COMP316-Natural Language Processing
Main Project

Final Report

Introduction

Choosing what foods to eat plays a big role in the lives of many. It's important to know what you're eating especially if it's for the first time. Online customer reviews are a helpful way for peers to provide constructive criticism, suggest and recommend their opinions to fellow parties of interest. A common metric to rate a certain product is through the use of a 'score' rating system which rates a product on a scale of 1 to 5 whereby a good review would have a rating of 4-5 and a bad review form 1-2 with 3 being neutral.

Project Objective: Predict a rating for what a reviewer says in their review text, based on the way all foodies talk in the data set of Amazon Fine Foods Reviews

[Q]: How to determine if a review is positive or negative i.e.) Classified correctly

[Ans]: By calculating the Word2Vec of each word and the AverageWord2Vec of an entire sentence we can predict a score/rating for a review based off of the word vectors of each review.

RESOURCES

Language

The language chosen to code the solution in was Python. Multiple languages were considered for use with the most likely alternative being Java, with access to the Stanford CoreNLP Library, however Python proved to be more suitable for the applications required. API's required were easily imported and data science resources were more readily available for reference and tutorials.

IDE

Jupyter Notebook was the selected platform to develop in, it has been developed to be a conducive environment to develop data science solutions in. The initial learning curve was minimal and use of all required API's proved to be simple and trouble free. Pycharm was the default IDE that I looked to however I encountered multiple issues with imports and compatibility resulting in the decision to revert to Jupyter Notebook.

API's

NumPy – NumPy is a fundamental package for scientific computing with Python

TQDM – Used to create and display training progress graphs

Pandas- pandas is an open-source library built on top of numPy providing high-performance, easy-to-use data structures and data analysis tools.

Matplotlib – Used to create graphs to depict attention mechanism

Seaborn- Seaborn is a Python data visualization library based on matplotlib.

Sklearn – The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, and clustering.

DataSet

The Amazon Fine Foods Reviews dataset consists of reviews for fine foods from Amazon.

Number of Reviews: 568,454
Number of users: 256,059
Number of Products: 74,258

ARCHITECTURE AND IMPLEMENTATION

DataSet Attribute information:

1. Id
2. ProductId
3. UserId
4. ProfileName
5. Helpfulness Denominator
6. Helpfulness Numerator
7. Score
8. Time
9. Summary
10. Text

Data Preparation and Text Preprocessing

The Amazon dataset is available as a '.csv' file.

In order to load the data we use numPy and pandas as it allows for fast analysis, data cleaning and preparation of '.csv' files with the use of efficient data structures.

Since we are only interested in predicting a score from the text of a review, we will purposely ignore Id, ProductId, UserId, ProfileName, Helpfulness Denominator and Helpfulness Numerator since our objective is independent of these attributes.

```
In [36]:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
import os
import math
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

```
In [2]:
#Import the data set using pandas
data2 = pd.read_csv('.//Reviews.csv')
#Head-Function to display the first 5 rows of set
data2.head()
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wid...

Unfortunately due to the limited functionality of my computer the dataset is reduced to a shortened- due the shortcomings of my machine computer, with a limit of only 2GB of ram to work with we shall reduce the data set to something that is easier to compile. Done so by reducing data set to 210 000 reviews to work with.

```
In [3]:
#Shortened data-set, as original data-set too large to compile()
data1 = data2.loc[:209999,:]
print("Shape: ",data1.shape)
```

```
Shape:  (210000, 10)
```

```
In [ ]:
```

```
In [4]:
#ValueCount-Function to display all reviews assigned to each score
data1['Score'].value_counts()
```

```
Out[4]: 5    132812
4     30547
1     19217
3     16305
2     11119
Name: Score, dtype: int64
```

Data Cleaning and De-duplication

It is observed that the reviews have many duplicate entries. Hence it was necessary to remove these duplicates in order to get unbiased results for the analysis of the data. We sort our data based on the ProductId and drop duplicates which are found based on UserId, ProfileName, Time, summary and text. We then proceed to clean the data by dropping columns which are not helpful to our calculations and only keep data needed for identification and analysis.

In [5]:
```
Final_Values = data1.sort_values('ProductId').drop_duplicates(subset = ['UserId','ProfileName','Time','Summary','Text'],keep=
data = Final_Values.drop(columns =[ 'UserId' , 'HelpfulnessNumerator' , 'HelpfulnessDenominator' , 'Time' ])
data.head()
```

Out[5]:

	Id	ProductId	ProfileName	Score	Summary	Text
150504	150505	0006641040	Jason A. Teeple "Nobody made a greater mistak...	4	A classic	Get the movie or sound track and sing along wi...
150512	150513	0006641040	tessarat	5	A classic	I remembered this book from my childhood and g...
150510	150511	0006641040	Laura Purdie Salas	4	Charming and childlike	A charming, rhyming book that describes the ci...
150509	150510	0006641040	Dr. Joshua Grossman	5	Professional Mentoring	TITLE: Chicken Soup with Rice AUTHOR: Mau...
150508	150509	0006641040	Teresa	5	A great way to learn the months	This is a book of poetry about the months of t...

Training and Test Data Splitting

We are required to split our data set into a training and test set. We do so by importing a **train_test_split** function from sklearn.metrics to split our data set at our required ratio. We choose to have a test size of 30% and a training size of 70% split randomly.

In [6]:
```
#Randomly split a data set into a training and test set with a Test-set size of 30% and Training-set size of 70%
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data["Text"].values, data["Score"].values, test_size = 0.3, random_state
```

We go on to create a **sentence_list()** function which will be used to create arrays of sentences out of our training set and test set. This enables us to control and maintain the data in a much easier fashion rather than using the raw data set.

In [7]:
```
#Function to create list of sentences
def sentence_list(x):
    list_of_sent = []
    for sent in tqdm(x):                #import TQDM(Taqqadum/Progress)
        words = []
        for w in sent.split():
            words.append(w)
        list_of_sent.append(words)
    return list_of_sent                 #Returns Array
```

Word2Vec Implementation

Word2vec is a technique for natural language processing. The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence. By importing **Word2Vec** from genism.models we can create a word2vec model of our training set. The purpose and usefulness of **Word2vec** is to group the vectors of similar words together in vector-space. That is, it detects similarities mathematically. **Word2vec** creates vectors that are distributed numerical representations of word features, features such as the context of individual words.

In [7]:
```
#implementing word2vec w/ Vec size=50,Minimum No. of Occurrences = 2 ; Workers = 4;
from gensim.models import Word2Vec
sent_train = sent_list(x_train)
w2v = Word2Vec(sent_train,size=50,min_count=2,workers=4)
```

```
100%|| 147000/147000 [00:03<00:00, 42644.83it/s]
```

Average Word2Vec

We define an **AverageWord2Vec** function **avgw2v()** which averages the word embedding's. This means that embedding's of all words are averaged, and thus we get a 1D vector of features corresponding to each review. This data format is what typical machine learning models expect, so in a sense it is convenient.

We then go on to create an average word2vec of the training data called **avgw2v_train** which we normalize using the StandardScaler imported from sklearn.preprocessing

In [8]:
```
#Function to create Average word2vec vector
def avgw2v(x):
    avgw2v_vec = []
    for sent in tqdm(x):
        sent_vec = np.zeros(50)            #each Vector up to size 50
        count = 0
        for word in sent:
            try:
                vec = w2v.wv[word]
                sent_vec+=vec
                count+=1
            except:
                pass
        sent_vec/=count
        avgw2v_vec.append(sent_vec)
    return avgw2v_vec
```

In [9]:
```
#Creating average word2vec training data
avgw2v_train = np.array(avgw2v(sent_train))
print("Shape of avg word2vec train data:- ",avgw2v_train.shape)    #Displaying shape
```

```
100%| 147000/147000 [01:24<00:00, 1734.43it/s]
Shape of avg word2vec train data:-  (147000, 50)
```

In [10]:
```
#Normalize Features by Standardizing avergae Word2Vec training data
sc = StandardScaler()
avgw2v_train = sc.fit_transform(avgw2v_train)
```


Shortcomings

However, this should be done very carefully because averaging word2vec does not take care of word order, for example:

- Our lecturer is a great teacher he will not fail us
- Our lecturer is not a great teacher he will fail us

These two sentences have the same words, and therefore will have same average word embedding, but these two sentences have very different meaning.
To address this issue, one might look into Sentence-BERT which is sentence-embedding using Siamese BERT- Network which is unfortunately outside this scope.

NUMERICAL ANALYSIS AND EVALUATION

Data Classification Analysis

We go on to test our model on the testing set. By creating an array **sentence_list** for our test set, we can find the average Word2vec for our entire test set. We then go on to use a logistic regression model as a discriminative classification method for our data to directly learn/predict 'score' values for our reviews.

In natural language processing, logistic regression is the baseline supervised machine learning algorithm for discriminative classification, and also has a very close relationship with neural networks. When comparing the actual 'score' values to the predicted 'score' values we can go on to calculate the Accuracy of the training and the test sets as well as calculate the precision score of each of our 'score' classes.

By predicting 'score' values we can test the effectiveness of using logistic regression as a classifier for average of our word2Vec model.

In [12]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
#Classification Method Analysis
#Using Linear Regression to Predict 'scores' based off of the AverageWord2Vec
#Testing the model on the test set
sent_test = sentence_list(x_test)
avgw2v_test = np.array(avgw2v(sent_test))
avgw2v_test = sc.transform(avgw2v_test)

classifier = LogisticRegression(C=1,penalty = 'l2',solver = 'sag')
classifier.fit(avgw2v_train,y_train)
y_pred = classifier.predict(avgw2v_test)
#Y_Pred are predicted 'score values' using Logistic Regression by finding a line of best fit to predct values
```

```
100%|| 51030/51030 [00:01<00:00, 43145.27it/s]
100%|| 51030/51030 [00:29<00:00, 1709.64it/s]
```

In [13]:
```
#Printing calulated values and scores used for Analysis
print("Avg Word2Vec test accuracy:- ",accuracy_score(y_test,y_pred))   #calculating Test Accuracy

print("Avg Word2Vec Training accuracy:- ",accuracy_score(y_train,classifier.predict(avgw2v_train))) #Calculating Training Acc

print("Precision Score for every Score class:- ",precision_score(y_test,y_pred, average = None))    #Calculating Precision of
```

```
Avg Word2Vec test accuracy:-  0.6610817166372722
Avg Word2Vec Training accuracy:-  0.6607036374478235
Precision Score for every Score class:-  [0.44516632 0.28640777 0.32258065 0.36864053 0.70434763]
```

Average Word2Vec test accuracy: **0.66214**
Average Word2Vec Training accuracy: **0.664149**
Precision Score for every score class: **[0.4570643, 0.26645768, 0.32086022, 0.37795808, 0.70479603]**

By using logistic regression, 0.66% of times our predicted value is equal to the actual value.

Confusion Matrix and Seaborn Heatmap

Confusion Matrix and Seaborn Heatmap of actual 'scores' vs. predicted 'scores' used to visually illustrated the error rates of 'scores' mapped incorrectly

In [23]:
```
#Numerical Prediction Analysis
#ConfusionMatrix of actual Test 'scores' vs. Predicted Test 'scores'
from sklearn.metrics import confusion_matrix
ConfusionMatrix = confusion_matrix(y_test,y_pred)
ConfusionMatrixNormalized = ConfusionMatrix / ConfusionMatrix.astype(np.float).sum(axis=0 , keepdims=True)
print(ConfusionMatrixNormalized ,)
```

Table 1: CONFUSION MATRIX GENERATED

```
[[0.44516632 0.30097087 0.12004231 0.04837364 0.05935834]
 [0.16008316 0.28640777 0.18984664 0.07923269 0.0371189 ]
 [0.14137214 0.23300971 0.32258065 0.20767306 0.0579228 ]
 [0.08809771 0.08737864 0.22051824 0.36864053 0.14125234]
 [0.16528067 0.09223301 0.14701216 0.29608007 0.70434763]]
```

In [24]:
```
fig , ax = plt.subplots(figsize=(10,10))
sns.heatmap(ConfusionMatrixNormalized, annot=True, fmt='.2f')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show(block=False)
```

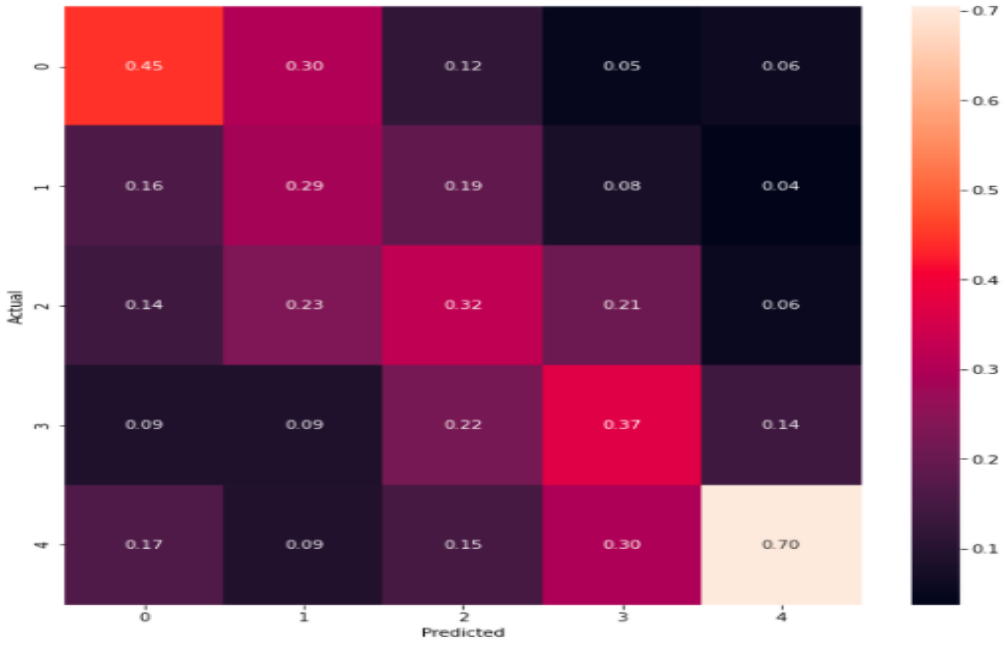

Table 2: HEATMAP where 0 1 2 3 4 correspond to rating scores 1 2 3 4 5 respectively.

We successfully built a model that predicts the given reviews 'scores' with an accuracy of 66%. For all scores, 1-rated scores were correctly classified 45% of the time, 2-rated scores 29%, 3-rates scores 32%, 4-rated scores 37% and 5-rated scores 70%. By analyzing the diagonals along our confusion matrix we can depict that no classification of score is predicted better than the other. Based off of our exploratory findings by analyzing the results we can analyze the effectiveness of using a logistic regression classifier

Since only 66% of data is correctly predicted, Logistic Regression is therefore not a good classification method for a Word2Vec Model. Alternative methods may yield more promising results such as KNeighboursClassifier from sklearn.model_selection or even a completely different modelling scheme which would classify a review as either negative or positive rather than by predicting a score for each review.

Numerical Prediction Analysis

We may further evaluate our prediction results through the use of common metrics Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). Both MAE and RMSE express average model prediction error in units of the variable of interest. Both metrics can range from 0 to ∞ and are indifferent to the direction of errors as well as are dependent on the variable of interest. They are negatively-oriented scores, which means lower values are better. Since we are dealing with review 'scores' our metrics shall range from 1 – 5 whereby hypothetically an error score of 0.5 would signify that our numeric prediction of 'score' values was off/deviates by this value.

Root Mean Square Error (RMSE):

By importing mean_squared_error from sklearn.metrics and importing Math, we can calculate both the Mean Squared Error and the Root Mean Squared Error of our 'score' results.

```
In [16]:
from sklearn.metrics import mean_squared_error
#Evaluating our Numerical Prediction by calculating the MeanSquareError of the data 'scores'
mse = mean_squared_error(y_test, y_pred)
print('Mean Square Error of Actual vs Predicted: ',mse)
#Finding the RootMeanSquareError by square-rooting the MSE
#Provides a more accurate estimation than MSE
rmse = math.sqrt(mse)
print('Root Mean Square Error of Actual vs Predicted: ',rmse)

Mean Square Error of Actual vs Predicted:  1.8299431706839113
Root Mean Square Error of Actual vs Predicted:  1.352753920964161
```

Mean Square Error of Actual vs. Predicted 'score' values: **1.8341111**

Root Mean Square Error of Actual vs. Predicted 'score' values: **1.35429**

Mean Absolute Error (MAE):

MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. By importing mean_absolute_error from sklearn.metrics we may calculate the MAE of our actual vs. predicted 'scores'.

```
In [17]:
#Evaluating our Numerical Prediction further by calculating the MeanAbsoluteError of the data 'scores'
from sklearn.metrics import mean_absolute_error

print('Mean Absolute Error of Actual vs Predicted Results: ' )
print('\t', mean_absolute_error(y_test, y_pred))

Mean Absolute Error of Actual vs Predicted Results:
        0.6821869488536155
```


Mean Absolute Error of Actual vs. Predicted 'score' values: **0.6818253**

FINAL RESULTS

- Average Word2Vec Test accuracy: **0.66214**
- Average Word2Vec Training accuracy: **0.664149**
- Precision Score for every score class: **[0.4570643, 0.26645768, 0.32086022, 0.37795808, 0.70479603]**

- Mean Square Error of Actual vs. Predicted 'score' values: **1.8341111**
- Root Mean Square Error of Actual vs. Predicted 'score' values: **1.35429**
- Mean Absolute Error of Actual vs. Predicted 'score' values: **0.6818253**

In our evaluation of the system we focus on accuracy achieved by the solution, efficiency of methods employed in developing the system, usage of NLP techniques, and solution of NLP problems within the solution.

Accuracy – In terms of accuracy the solution performs mildly. There aren't many similar level systems to compare the solution to. A professional system for comparison would be Ratings prediction system using sentiment analysis and a Naïve Bayes classifier to predict a rating off a reviews semantics.

Efficiency – The methods used to prepare data, process data, and process answers utilized in the solution are extremely efficient. Simple procedures were employed where need be, certain tasks were easily solved by use string processing and these methods were easily employed through the complicated sklearn functions.

NLP Techniques – Simple NLP techniques were utilized within the first division of the solution such as Data Retrieval, Text Pre-processing. The second division of the solution implemented a deep learning approach to NLP which involved the use of word embedding's and vectorized data in a neural network.

NLP Problems Addressed – The solution solves numerous NLP problems in its implementation through the choice of data set and methods used to achieve the desired result. This type of process allows to create a reliable method of reviewing a system by forcing a link between the predicted score of a review and the text written for a review. This can further help to rid the review system of fake/spam reviews as well as reviews that make no sense i.e.) a lazy review that has no text and only a score rating ,or a spam review that contains gibberish text and a false score rating.

CONCLUSION

Model classification and numerical prediction are a complex and diverse area of application within Natural Language Processing. The solution developed focuses on a single area of application within the vast domain of review systems. Even though I found that I didn't go about things in the perfect way, I was still able to provide a useful service to the food reviewing industry.