, y\_test = tts(X, y,

test\_size=test\_split, random\_state=seed)

print(X\_train[0:3]) # df['text\_string\_lem']

print('X\_train shape-type: {}-{}'.format(X\_train.shape, type(X\_train)))

print('X\_test shape: {}'.format(X\_test.shape))

print('y\_train shape-type: {}-{}'.format(y\_train.shape, type(y\_train)))

print('y\_test shape: {}'.format(y\_test.shape))

# ## model #1 - keras(Sequential)

# ### n\_token\_words =

# In[21]:

# second tokenizer words -> numbers

n\_token\_words = 4425 # best so far = 5000

tokenizer = Tokenizer(num\_words=n\_token\_words)

#print('\ntype: {}\nbefore:\n{}'.format(type(X\_test), X\_test[0]))

tokenizer.fit\_on\_texts(X\_train)

X\_train = tokenizer.texts\_to\_sequences(X\_train) # ndarry/df -> list

X\_test = tokenizer.texts\_to\_sequences(X\_test)

#print('\ntype: {}\nafter:\n{}'.format(type(X\_test), X\_test[0])) # now a list

# In[22]:

print(type(X\_test))

# In[23]:

#X\_train[0:3] # tokenized

# ### vocab\_size =

# ### maxlen =

# In[24]:

# Adding 1 because of reserved 0 index

vocab\_size = len(tokenizer.word\_index) + 1

maxlen = 64

X\_train = pad\_sequences(X\_train, padding='post', maxlen=maxlen)

X\_test = pad\_sequences(X\_test, padding='post', maxlen=maxlen)

np.set\_printoptions(threshold=np.inf)

print('vocab\_size: {}'.format(vocab\_size))

print('maxlen: {}'.format(maxlen))

X\_test[0] # now a padded list

# In[25]:

# reset options

#pd.reset\_option('all')

# In[26]:

#X\_train[0:3] # padded

# ### dropout =

# ### output\_dim =

# In[27]:

# define model

dropout = 0.4 # use dropout = 0 to specify not dropout layer

output\_dim = 2000 # vocab\_size # 1-1 mapping to vocab word

model = models.Sequential()

model.add(layers.Embedding(input\_dim=vocab\_size, output\_dim=output\_dim, input\_length=maxlen))

if(dropout > 0):

model.add(layers.Dropout(dropout))

model.add(layers.Flatten())

model.add(layers.Dense(1, activation='sigmoid'))

print(model.summary())

# In[28]:

# compile model

model.compile(

optimizer='adam',

loss='binary\_crossentropy',

metrics=['acc'])

# In[29]:

# save model in SavedModel format

# prior to saving the model, you need to compile the model

from datetime import datetime

now = datetime.now() # current date and time

date\_time\_stamp = now.strftime("\_%y%m%d\_%H%M")

model.save('models/final' + date\_time\_stamp)

# ### val\_split =

# In[30]:

val\_split = .2 # .3 or .4 working best so far

len(X\_train)

val\_split = int(val\_split \* len(X\_train))

x\_val = X\_train[:val\_split]

partial\_x\_train = X\_train[val\_split:]

y\_val = y\_train[:val\_split]

partial\_y\_train = y\_train[val\_split:]

# ### batch\_size =

# ### n\_epochs =

# In[31]:

batch\_size = 32 # 256 best so far

n\_epochs = 300 # 100-200 best so far

history = model.fit(partial\_x\_train,

partial\_y\_train,

batch\_size=batch\_size,

epochs=n\_epochs,

verbose=0,

validation\_data=(x\_val, y\_val))

# "Usually training should be better than validation..."

# validation loss goes down but then increases - overfit

# ## custom\_loss\_acc\_plot

# In[32]:

import matplotlib.pyplot as plt

import matplotlib.axes as ax

# adapted from Assaker (2022)

def custom\_loss\_acc\_plot(

ax: ax,

hist: dict,

title: str,

n\_epochs: int,

batch\_size: int,

vocab\_size: int,

output\_dim: int,

test\_split: int,

val\_split: int,

maxlen: int,

seed: int,

summary: str,

top: int,

score: np.ndarray,

n\_token\_words: int,

dropout: float

) -> ax:

"""

custom subplot returns

"""

# plot loss on axis=0

y1 = hist['loss']

y2 = hist['val\_loss']

x = range(1, len(y1) + 1) # x-axis = Epochs

ax[0].plot(x, y1, 'b+', label='Training loss')

ax[0].plot(x, y2, 'b', label='Validation loss')

ax[0].set\_title('Loss')

ax[0].text(.05 \* n\_epochs, top - .5, 'n\_epochs: ' + str(n\_epochs), fontsize=10)

ax[0].text(.05 \* n\_epochs, top - .8, 'batch\_size: ' + str(batch\_size), fontsize=10)

ax[0].text(.05 \* n\_epochs, top - 1.1, 'vocab\_size: ' + str(vocab\_size), fontsize=10)

ax[0].text(.05 \* n\_epochs, top - 1.4, 'test\_split: ' + str(test\_split), fontsize=10)

ax[0].text(.05 \* n\_epochs, top - 1.7, 'val\_split: ' + str(val\_split), fontsize=10)

ax[0].text(.05 \* n\_epochs, top - 2.0, 'maxlen: ' + str(maxlen), fontsize=10)

ax[0].text(.05 \* n\_epochs, top - 2.3, 'seed: ' + str(seed), fontsize=10)

ax[0].text(.05 \* n\_epochs, top - 2.6, 'test scores: ' + str(score), fontsize=10)

ax[0].text(.05 \* n\_epochs, top - 2.9, 'output\_dim: ' + str(output\_dim), fontsize=10)

ax[0].text(.05 \* n\_epochs, top - 3.2, 'n\_token\_words: ' + str(n\_token\_words), fontsize=10)

if(dropout > 0):

ax[0].text(.05 \* n\_epochs, top - 3.5, 'dropout: ' + str(dropout), fontsize=10)

ax[0].grid(True)

ax[0].axis('on')

ax[0].set\_ylim(0,5)

#ax[0].set\_ylim(0,1)

#ax[0].yaxis.set\_major\_locator((integer=True))

ax[0].legend()

# plot acc on axis=1

y1 = hist['acc']

y2 = hist['val\_acc']

x = range(1, len(y1) + 1) # x-axis = Epochs

ax[1].plot(x, y1, 'b+', label='Training acc')

ax[1].plot(x, y2, 'b', label='Validation acc')

ax[1].set\_title('Accuracy')

ax[1].set\_xlabel('Epoch')

ax[1].grid(True)

ax[1].axis('on')

ax[1].legend()

# plot model summary on axis=2

ax[2].text(0, -.2, summary, fontsize=10)

ax[2].grid(False)

ax[2].axis('off')

return (ax)

title = 'Loss-Accuracy'

fig, ax = plt.subplots(3, sharex=False, figsize=(7,10))

stringlist = []

model.summary(print\_fn=lambda x: stringlist.append(x))

short\_model\_summary = "\n".join(stringlist)

score = model.evaluate(X\_test, y\_test, verbose=0)

top = 5

custom\_loss\_acc\_plot(

ax,

history.history,

title,

n\_epochs,

batch\_size,

vocab\_size,

output\_dim,

test\_split,

val\_split,

maxlen,

seed,

short\_model\_summary,

top,

score,

n\_token\_words,

dropout

)

from datetime import datetime

now = datetime.now() # current date and time

title += now.strftime("\_%y%m%d\_%H%M")

ax[0].text(.05 \* n\_epochs, top, title, fontsize=12)

fig.savefig('figures\\' + title, dpi=150)

plt.close()

# ## end of notebook

# In[33]:

# beeps to indicate end of notebook

import winsound

n\_beeps = int((score[1]\*10-5))

for i in range(5):

winsound.Beep(700, 100)

for i in range(n\_beeps):

winsound.Beep(500, 200)

# In[ ]: