SENTIMENT ANALYSIS USING RNN

A INTERNSHIP REPORT

Submitted by

RISHAV KUMAR MISHRA [Reg No: RA1811003030177]

Under the guidance of

MR. NISHANT KUMAR SINGH

(Assistant Professor, Department of Computer Sciene & Engineering)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM
INSTITUTE OF SCIENCE AND TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)
DELHI-NCR CAMPUS, GHAZIABAD (U.P)

SRM INSTITUTE OF SCIENCE & TECHNOLOGY, NCR CAMPUS

MAY 2022

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this internship report titled “**SENTIMENT ANALYSIS USING RNN**” is the bonafide work of “ **RISHAV KUMAR MISHRA [Reg No: RA1811003030177]** ”, who carried out the internship work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other internship report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE	**SIGNATURE**
MR. NISHANT KUMAR SINGH **GUIDE** Assistant Professor Dept. of Computer Sciene & Engineering	Dr. R.P.Mahapatra **HEAD OF THE DEPARTMENT** Dept. of Computer Science & Engineering
Signature of the Internal Examiner	Signature of the External Examiner

ABSTRACT

To check the Sentiment of the people using Highradius's Analytics360 Dashboard as we will soon be implementing this and it will give us a brief idea how good is our implementation of Dashboards , adding new functionalities to Dashboard and what are their views on it.

I will be implementing this for the feedback for our system as our analytics360 team is building so many new interactive dashboards using tableau and there are more dashboards are in progress. Due to which we want a system will be giving us the data like what people like and what they want should be improved. So for this we are building RNN(Recurrent Neural Architecture) based Sentiment Analysis for our Dashboards.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my guide, MR. NISHANT KUMAR SINGH for her valuable guidance, consistent encouragement, personal caring, timely help and providing me with an excellent atmosphere for doing research. All through the work, in spite of his busy schedule, he has extended cheerful and cordial support to me for completing this research work.

Author

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Sentiment Analysis

The number one priority of a organisation is to understand the mood of the customers by looking at their feedback text. Whether they are happy, sad, or angry! Sentiment Analysis is the term for this.

Sentiment analysis is a type of text mining that discovers and extracts subjective information from source material in order to help a firm better understand the social sentiment of its brand, product, or service while monitoring online interactions. The majority of social media stream analysis, on the other hand, is limited to sentiment analysis and count-based metrics. This is comparable to only scratching the surface and missing out on high-value discoveries waiting to be discovered.

In this project, I have integrated neural network architecture with NLP (natural language processing) to improve accuracy over other machine learning systems.

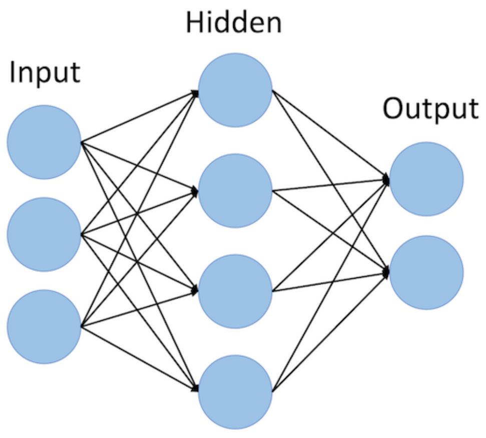


Figure 1.1: Neural Network Architecture

1.1.1 Neural Network

I am using Neural Network Architecture in this project to train the model for training the model. Have a look at Figure 1.1 for architecture.

1.1.2 Natural Language Processing

I am using NLP for processing the text such that the neural network should take the words in correct input and they should be such that we can see the dependency of a single word in sentence.

CHAPTER 2

LITERATURE SURVEY

2.1 "Expressively vulgar : The socio - dynamics of vulgarity and its effects on sentiment analysis in social media"

At The University of Texas in Austin. Isabel Cachola, Eric Holgate, Daniel Preotiuc-Pietro, Junyi Jessy Li wrote this paper in the year 2009. In this paper they used Bi-LSTM + Masking + Token Insertion + Concatenation and got the overall accuracy of 75%.

2.2 "Multilingual Twitter Sentiment Classification: The Role of Human Annotators"

At The Department of Knowledge Technologies, Jozef Stefan Institute, Ljubljana, Slovenia Igor Mozetic, Miha Grcar, Jasmina Smailovic wrote this paper in the year 2016. In this paper they used NaiveBayes and got the overall accuracy of 65%.

2.3 "Toward multi-label sentiment analysis: a transfer learning based approach"

At The Journal of Big Data, J., Fang, X. wrote this paper in the year 2020. In this paper they used XLNet and got the accuracy around 90%.

2.4 "Deep Learning for Hate Speech Detection in Tweets"

At The IIIT-H, Hyderabad, India in colaboration with Microsoft, India. Pinkesh Badjatiya, Shashank Gupta, Manish Gupta, Vasudeva Varma wrote this paper in the year 2017. In this paper they used LSTM + GloVe + GBDT and got the overall accuracy of 86%.

2.5 "UBham: Lexical Resources and Dependency Parsing for Aspect-Based Sentiment Analysis"

At University of Birmingham. Viktor Pekar, Naveed Afzal, Bernd Bohnet wrote this paper in the year 2014. In this paper they used Linear SVM and got the overall accuracy of 61%.

CHAPTER 3

SYSTEM ARCHITECTURE

3.1 EDA (Exploratory Data Analysis)

3.1.1 Fetching data from DB

This dataset was acquired from Kaggle and comes from the popular movie database IMDB. The information is in the txt format. The information is split into two files. One file contains all of the customer reviews (reviews.txt), while the other provides a label for each review (labels.txt) that indicates whether the review is positive or negative. Let's have a look at the data once it's been imported into the notebook and connected to form the dataset.

3.1.2 Tokenization of words

Tokenization is the process that breaks a phrase, sentence, paragraph, and an entire text content into smaller components like individual words or phrases. Tokens are the names given to each of these small parts. Words, numbers, or punctuations could be used as tokens. However, I am simply utilising words as tokens in this project.

3.1.3 Visualizing dataset

The graphical depiction of information or data is known as data visualisation. Data visualisation systems make it possible to examine and comprehend trends, outliers, and patterns in data by leveraging visual elements like charts, graphs, and maps. Graphs, charts, word clouds, maps, networks, timelines, and other tools are commonly used to display text. Humans are able to read the most important features of a large amount of data thanks to these graphical outcomes.

3.2 Model Building

3.2.1 RNN architect

RNNs are a kind of Neural Network architecture where the output from the last step is being used to give input in the next phase. All of the inputs and outputs of the standard neural networks are independent of each other, looking at some other instances, such as when predicting the next word of a phrase or a sentence, the prior words are necessary, and so the previous words must always be remembered. For that reason new neural network RNN was developed, which used a invisible layer to overcome the problem. The Invisible state, which remembers certain information about a sequence of the words, is the most important feature of RNN.

Types of RNN architecture :

1. One-to-one (T$_x = T_y = 1$)

It is used as Traditional neural network.

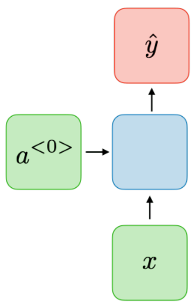


Figure 3.1: One-to-one

2. One-to-many (T$_x = 1, T_y > 1$)

It is used in Music generation.

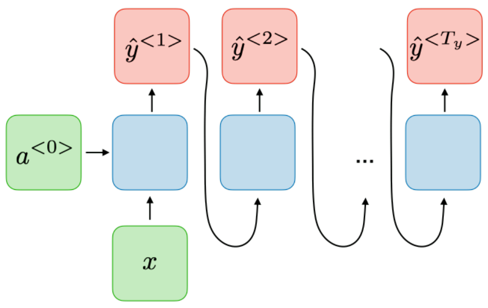


Figure 3.2: One-to-many

3. Many-to-one (T$_x > 1, T_y = 1$)

It is used in Sentiment classification. I'll be using this in my project.

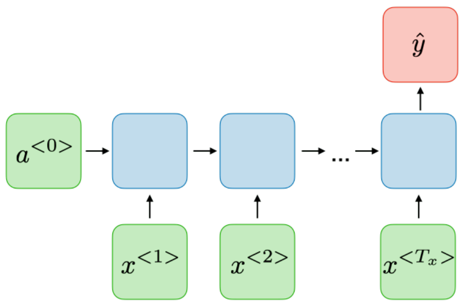


Figure 3.3: Many-to-one

4. Many-to-many (T$_x = T_y$)

It is used in Name entity recognition.

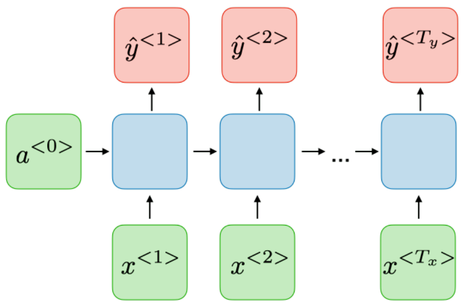


Figure 3.4: Many-to-many

5. Many-to-many (T$_x \neq T_y$)

It is used in Machine translation.

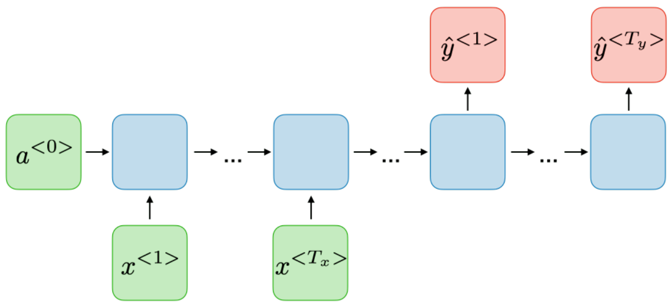


Figure 3.5: Many-to-many

3.2.2 Checking Sentiment

I'll verify the sentiment of a sentence in ad hoc to model for testing purposes, as it will assist us determine whether or not our model is working.

3.3 Accuracy

3.3.1 Score on Training Data

The training data is utilised to begin the construction of a model. The training score reflects how well the trained data matches the original dataset. Over-Fitting may occur when the training score is high and the testing score is low, or underfitting may occur if they both perform badly. So, in order to avoid over-fitting or under-fitting, we use,variety of techniques.

3.3.2 Score on Testing Data

This is when our model run will be finished. This data-set had never been touched before this stage. As a result, this is a realistic possibility. The higher the score, the more flexible the model is.

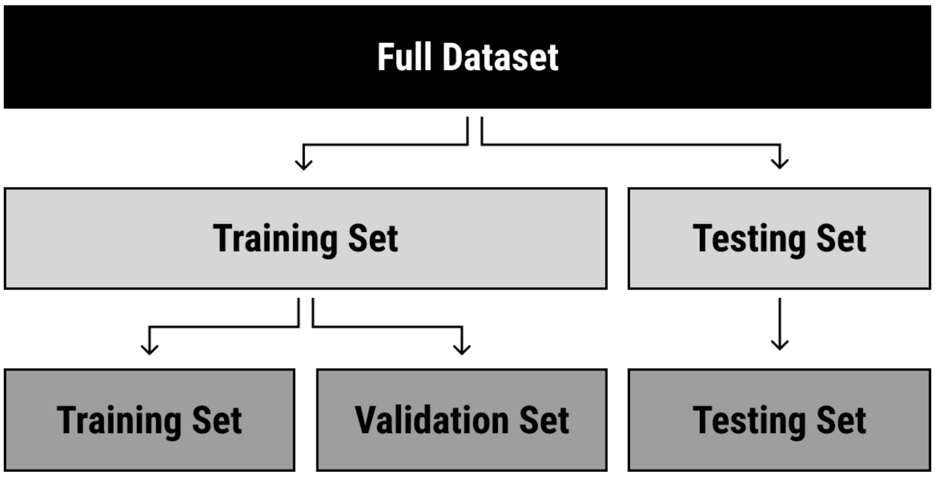


Figure 3.6: Score

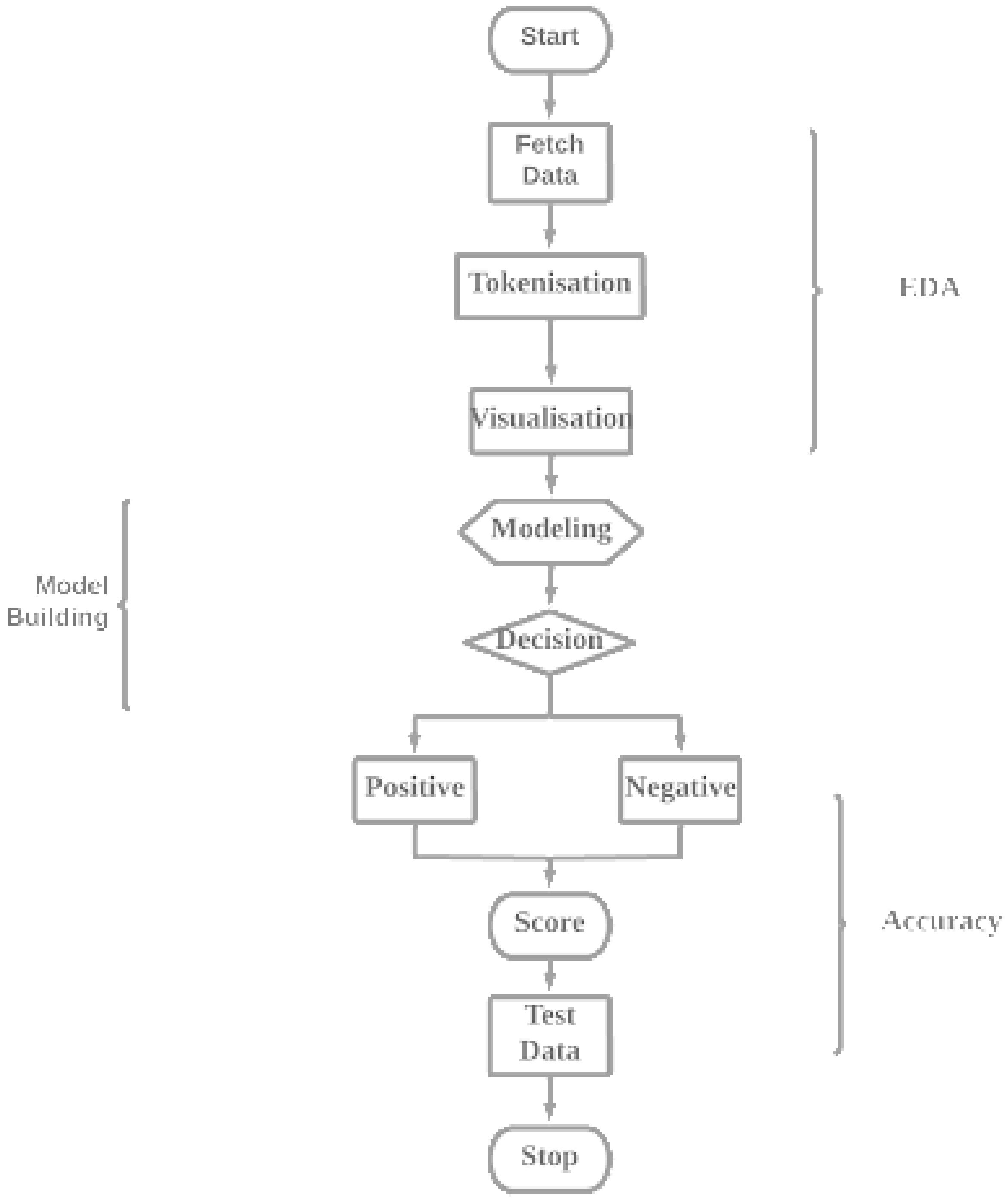


Figure 3.7: System Architecture

CHAPTER 4

SYSTEM DESIGN

4.1 Natural Language Processing (NLP)

4.1.1 Tokenisation

In our project, we will first create smaller tokens of words from larger reviews phrases, which will assist us and even the system in quickly interpreting the data and determining the data's purpose.

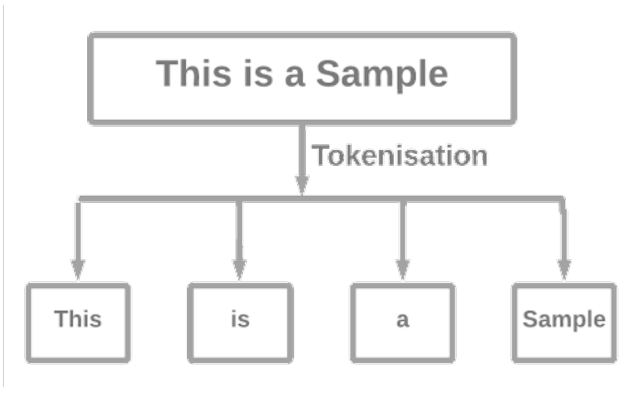


Figure 4.1: Tokenisation

4.1.2 Normalisation

In our project while working with NLP we also have to normalize the dataset after tokenization as it is a good practice which would lead our dataset to some new findings. So while looking at our dataset of tokens we will find that some of the terms like "and", "an", "the" and some other terms are appearing frequently. Now we have to normalize them to decrease their counts. So for doing that we have to first check the "positive-to-negative" ratio with the formula "Positive counts of the words will be divided by float(negative counts of words) and adding one in the denominator". Here in this formula we will always have "+1" in the denominator to make it sure, whenever a word is there in positive and not there in negative so it won't be "0" it has positive value of "1" with it.

Impact Analysis on Project: :

I. Whenever we see the words like "amazing" or "good" then we will expect these words to appear in positive evaluations and their ratios will always be greater than "1". When "positive-to-negative" ratio deviates from "1", the word will be more skewed towards the positive direction of the graph.

II. Words like "terrible" or "bad" then we always declare them as negative words and they would appear in the negative evaluation of the words and while looking at ratios we can say that it will be less than "1". When the "positive-to-negative" ratio of a word is closer to "0", the word will be more skewed towards negative direction of the graph.

III. There is not only positive and negative words but also some neutral words like "and", "an" and "the". These words will never tells us about the sentiment of any person. And even we'd expect that these words will appear in all kinds of positive or negative sentences. While looking at their "positive-to-negative" ratios we can predict that it will be quite close to "1.". The "+1" suggested in the denominator is biased towards negative but we will ignore that words that will be too close to neutral because they will be minor bias.

While Looking at the ratios of the tokens that we calculated, it will tell us like which all terms will be appearing more periodically in positive or negative reviews, but the ratios that we have calculated of the words are tough to work with. Let's look at some scenarios like: A exceedingly positive term such as "perfection," has value more than "4", and even exceedingly negative words like "bad" has value near to 0.184. It's difficult to evaluate those data for a plethora of purposes:

1. Despite knowing the fact that "1" is currently thought to be neutral, overwhelmingly positive phrase "positive-to-negative" ratios have a higher actual value than "negative-to-positive" ratios. Consequently, We can't find any method to compare 2 numbers directly to determine if any one of the word has the same level of positive sentiment compared to another. As a result, we will now be focusing on all the values around neutral words in tokens, so the values assigned for the neutral words in the "positive-to-negative" ratio of a single token of word would tell us the sentiment value that has been conveyed.

2. It's helpful to evaluate absolute values around zero than it is around one.
To address these concerns, we'll use log(Logarithms) to convert all of our ratios that we found out to new values that will help us in calculation.

At the end of the Normalisation project we will exceedingly get positive word and extra negative word will be having "positive-to-negative" ratios with same number of values and different sign(+ or -).

If everything goes well, we will be seeing neutral terms with values near zero. Now Have a look at some example of tokens: word "the" has it's value very close to "0" but it is moderately positive. In this case, we can say that it is mostly used in positive reviews and less in negative reviews. Lets look at other cases the word "amazing" the value is above 1, this shows that the word has strong positive connotation. Simultaneously, let's have a look at word "insult" it also has same value but with negative sign. Now this will clear the thoughts like every word is associated to a specific conflicting attitudes.

4.1.3 Python

I used Python to label the words with numbers so that they could be fed into the RNN system and get trained, and then I numbered the positive and negative labels as '1' and '0' respectively.

4.2 Neural Network & their Operations

4.2.1 Recurrent Neural network (RNN)

Over here, I'm using many to one rnn. It takes sequence data as input and outputs informative data such as labels. As a result, we can identify it. Assume that each sentence's sentiment is defined by someone, and that the model is trained using the many-to-one type. When the model receives the unseen sentence, it predicts the phrase's intention, whether positive or negative.

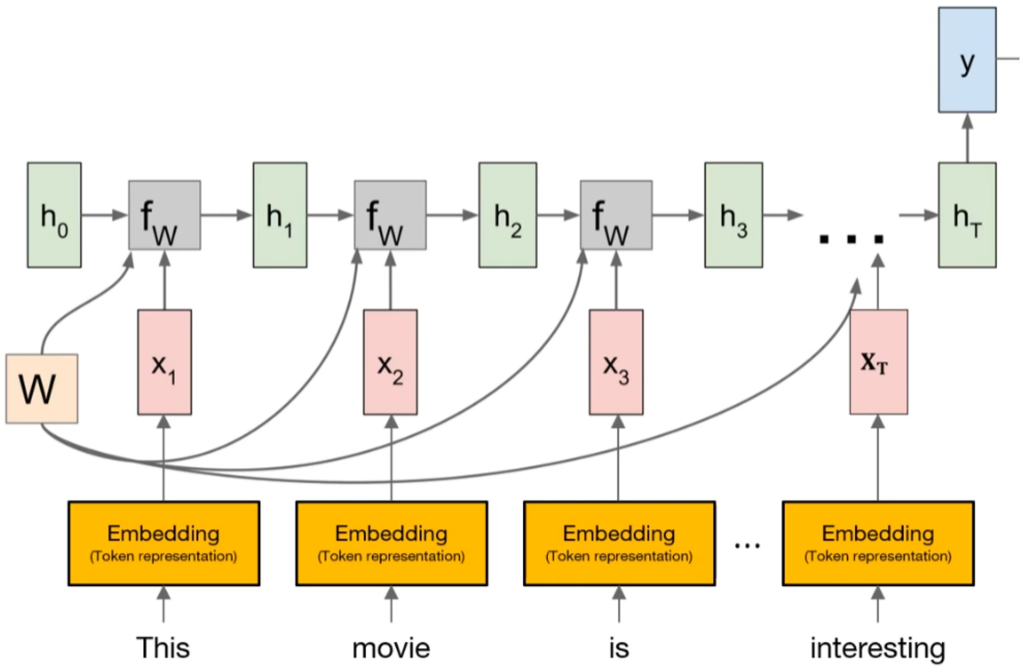

y
h0
fW
h1
fW
h2
fW
h3
. . .
hT
W
x1
x2
x3
xT
Embedding (Token representation)
Embedding (Token representation)
Embedding (Token representation)
...
Embedding (Token representation)
This
movie
is
interesting

Figure 4.2: many-to-one RNN Architecture

RNN's Operation:

I created this RNN in such a way that we can follow along with it in a step-by-step manner to properly understand it.

Step 1 :

First of all I am checking how a sentence looks like when it has positive word with it in it. Here in the example, We are checking it with the excellent word which is a positive word. And then it show it is a positive response in sentiment.

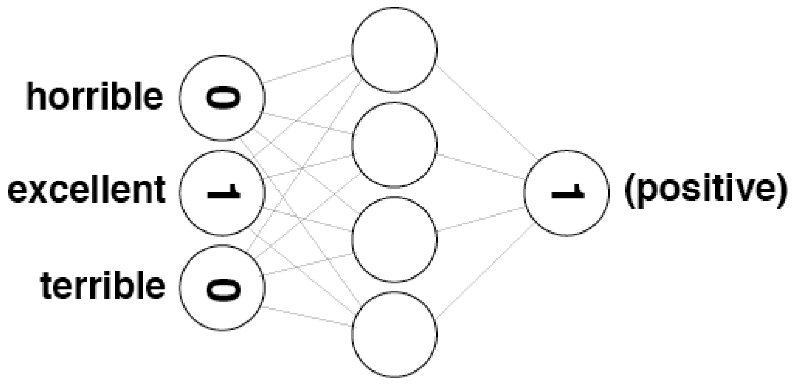


Figure 4.3: Positive

Step 2 :

Now, I am checking how a sentence looks like when it has negative word with it in it. Here in the example, We are checking it with the horrible and terrible like words which is a negative word. And then it show it is a negative response in sentiment.

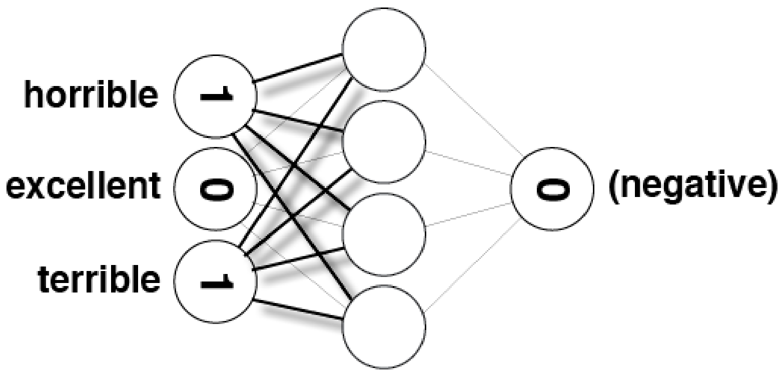


Figure 4.4: Negative

Step 3 :

Now I’m putting first RNN layers that is invisible. It is the initial layer of an RNN that takes the input data and reads the label it has been given, "How they were given such labels with the help of their neural network training."

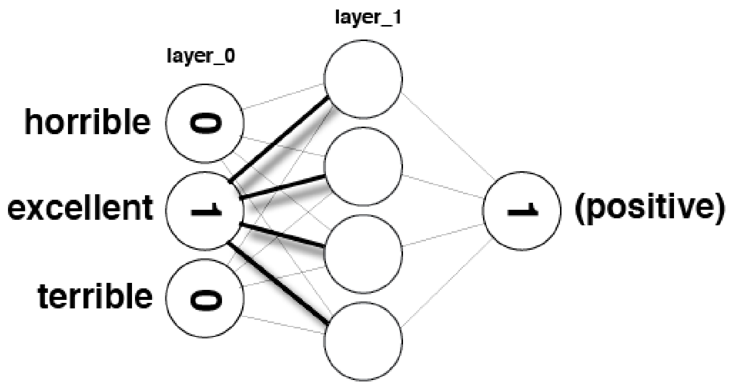


Figure 4.5: Layer 1

Step 4 :

Now I’m combining all of the RNN layers to get the forecast. It is the last layer of an RNN that provides the output data and informs the reader of the label that has been assigned to them after training, such as Positive and Negative.

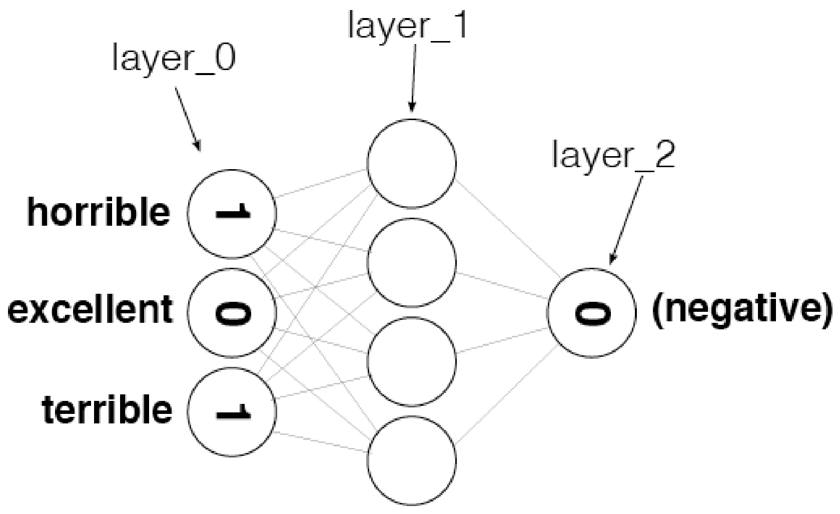


Figure 4.6: Layer 2

Step 5 :

This image is the result of the neural network's work. This will show the review prediction made by the Dashboard user. What did they think of the dashboard and how could it be improved?

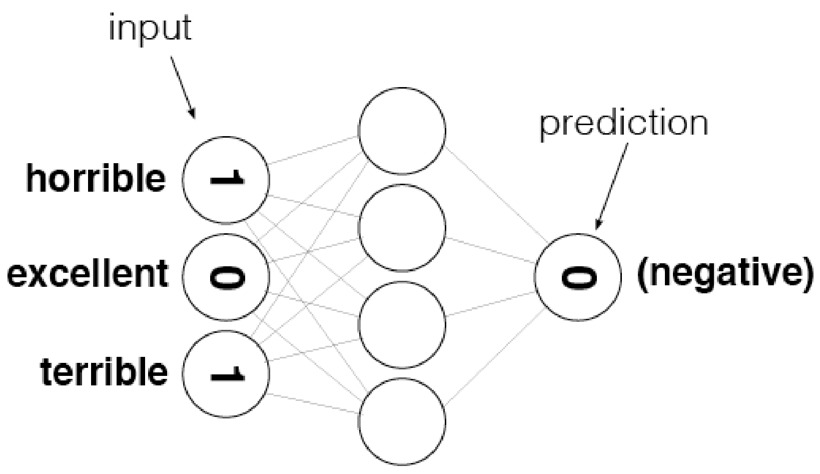


Figure 4.7: Prediction

4.2.2 t-SNE(t-distributed stochastic neighbour embedding)

It is known for its statistical method for analyzing high-dimensional data by assigning a single two- or three-dimensional map to each datapoint.

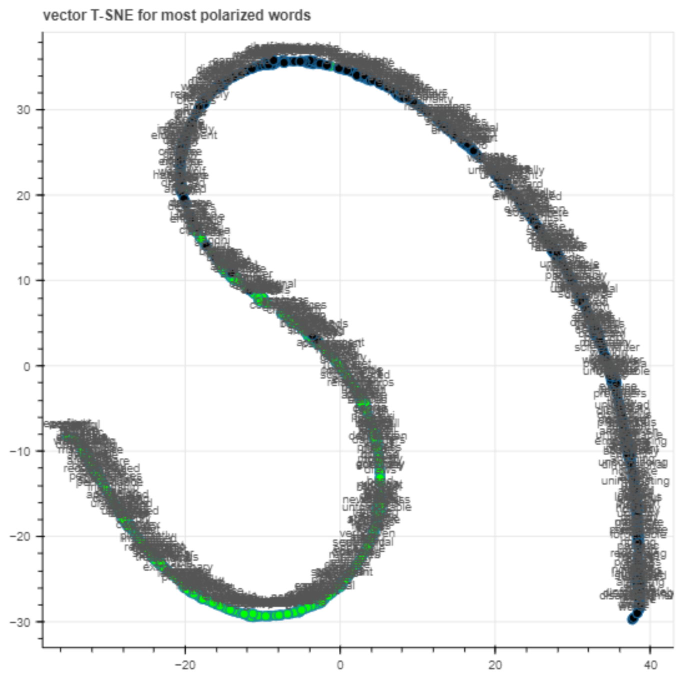


Figure 4.8: t-SNE Plot

CHAPTER 5

CODING, TESTING

5.1 Coding Requirements

In this chapter, the program coding is related to our work using Python and neural network are presented over here.

1. **Python**

   Python is a high-level, general-purpose language of programming that is interpreted. The use of significant indentation in its design philosophy promotes readability of code. Its linguistic elements and object-oriented approach towards a problems that are aimed to assist coders in writing clean, reasonable code for projects.

   For the implementation of this project, I'm using Python 3.8. For the purposes of developing this system, I am using counters and all other functions that came with python to use.

2. **Deep Learning(Neural Network)**

   Deep learning is a subset of a wider family of machine learning algorithms based on representation learning and artificial neural network (ANN). To simulate the human mind, a neural network is used. As a result, it may include n number of neurons(n can be any number).

   It is used to develop artificial neural networks, namely RNNs (recurrent neural networks) for this project.

3. **Natural Language Processing(NLP)**

   Natural language processing is a branch of linguistics, computer science, and artificial intelligence concerned with human computer, particularly how to design computers to process and evaluate huge volumes of natural language(sound) data.

   In this project, i used Tokenization, Normalization, and their kinds stemming and lemmatization.

Python Libraries installation for this project are as follows:

- ✲ **Time**: To check how many reviews passed per second.
- ✲ **Sys**: It manipulate different parts of the Python runtime environment.
- ✲ **Collections**: Collections is a library that stores a variety of items and provides a way to access and iterate over them.
- ✲ **Numpy**: Numpy is a library used for working with arrays with python.
- ✲ **Pandas**: Pandas library will help us to work with multi-dimensional arrays.
- ✲ **Matplotlib**: Matplotlib is used to creating static, animated, and interactive visualizations in Python.
- ✲ **Sklearn**: It is used for classification, regression and clustering algorithms.
- ✲ **Bokeh**: It is used to create interactive visualizations for modern web browsers.

5.2 Testing

Looking at our testing Data.

Input Data	Output Data	Excepted Output	Actual Output	Status
Sweet	Positive	Positive	Positive	Pass
Masterpiece	Positive	Positive	Positive	Pass
Beautifully	Positive	Positive	Positive	Pass
Knife	Negative	Positive	Negative	Fail
Insult	Negative	Negative	Negative	Pass
Fake	Negative	Negative	Negative	Pass

Table 5.1: Testing table

Results and Analysis

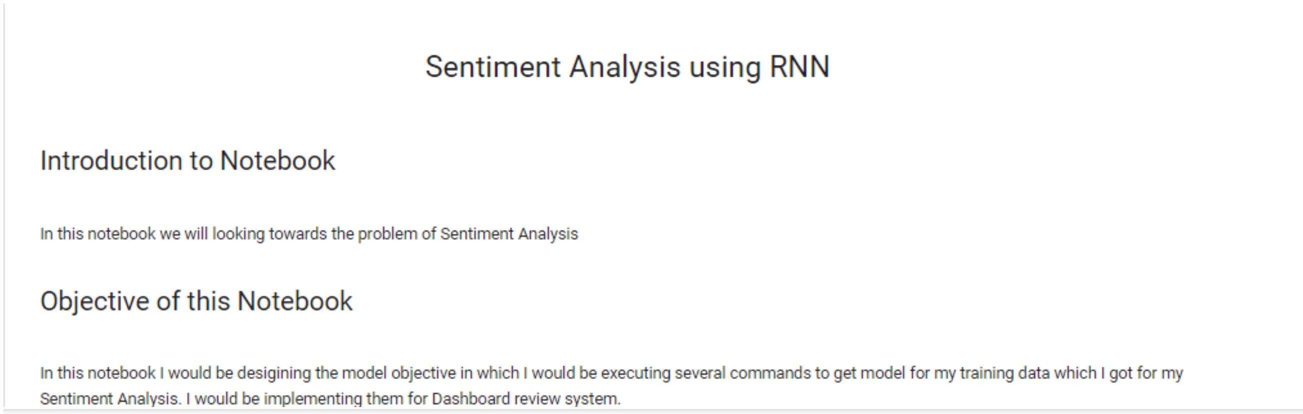
Label	Accuracy	Ran On	Correct
Training Accuracy	85.6%	24000	20552
Testing Accuracy	82.2%	1000	822

Table 5.2: Accuracy

5.3 Code

For Dataset Coding Reference have a look at:

https://github.com/RishavMishraRM/Sentiment_Analysis/tree/development



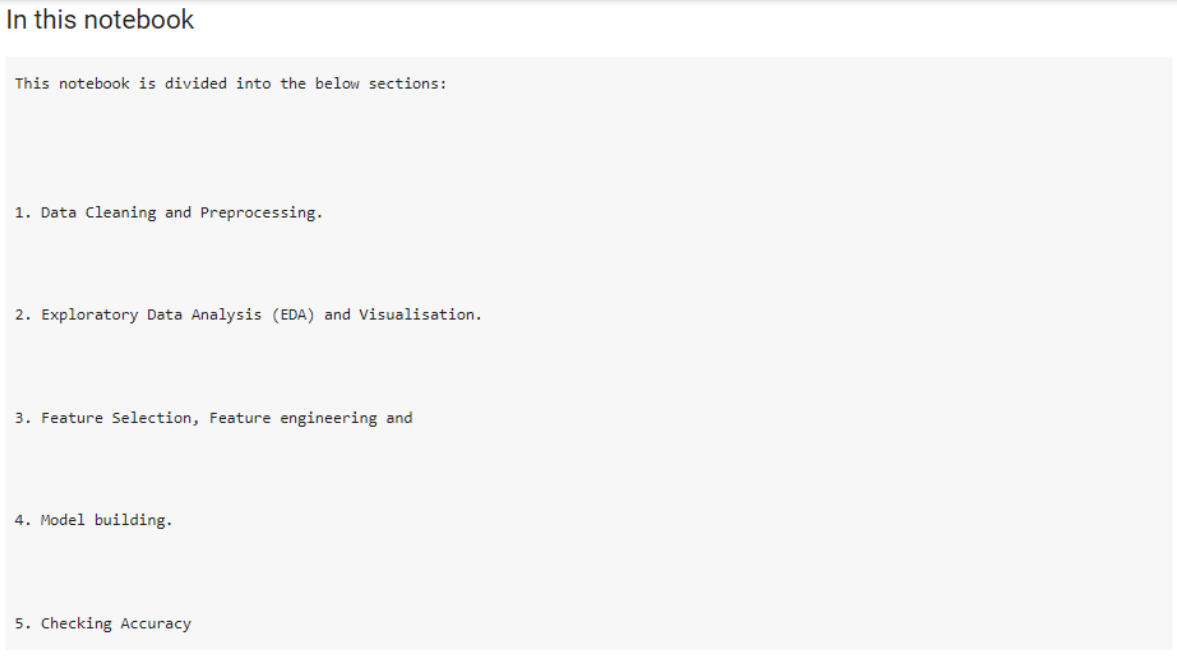
Sentiment Analysis using RNN

Introduction to Notebook

In this notebook we will looking towards the problem of Sentiment Analysis

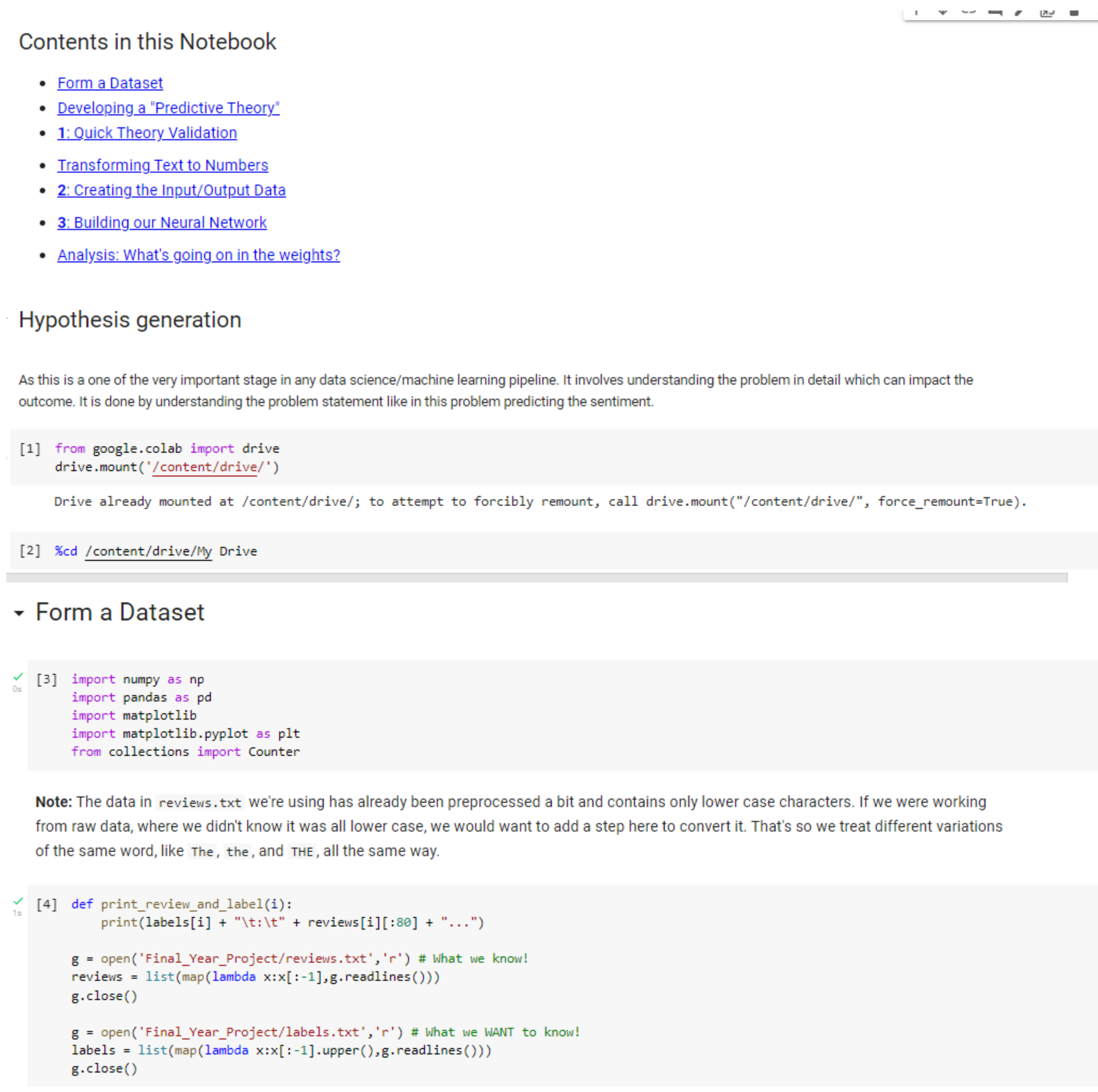
Objective of this Notebook

In this notebook I would be desigining the model objective in which I would be executing several commands to get model for my training data which I got for my Sentiment Analysis. I would be implementing them for Dashboard review system.



In this notebook

```
This notebook is divided into the below sections:



1. Data Cleaning and Preprocessing.


2. Exploratory Data Analysis (EDA) and Visualisation.


3. Feature Selection, Feature engineering and


4. Model building.


5. Checking Accuracy
```



Contents in this Notebook

Hypothesis generation

As this is a one of the very important stage in any data science/machine learning pipeline. It involves understanding the problem in detail which can impact the outcome. It is done by understanding the problem statement like in this problem predicting the sentiment.

```
[1] from google.colab import drive
    drive.mount('/content/drive/')

    Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_remount=True).
```

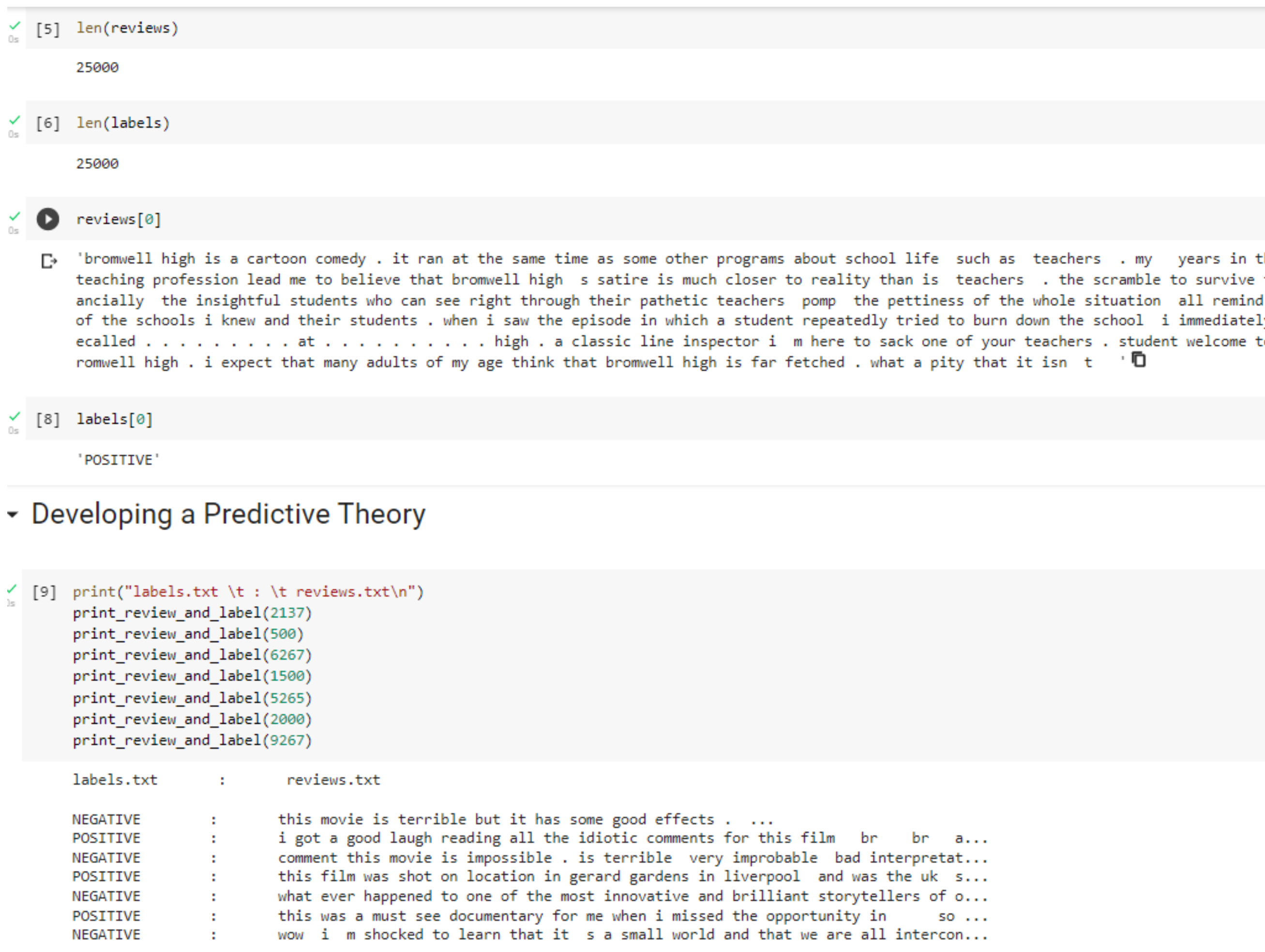
```
[2] %cd /content/drive/My Drive
```

Form a Dataset

```
[3] import numpy as np
    import pandas as pd
    import matplotlib
    import matplotlib.pyplot as plt
    from collections import Counter
```

Note: The data in `reviews.txt` we're using has already been preprocessed a bit and contains only lower case characters. If we were working from raw data, where we didn't know it was all lower case, we would want to add a step here to convert it. That's so we treat different variations of the same word, like `The`, `the`, and `THE`, all the same way.

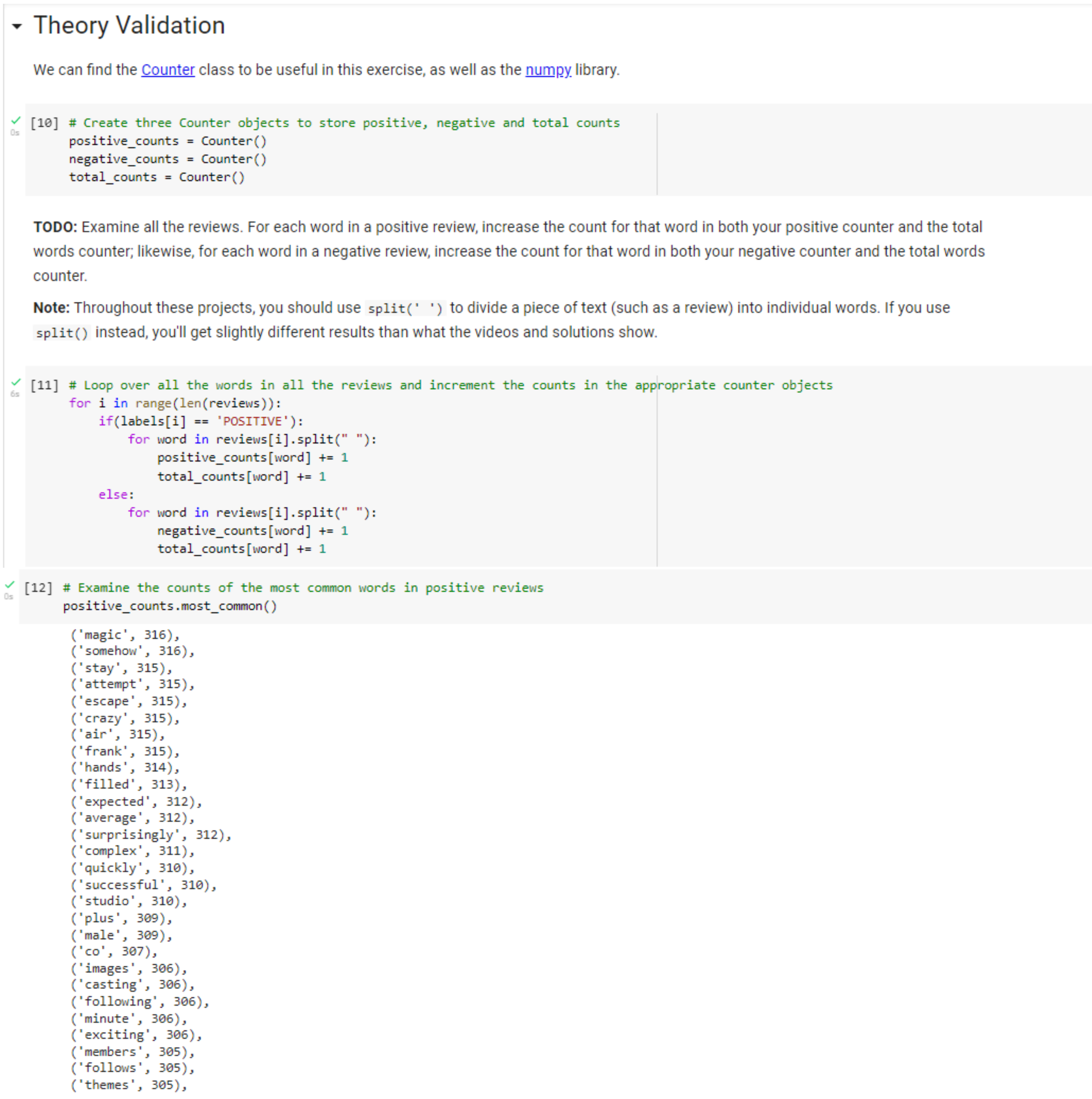
```
[4] def print_review_and_label(i):
        print(labels[i] + "\t:\t" + reviews[i][:80] + "...")

    g = open('Final_Year_Project/reviews.txt','r') # What we know!
    reviews = list(map(lambda x:x[:-1],g.readlines()))
    g.close()

    g = open('Final_Year_Project/labels.txt','r') # What we WANT to know!
    labels = list(map(lambda x:x[:-1].upper(),g.readlines()))
    g.close()
```



```
[5] len(reviews)

    25000

[6] len(labels)

    25000

    reviews[0]

    'bromwell high is a cartoon comedy . it ran at the same time as some other programs about school life  such as  teachers  . my   years in t
    teaching profession lead me to believe that bromwell high  s satire is much closer to reality than is  teachers  . the scramble to survive
    ancially  the insightful students who can see right through their pathetic teachers  pomp  the pettiness of the whole situation  all remind
    of the schools i knew and their students . when i saw the episode in which a student repeatedly tried to burn down the school  i immediatel
    ecalled . . . . . . . . . at . . . . . . . . . . . high . a classic line inspector i  m here to sack one of your teachers . student welcome t
    romwell high . i expect that many adults of my age think that bromwell high is far fetched . what a pity that it isn  t   '

[8] labels[0]

    'POSITIVE'
```

Developing a Predictive Theory

```
[9] print("labels.txt \t : \t reviews.txt\n")
    print_review_and_label(2137)
    print_review_and_label(500)
    print_review_and_label(6267)
    print_review_and_label(1500)
    print_review_and_label(5265)
    print_review_and_label(2000)
    print_review_and_label(9267)

    labels.txt       :       reviews.txt

    NEGATIVE        :       this movie is terrible but it has some good effects .  ...
    POSITIVE        :       i got a good laugh reading all the idiotic comments for this film   br    br   a...
    NEGATIVE        :       comment this movie is impossible . is terrible  very improbable  bad interpretat...
    POSITIVE        :       this film was shot on location in gerard gardens in liverpool  and was the uk  s...
    NEGATIVE        :       what ever happened to one of the most innovative and brilliant storytellers of o...
    POSITIVE        :       this was a must see documentary for me when i missed the opportunity in      so ...
    NEGATIVE        :       wow  i  m shocked to learn that it  s a small world and that we are all intercon...
```



Theory Validation

We can find the Counter class to be useful in this exercise, as well as the numpy library.

```
[10] # Create three Counter objects to store positive, negative and total counts
     positive_counts = Counter()
     negative_counts = Counter()
     total_counts = Counter()
```

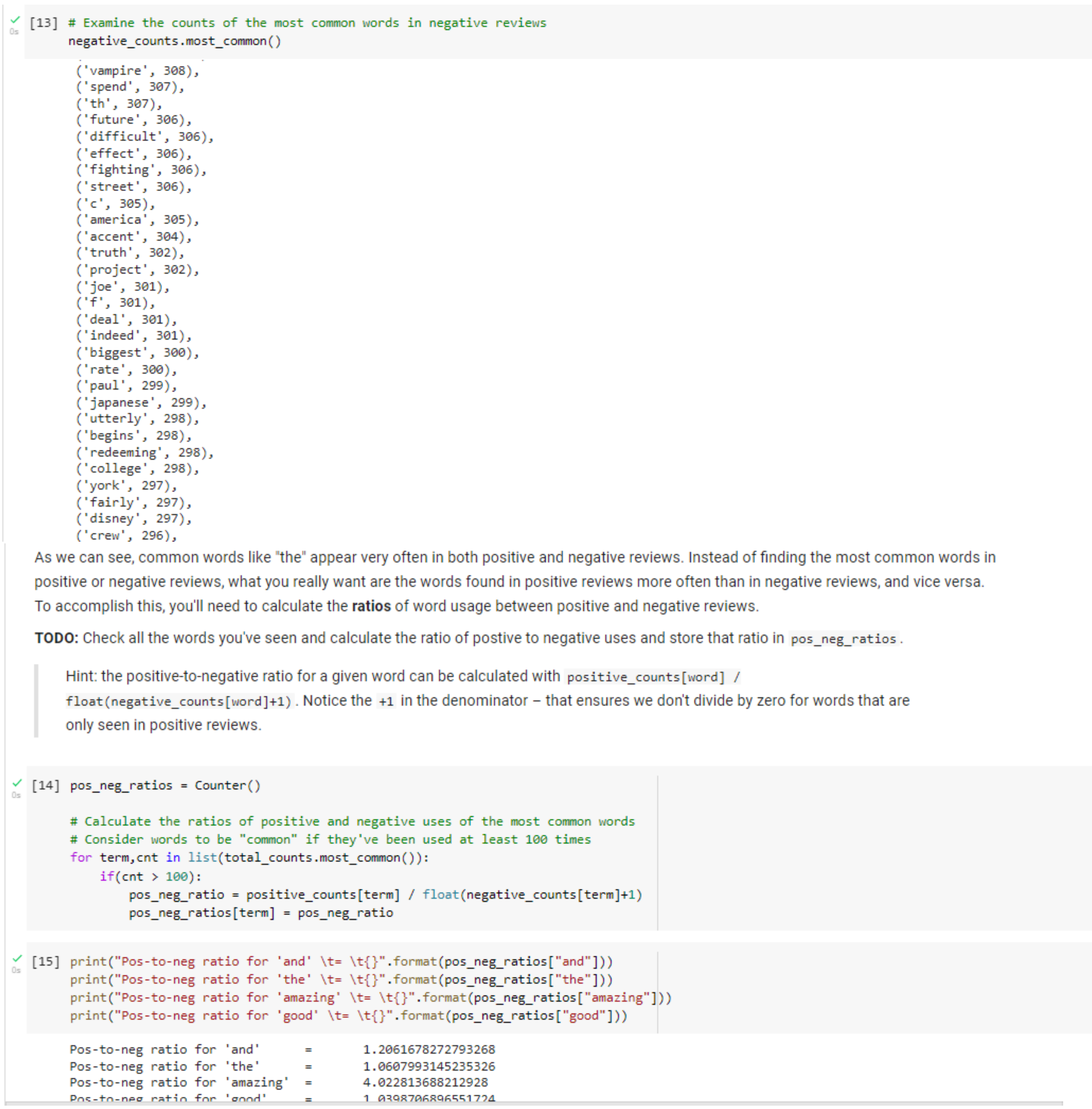
TODO: Examine all the reviews. For each word in a positive review, increase the count for that word in both your positive counter and the total words counter; likewise, for each word in a negative review, increase the count for that word in both your negative counter and the total words counter.

Note: Throughout these projects, you should use `split(' ')` to divide a piece of text (such as a review) into individual words. If you use `split()` instead, you'll get slightly different results than what the videos and solutions show.

```
[11] # Loop over all the words in all the reviews and increment the counts in the appropriate counter objects
     for i in range(len(reviews)):
         if(labels[i] == 'POSITIVE'):
             for word in reviews[i].split(" "):
                 positive_counts[word] += 1
                 total_counts[word] += 1
         else:
             for word in reviews[i].split(" "):
                 negative_counts[word] += 1
                 total_counts[word] += 1
```

```
[12] # Examine the counts of the most common words in positive reviews
     positive_counts.most_common()
```

```
 ('magic', 316),
 ('somehow', 316),
 ('stay', 315),
 ('attempt', 315),
 ('escape', 315),
 ('crazy', 315),
 ('air', 315),
 ('frank', 315),
 ('hands', 314),
 ('filled', 313),
 ('expected', 312),
 ('average', 312),
 ('surprisingly', 312),
 ('complex', 311),
 ('quickly', 310),
 ('successful', 310),
 ('studio', 310),
 ('plus', 309),
 ('male', 309),
 ('co', 307),
 ('images', 306),
 ('casting', 306),
 ('following', 306),
 ('minute', 306),
 ('exciting', 306),
 ('members', 305),
 ('follows', 305),
 ('themes', 305),
```



```
[13] # Examine the counts of the most common words in negative reviews
     negative_counts.most_common()
```

```
  ('vampire', 308),
  ('spend', 307),
  ('th', 307),
  ('future', 306),
  ('difficult', 306),
  ('effect', 306),
  ('fighting', 306),
  ('street', 306),
  ('c', 305),
  ('america', 305),
  ('accent', 304),
  ('truth', 302),
  ('project', 302),
  ('joe', 301),
  ('f', 301),
  ('deal', 301),
  ('indeed', 301),
  ('biggest', 300),
  ('rate', 300),
  ('paul', 299),
  ('japanese', 299),
  ('utterly', 298),
  ('begins', 298),
  ('redeeming', 298),
  ('college', 298),
  ('york', 297),
  ('fairly', 297),
  ('disney', 297),
  ('crew', 296),
```

As we can see, common words like "the" appear very often in both positive and negative reviews. Instead of finding the most common words in positive or negative reviews, what you really want are the words found in positive reviews more often than in negative reviews, and vice versa. To accomplish this, you'll need to calculate the **ratios** of word usage between positive and negative reviews.

TODO: Check all the words you've seen and calculate the ratio of postive to negative uses and store that ratio in `pos_neg_ratios`.

> Hint: the positive-to-negative ratio for a given word can be calculated with `positive_counts[word] / float(negative_counts[word]+1)`. Notice the `+1` in the denominator – that ensures we don't divide by zero for words that are only seen in positive reviews.

```
[14] pos_neg_ratios = Counter()

     # Calculate the ratios of positive and negative uses of the most common words
     # Consider words to be "common" if they've been used at least 100 times
     for term,cnt in list(total_counts.most_common()):
         if(cnt > 100):
             pos_neg_ratio = positive_counts[term] / float(negative_counts[term]+1)
             pos_neg_ratios[term] = pos_neg_ratio
```

```
[15] print("Pos-to-neg ratio for 'and' \t= \t{}".format(pos_neg_ratios["and"]))
     print("Pos-to-neg ratio for 'the' \t= \t{}".format(pos_neg_ratios["the"]))
     print("Pos-to-neg ratio for 'amazing' \t= \t{}".format(pos_neg_ratios["amazing"]))
     print("Pos-to-neg ratio for 'good' \t= \t{}".format(pos_neg_ratios["good"]))
```

```
Pos-to-neg ratio for 'and'      =       1.2061678272793268
Pos-to-neg ratio for 'the'      =       1.0607993145235326
Pos-to-neg ratio for 'amazing'  =       4.022813688212928
Pos-to-neg ratio for 'good'     =       1.0398706896551724
```

Looking closely at the values you just calculated, we see the following:

- Words that you would expect to see more often in positive reviews – like "amazing" – have a ratio greater than 1. The more skewed a word is toward postive, the farther from 1 its positive-to-negative ratio will be.
- Words that you would expect to see more often in negative reviews – like "terrible" – have positive values that are less than 1. The more skewed a word is toward negative, the closer to zero its positive-to-negative ratio will be.
- Neutral words, which don't really convey any sentiment because you would expect to see them in all sorts of reviews – like "the" – have values very close to 1. A perfectly neutral word – one that was used in exactly the same number of positive reviews as negative reviews – would be almost exactly 1. The `+1` we suggested you add to the denominator slightly biases words toward negative, but it won't matter because it will be a tiny bias and later we'll be ignoring words that are too close to neutral anyway.

Ok, the ratios tell us which words are used more often in postive or negative reviews, but the specific values we've calculated are a bit difficult to work with. A very positive word like "amazing" has a value above 4, whereas a very negative word like "terrible" has a value around 0.18. Those values aren't easy to compare for a couple of reasons:

- Right now, 1 is considered neutral, but the absolute value of the postive-to-negative ratios of very postive words is larger than the absolute value of the ratios for the very negative words. So there is no way to directly compare two numbers and see if one word conveys the same magnitude of positive sentiment as another word conveys negative sentiment. So we should center all the values around netural so the absolute value for neutral of the postive-to-negative ratio for a word would indicate how much sentiment (positive or negative) that word conveys.
- When comparing absolute values it's easier to do that around zero than one. To fix these issues, we'll convert all of our ratios to new values using logarithms.

TODO: Go through all the ratios you calculated and convert them to logarithms. (i.e. use `np.log(ratio)`)

In the end, extremely positive and extremely negative words will have positive-to-negative ratios with similar magnitudes but opposite signs.

```
[16] # Convert ratios to logs
     for word,ratio in pos_neg_ratios.most_common():
         pos_neg_ratios[word] = np.log(ratio)
```

```
[17] print("Pos-to-neg ratio for 'and' \t= \t{}".format(pos_neg_ratios["and"]))
     print("Pos-to-neg ratio for 'the' \t= \t{}".format(pos_neg_ratios["the"]))
     print("Pos-to-neg ratio for 'amazing' \t= \t{}".format(pos_neg_ratios["amazing"]))
     print("Pos-to-neg ratio for 'good' \t= \t{}".format(pos_neg_ratios["good"]))
```

```
     Pos-to-neg ratio for 'and'      =       0.18744824888788403
     Pos-to-neg ratio for 'the'      =       0.05902269426102881
     Pos-to-neg ratio for 'amazing'  =       1.3919815802404802
     Pos-to-neg ratio for 'good'     =       0.03909636855278532
```

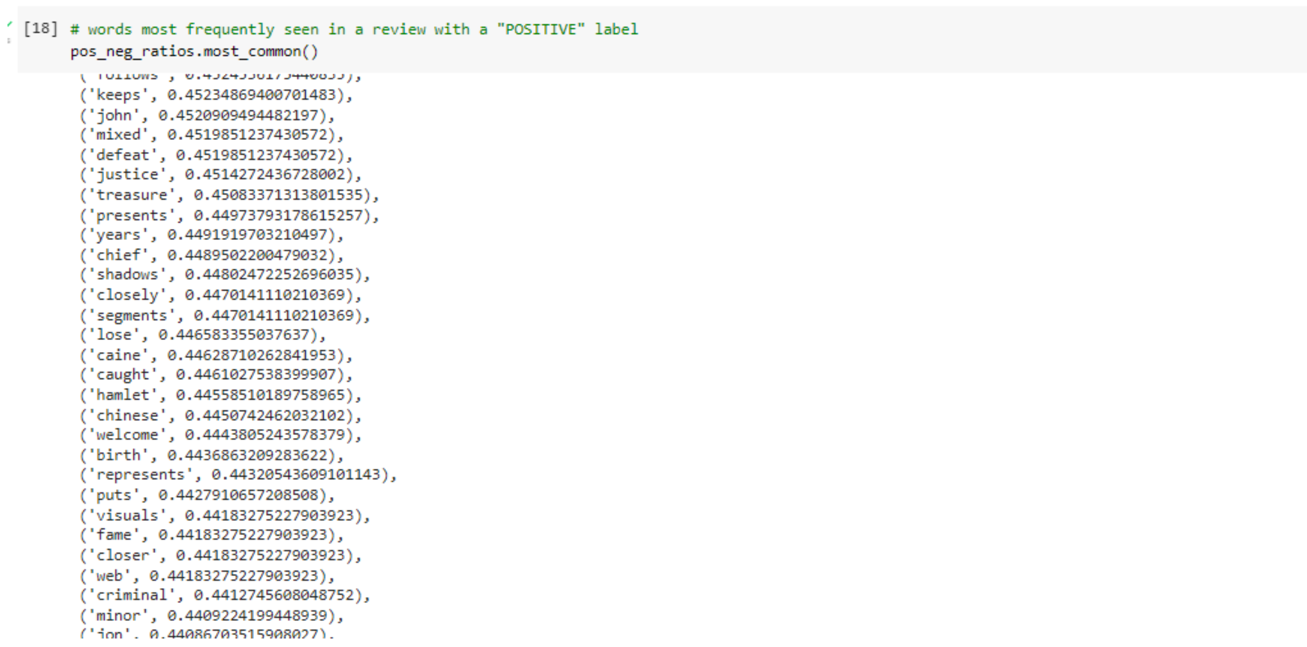
If everything worked, now you should see neutral words with values close to zero. In this case, "the" is near zero but slightly positive, so it was probably used in more positive reviews than negative reviews. But look at "amazing"'s ratio - it's above `1`, showing it is clearly a word with positive sentiment. And "terrible" has a similar score, but in the opposite direction, so it's below `-1`. It's now clear that both of these words are associated with specific, opposing sentiments.

Now run the following cells to see more ratios.

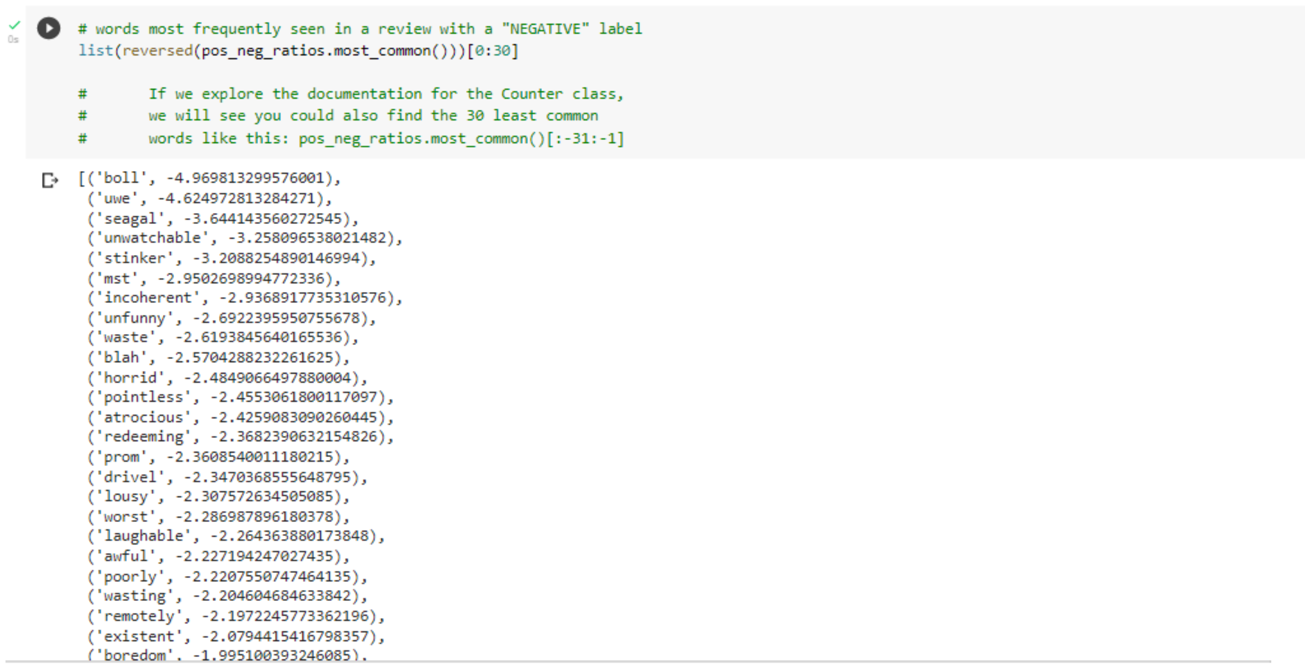
The first cell displays all the words, ordered by how associated they are with postive reviews. (Your notebook will most likely truncate the output so you won't actually see *all* the words in the list.)

The second cell displays the 30 words most associated with negative reviews by reversing the order of the first list and then looking at the first 30 words. (If you want the second cell to display all the words, ordered by how associated they are with negative reviews, you could just write `reversed(pos_neg_ratios.most_common())`.)

You should continue to see values similar to the earlier ones we checked – neutral words will be close to `0`, words will get more positive as their ratios approach and go above `1`, and words will get more negative as their ratios approach and go below `-1`. That's why we decided to use the logs instead of the raw ratios.

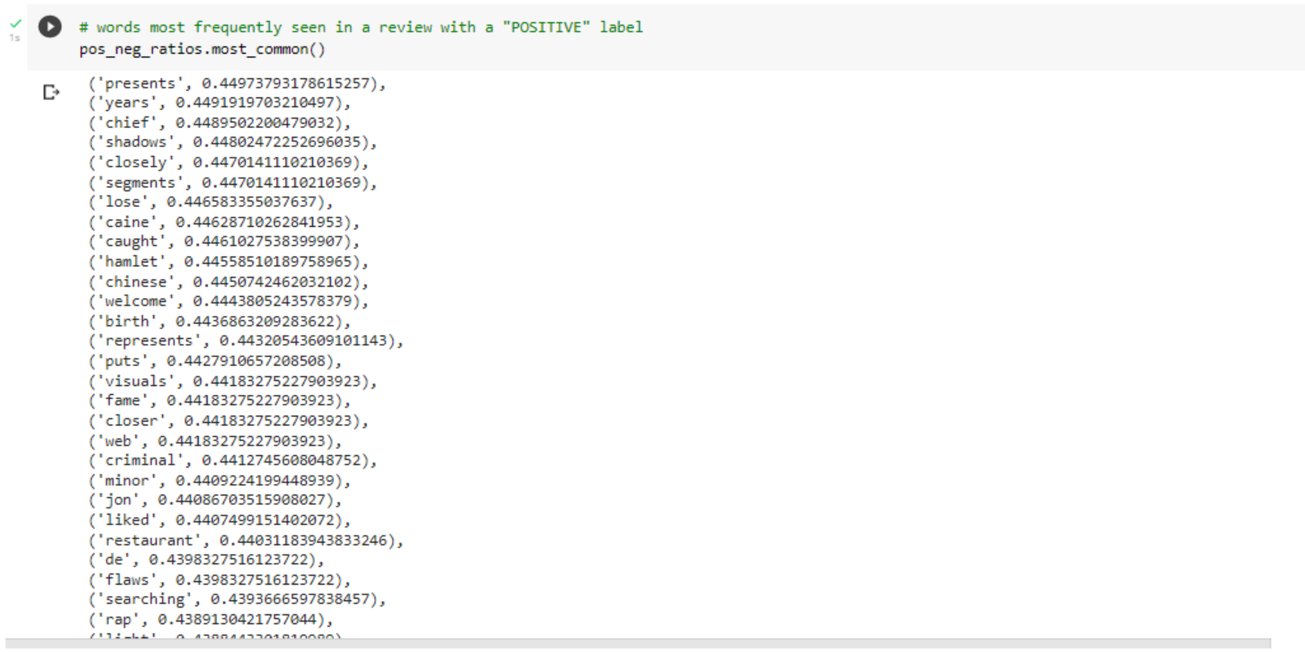


```
[18] # words most frequently seen in a review with a "POSITIVE" label
     pos_neg_ratios.most_common()
      ('follows', 0.452[illegible]),
      ('keeps', 0.45234869400701483),
      ('john', 0.4520909494482197),
      ('mixed', 0.4519851237430572),
      ('defeat', 0.4519851237430572),
      ('justice', 0.4514272436728002),
      ('treasure', 0.45083371313801535),
      ('presents', 0.44973793178615257),
      ('years', 0.4491919703210497),
      ('chief', 0.4489502200479032),
      ('shadows', 0.44802472252696035),
      ('closely', 0.4470141110210369),
      ('segments', 0.4470141110210369),
      ('lose', 0.446583355037637),
      ('caine', 0.44628710262841953),
      ('caught', 0.4461027538399907),
      ('hamlet', 0.44558510189758965),
      ('chinese', 0.4450742462032102),
      ('welcome', 0.4443805243578379),
      ('birth', 0.4436863209283622),
      ('represents', 0.44320543609101143),
      ('puts', 0.4427910657208508),
      ('visuals', 0.44183275227903923),
      ('fame', 0.44183275227903923),
      ('closer', 0.44183275227903923),
      ('web', 0.44183275227903923),
      ('criminal', 0.4412745608048752),
      ('minor', 0.4409224199448939),
      ('jon', 0.44086703515908027),
```



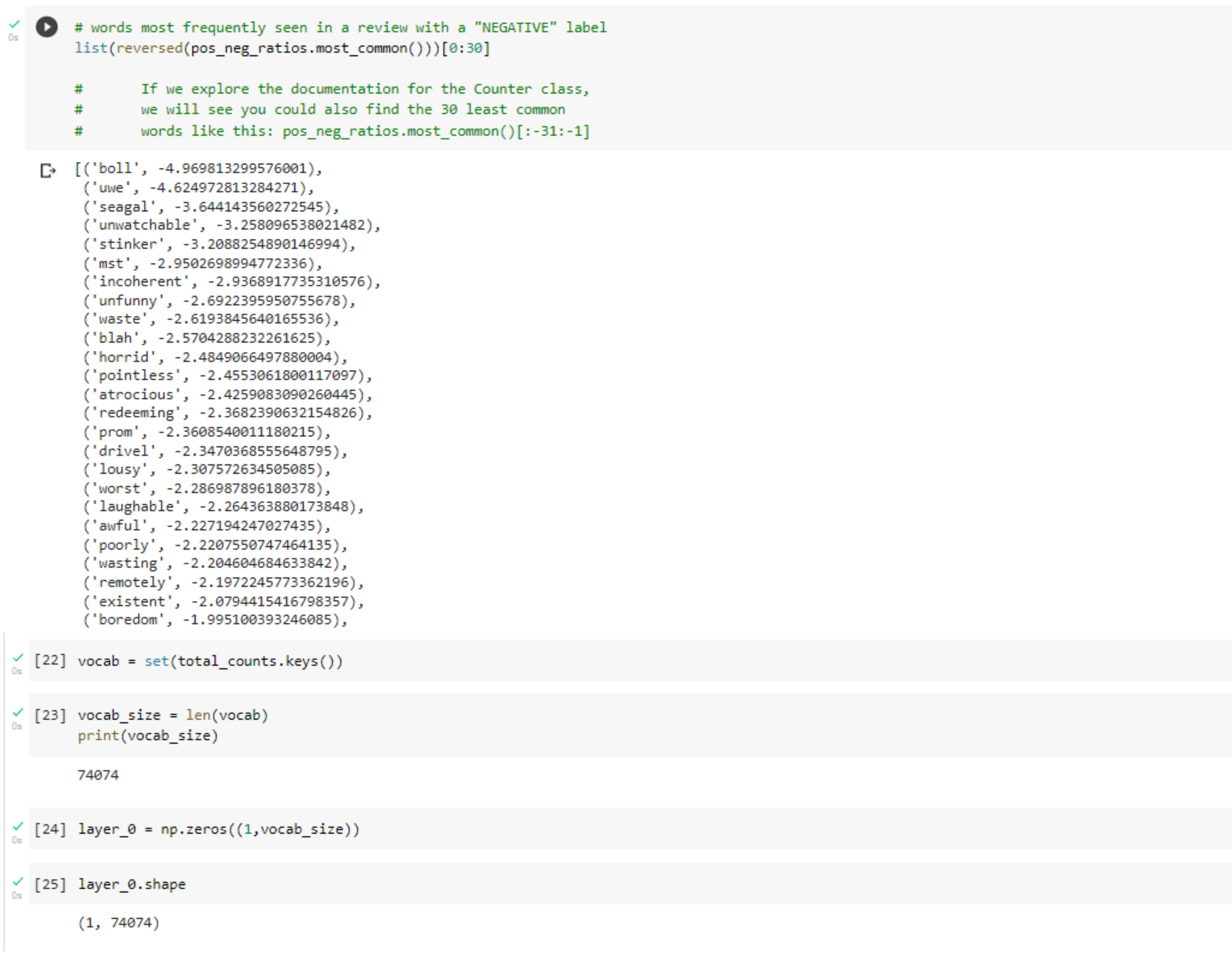
```
# words most frequently seen in a review with a "NEGATIVE" label
list(reversed(pos_neg_ratios.most_common()))[0:30]

#       If we explore the documentation for the Counter class,
#       we will see you could also find the 30 least common
#       words like this: pos_neg_ratios.most_common()[:-31:-1]
```

```
[('boll', -4.969813299576001),
 ('uwe', -4.624972813284271),
 ('seagal', -3.644143560272545),
 ('unwatchable', -3.258096538021482),
 ('stinker', -3.2088254890146994),
 ('mst', -2.9502698994772336),
 ('incoherent', -2.9368917735310576),
 ('unfunny', -2.6922395950755678),
 ('waste', -2.6193845640165536),
 ('blah', -2.5704288232261625),
 ('horrid', -2.4849066497880004),
 ('pointless', -2.4553061800117097),
 ('atrocious', -2.4259083090260445),
 ('redeeming', -2.3682390632154826),
 ('prom', -2.3608540011180215),
 ('drivel', -2.3470368555648795),
 ('lousy', -2.307572634505085),
 ('worst', -2.286987896180378),
 ('laughable', -2.264363880173848),
 ('awful', -2.227194247027435),
 ('poorly', -2.2207550747464135),
 ('wasting', -2.204604684633842),
 ('remotely', -2.1972245773362196),
 ('existent', -2.0794415416798357),
 ('boredom', -1.995100393246085),
```



```
# words most frequently seen in a review with a "POSITIVE" label
pos_neg_ratios.most_common()
```

```
 ('presents', 0.4497379317861525 7),
 ('years', 0.4491919703210497),
 ('chief', 0.4489502200479032),
 ('shadows', 0.4480247225269603 5),
 ('closely', 0.447014111021036 9),
 ('segments', 0.447014111021036 9),
 ('lose', 0.446583355037637),
 ('caine', 0.4462871026284195 3),
 ('caught', 0.4461027538399907),
 ('hamlet', 0.4455851018975896 5),
 ('chinese', 0.4450742462032102),
 ('welcome', 0.4443805243578379),
 ('birth', 0.443686320928362 2),
 ('represents', 0.4432054360910114 3),
 ('puts', 0.4427910657208508),
 ('visuals', 0.4418327522790392 3),
 ('fame', 0.4418327522790392 3),
 ('closer', 0.4418327522790392 3),
 ('web', 0.4418327522790392 3),
 ('criminal', 0.4412745608048752),
 ('minor', 0.4409224199448939),
 ('jon', 0.4408670351590802 7),
 ('liked', 0.4407499151402072),
 ('restaurant', 0.4403118394383324 6),
 ('de', 0.4398327516123722),
 ('flaws', 0.4398327516123722),
 ('searching', 0.4393666597838457),
 ('rap', 0.4389130421757044),
```



```
# words most frequently seen in a review with a "NEGATIVE" label
list(reversed(pos_neg_ratios.most_common()))[0:30]

#       If we explore the documentation for the Counter class,
#       we will see you could also find the 30 least common
#       words like this: pos_neg_ratios.most_common()[:-31:-1]
```

```
[('boll', -4.969813299576001),
 ('uwe', -4.624972813284271),
 ('seagal', -3.644143560272545),
 ('unwatchable', -3.258096538021482),
 ('stinker', -3.2088254890146994),
 ('mst', -2.9502698994772336),
 ('incoherent', -2.9368917735310576),
 ('unfunny', -2.6922395950755678),
 ('waste', -2.6193845640165536),
 ('blah', -2.5704288232261625),
 ('horrid', -2.4849066497880004),
 ('pointless', -2.4553061800117097),
 ('atrocious', -2.4259083090260445),
 ('redeeming', -2.3682390632154826),
 ('prom', -2.3608540011180215),
 ('drivel', -2.3470368555648795),
 ('lousy', -2.307572634505085),
 ('worst', -2.286987896180378),
 ('laughable', -2.264363880173848),
 ('awful', -2.227194247027435),
 ('poorly', -2.2207550747464135),
 ('wasting', -2.204604684633842),
 ('remotely', -2.1972245773362196),
 ('existent', -2.0794415416798357),
 ('boredom', -1.995100393246085),
```

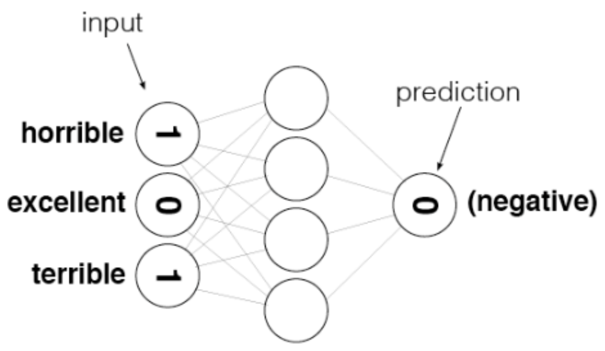
```
[22] vocab = set(total_counts.keys())
```

```
[23] vocab_size = len(vocab)
     print(vocab_size)

     74074
```

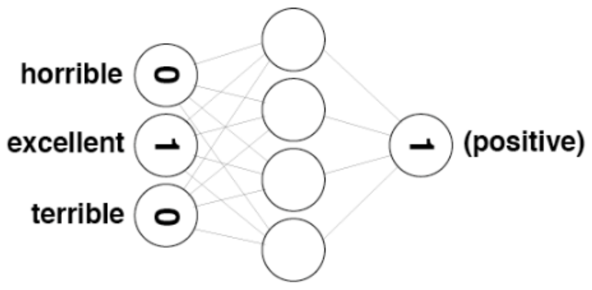
```
[24] layer_0 = np.zeros((1,vocab_size))
```

```
[25] layer_0.shape

     (1, 74074)
```

```
[26] from IPython.display import Image

     review = "This was a horrible, terrible movie."

     Image(filename='Final_Year_Project/Images/sentiment_network.png')
```



```
[27] review = "The movie was excellent"

     Image(filename='Final_Year_Project/Images/sentiment_network_pos.png')
```



```
[28] def update_input_layer(review):
         global layer_0
         # clear out previous state, reset the layer to be all 0s
         layer_0 *= 0
         # count how many times each word is used in the given review and store the results in layer_0
         for word in review.split(" "):
             layer_0[0][word2index[word]] += 1
```

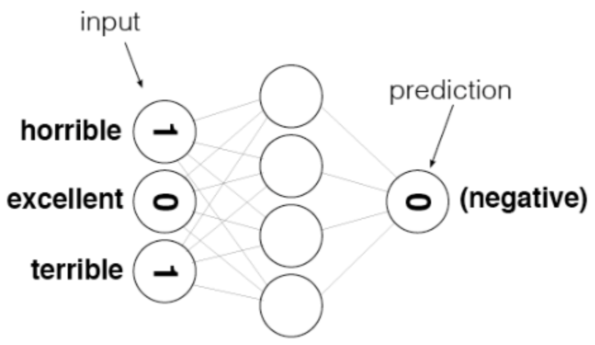
Running the following cell to test updating the input layer with the first review. The indices assigned may not be the same as in the solution, but hopefully we'll see some non-zero values in `layer_0`.

```
[29] layer_0 = np.zeros((1,vocab_size))
```

```
[30] layer_0.shape
```

```
(1, 74074)
```

```
[31] from IPython.display import Image
     Image(filename='Final_Year_Project/Images/sentiment_network.png')
```



`layer_0` contains one entry for every word in the vocabulary, as shown in the above image. We need to make sure we know the index of each word, so run the following cell to create a lookup table that stores the index of every word.

```
[32] # Create a dictionary of words in the vocabulary mapped to index positions
     # (to be used in layer_0)
     word2index = {}
     for i,word in enumerate(vocab):
         word2index[word] = i

     # display the map of words to indices
     word2index
```

```
      'decca': 836,
      'plainness': 837,
      'matkondar': 838,
      'vandalizes': 839,
      'looked': 840,
      'bandes': 841,
      'mics': 842,
      'persbrandt': 843,
      'immigrant': 844,
      'kangaroo': 845,
      'peterson': 846,
      'affectations': 847,
      'fim': 848,
      'savitri': 849,
      'agey': 850,
```

Implementation of `update_input_layer`. It should count how many times each word is used in the given review, and then store those counts at the appropriate indices inside `layer_0`.

```
def update_input_layer(review):
    global layer_0
    # clear out previous state, reset the layer to be all 0s
    layer_0 *= 0
    # count how many times each word is used in the given review and store the results in layer_0
    for word in review.split(" "):
        layer_0[0][word2index[word]] += 1
```

```
[34] update_input_layer(reviews[0])
     layer_0
```

```
array([[18.,  0.,  0., ...,  0.,  0.,  0.]])
```

Implementation of `get_target_for_labels`. It should return `0` or `1`, depending on whether the given label is `NEGATIVE` or `POSITIVE`, respectively.

```
def get_target_for_label(label):
    if(label == 'POSITIVE'):
        return 1
    else:
        return 0
```

Run the following two cells. They should print out `'POSITIVE'` and `1`, respectively.

```
[36] labels[0]
```

```
'POSITIVE'
```

```
[37] get_target_for_label(labels[0])
```

```
1
```

Run the following two cells. They should print out `'NEGATIVE'` and `0`, respectively.

```
[38] labels[1]
```

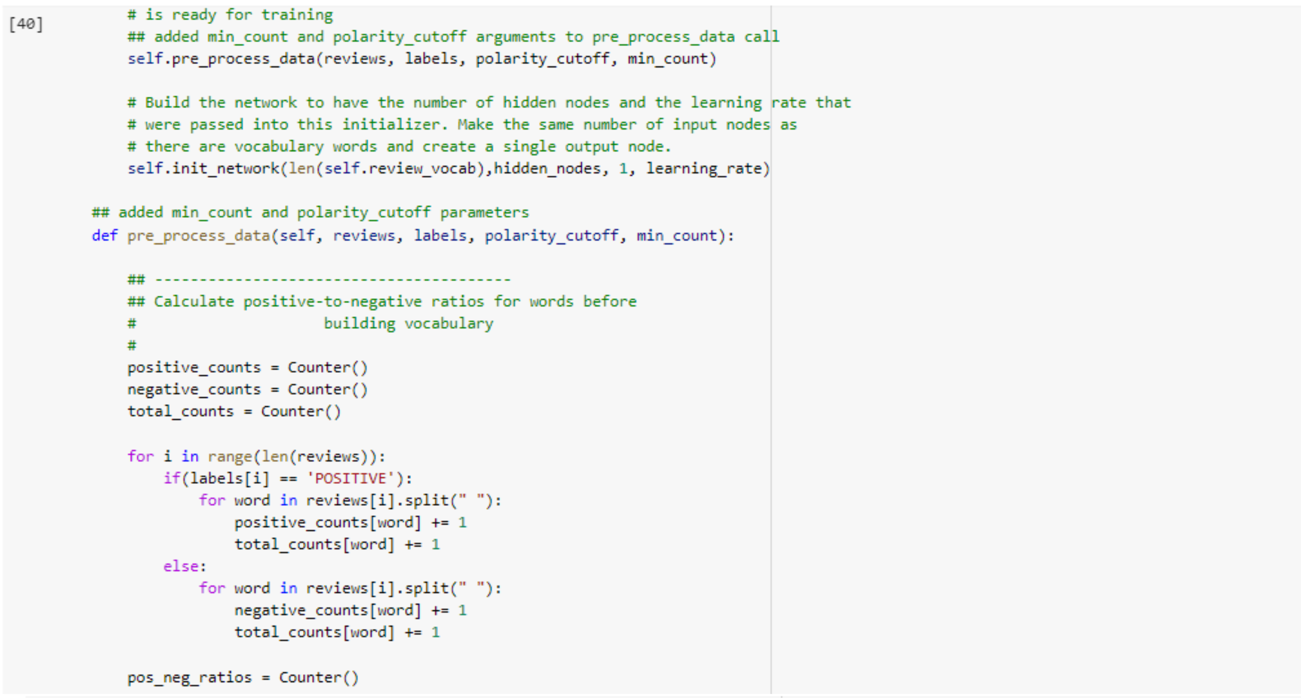
```
'NEGATIVE'
```

```
[39] get_target_for_label(labels[1])
```

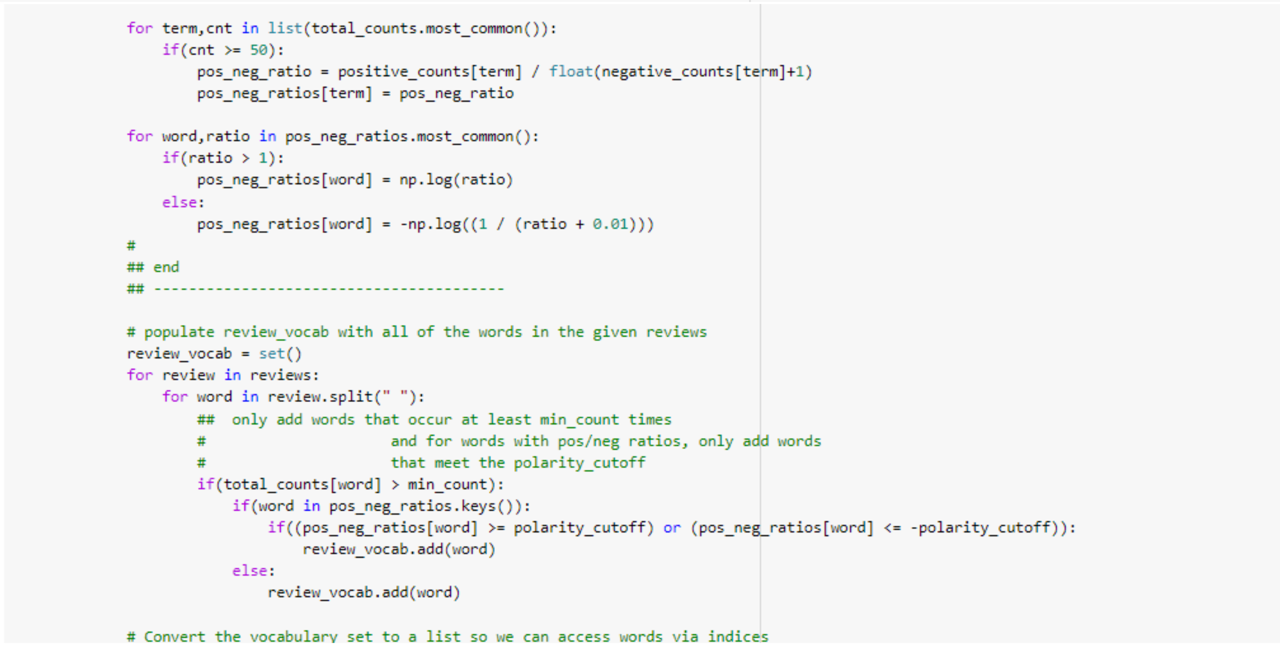
```
0
```

Building a Neural Network

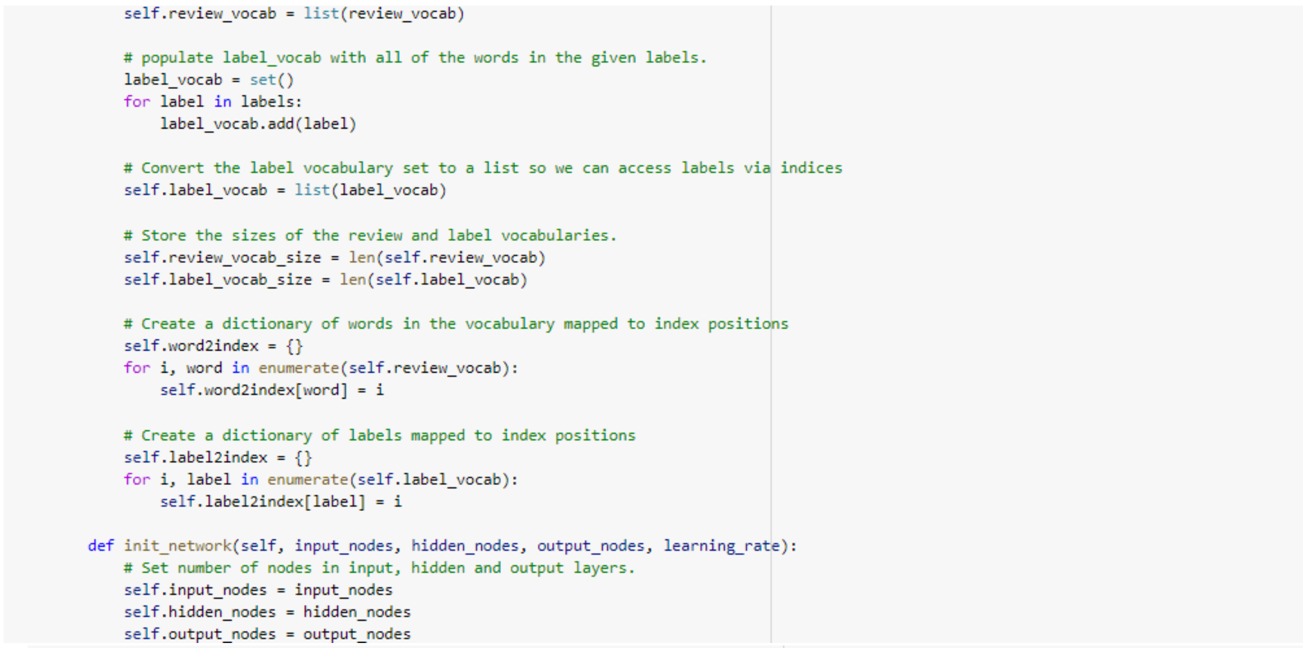
```
[40] import time
     import sys
     import numpy as np

     # Encapsulate our neural network in a class
     class SentimentNetwork:
         ## added min_count and polarity_cutoff parameters
         def __init__(self, reviews,labels,min_count = 10,polarity_cutoff = 0.1,hidden_nodes = 10, learning_rate = 0.1):
             """Create a SentimenNetwork with the given settings
             Args:
                 reviews(list) - List of reviews used for training
                 labels(list) - List of POSITIVE/NEGATIVE labels associated with the given reviews
                 min_count(int) - Words should only be added to the vocabulary
                                  if they occur more than this many times
                 polarity_cutoff(float) - The absolute value of a word's positive-to-negative
                                          ratio must be at least this big to be considered.
                 hidden_nodes(int) - Number of nodes to create in the hidden layer
                 learning_rate(float) - Learning rate to use while training

             """
             # Assign a seed to our random number generator to ensure we get
             # reproducable results during development
             np.random.seed(1)

             # process the reviews and their associated labels so that everything
             # is ready for training
             ## added min_count and polarity_cutoff arguments to pre_process_data call
```



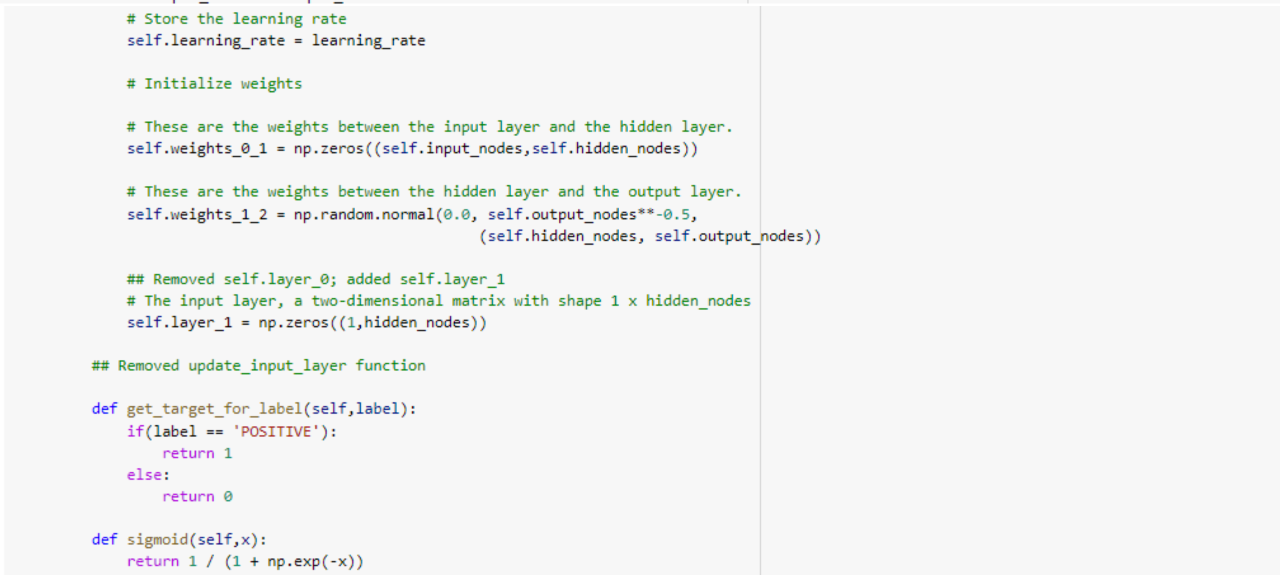
```
[40]        # is ready for training
        ## added min_count and polarity_cutoff arguments to pre_process_data call
        self.pre_process_data(reviews, labels, polarity_cutoff, min_count)

        # Build the network to have the number of hidden nodes and the learning rate that
        # were passed into this initializer. Make the same number of input nodes as
        # there are vocabulary words and create a single output node.
        self.init_network(len(self.review_vocab),hidden_nodes, 1, learning_rate)

    ## added min_count and polarity_cutoff parameters
    def pre_process_data(self, reviews, labels, polarity_cutoff, min_count):

        ## ----------------------------------------
        ## Calculate positive-to-negative ratios for words before
        #                    building vocabulary
        #
        positive_counts = Counter()
        negative_counts = Counter()
        total_counts = Counter()

        for i in range(len(reviews)):
            if(labels[i] == 'POSITIVE'):
                for word in reviews[i].split(" "):
                    positive_counts[word] += 1
                    total_counts[word] += 1
            else:
                for word in reviews[i].split(" "):
                    negative_counts[word] += 1
                    total_counts[word] += 1

        pos_neg_ratios = Counter()
```



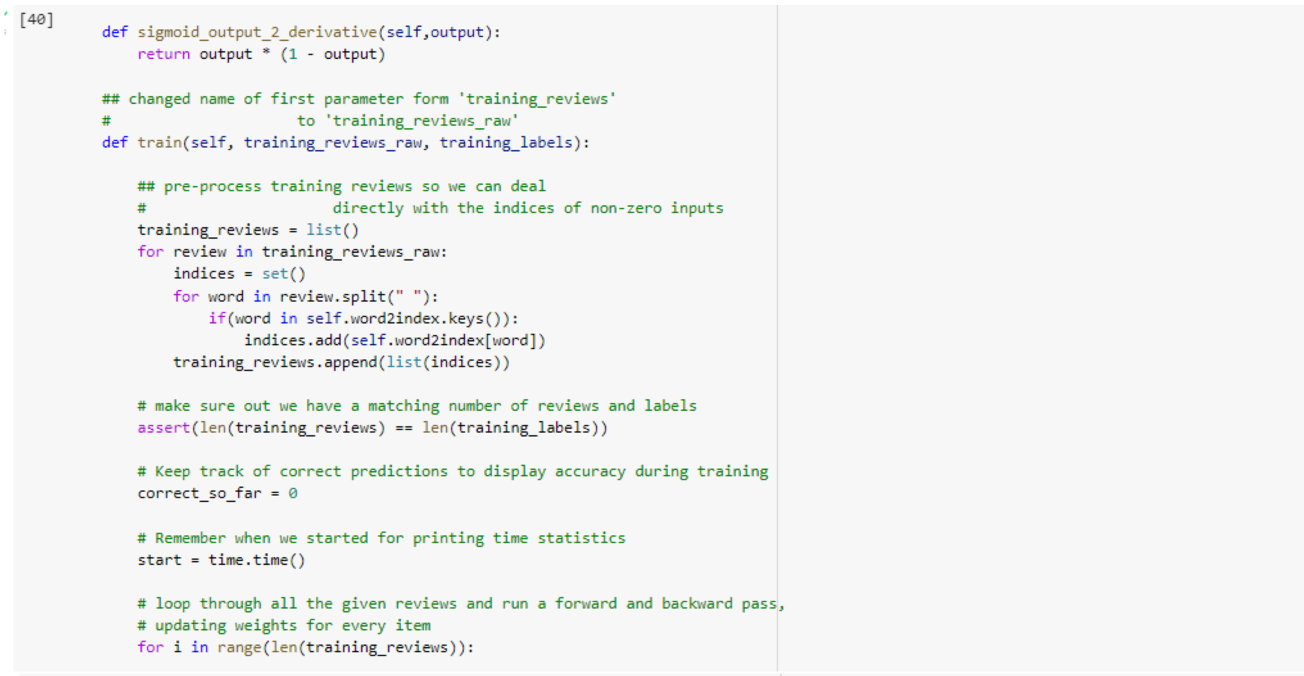
```
        for term,cnt in list(total_counts.most_common()):
            if(cnt >= 50):
                pos_neg_ratio = positive_counts[term] / float(negative_counts[term]+1)
                pos_neg_ratios[term] = pos_neg_ratio

        for word,ratio in pos_neg_ratios.most_common():
            if(ratio > 1):
                pos_neg_ratios[word] = np.log(ratio)
            else:
                pos_neg_ratios[word] = -np.log((1 / (ratio + 0.01)))
        #
        ## end
        ## ----------------------------------------

        # populate review_vocab with all of the words in the given reviews
        review_vocab = set()
        for review in reviews:
            for word in review.split(" "):
                ##  only add words that occur at least min_count times
                #                     and for words with pos/neg ratios, only add words
                #                     that meet the polarity_cutoff
                if(total_counts[word] > min_count):
                    if(word in pos_neg_ratios.keys()):
                        if((pos_neg_ratios[word] >= polarity_cutoff) or (pos_neg_ratios[word] <= -polarity_cutoff)):
                            review_vocab.add(word)
                    else:
                        review_vocab.add(word)

        # Convert the vocabulary set to a list so we can access words via indices
```



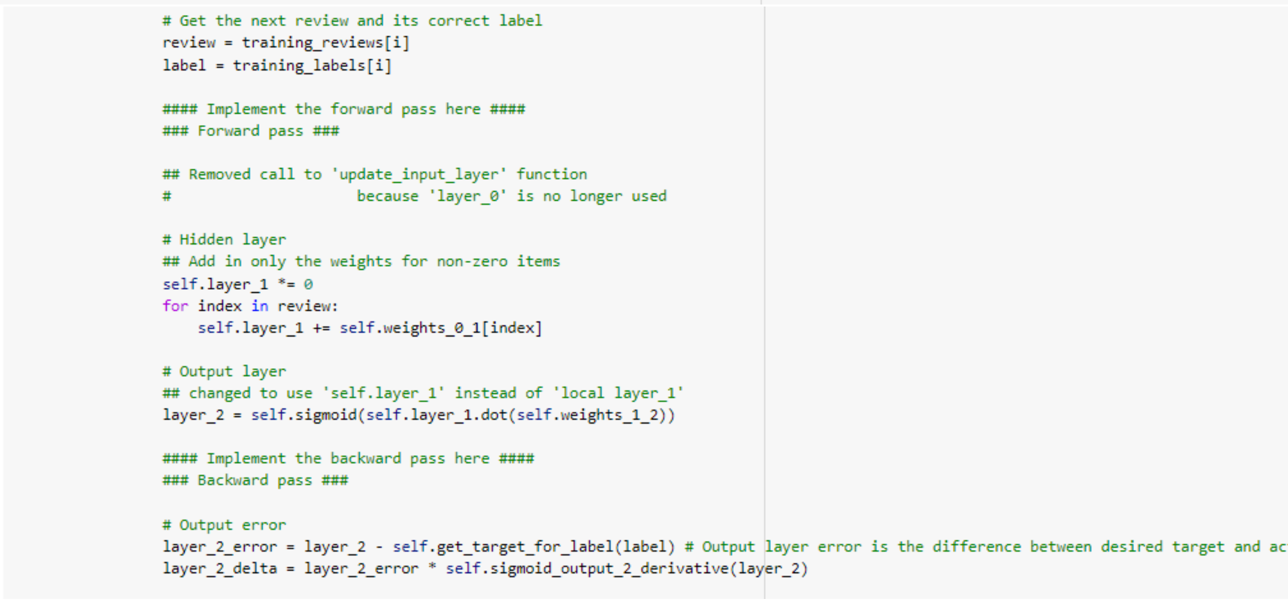
```python
        self.review_vocab = list(review_vocab)

        # populate label_vocab with all of the words in the given labels.
        label_vocab = set()
        for label in labels:
            label_vocab.add(label)

        # Convert the label vocabulary set to a list so we can access labels via indices
        self.label_vocab = list(label_vocab)

        # Store the sizes of the review and label vocabularies.
        self.review_vocab_size = len(self.review_vocab)
        self.label_vocab_size = len(self.label_vocab)

        # Create a dictionary of words in the vocabulary mapped to index positions
        self.word2index = {}
        for i, word in enumerate(self.review_vocab):
            self.word2index[word] = i

        # Create a dictionary of labels mapped to index positions
        self.label2index = {}
        for i, label in enumerate(self.label_vocab):
            self.label2index[label] = i

    def init_network(self, input_nodes, hidden_nodes, output_nodes, learning_rate):
        # Set number of nodes in input, hidden and output layers.
        self.input_nodes = input_nodes
        self.hidden_nodes = hidden_nodes
        self.output_nodes = output_nodes
```



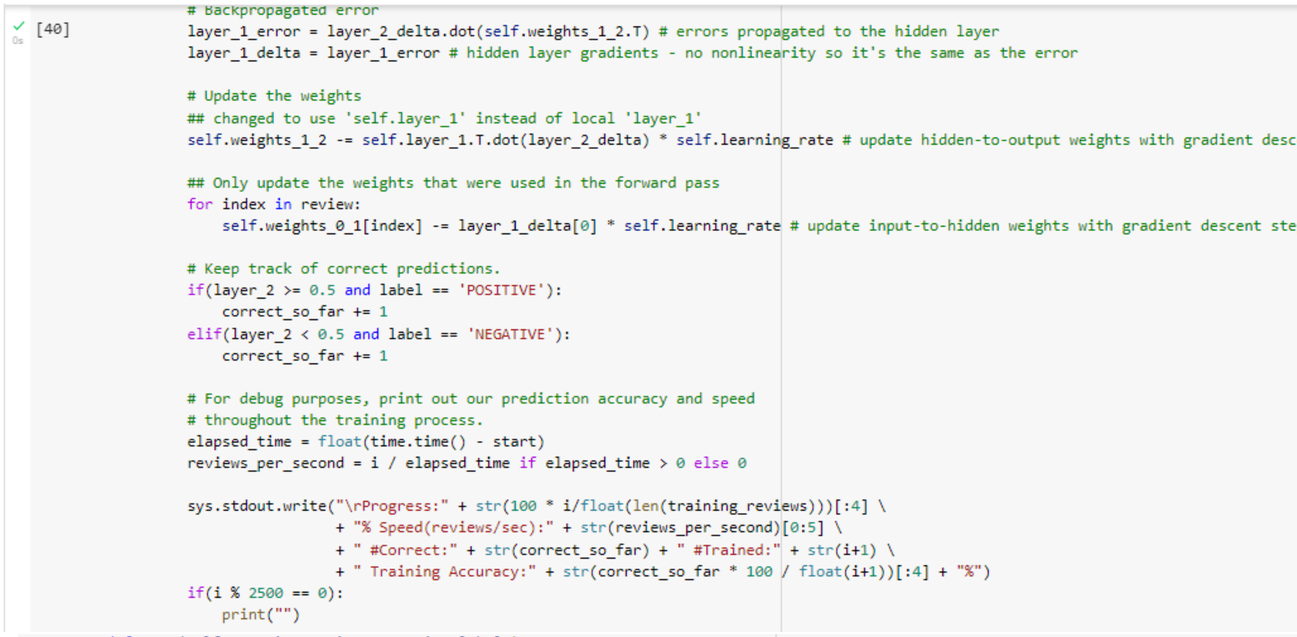
```python
        # Store the learning rate
        self.learning_rate = learning_rate

        # Initialize weights

        # These are the weights between the input layer and the hidden layer.
        self.weights_0_1 = np.zeros((self.input_nodes,self.hidden_nodes))

        # These are the weights between the hidden layer and the output layer.
        self.weights_1_2 = np.random.normal(0.0, self.output_nodes**-0.5,
                                            (self.hidden_nodes, self.output_nodes))

        ## Removed self.layer_0; added self.layer_1
        # The input layer, a two-dimensional matrix with shape 1 x hidden_nodes
        self.layer_1 = np.zeros((1,hidden_nodes))

    ## Removed update_input_layer function

    def get_target_for_label(self,label):
        if(label == 'POSITIVE'):
            return 1
        else:
            return 0

    def sigmoid(self,x):
        return 1 / (1 + np.exp(-x))
```



```
[40]
    def sigmoid_output_2_derivative(self,output):
        return output * (1 - output)

    ## changed name of first parameter form 'training_reviews'
    #                     to 'training_reviews_raw'
    def train(self, training_reviews_raw, training_labels):

        ## pre-process training reviews so we can deal
        #                      directly with the indices of non-zero inputs
        training_reviews = list()
        for review in training_reviews_raw:
            indices = set()
            for word in review.split(" "):
                if(word in self.word2index.keys()):
                    indices.add(self.word2index[word])
            training_reviews.append(list(indices))

        # make sure out we have a matching number of reviews and labels
        assert(len(training_reviews) == len(training_labels))

        # Keep track of correct predictions to display accuracy during training
        correct_so_far = 0

        # Remember when we started for printing time statistics
        start = time.time()

        # loop through all the given reviews and run a forward and backward pass,
        # updating weights for every item
        for i in range(len(training_reviews)):
```

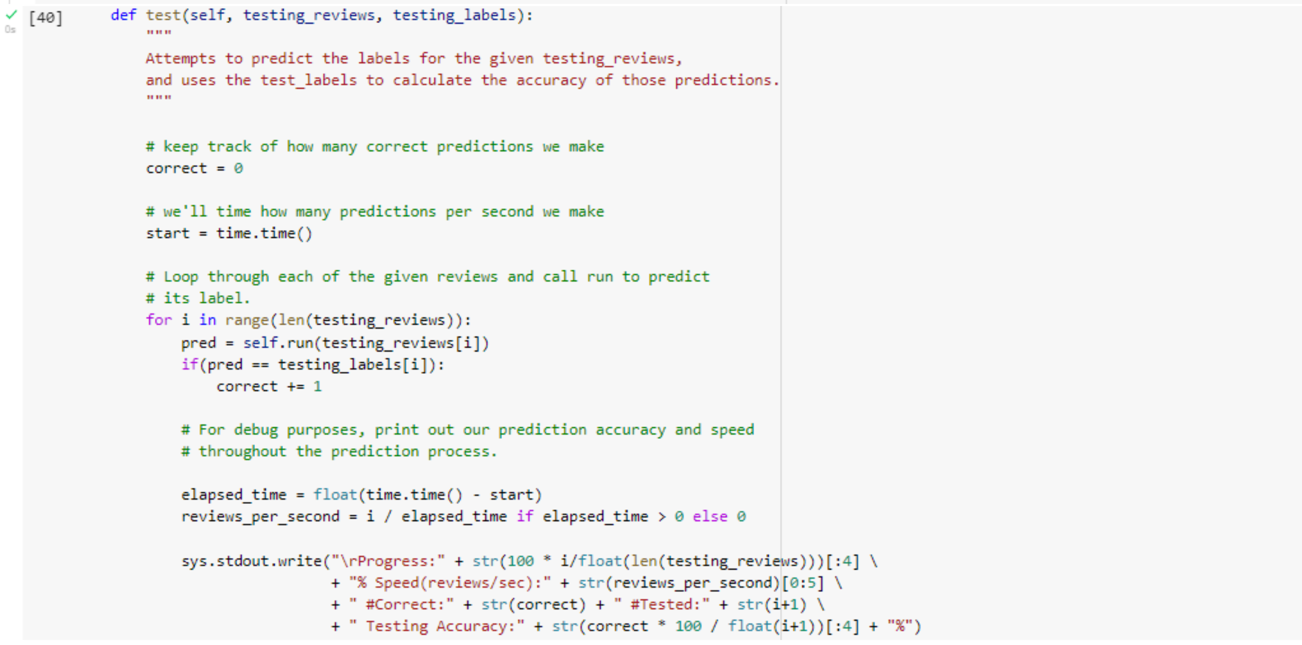


```
            # Get the next review and its correct label
            review = training_reviews[i]
            label = training_labels[i]

            #### Implement the forward pass here ####
            ### Forward pass ###

            ## Removed call to 'update_input_layer' function
            #                     because 'layer_0' is no longer used

            # Hidden layer
            ## Add in only the weights for non-zero items
            self.layer_1 *= 0
            for index in review:
                self.layer_1 += self.weights_0_1[index]

            # Output layer
            ## changed to use 'self.layer_1' instead of 'local layer_1'
            layer_2 = self.sigmoid(self.layer_1.dot(self.weights_1_2))

            #### Implement the backward pass here ####
            ### Backward pass ###

            # Output error
            layer_2_error = layer_2 - self.get_target_for_label(label) # Output layer error is the difference between desired target and ac
            layer_2_delta = layer_2_error * self.sigmoid_output_2_derivative(layer_2)
```



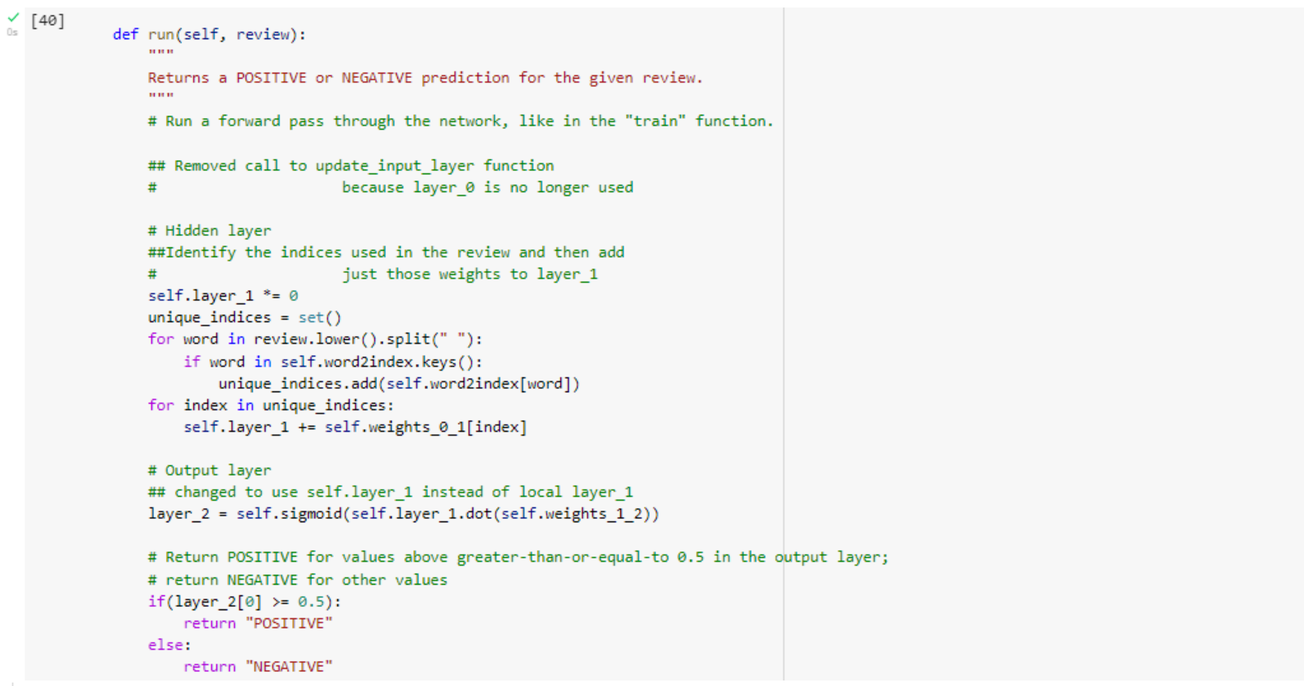
[40]

```
        # Backpropagated error
        layer_1_error = layer_2_delta.dot(self.weights_1_2.T) # errors propagated to the hidden layer
        layer_1_delta = layer_1_error # hidden layer gradients - no nonlinearity so it's the same as the error

        # Update the weights
        ## changed to use 'self.layer_1' instead of local 'layer_1'
        self.weights_1_2 -= self.layer_1.T.dot(layer_2_delta) * self.learning_rate # update hidden-to-output weights with gradient desc

        ## Only update the weights that were used in the forward pass
        for index in review:
            self.weights_0_1[index] -= layer_1_delta[0] * self.learning_rate # update input-to-hidden weights with gradient descent ste

        # Keep track of correct predictions.
        if(layer_2 >= 0.5 and label == 'POSITIVE'):
            correct_so_far += 1
        elif(layer_2 < 0.5 and label == 'NEGATIVE'):
            correct_so_far += 1

        # For debug purposes, print out our prediction accuracy and speed
        # throughout the training process.
        elapsed_time = float(time.time() - start)
        reviews_per_second = i / elapsed_time if elapsed_time > 0 else 0

        sys.stdout.write("\rProgress:" + str(100 * i/float(len(training_reviews)))[:4] \
                         + "% Speed(reviews/sec):" + str(reviews_per_second)[0:5] \
                         + " #Correct:" + str(correct_so_far) + " #Trained:" + str(i+1) \
                         + " Training Accuracy:" + str(correct_so_far * 100 / float(i+1))[:4] + "%")
        if(i % 2500 == 0):
            print("")
```

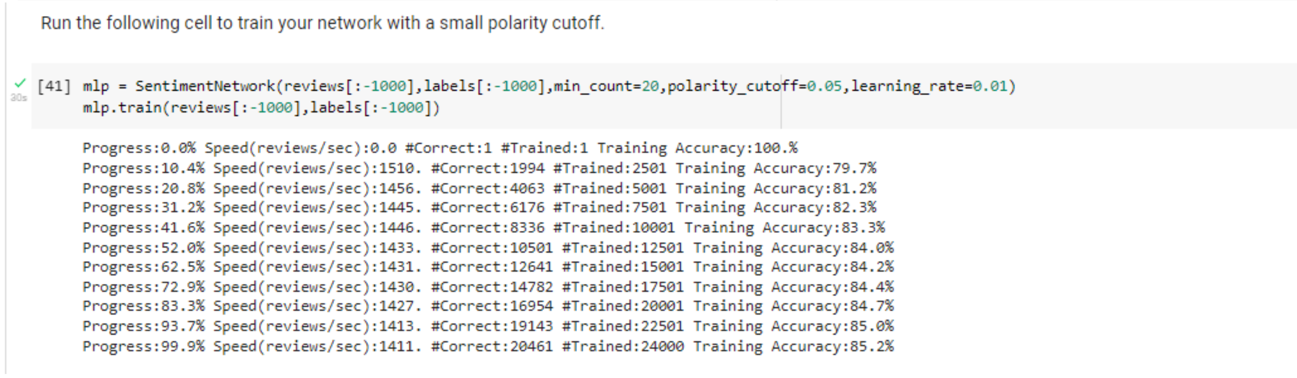


[40]

```
    def test(self, testing_reviews, testing_labels):
        """
        Attempts to predict the labels for the given testing_reviews,
        and uses the test_labels to calculate the accuracy of those predictions.
        """

        # keep track of how many correct predictions we make
        correct = 0

        # we'll time how many predictions per second we make
        start = time.time()

        # Loop through each of the given reviews and call run to predict
        # its label.
        for i in range(len(testing_reviews)):
            pred = self.run(testing_reviews[i])
            if(pred == testing_labels[i]):
                correct += 1

            # For debug purposes, print out our prediction accuracy and speed
            # throughout the prediction process.

            elapsed_time = float(time.time() - start)
            reviews_per_second = i / elapsed_time if elapsed_time > 0 else 0

            sys.stdout.write("\rProgress:" + str(100 * i/float(len(testing_reviews)))[:4] \
                             + "% Speed(reviews/sec):" + str(reviews_per_second)[0:5] \
                             + " #Correct:" + str(correct) + " #Tested:" + str(i+1) \
                             + " Testing Accuracy:" + str(correct * 100 / float(i+1))[:4] + "%")
```

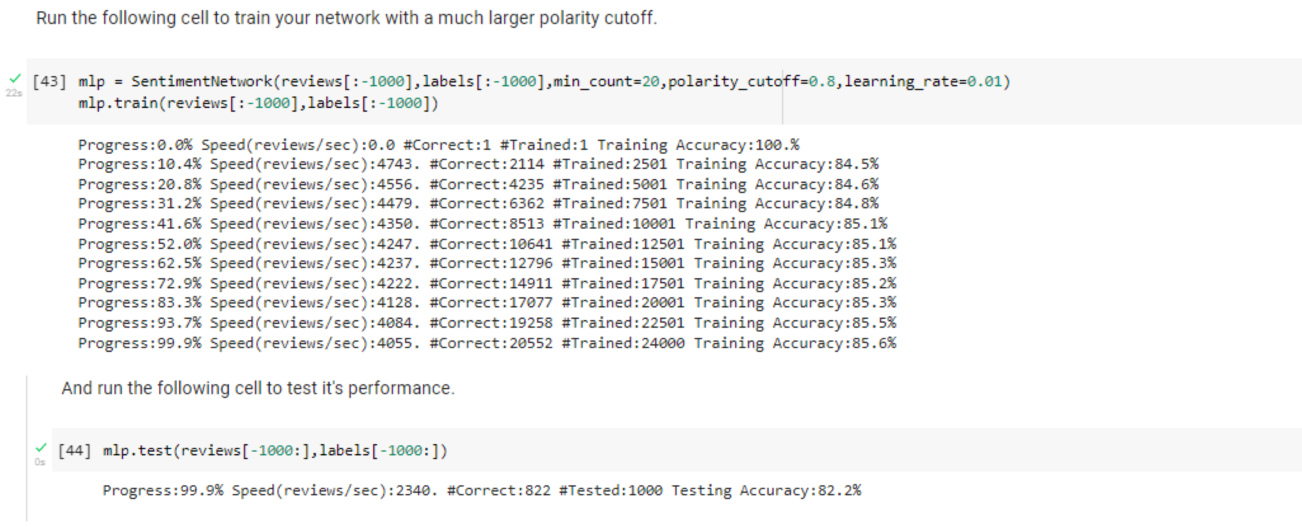


```
[40]
        def run(self, review):
            """
            Returns a POSITIVE or NEGATIVE prediction for the given review.
            """
            # Run a forward pass through the network, like in the "train" function.

            ## Removed call to update_input_layer function
            #                     because layer_0 is no longer used

            # Hidden layer
            ##Identify the indices used in the review and then add
            #                     just those weights to layer_1
            self.layer_1 *= 0
            unique_indices = set()
            for word in review.lower().split(" "):
                if word in self.word2index.keys():
                    unique_indices.add(self.word2index[word])
            for index in unique_indices:
                self.layer_1 += self.weights_0_1[index]

            # Output layer
            ## changed to use self.layer_1 instead of local layer_1
            layer_2 = self.sigmoid(self.layer_1.dot(self.weights_1_2))

            # Return POSITIVE for values above greater-than-or-equal-to 0.5 in the output layer;
            # return NEGATIVE for other values
            if(layer_2[0] >= 0.5):
                return "POSITIVE"
            else:
                return "NEGATIVE"
```



Run the following cell to train your network with a small polarity cutoff.

```
[41] mlp = SentimentNetwork(reviews[:-1000],labels[:-1000],min_count=20,polarity_cutoff=0.05,learning_rate=0.01)
     mlp.train(reviews[:-1000],labels[:-1000])

     Progress:0.0% Speed(reviews/sec):0.0 #Correct:1 #Trained:1 Training Accuracy:100.%
     Progress:10.4% Speed(reviews/sec):1510. #Correct:1994 #Trained:2501 Training Accuracy:79.7%
     Progress:20.8% Speed(reviews/sec):1456. #Correct:4063 #Trained:5001 Training Accuracy:81.2%
     Progress:31.2% Speed(reviews/sec):1445. #Correct:6176 #Trained:7501 Training Accuracy:82.3%
     Progress:41.6% Speed(reviews/sec):1446. #Correct:8336 #Trained:10001 Training Accuracy:83.3%
     Progress:52.0% Speed(reviews/sec):1433. #Correct:10501 #Trained:12501 Training Accuracy:84.0%
     Progress:62.5% Speed(reviews/sec):1431. #Correct:12641 #Trained:15001 Training Accuracy:84.2%
     Progress:72.9% Speed(reviews/sec):1430. #Correct:14782 #Trained:17501 Training Accuracy:84.4%
     Progress:83.3% Speed(reviews/sec):1427. #Correct:16954 #Trained:20001 Training Accuracy:84.7%
     Progress:93.7% Speed(reviews/sec):1413. #Correct:19143 #Trained:22501 Training Accuracy:85.0%
     Progress:99.9% Speed(reviews/sec):1411. #Correct:20461 #Trained:24000 Training Accuracy:85.2%
```

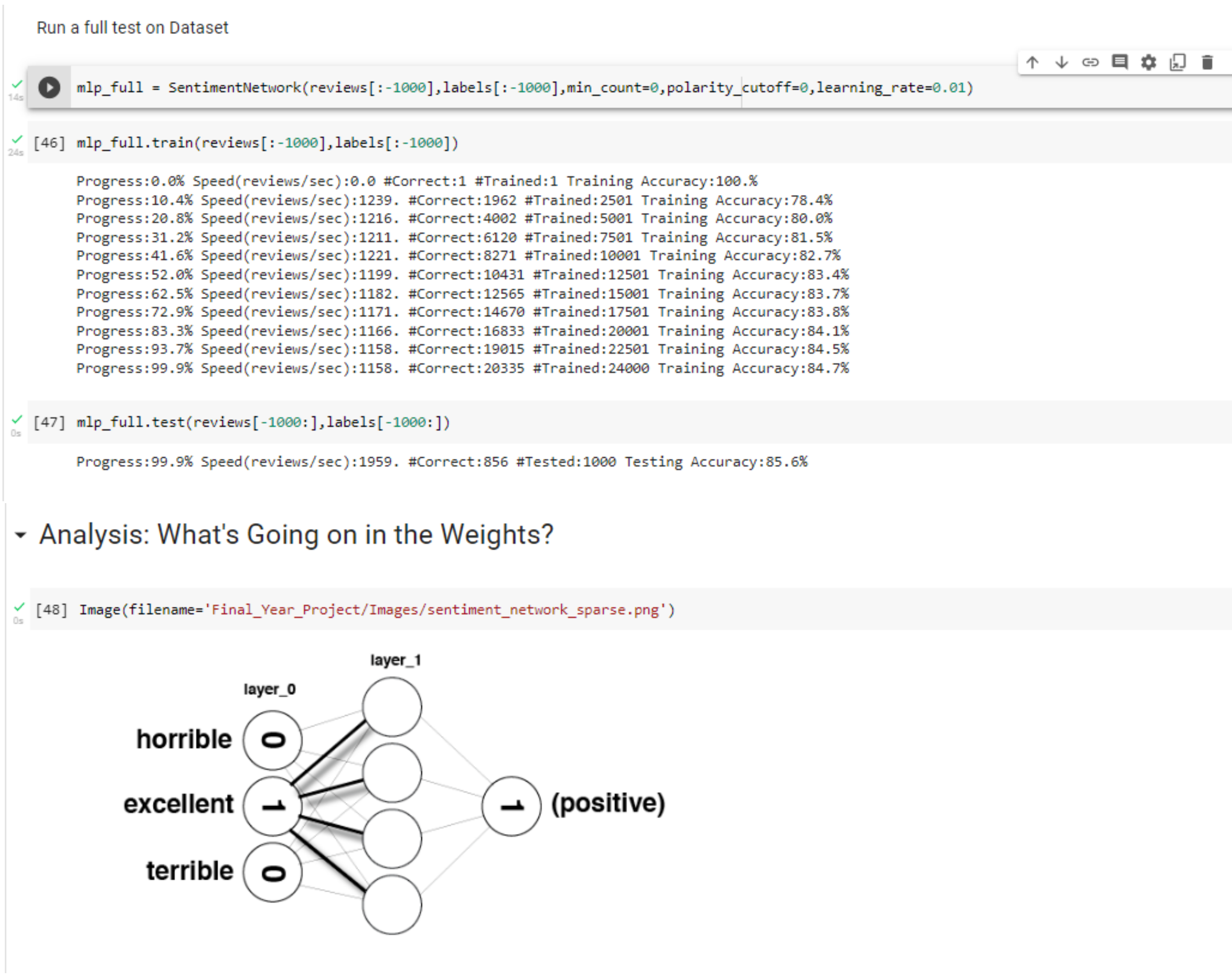


Run the following cell to train your network with a much larger polarity cutoff.

```
[43] mlp = SentimentNetwork(reviews[:-1000],labels[:-1000],min_count=20,polarity_cutoff=0.8,learning_rate=0.01)
     mlp.train(reviews[:-1000],labels[:-1000])

     Progress:0.0% Speed(reviews/sec):0.0 #Correct:1 #Trained:1 Training Accuracy:100.%
     Progress:10.4% Speed(reviews/sec):4743. #Correct:2114 #Trained:2501 Training Accuracy:84.5%
     Progress:20.8% Speed(reviews/sec):4556. #Correct:4235 #Trained:5001 Training Accuracy:84.6%
     Progress:31.2% Speed(reviews/sec):4479. #Correct:6362 #Trained:7501 Training Accuracy:84.8%
     Progress:41.6% Speed(reviews/sec):4350. #Correct:8513 #Trained:10001 Training Accuracy:85.1%
     Progress:52.0% Speed(reviews/sec):4247. #Correct:10641 #Trained:12501 Training Accuracy:85.1%
     Progress:62.5% Speed(reviews/sec):4237. #Correct:12796 #Trained:15001 Training Accuracy:85.3%
     Progress:72.9% Speed(reviews/sec):4222. #Correct:14911 #Trained:17501 Training Accuracy:85.2%
     Progress:83.3% Speed(reviews/sec):4128. #Correct:17077 #Trained:20001 Training Accuracy:85.3%
     Progress:93.7% Speed(reviews/sec):4084. #Correct:19258 #Trained:22501 Training Accuracy:85.5%
     Progress:99.9% Speed(reviews/sec):4055. #Correct:20552 #Trained:24000 Training Accuracy:85.6%
```

And run the following cell to test it's performance.

```
[44] mlp.test(reviews[-1000:],labels[-1000:])

     Progress:99.9% Speed(reviews/sec):2340. #Correct:822 #Tested:1000 Testing Accuracy:82.2%
```



Run a full test on Dataset

```
mlp_full = SentimentNetwork(reviews[:-1000],labels[:-1000],min_count=0,polarity_cutoff=0,learning_rate=0.01)
```

```
[46] mlp_full.train(reviews[:-1000],labels[:-1000])
```

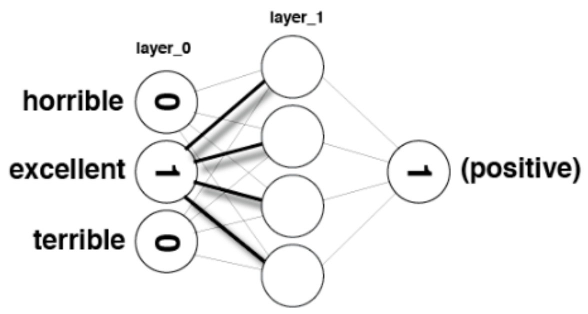
```
Progress:0.0% Speed(reviews/sec):0.0 #Correct:1 #Trained:1 Training Accuracy:100.%
Progress:10.4% Speed(reviews/sec):1239. #Correct:1962 #Trained:2501 Training Accuracy:78.4%
Progress:20.8% Speed(reviews/sec):1216. #Correct:4002 #Trained:5001 Training Accuracy:80.0%
Progress:31.2% Speed(reviews/sec):1211. #Correct:6120 #Trained:7501 Training Accuracy:81.5%
Progress:41.6% Speed(reviews/sec):1221. #Correct:8271 #Trained:10001 Training Accuracy:82.7%
Progress:52.0% Speed(reviews/sec):1199. #Correct:10431 #Trained:12501 Training Accuracy:83.4%
Progress:62.5% Speed(reviews/sec):1182. #Correct:12565 #Trained:15001 Training Accuracy:83.7%
Progress:72.9% Speed(reviews/sec):1171. #Correct:14670 #Trained:17501 Training Accuracy:83.8%
Progress:83.3% Speed(reviews/sec):1166. #Correct:16833 #Trained:20001 Training Accuracy:84.1%
Progress:93.7% Speed(reviews/sec):1158. #Correct:19015 #Trained:22501 Training Accuracy:84.5%
Progress:99.9% Speed(reviews/sec):1158. #Correct:20335 #Trained:24000 Training Accuracy:84.7%
```

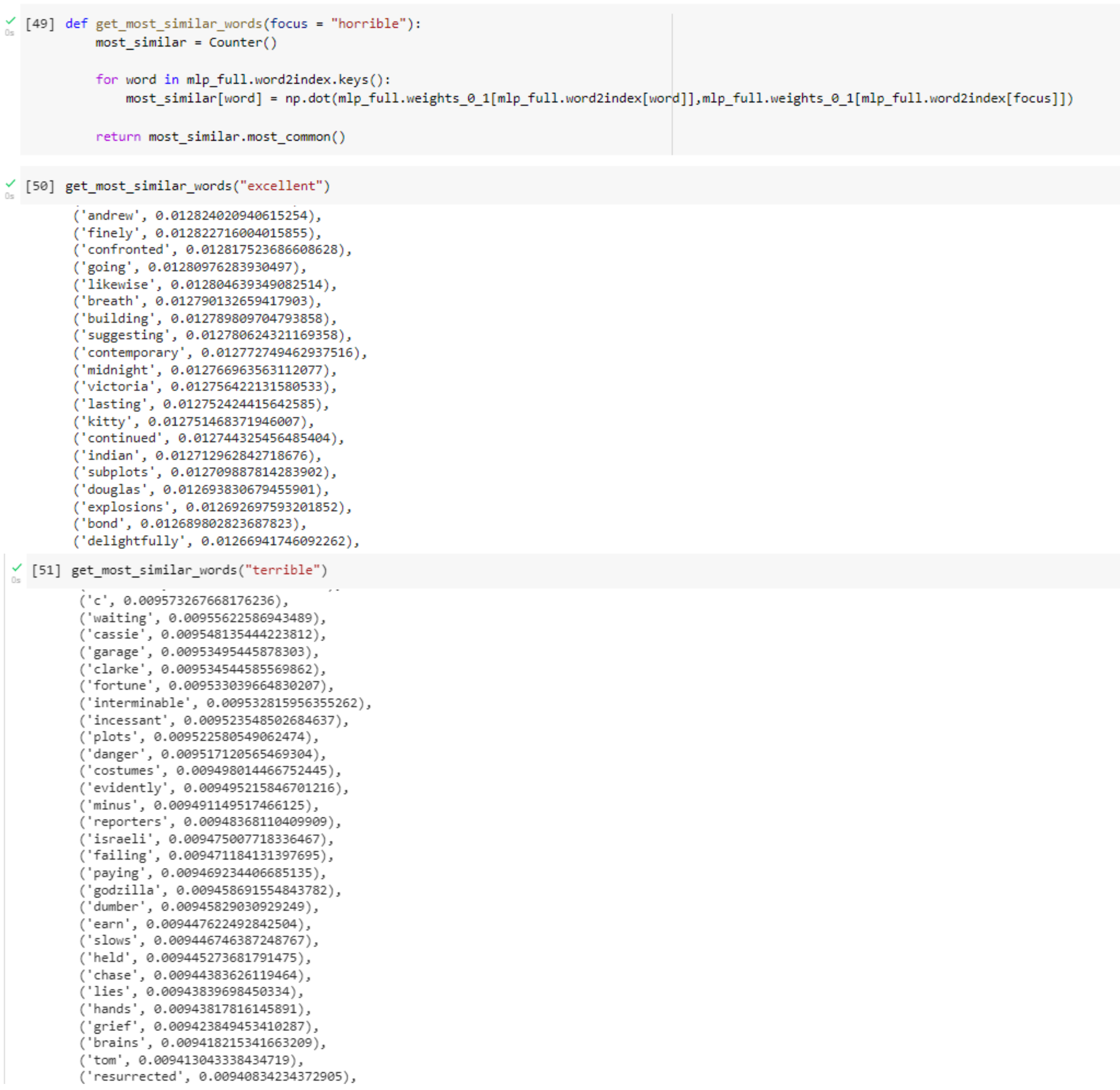
```
[47] mlp_full.test(reviews[-1000:],labels[-1000:])
```

```
Progress:99.9% Speed(reviews/sec):1959. #Correct:856 #Tested:1000 Testing Accuracy:85.6%
```

Analysis: What's Going on in the Weights?

```
[48] Image(filename='Final_Year_Project/Images/sentiment_network_sparse.png')
```





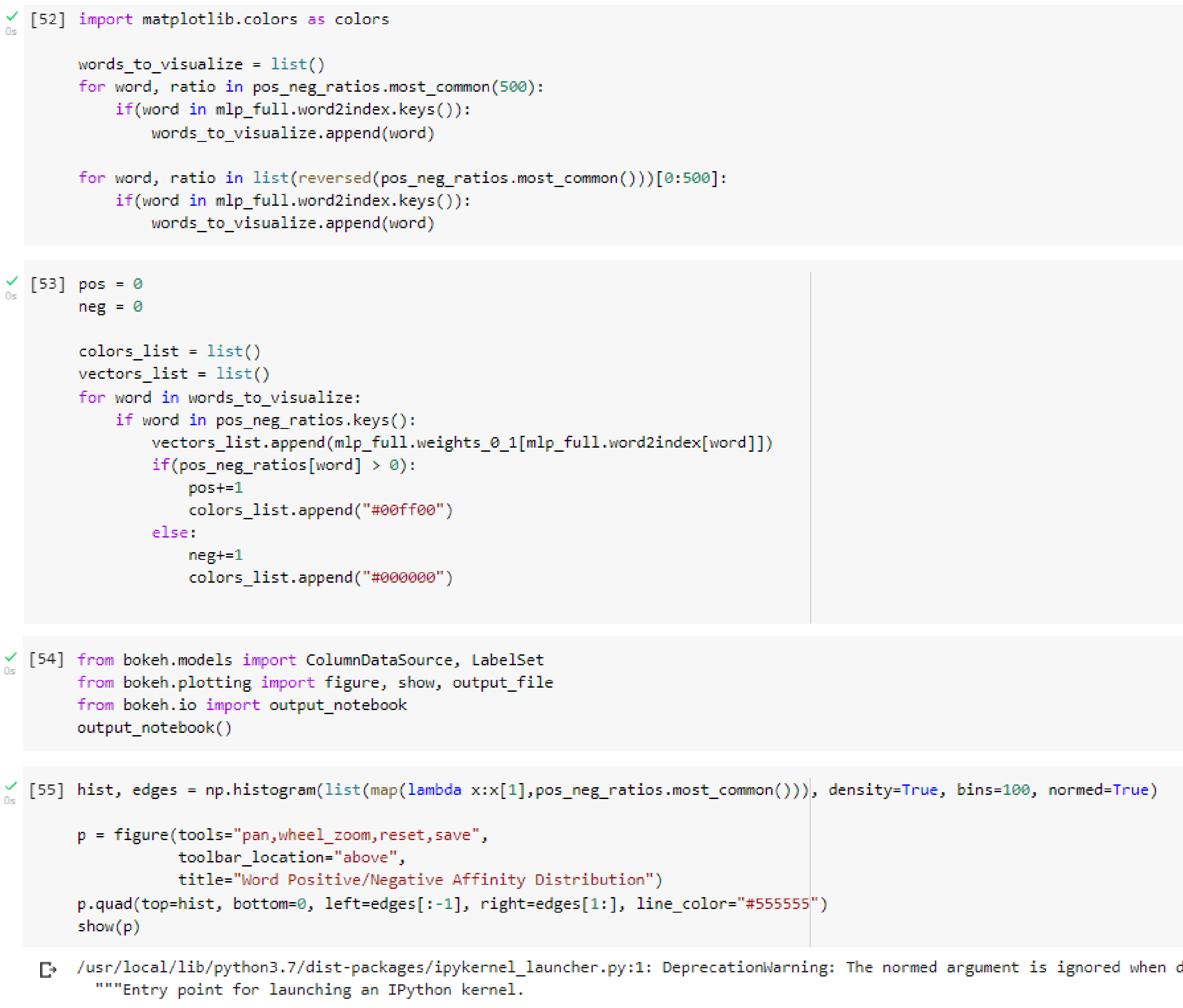
```
[49] def get_most_similar_words(focus = "horrible"):
         most_similar = Counter()

         for word in mlp_full.word2index.keys():
             most_similar[word] = np.dot(mlp_full.weights_0_1[mlp_full.word2index[word]],mlp_full.weights_0_1[mlp_full.word2index[focus]])

         return most_similar.most_common()
```

```
[50] get_most_similar_words("excellent")
```

```
     ('andrew', 0.012824020940615254),
     ('finely', 0.012822716004015855),
     ('confronted', 0.012817523686608628),
     ('going', 0.012809762839304997),
     ('likewise', 0.012804639349082514),
     ('breath', 0.012790132659417903),
     ('building', 0.012789809704793858),
     ('suggesting', 0.012780624321169358),
     ('contemporary', 0.012772749462937516),
     ('midnight', 0.012766963563112077),
     ('victoria', 0.012756422131580533),
     ('lasting', 0.012752424415642585),
     ('kitty', 0.012751468371946007),
     ('continued', 0.012744325456485404),
     ('indian', 0.012712962842718676),
     ('subplots', 0.012709887814283902),
     ('douglas', 0.012693830679455901),
     ('explosions', 0.012692697593201852),
     ('bond', 0.012689802823687823),
     ('delightfully', 0.012669417460922662),
```

```
[51] get_most_similar_words("terrible")
```

```
     ('c', 0.009573267668176236),
     ('waiting', 0.009556225869434489),
     ('cassie', 0.009548135444223812),
     ('garage', 0.009534954458783030),
     ('clarke', 0.009534544585569862),
     ('fortune', 0.009533039664830207),
     ('interminable', 0.009532815956355262),
     ('incessant', 0.009523548502684637),
     ('plots', 0.009522580549062474),
     ('danger', 0.009517120565469304),
     ('costumes', 0.009498014466752445),
     ('evidently', 0.009495215846701216),
     ('minus', 0.009491149517466125),
     ('reporters', 0.009483681104099909),
     ('israeli', 0.009475007718336467),
     ('failing', 0.009471184131397695),
     ('paying', 0.009469234406685135),
     ('godzilla', 0.009458691554843782),
     ('dumber', 0.009458290309292249),
     ('earn', 0.009447622492842504),
     ('slows', 0.009446746387248767),
     ('held', 0.009445273681791475),
     ('chase', 0.009443836261194464),
     ('lies', 0.009438396984503334),
     ('hands', 0.009438178161458891),
     ('grief', 0.009423849453410287),
     ('brains', 0.009418215341663209),
     ('tom', 0.009413043338434719),
     ('resurrected', 0.009408342343729005),
```



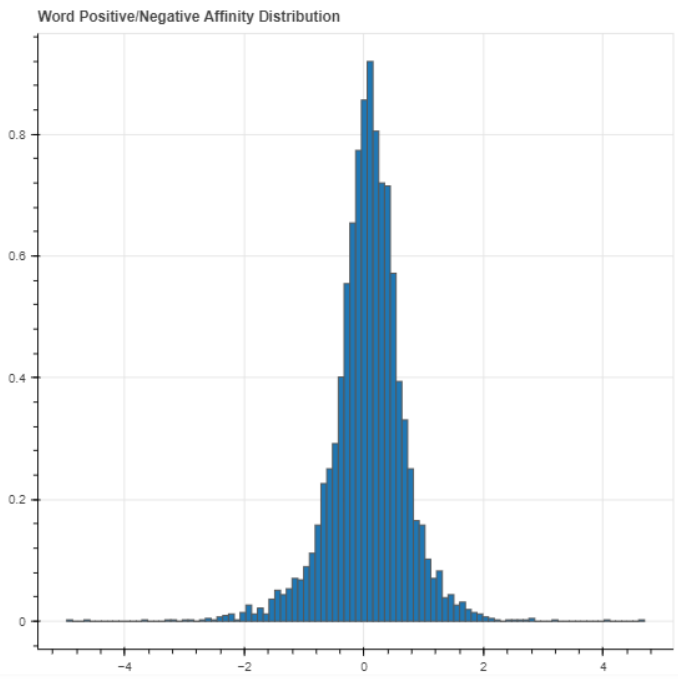
```
[52] import matplotlib.colors as colors

     words_to_visualize = list()
     for word, ratio in pos_neg_ratios.most_common(500):
         if(word in mlp_full.word2index.keys()):
             words_to_visualize.append(word)

     for word, ratio in list(reversed(pos_neg_ratios.most_common()))[0:500]:
         if(word in mlp_full.word2index.keys()):
             words_to_visualize.append(word)
```

```
[53] pos = 0
     neg = 0

     colors_list = list()
     vectors_list = list()
     for word in words_to_visualize:
         if word in pos_neg_ratios.keys():
             vectors_list.append(mlp_full.weights_0_1[mlp_full.word2index[word]])
             if(pos_neg_ratios[word] > 0):
                 pos+=1
                 colors_list.append("#00ff00")
             else:
                 neg+=1
                 colors_list.append("#000000")
```

```
[54] from bokeh.models import ColumnDataSource, LabelSet
     from bokeh.plotting import figure, show, output_file
     from bokeh.io import output_notebook
     output_notebook()
```

```
[55] hist, edges = np.histogram(list(map(lambda x:x[1],pos_neg_ratios.most_common())), density=True, bins=100, normed=True)

     p = figure(tools="pan,wheel_zoom,reset,save",
                toolbar_location="above",
                title="Word Positive/Negative Affinity Distribution")
     p.quad(top=hist, bottom=0, left=edges[:-1], right=edges[1:], line_color="#555555")
     show(p)
```

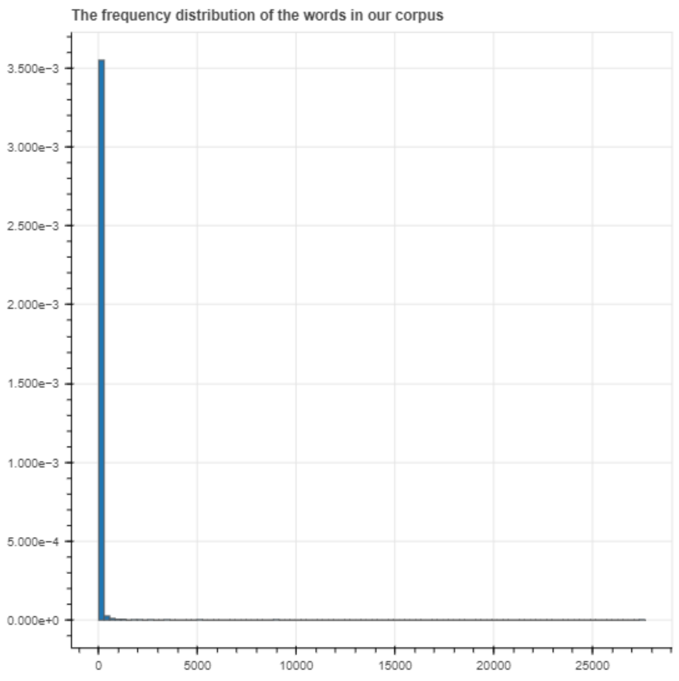
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: The normed argument is ignored when d
  """Entry point for launching an IPython kernel.
```

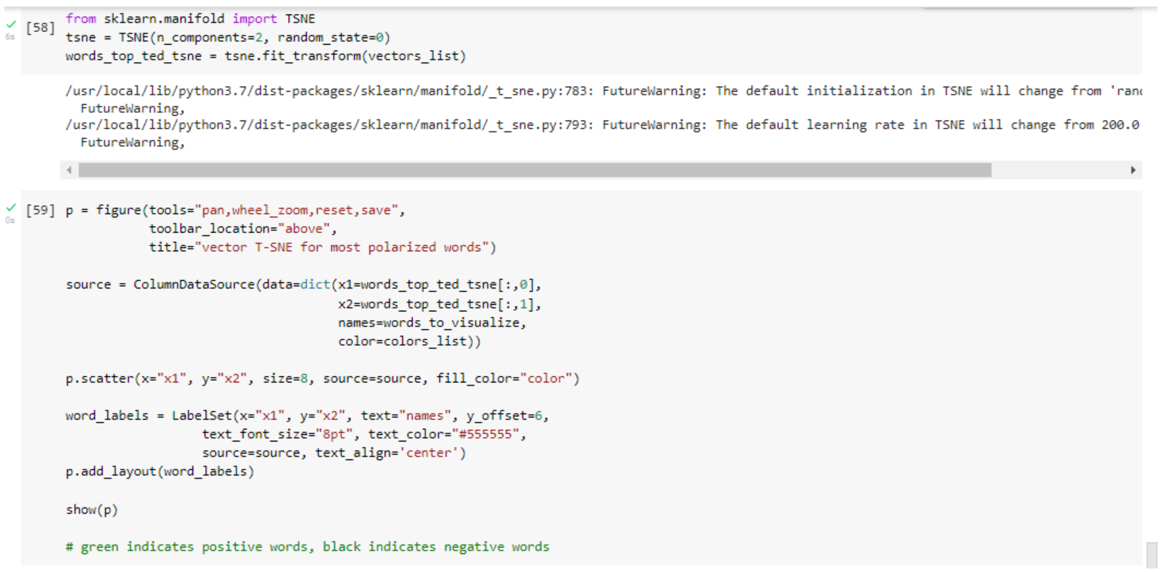


```
[56] frequency_frequency = Counter()

     for word, cnt in total_counts.most_common():
         frequency_frequency[cnt] += 1
```

```
[57] hist, edges = np.histogram(list(map(lambda x:x[1],frequency_frequency.most_common())), density=True, bins=100, normed=True)

     p = figure(tools="pan,wheel_zoom,reset,save",
                toolbar_location="above",
                title="The frequency distribution of the words in our corpus")
     p.quad(top=hist, bottom=0, left=edges[:-1], right=edges[1:], line_color="#555555")
     show(p)
```

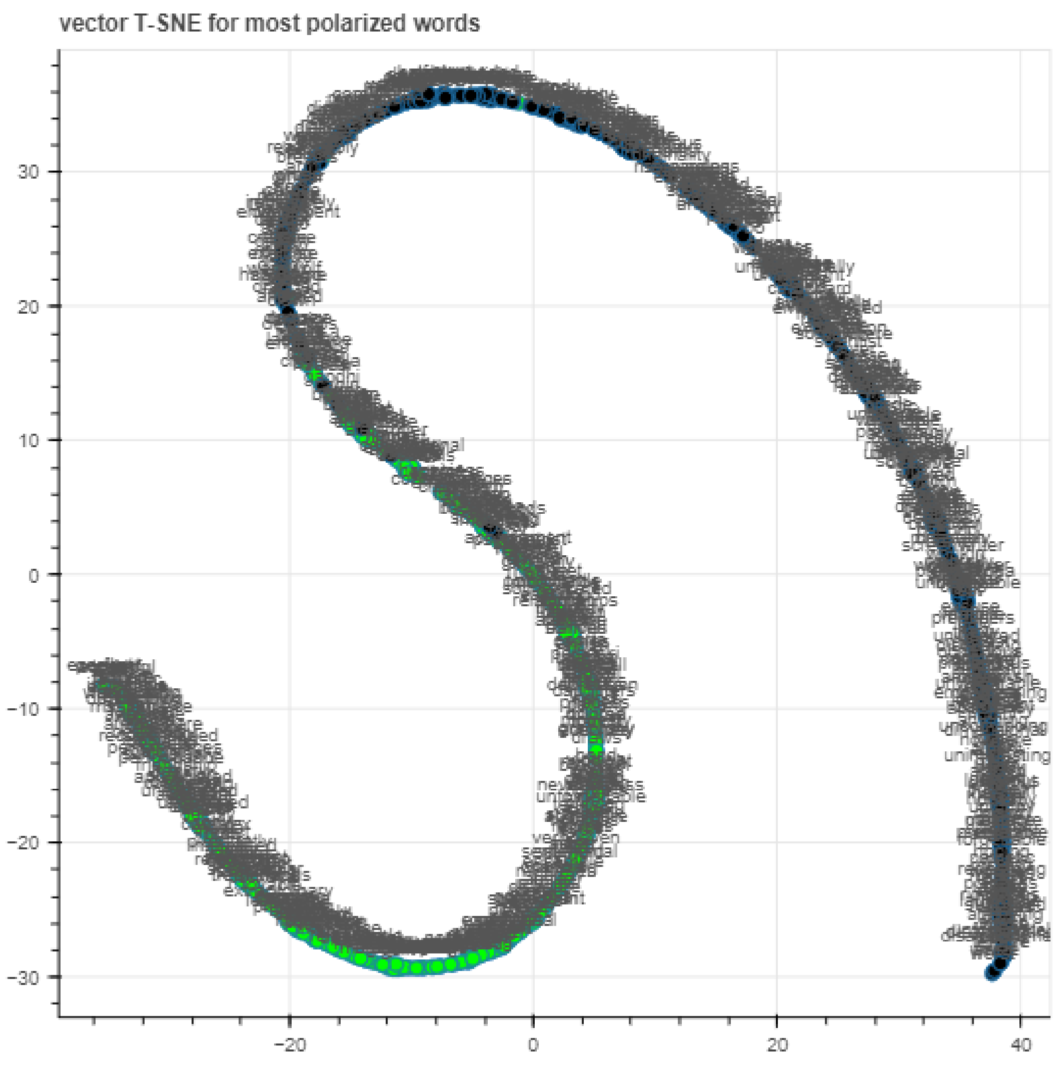
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: The normed argument is ignored when density is provided
  """Entry point for launching an IPython kernel.
```





```
[58] from sklearn.manifold import TSNE
     tsne = TSNE(n_components=2, random_state=0)
     words_top_ted_tsne = tsne.fit_transform(vectors_list)

     /usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning: The default initialization in TSNE will change from 'ran
       FutureWarning,
     /usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: The default learning rate in TSNE will change from 200.0
       FutureWarning,
```

```
[59] p = figure(tools="pan,wheel_zoom,reset,save",
                toolbar_location="above",
                title="vector T-SNE for most polarized words")

     source = ColumnDataSource(data=dict(x1=words_top_ted_tsne[:,0],
                                         x2=words_top_ted_tsne[:,1],
                                         names=words_to_visualize,
                                         color=colors_list))

     p.scatter(x="x1", y="x2", size=8, source=source, fill_color="color")

     word_labels = LabelSet(x="x1", y="x2", text="names", y_offset=6,
                       text_font_size="8pt", text_color="#555555",
                       source=source, text_align='center')
     p.add_layout(word_labels)

     show(p)

     # green indicates positive words, black indicates negative words
```



CHAPTER 6

CONCLUSION

In this report, I looked into how deep neural network architectures could be used for sentiment analysis.

They greatly outperformed the conventional methods, in my opinion. When Deep Neural Network models' embeddings were merged with Natural Language Processing, the best accuracy values were obtained. We intend to investigate the role of other neural network elements in the challenge in the future.

CHAPTER 7

FUTURE ENHANCEMENT

In order to improve this project further, we will try to implement Bert model instead of RNN and evaluate the outcomes. We'll be putting this on the cloud platform soon, as well as in the dashboards used by the analytics360 team. As soon as our system is up and running, we'll combine the reviewed data with the person who reviewed the dashboard data and build a dashboard for our team to use for further improvements. Our Dashboard will look like in the below image:

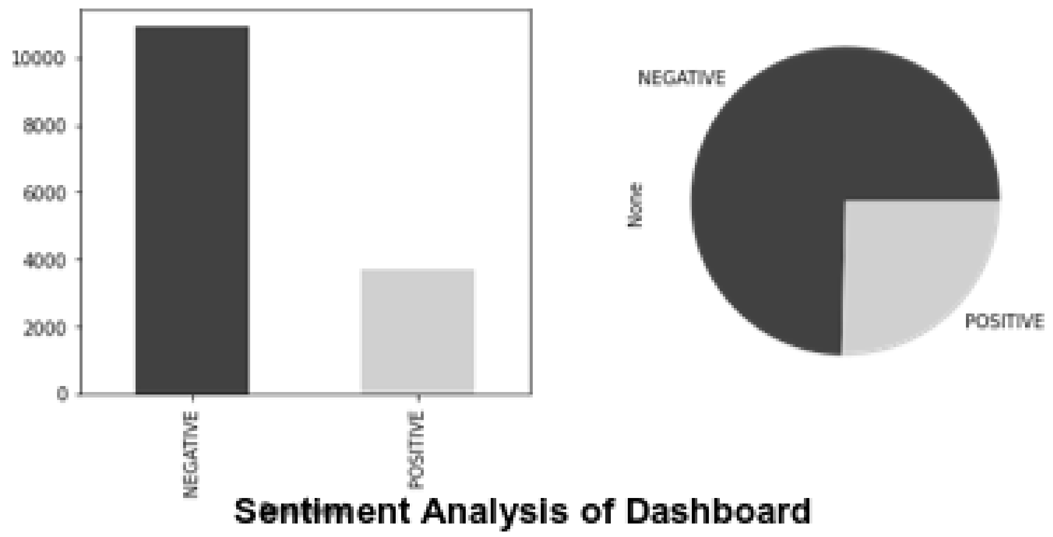


Figure 7.1: Example Dashboard Look

REFERENCES

1." Expressively vulgar: The socio-dynamics of vulgarity and its effects on sentiment analysis in social media" Department of Mathematics, Department of Linguistics, The University of Texas at Austin

2. "Multilingual Twitter Sentiment Classification: The Role of Human Annotators , Department of Knowledge Technologies,Jozef Stefan Institute, Ljubljana, Slovenia"

3. "Toward multi-label sentiment analysis: a transfer learning based approach, Journal of Big Data"

4. "IIIT-Hyderabad India , Microsoft India Deep Learning for Hate Speech Detection in Tweets"

5. "Aspect-based Sentiment Analysis using Dependency Parsing"

paper

ORIGINALITY REPORT

13%	5%	8%	11%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Neelam Chandolikar, Chaitanya Joshi, Prateek Roy, Abhijeet Gawas, Mini Vishwakarma. "Voice Recognition: A Comprehensive Survey", 2022 International Mobile and Embedded Technology Conference (MECON), 2022 Publication	2%
2	Submitted to Technological Institute of the Philippines Student Paper	1%
3	Submitted to De Montfort University Student Paper	1%
4	Submitted to The University of Wolverhampton Student Paper	1%
5	www.analyticssteps.com Internet Source	1%
6	Submitted to University of Asia Pacific Student Paper	1%
7	Submitted to The Robert Gordon University Student Paper	1%