Capstone Project – 006

**Multi-Class Text Sentiment Analysis**

(using Python)

By

**Abhiteja Achanta, M13023084**

MS Business Analytics, University of Cincinnati

**First Reader: Dr. Yichen Qin**

**Second Reader: Dr. Liwei Chen**

**Abstract**

The goal of sentiment analysis is to extract human emotions from text. This project applies various machine learning algorithms to predict sentiment of reviewer from his textual review on Amazon food products. Metrics such as accuracy of prediction and precision/recall are presented to gauge the success of these different algorithms. Main purpose of this project is to introduce and apply different feature engineering techniques to convert text to numeric data and see how different Machine learning and Deep Learning algorithms perform with this data.

We have built following models:

1. Multinomial Naïve Bayes

2. Logistic Regression

3. Random Forest

4. Xgboost Classifier

5. Dense Neural Network

6. Recurrent Neural Network (LSTM)

7. Convolutional Neural Network (CNN)

**Table 4.1 Comparison of different classifiers with different embeddings**

A screenshot of a cell phone

Description automatically generated

**Table of Contents**

1. Abstract………………………………………………………………………………………………….1
2. Introduction……………………………………………………………………………………………3
3. Objective………………………………………………………………….…………………………….3
4. Exploratory Data Analysis……………………………………………………………………….3
   1. Data Sources…………………………………………………………………………………….3
   2. Data Definition…………………………………………………………………………………4
   3. Data Wrangling…………………………………………………………………………………4
   4. Data Visualization…………………………………………………………………………….4
   5. Data Cleaning…………………………………………………………………………………….7
5. Feature Engineering…………………………………………………………………………………7
   1. Count Vectorizer……………………………………………………………………………….7
   2. TF-IDF Vectorizer……………………………………………………………………………….8
   3. Word2Vec………………………………………………………………………………………….8
6. Model Building and evaluation…………………………………………………………………9
   1. Machine Learning…….…………………………………………………………………………9
   2. Deep Learning………………………………………………………………………………......17
7. Conclusion………………………………………………………………….…………………………….20
8. References………………………………………………………………………………………………..20
9. Code………………………………………………………………………………………………………….25
10. **Introduction:**

Text sentiment analysis is an important research topic for its wide applicability in real-world applications, and recent breakthroughs in text embedding and classification models led to state-of-the-art results. This project aims to apply recent innovations in machine learning to Amazon reviews. Different types of ML/DL models are applied to text data like Naïve Bayes, Logistic Regression, Random Forest, Xgboost Classifier, Dense Neural Network, Recurrent Neural Network (LSTM), Convolutional Neural Network (CNN).

1. **Objective:**

The objective of this project is to predict one of the three (positive, neutral, negative) sentiment classes given an Amazon food review.

1. **Exploratory Data Analysis:**

This is the first step in any data analysis project. We need to completely understand the data even before we implement machine learning models on the data.

**3.1 Data Sources:**

This dataset consists of reviews of fine foods from amazon. The data span a period of more than 10 years, including all ~500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories.

Data includes:

* Reviews from Oct 1999 - Oct 2012
* 568,454 reviews
* 256,059 users
* 74,258 products
* 260 users with > 50 reviews

We are going to use the subset of this dataset with close to 100,000 reviews due to the memory and capacity constraint of the machine.

* 1. **Data Definition:**
* **Id:** Row Id.
* **ProductId:** Unique identifier for the product.
* **UserId:** Unique identifier for the user.
* **ProfileName:** Profile name of the user.
* **HelpfulnessNumerator:** Number of users who found the review helpful.
* **HelpfulnessDenominator:** Number of users who indicated whether they found the review helpful or not.
* **Score:** Rating between 1 and 5.
* **Time:** Timestamp for the review.
* **Summary:** Brief summary of the review.
* **Text:** Text of the review.
  1. **Data Wrangling:**

We perform couple of data wrangling tasks on the actual dataset to get the dataset into desired format for our analysis and modelling. Following manipulations are carried out on original dataset:

* Remove unnecessary columns from dataset. We only need Text and score columns.
* Remove the rows with missing values.
* Remove the duplicate rows in the dataset.
* Change the Score column such that ratings less than 3 are converted to 0, ratings with 3 are converted to 1 and ratings greater than 3 are converted to 2.
* Hence, we have 3 classes (positive-2, neutral-1, negative-0).
* After performing above tasks, take the subset of data with similar percentages of three classes to carry out further analysis and modelling because of memory and capacity constraint.
  1. **Data Visualization:**

Using the dataset, we build couple of visualizations to get more understanding of the data. As we have text data, I made use of word clouds to highlight the words present each category of reviews.

**Fig 3.1 Number of reviews by category**

**A screenshot of a cell phone

Description automatically generated**

* Most of the reviews are positive.
* From the above chart, we can confirm this is highly imbalanced dataset.
* We can use techniques like under sampling or over sampling to deal with this issue.

**Fig 3.2 Distribution of length of reviews**

**A screenshot of a social media post

Description automatically generated**

* We can observe that above distribution is right skewed.

**Fig 3.3 Mean length of comments by category**

**A screenshot of a cell phone

Description automatically generated**

* Length of neutral reviews is longer on average compared to positive, negative reviews.
  1. **Data Cleaning:**

Before the feature engineering and modelling, we clean the text data after examining couple of reviews from each category. Following data cleaning steps are carried out:

* Converting entire text into Lowercase to avoid considering single word as two different words.
* Remove new line characters.
* Converting words like don't to do not using custom dictionary.
* Remove all the html tags like the one in the above neutral review.
* Remove all the punctuation marks.
* Remove all the numeric characters from the text.
* Remove the stop words like a, the etc. for the better performance of model.
* Carry out lemmatization to convert word like rocks to rock.

1. **Feature Engineering:**

Now that we have cleaned text, next step is to convert text data to numeric data which can be fed to Machine Learning/ Deep Learning algorithms. There are multiple techniques to accomplish this task. We will explore three major techniques in our project.

**4.1 Count Vectorizer:**

To use textual data for predictive modeling, the text must be parsed to remove certain words – this process is called **tokenization**. These words need to then be encoded as integers, or floating-point values, for use as inputs in machine learning algorithms. This process is called **feature extraction (or vectorization)**.

Scikit-learn’s Count Vectorizer is used to convert a collection of text documents to a vector of term/token counts. It also enables the ​pre-processing of text data prior to generating the vector representation. This functionality makes it a highly flexible feature representation module for text.

A screenshot of a cell phone

Description automatically generated

* 1. **TF-IDF Vectorizer:**

One issue with simple counts is that some words like “*the*” will appear many times and their large counts will not be very meaningful in the encoded vectors.

An alternative is to calculate word frequencies, and by far the most popular method is called [TF-IDF](https://en.wikipedia.org/wiki/Tf%E2%80%93idf). This is an acronym than stands for “*Term Frequency – Inverse Document*” Frequency which are the components of the resulting scores assigned to each word.

* **Term Frequency**: This summarizes how often a given word appears within a document.
* **Inverse Document Frequency**: This downscales words that appear a lot across documents.

To put it in more formal mathematical terms, the TF-IDF score for the word t in the document d from the document set D is calculated as follows:



Where:





We can use TfidfVectorizer function from sklearn to perform this task.

* 1. **Word2Vec**

Word2Vec is a type of word embedding. Word embedding is a form of representing words and documents using a dense vector representation. The position of a word within the vector space is learned from text and is based on the words that surround the word when it is used. Word embeddings can be trained using the input corpus itself or can be generated using pre-trained word embeddings such as **Glove, FastText,**and**Word2Vec**.

Word vectors are positioned in the vector space such that words that share common contexts in the corpus are in the proximity to one another in space. Word2Vec is particularly computationally effective predictive model for learning word embeddings from raw text.

It comes in two flavors, the Continuous Bag-of-Words (CBOW) model and the Skip-Gram model.  
Algorithmically, these models are similar.

Word2Vec can capture multiple different degrees of similarity between words, such that semantic and syntactic patterns can be reproduced using vector arithmetic. Patterns such as “Man is to Woman as Brother is to Sister” can be generated through algebraic operations on the vector representations of these words such that the vector representation of “Brother” - ”Man” + ”Woman” produces a result which is closest to the vector representation of “Sister” in the model. Such relationships can be generated for a range of semantic relations (such as Country—Capital) as well as syntactic relations (e.g. present tense—past tense).

A close up of a map

Description automatically generated

We use genism package in Python to create word embeddings using Word2Vec. There are two ways we can use embeddings. We can create word embeddings from our corpus, or we can use pre trained word embeddings by downloading them online like word2vec-google-news-300 where each is represented in 300-dimensional vector space. In this project we explored both the techniques for at least one model.

1. **Model Building and Evaluation:**

We perform feature engineering on the dataset using above three techniques mentioned. These three different datasets are used to build machine learning models and their evaluation metrics like accuracy/ f1 score etc. are compared. For each algorithm, we report training accuracy, test accuracy, classification report with metrics like precision/recall and confusion matrix. For naïve Bayes algorithm we do not use dataset from Word2Vec because it cannot handle negative values in the dataset.

* 1. **Machine Learning:**
     1. **Multinomial Naïve Bayes**

**A screenshot of a cell phone

Description automatically generated**

* Accuracy is little better than baseline accuracy of 0.75
* F1 score for positive and classes are very good but for the neutral class it is decent

**A screenshot of a cell phone

Description automatically generated**

* Accuracy has been improved to 0.82 but F1 score of neutral reviews is very less.
  + 1. **Logistic Regression**

**A screenshot of a cell phone

Description automatically generated**

* Accuracy score has been improved compared to MNB and F1 score is good for positive, negative sentiments and decent for neutral sentiment.

**A screenshot of a cell phone

Description automatically generated**

* This is the best accuracy we got for Machine learning models.
* Precision for neutral reviews is good but recall and overall f1 score is below par

**A screenshot of a cell phone

Description automatically generated**

* Accuracy of this classifier is 0.81
* Precision of neutral reviews is decent at 0.34 but recall and overall f1 score are low for neutral reviews
  + 1. **Random Forest**

**A screenshot of a cell phone

Description automatically generated**

* Huge difference between training and testing classifier indicates overfitting.
* Precision of neutral reviews is decent at 0.33 but recall and overall f1 score are low for neutral reviews

**A screenshot of a cell phone

Description automatically generated**

* Huge difference between training and testing classifier indicates overfitting.
* Precision of neutral reviews is decent at 0.36 but recall and overall f1 score are low for neutral reviews

A screenshot of a cell phone

Description automatically generated

* Huge difference between training and testing classifier indicates overfitting.
* All the metrics for neutral reviews are bad in this model but for positive, negative reviews it is good.
  + 1. **Xgboost Classifier**

**A screenshot of a cell phone

Description automatically generated**

* Though accuracy of the model is decent at 0.80, Precision of all the reviews including neutral reviews is very good.
* Hence, we can use XGBoost model if precision is important factor.

A screenshot of a cell phone

Description automatically generated

* Though accuracy of the model is decent at 0.80, Precision of all the reviews including neutral reviews is very good.
* Hence, we can use XGBoost model if precision is important factor.

A screenshot of a cell phone

Description automatically generated

* Precision of neutral reviews is decent at 0.38 but recall and overall f1 score are low for neutral reviews
  + 1. **Logistic Regression with Oversampling**

From the above table, we can see how different algorithms are performing with different word embeddings. As this dataset is highly imbalanced with very few neutral reviews, f1 score for that category is very less even though accuracy overall is decent. Generally, there are couple of techniques like Under sampling and Over sampling that can be used to overcome this issue. I am performing Over sampling with Logistic Regression which is best performed among all algorithms. I will be using Synthetic Minority Oversampling Technique (SMOTE) to perform over sampling.

A screenshot of a cell phone

Description automatically generated

* Accuracy of this model is decent at 0.80
* Precision recall and f1 score for neutral reviews in this classifier are evenly distributed.

A screenshot of a cell phone

Description automatically generated

* Accuracy of this model is below par at 0.78
* Recall for all the sentiments including neutral reviews is very good and Precision, f1 score are decent.

A screenshot of a cell phone

Description automatically generated

* Accuracy of this model is very bad at 0.70
* Recall for all the sentiments including neutral reviews is very good and Precision, f1 score are decent.
  1. **Deep Learning:**

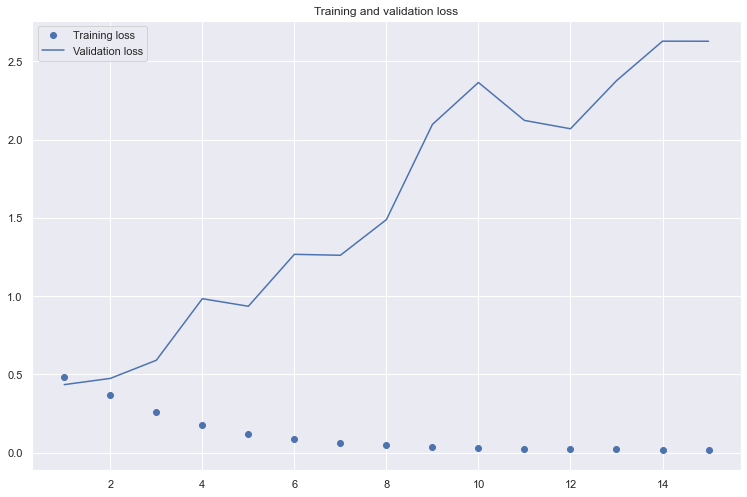
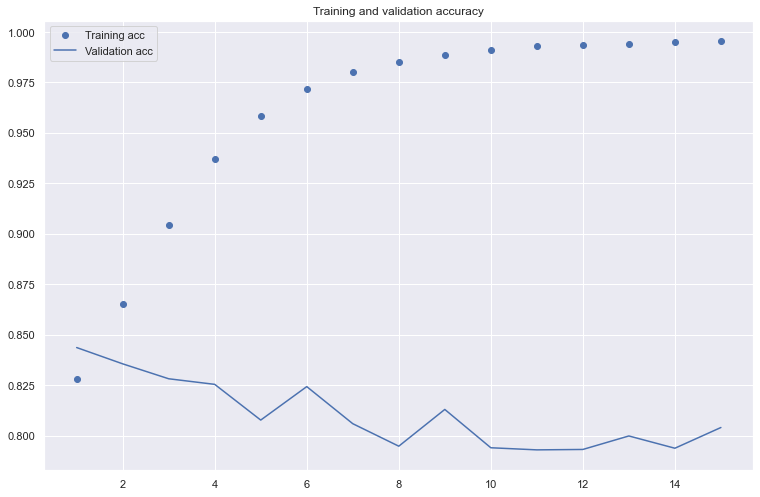
After using the machine learning algorithms, we will now explore more complicated deep learning algorithms. Word embeddings for this model can be built in couple of ways. We can learn embeddings while fitting the model or we can pre train word embeddings and directly feed them to embedding layer. We will apply prior method with Linear NN, LSTM and later with Text CNN.

* + 1. **Linear Neural Network:**

We are training Linear Neural Network with an embedding layer, two hidden dense layers along with drop out layer which takes care of Overfitting. Model summary can be seen below:

A screenshot of a cell phone

Description automatically generated



Training accuracy for Linear Neural Network is : 0.99

Test accuracy for Linear Neural Network is : 0.80

We can clearly observe that even after using dropout technique it is Overfitting.

* + 1. **Recurrent Neural Network (LSTM)**

**LSTM ARCHITECTURE:**

**A screenshot of a cell phone

Description automatically generated**

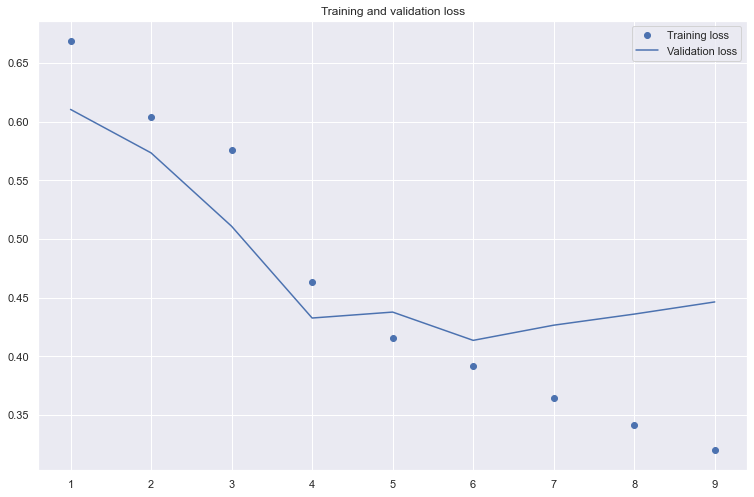
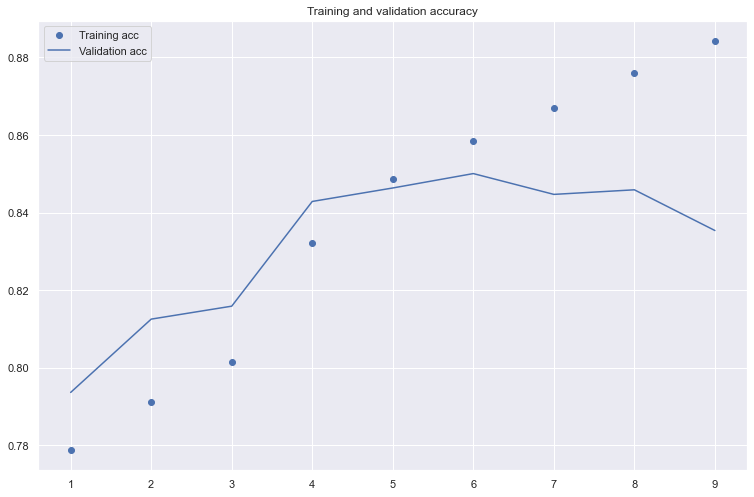
The layers are as follows:

1. Tokenize: This is not a layer for LSTM network but a mandatory step of converting our words into tokens (integers)
2. An [embedding layer](https://pytorch.org/docs/stable/nn.html#embedding) that converts our word tokens (integers) into embeddings of a specific size.
3. An [LSTM layer](https://pytorch.org/docs/stable/nn.html#lstm) defined by a hidden state size and number of layers
4. A fully connected output layer that maps the LSTM layer outputs to a desired output size
5. A sigmoid activation layer which turns all outputs into a value 0–1; return **only the last sigmoid output** as the output of this network.
6. Output: Sigmoid output from the last timestep is considered as the final output of this network

We are training LSTM model with an embedding layer, LSTM layer and 2 dense layers along with dropout layer. Model summary and accuracy/loss plots can be seen below:

A screenshot of a cell phone

Description automatically generated



Training accuracy for Linear Neural Network is : 0.88

Test accuracy for Linear Neural Network is : 0.84

* + 1. **Convolutional Neural Network (Text CNN)**

Among the existing studies using deep learning to classify texts, the CNN takes advantage of the so-called convolutional filters that automatically learn features suitable for the given task. For example, if we use the CNN for the sentiment classification, the convolutional filters may capture inherent syntactic and semantic features of sentimental expressions. It has been shown that a single convolutional layer, a combination of convolutional filters, might achieve comparable performance even without any special hyperparameter adjustment. Furthermore, the CNN does not require expert knowledge about the linguistic structure of a target language. Thanks to these advantages, the CNN has been successfully applied to various text analyses: semantic parsing, search by query, sentence modeling. Below is the architecture for CNN implementation:

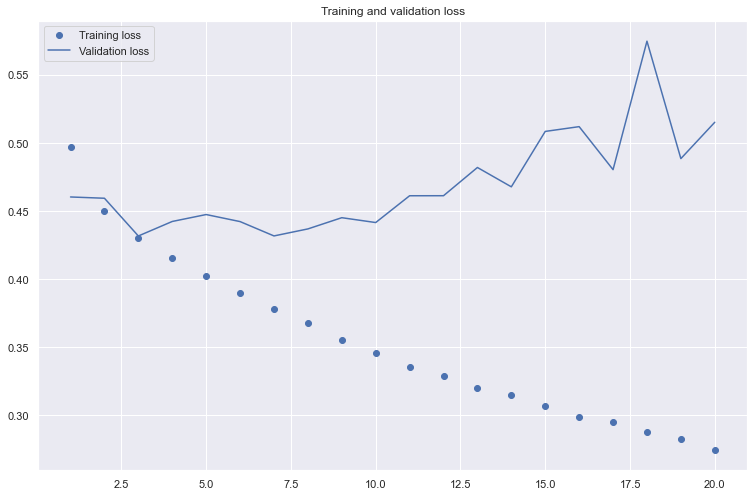
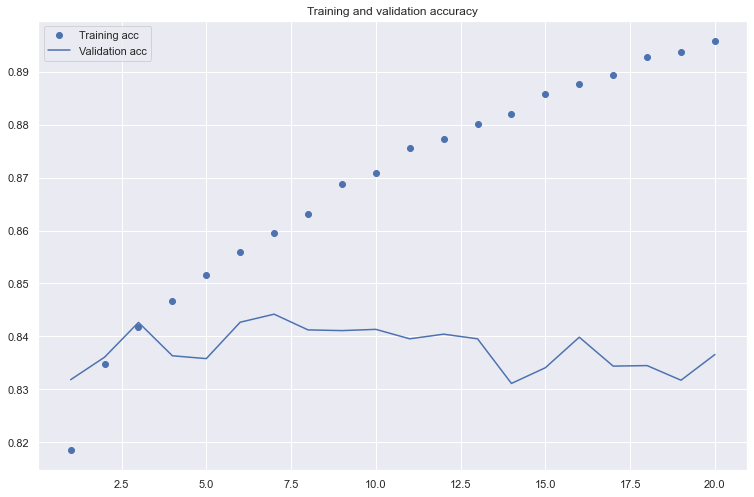
A close up of a map

Description automatically generated

In this model, we train the word embeddings prior using Word2Vec function in genism. We add the embedding layer directly after converting the input to embedding matrix using above trained embeddings. To this, we add 2 CNN layers and finally one dense layer. Model summary and accuracy/loss plots can be seen below:

A screenshot of a cell phone

Description automatically generated



Training accuracy for Linear Neural Network is : 0.89

Test accuracy for Linear Neural Network is : 0.84

1. **Conclusion:**

In this project, we implemented multiple ML/DL algorithms for amazon reviews to carry out text sentiment classification and evaluated them against themselves and each other. We experimented with different sampling, embedding techniques, and utilized various visualizations to get insight of model performance.

We can observe that more complicated LSTM and Text CNN models are not giving any significant improvement over Logistic Regression with TF-IDF Vectorizer. This is because sentiment of the model is based on the usage of certain words rather the semantic or order in which words are used. These sophisticated deep learning algorithms with word embeddings will be extremely beneficial when we are performing tasks like Language Translation, Question answering etc.

From here, we can extend our project to doing hyperparameter tuning which cannot be done on local machine because of memory and capacity constraints. We can use the entire dataset and extend this project to predict numerical rating from 1 to 5 instead of 3 classes.

1. **References:**

* <https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/>
* <https://stackabuse.com/python-for-nlp-movie-sentiment-analysis-using-deep-learning-in-keras/>
* <https://towardsdatascience.com/machine-learning-word-embedding-sentiment-classification-using-keras-b83c28087456>
* <https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/>
* <https://medium.com/@lamiae.hana/a-step-by-step-guide-on-sentiment-analysis-with-rnn-and-lstm-3a293817e314>
* <https://www.aclweb.org/anthology/D14-1181/>
* Deep Learning with Python – Book by François Chollet

1. **Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from wordcloud import WordCloud, STOPWORDS

import re

import pickle

from nltk.corpus import stopwords

from nltk import pos\_tag

from nltk.stem.wordnet import WordNetLemmatizer

from nltk.tokenize import word\_tokenize

stop\_words = set(stopwords.words('english'))

lemma = WordNetLemmatizer()

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

from sklearn.feature\_extraction.text import TfidfVectorizer, CountVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import MultinomialNB

from sklearn.svm import SVC

from xgboost import XGBClassifier

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score,f1\_score

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

from keras.models import Sequential

from keras.layers import Embedding,Dense,LSTM,Conv1D,Flatten,Dropout,MaxPooling1D,GlobalMaxPooling1D

from keras.models import load\_model

from keras.initializers import Constant

from keras.callbacks import EarlyStopping

from keras.utils.np\_utils import to\_categorical

import warnings

warnings.filterwarnings("ignore")

data = pd.read\_csv("Reviews.csv")

data.head()

data.shape

data.columns

data.drop(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator','HelpfulnessDenominator', 'Time'],axis=1,inplace=True)

data.isnull().sum()

data[data['Summary'].isnull()]

data1 = data[data['Summary'].notnull()].reset\_index(drop=True)

data1.info()

data\_clean = data1.drop\_duplicates()

def convert\_score(x):

if x<=2:

return 0

elif x==3:

return 1

else:

return 2

data\_clean['Score'] = data\_clean['Score'].apply(lambda x: convert\_score(x))

data\_clean2=data\_clean.copy()

train,test = train\_test\_split(data\_clean2,stratify=data\_clean2['Score'],test\_size=0.3)

data\_clean=test.reset\_index(drop=True)

reviews\_by\_category=pd.DataFrame(data\_clean['Score'].value\_counts()).reset\_index().rename(columns={'index':'Score','Score':'Count'})

sns.set(rc={'figure.figsize':(12.7,8.27)})

plt.title("Number of reviews by Category",fontsize=20)

ax=sns.barplot(x='Score',y='Count',data=reviews\_by\_category)

Comment\_length=data\_clean['Text'].str.len()

plt.title("Distribution of length of reviews",fontsize=20)

sns.distplot(Comment\_length)

mean\_length = dict()

for i in range(3):

mean\_length[i]=data\_clean[data\_clean['Score']==i]['Text'].str.len().mean()

sns.set(rc={'figure.figsize':(12.7,8.27)})

plt.title("Mean Length of comments by Category",fontsize=20)

ax=sns.barplot(x=list(mean\_length.keys()),y=list(mean\_length.values()))

score0\_reviews = data\_clean[data\_clean['Score']==0]

score0\_text=" ".join([i for i in score0\_reviews['Text'].values])

wordcloud = WordCloud(stopwords=STOPWORDS).generate(score0\_text)

plt.title("Type of words in negative reviews", fontsize=20)

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

plt.axis("off")

plt.show()

score1\_reviews = data\_clean[data\_clean['Score']==1]

score1\_text=" ".join([i for i in score1\_reviews['Text'].values])

wordcloud = WordCloud(stopwords=STOPWORDS).generate(score1\_text)

plt.title("Type of words in nuetral reviews", fontsize=20)

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

plt.axis("off")

plt.show()

score2\_reviews = data\_clean[data\_clean['Score']==2]

score2\_text=" ".join([i for i in score2\_reviews['Text'].values])

wordcloud = WordCloud(stopwords=STOPWORDS).generate(score2\_text)

plt.title("Type of words in positive reviews", fontsize=20)

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

plt.axis("off")

plt.show()

# Custom dictionary

abbv = {

"aren't" : "are not",

"can't" : "cannot",

"couldn't" : "could not",

"didn't" : "did not",

"doesn't" : "does not",

"don't" : "do not",

"hadn't" : "had not",

"hasn't" : "has not",

"haven't" : "have not",

"he'd" : "he would",

"he'll" : "he will",

"he's" : "he is",

"i'd" : "I would",

"i'd" : "I had",

"i'll" : "I will",

"i'm" : "I am",

"isn't" : "is not",

"it's" : "it is",

"it'll":"it will",

"i've" : "I have",

"let's" : "let us",

"mightn't" : "might not",

"mustn't" : "must not",

"shan't" : "shall not",

"she'd" : "she would",

"she'll" : "she will",

"she's" : "she is",

"shouldn't" : "should not",

"that's" : "that is",

"there's" : "there is",

"they'd" : "they would",

"they'll" : "they will",

"they're" : "they are",

"they've" : "they have",

"we'd" : "we would",

"we're" : "we are",

"weren't" : "were not",

"we've" : "we have",

"what'll" : "what will",

"what're" : "what are",

"what's" : "what is",

"what've" : "what have",

"where's" : "where is",

"who'd" : "who would",

"who'll" : "who will",

"who're" : "who are",

"who's" : "who is",

"who've" : "who have",

"won't" : "will not",

"wouldn't" : "would not",

"you'd" : "you would",

"you'll" : "you will",

"you're" : "you are",

"you've" : "you have",

"'re": " are",

"wasn't": "was not",

"we'll":" will",

"didn't": "did not",

"tryin'":"trying"

}

def clean\_text(data):

#Convert to Lower case

data.loc[:,'Text'] = data['Text'].apply(lambda x: x.lower())

#Remove new line \n

data.loc[:,'Text'] = data['Text'].apply(lambda x: x.replace("\n"," "))

#Remove html tags

data.loc[:,'Text'] = data['Text'].apply(lambda x: re.sub(r'<.\*?>',' ',x))

#Get abbrevations from custom dictionary above

data.loc[:,'Text'] = data['Text'].apply(lambda x: " ".join([abbv.get(i,i) for i in x.split()]))

#Remove punctuations

data.loc[:,'Text'] = data['Text'].apply(lambda x: re.sub(r'[^\w\s]',' ',x))

#Remove numeric values from text

data.loc[:,'Text'] = data['Text'].apply(lambda x: re.sub(r'[\d]',' ',x))

#Removing Stop words

data.loc[:,'Text'] = data['Text'].apply(lambda x: " ".join([i.strip() for i in x.split() if i not in stop\_words]))

#Lemmatization

data.loc[:,'Text'] = data['Text'].apply(lambda x: " ".join([lemma.lemmatize(i, "v") for i in x.split()]))

return data

data\_final.drop('Summary',axis=1,inplace=True)

data\_final.head()

data\_final=data\_final[data\_final['Text']!='']

X=data\_final['Text']

y=data\_final['Score']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,stratify=y,test\_size=0.25)

#Count vectorizer

cnt\_vec = CountVectorizer(max\_features=10000, min\_df=5, max\_df=0.7,stop\_words='english',ngram\_range=(1,2))

X\_train\_cnt = cnt\_vec.fit\_transform(X\_train)

X\_test\_cnt = cnt\_vec.transform(X\_test)

#TF-IDF Vectorizer

tfidf\_vec = TfidfVectorizer(max\_features=10000, min\_df=5, max\_df=0.7, stop\_words='english',ngram\_range=(1,2))

X\_train\_tfidf = tfidf\_vec.fit\_transform(X\_train)

X\_test\_tfidf = tfidf\_vec.transform(X\_test)

#Word2Vec

import gensim

from gensim.models import Word2Vec,KeyedVectors

model = KeyedVectors.load\_word2vec\_format('C:\\Users\\abhi1\\gensim-data\\word2vec-google-news-300\\word2vec-google-news-300.gz', binary=True)

model.init\_sims(replace=True)

vocab = model.vocab.keys()

wordsInVocab = len(vocab)

clean\_train\_reviews = []

for review in X\_train:

clean\_train\_reviews.append(review.split())

clean\_test\_reviews = []

for review in X\_test:

clean\_test\_reviews.append(review.split())

def sent\_vectorizer(sent, model):

sent\_vec =[]

numw = 0

for w in sent:

try:

if numw == 0:

sent\_vec = model[w]

else:

sent\_vec = np.add(sent\_vec, model[w])

numw+=1

except:

pass

return np.asarray(sent\_vec) / numw

X\_train\_vectorized = np.zeros((len(clean\_train\_reviews), 300))

num=0

for index,sentence in enumerate(clean\_train\_reviews):

X\_train\_vectorized[index,:]=sent\_vectorizer(sentence, model)

num+=1

if num%10000==0:

print(str(num)+": reviews")

X\_train\_vectorized\_df = pd.DataFrame(X\_train\_vectorized)

X\_train\_vectorized\_df.shape

X\_test\_vectorized=np.zeros((len(clean\_test\_reviews), 300))

num=0

for index,sentence in enumerate(clean\_test\_reviews):

X\_test\_vectorized[index,:]=sent\_vectorizer(sentence, model)

num+=1

if num%10000==0:

print(str(num)+": reviews")

X\_test\_vectorized\_df = pd.DataFrame(X\_test\_vectorized)

X\_test\_vectorized\_df.shape

def model\_build\_evaluate(clf,x\_train,y\_train,x\_test,y\_test,classifier,Vectorizer,df):

clf.fit(x\_train,y\_train)

train\_accuracy=clf.score(x\_train,y\_train)

test\_accuracy=clf.score(x\_test,y\_test)

y\_pred = clf.predict(x\_test)

f1score = f1\_score(y\_test,y\_pred,average=None)

f1score=[round(i,2) for i in f1score]

print("Train accuracy of "+classifier+"Classifier with "+Vectorizer+" is " + str(train\_accuracy))

print("Test accuracy of "+classifier+"Classifier with "+Vectorizer+" is " + str(test\_accuracy))

print("")

print("Classification report is :")

print(classification\_report(y\_test,y\_pred))

cnf\_matrix = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(12.8,6))

sns.heatmap(cnf\_matrix, annot=True,cmap="Blues",fmt='g',annot\_kws={"size": 20})

plt.ylabel('Predicted')

plt.xlabel('Actual')

plt.title('Confusion matrix')

plt.show()

df\_length = len(df)

df.loc[df\_length] = [classifier,Vectorizer,train\_accuracy,test\_accuracy,f1score]

return clf,df

classifier\_comp = pd.DataFrame(columns = ['Classifier','Word Embeddings','Train Accuracy','Test Accuracy','f1 score'])

mnb\_clf = MultinomialNB()

mnb\_clf\_cnt,classifier\_comp = model\_build\_evaluate(mnb\_clf,X\_train\_cnt,y\_train,X\_test\_cnt,y\_test,

'Multinomial Naive Bayes','Count Vectorizer',classifier\_comp)

mnb\_clf\_tfidf,classifier\_comp = model\_build\_evaluate(mnb\_clf,X\_train\_tfidf,y\_train,X\_test\_tfidf,y\_test,

'Multinomial Naive Bayes','TF-IDF Vectorizer',classifier\_comp)

logreg\_clf =LogisticRegression()

logreg\_clf\_cnt,classifier\_comp = model\_build\_evaluate(logreg\_clf,X\_train\_cnt,y\_train,X\_test\_cnt,y\_test,

'Logistic Regression','Count Vectorizer',classifier\_comp)

logreg\_clf\_tfidf,classifier\_comp = model\_build\_evaluate(logreg\_clf,X\_train\_tfidf,y\_train,X\_test\_tfidf,y\_test,

'Logistic Regression','TF-IDF Vectorizer',classifier\_comp)

logreg\_clf\_w2v,classifier\_comp = model\_build\_evaluate(logreg\_clf,X\_train\_vectorized\_df,y\_train,X\_test\_vectorized\_df,y\_test,

'Logistic Regression','Word2Vec',classifier\_comp)

forest\_clf = RandomForestClassifier()

forest\_clf\_cnt,classifier\_comp = model\_build\_evaluate(forest\_clf,X\_train\_cnt,y\_train,X\_test\_cnt,y\_test,

'Random Forest','Count Vectorizer',classifier\_comp)

forest\_clf\_tfidf,classifier\_comp = model\_build\_evaluate(forest\_clf,X\_train\_tfidf,y\_train,X\_test\_tfidf,y\_test,

'Random Forest','TF-IDF Vectorizer',classifier\_comp)

forest\_clf\_w2v,classifier\_comp = model\_build\_evaluate(forest\_clf,X\_train\_vectorized\_df,y\_train,X\_test\_vectorized\_df,y\_test,

'Random Forest','Word2Vec',classifier\_comp)

xgb\_clf = XGBClassifier(objective='multi:softmax')

xgb\_clf\_cnt,classifier\_comp = model\_build\_evaluate(xgb\_clf,X\_train\_cnt,y\_train,X\_test\_cnt,y\_test,

'XGBoost Classifier','Count Vectorizer',classifier\_comp)

xgb\_clf\_tfidf,classifier\_comp = model\_build\_evaluate(xgb\_clf,X\_train\_tfidf,y\_train,X\_test\_tfidf,y\_test,

'XGBoost Classifier','TF-IDF Vectorizer',classifier\_comp)

xgb\_clf\_w2v,classifier\_comp = model\_build\_evaluate(xgb\_clf,X\_train\_vectorized\_df,y\_train,X\_test\_vectorized\_df,y\_test,

'XGBoost Classifier','Word2Vec',classifier\_comp)

from imblearn.over\_sampling import SMOTE

smt1 = SMOTE()

X\_train\_cnt\_smt,y\_train\_smt=smt1.fit\_sample(X\_train\_cnt,y\_train)

smt2 = SMOTE()

X\_train\_tfidf\_smt,y\_train\_smt\_tfidf=smt2.fit\_sample(X\_train\_tfidf,y\_train)

smt3 = SMOTE()

X\_train\_w2v\_smt,y\_train\_smt\_w2v=smt3.fit\_sample(X\_train\_vectorized\_df,y\_train)

logreg\_clf\_cnt,classifier\_comp = model\_build\_evaluate(LogisticRegression(),X\_train\_cnt\_smt,y\_train\_smt,X\_test\_cnt,y\_test,

'Logistic Regression','Count Vectorizer',classifier\_comp)

logreg\_clf\_tfidf,classifier\_comp = model\_build\_evaluate(LogisticRegression(),X\_train\_tfidf\_smt,y\_train\_smt\_tfidf,X\_test\_tfidf,y\_test,

'Logistic Regression','TF-IDF Vectorizer',classifier\_comp)

logreg\_clf\_w2v,classifier\_comp = model\_build\_evaluate(LogisticRegression(),X\_train\_w2v\_smt,y\_train\_smt\_w2v,X\_test\_vectorized\_df,y\_test,

'Logistic Regression','Word2Vec',classifier\_comp)

tokenizer\_obj = Tokenizer()

total\_reviews = list(X\_train.values)+list(X\_test.values)

max\_len = 100

tokenizer\_obj.fit\_on\_texts(total\_reviews)

X\_train\_seq = tokenizer\_obj.texts\_to\_sequences(X\_train)

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

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

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

one\_hot\_train\_labels = to\_categorical(y\_train)

one\_hot\_test\_labels = to\_categorical(y\_test)

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

Embedding\_dim = 200

model = Sequential()

model.add(Embedding(vocab\_size,Embedding\_dim,input\_length=max\_len))

model.add(Flatten())

model.add(Dense(128,activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(64,activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(3,activation='softmax'))

model.compile(optimizer='rmsprop',loss='categorical\_crossentropy',metrics=['accuracy'])

print(model.summary())

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3)

,callbacks = [early\_stopping]

history = model.fit(X\_train\_seq,one\_hot\_train\_labels,batch\_size=64,epochs=15,

validation\_data=(X\_test\_seq,one\_hot\_test\_labels))

with open('Neural\_network\_model', 'wb') as file\_pi:

pickle.dump(history.history, file\_pi)

import matplotlib.pyplot as plt

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()

model = Sequential()

model.add(Embedding(vocab\_size,Embedding\_dim,input\_length=max\_len))

model.add(LSTM(128,dropout=0.2,recurrent\_dropout=0.2))

model.add(Dense(64,activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(3,activation='softmax'))

model.compile(optimizer='rmsprop',loss='categorical\_crossentropy',metrics=['accuracy'])

print(model.summary())

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3)

history1 = model.fit(X\_train\_seq,one\_hot\_train\_labels,batch\_size=64,epochs=20,

validation\_data=(X\_test\_seq,one\_hot\_test\_labels),callbacks = [early\_stopping])

acc = history1.history['accuracy']

val\_acc = history1.history['val\_accuracy']

loss = history1.history['loss']

val\_loss = history1.history['val\_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()

with open('LSTM\_model', 'wb') as file\_pi:

pickle.dump(history1.history, file\_pi)

all\_reviews = clean\_train\_reviews+clean\_test\_reviews

model\_w2v = Word2Vec(all\_reviews,size=Embedding\_dim,min\_count=2)

words=list(model\_w2v.wv.vocab)

model\_w2v.wv.save\_word2vec\_format("amazon\_food\_reviews\_word2vec.txt",binary=False)

embeddings\_dictionary = dict()

embedding\_file = open('amazon\_food\_reviews\_word2vec.txt', encoding="utf8")

for line in embedding\_file:

records = line.split()

word = records[0]

vector\_dimensions = np.asarray(records[1:], dtype='float32')

embeddings\_dictionary [word] = vector\_dimensions

embedding\_file.close()

embedding\_matrix = np.zeros((vocab\_size, Embedding\_dim))

for word, index in tokenizer\_obj.word\_index.items():

embedding\_vector = embeddings\_dictionary.get(word)

if embedding\_vector is not None:

embedding\_matrix[index] = embedding\_vector

model = Sequential()

model.add(Embedding(vocab\_size,Embedding\_dim,input\_length=max\_len,

embeddings\_initializer=Constant(embedding\_matrix),trainable=False))

model.add(Conv1D(64, 7, activation='relu'))

model.add(MaxPooling1D(5))

model.add(Dropout(0.2))

model.add(Conv1D(32, 7, activation='relu'))

model.add(GlobalMaxPooling1D())

model.add(Dropout(0.2))

model.add(Dense(32,activation='relu'))

model.add(Dense(3,activation='softmax'))

model.compile(optimizer='rmsprop',loss='categorical\_crossentropy',metrics=['accuracy'])

print(model.summary())

history2 = model.fit(X\_train\_seq,one\_hot\_train\_labels,batch\_size=64,epochs=20,

validation\_data=(X\_test\_seq,one\_hot\_test\_labels))

with open('TextCNN\_word2vec\_model', 'wb') as file\_pi:

pickle.dump(history2.history, file\_pi)

acc = history2.history['accuracy']

val\_acc = history2.history['val\_accuracy']

loss = history2.history['loss']

val\_loss = history2.history['val\_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()