Unsupervised Sentiment Analysis of Yelp Reviews Using Natural Language Processing

Ben Chamblee - Data Scientist - *Springboard School of Data*
https://github.com/Bench-amblee/springboard
June 15th, 2021

Abstract : This paper will review the development and application of Natural Language Processing techniques on text based reviews gathered from Yelp.com. This paper will review all steps of the data science method in depth and go through the performance of the statistical model developed using multi classified testing data and how it can be applied in a business enviornment.

Keywords: Natural Language Processing, Sentiment Analysis, Data Science, Machine Learning

1. INTRODUCTION

Quantifying a thought or an opinion has always been a challenge for modern applications. Most sites today allow users to give feedback on products or services using text and a five-star ranking system. As good as the star ranking system is, it's not perfect for quantifying how someone truly feels about something. A four-star ranking to some might say that that a product is amazing and to others, it might say that it's just above average. There's a reason that so many sites use this system - it's simple and easy to understand for people and programs alike. In order to get a more accurate version of the opinions of these reviewers, the full-text review should be read instead. For regular people though this is not an option. What if instead, we made the programs read the full text instead?

Sentiment analysis allows us to take advantage of text-based data. This unstructured data accounts for about 80% of all data on the internet.[1] It's easy for a human to understand what words on a screen are saying but not as easy for a program. On the contrary, it's very easy for a program to read through millions of lines of code in a short amount of time but impossible for humans to do the same. The main drawback of sentiment analysis is that even the most complex models are not 100% accurate.

Yelp.com gives its users the ability to review restaurants and other businesses using the five-star rating system along with an open-ended text review. For this analysis, the focus will be solely on text-based reviews.

2. DATA COLLECTION

All data was collected directly from Yelp. The goal was to get at least 10,000 reviews from a good variety of popular restaurants. The restaurants were chosen from the NYC Open Data directory of restaurants and then, using Yelp's Fusion API, were converted to valid URLs. In total, 11067 reviews from 220 unique restaurants were used to create the training dataset.

In order to get the data in a usable format, we needed to filter out any rouge HTML tags from the initial web scrape. The reviews were then stored as strings in a pandas data frame along with the restaurant name, location, and other identifying details. This data frame was then exported as a CSV file for further analysis.

3. EXPLORATORY DATA ANALYSIS

The goal of EDA is to explore the relationships between variables in a dataset. In this instance, we also want to apply basic natural language processing techniques to see how this data will be used for the prediction model. Because this data is primarily text-based, the first few steps were to remove stop words, tokenize each review, and apply stemming or lemmatization methods.

Stop words like 'the', 'to', or 'and' don't add much for sentiment analysis. We want to remove these for two main reasons: to save memory and processing power, and to make sure each review's most frequent words actually have a sentiment.

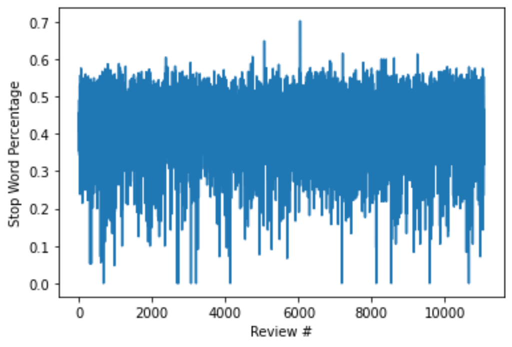
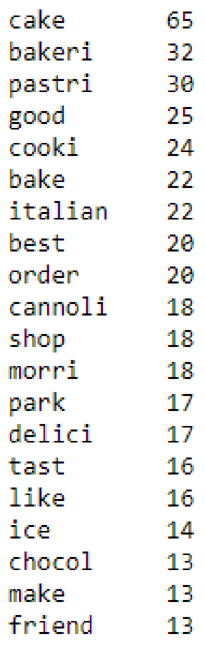


Fig 1. Percentage of Stop Words for Each Review

As you can see, most yelp reviews are about 40% stop words on average - removing them will save significant time and computing power.

Stemming is the process of reducing inflection in words to their root forms such as mapping a group of words to the same stem even if the stem itself is not a valid word.[2] This is another way to save time and computing power when working with large amounts of text data. There will be all kinds of different conjugations of the word 'quick' for example, to have them all in the same root form will make our dataset smaller and the frequency analysis more accurate. Now that the stop words have been removed and all words stemmed, the reviews should have only meaningful words. Below you can see an example of what the top 20 words look like after these first two steps



```
cake        65
bakeri      32
pastri      30
good        25
cooki       24
bake        22
italian     22
best        20
order       20
cannoli     18
shop        18
morri       18
park        17
delici      17
tast        16
like        16
ice         14
chocol      13
make        13
friend      13
```

Fig 2. Word Frequency for 56 Bake Shop Reviews

The next thing to figure out is how we quantify the meaning behind these words. For that, we need to convert words into vectors using the Word2Vec library. Similar to the frequency, the Word2Vec stores each unique word occurrence as a variable in a series. As you add multiple lines for multiple reviews, this becomes a matrix where the columns are each unique word and the rows are the total number of times that word appears. These word vectors can then be compared to each other and you can start to see the numerical similarity between words. To find out if a word or phrase is positive or not, we can calculate its Polarity value based on the TextBlob library's positivity index.

The polarity ranges from -1 to 1 and measures how similar a text vector is to known positive and negative text vectors. A polarity of 1 implies 100% positivity and a score of -1 implies negativity. The TextBlob function takes each word vector in the review and gives a polarity score, then assigns a score for the review as a whole. This gives us a way to classify each review and determine the sentiment without relying on the star rating system.

The last relationship that was looked into was the percentage of positive words in each review. If there's a noticeable increase in the percentage of positive words in a review that also has a high polarity and vice versa for negative reviews, it will make the model even more accurate. In order to confirm the percentage of positive words in each review, there needed to be a universal list of words to compare to. This can be done using the Positive Opinion Lexicon[3] which is a collected list of over 200 known English positive words that we can format the same as our stemmed reviews. Once the process of comparing each word to the words in the Lexion has been iterated for each review, you can divide the number of positive words by the number of total words to get a positivity percentage.

After confirming that more positive words correlated with a higher polarity score, the dataset was ready to move on to preprocessing for eventual model creation.

4. PRE-MODEL DATA PROCESSING

In order to create a model that can accurately assign each review to a sentiment class, the model needs to be able to process two different kinds of data:

- Text-based, like word matrices or Term Frequency Inverse Document Frequency vectors
- Number based like positive word percentage or classification indices

To prepare the dataset to be converted into a train/test split, we need to select the right features in order to not under or overfit the model. Currently, the sentiment distribution looks like this:

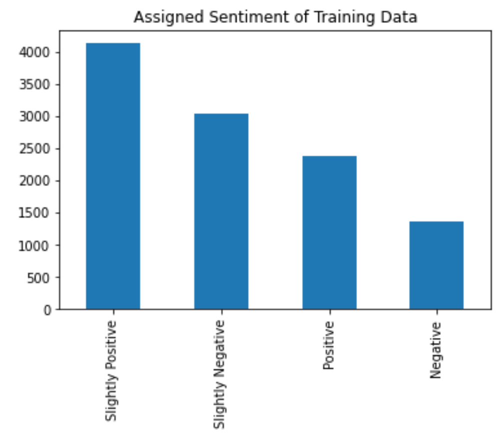


Fig 3. Review Classifications for Training Data

Let's take a look at the numerical features to see which ones would be a good fit for the model.

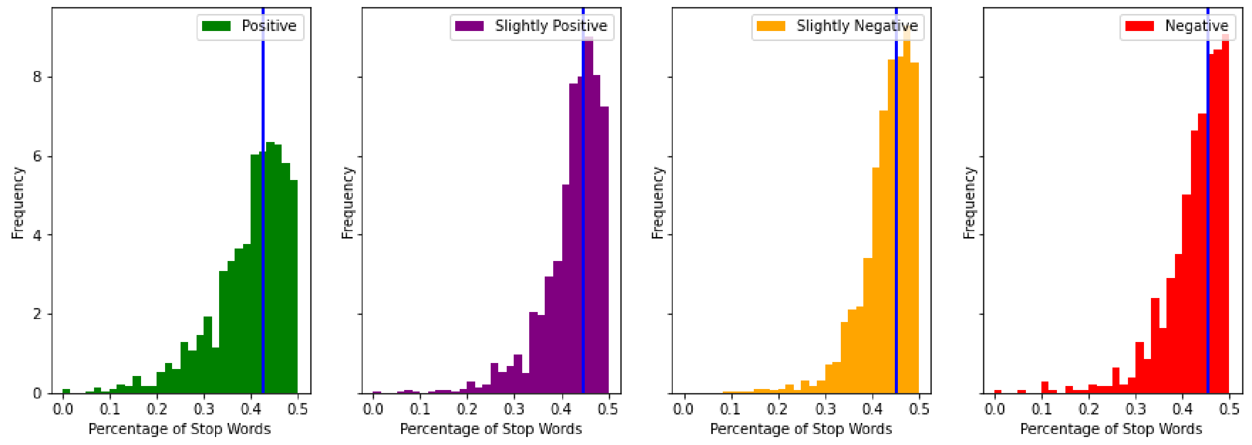


Fig 4. Percentage of Stop Words For Each Review Class

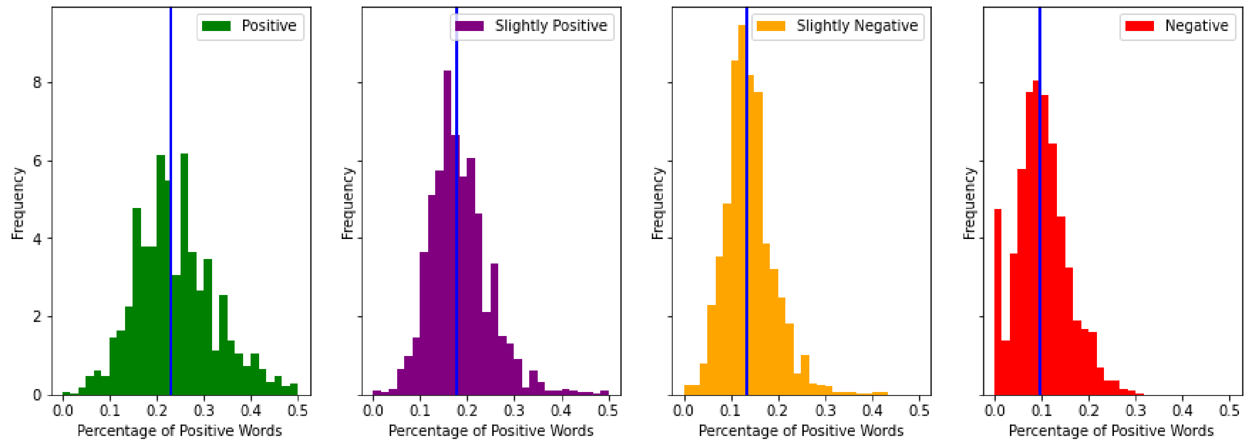


Fig 5. Percentage of Positive Words For Each Review Class

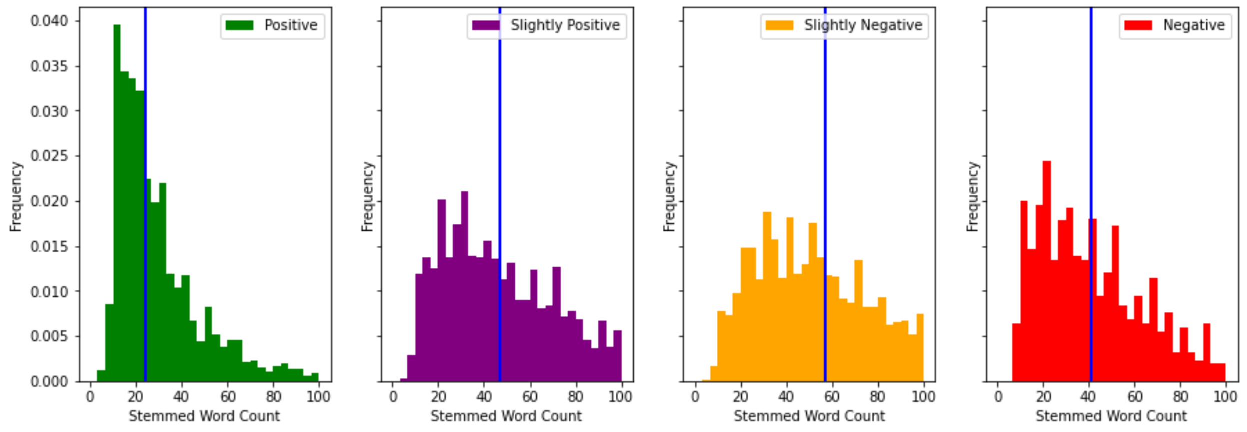


Fig 6. Stemmed Word Count For Each Review Class

Of the three numerical features observed, the percentage of positive words seemed to be the best one for training the model. The percentage of stop words was around 40% for each category and the stemmed word count was practically the same for three of the four groups. For positive word percentage though, each review had a distinct average that could be used to improve the model's predictions. Usually, there would be more numerical features, but for this classification, the simple solution seems to work better.

Now that numerical features are selected, the text-based features are the next step. In order to convert the text from each review into a number that the model can then interpret, the text needs to be converted to a matrix using either CountVectorizer or TFIDFVectorizer. To ensure the best accuracy, we'll be testing both to see which one performs the best. In addition to the two vectorizers, it is also important to determine the ideal number of ngrams (token or token pairs) and minimum/maximum document frequency for the TFIDFVectorizer.

To test each feature, a function was written to run tests on a simple logistic regression to see which set of features returned the highest accuracy score. The testing pipeline tested TFIDF vs. Countvectorizer first, then minimum document frequency, and finally, the number of ngrams.

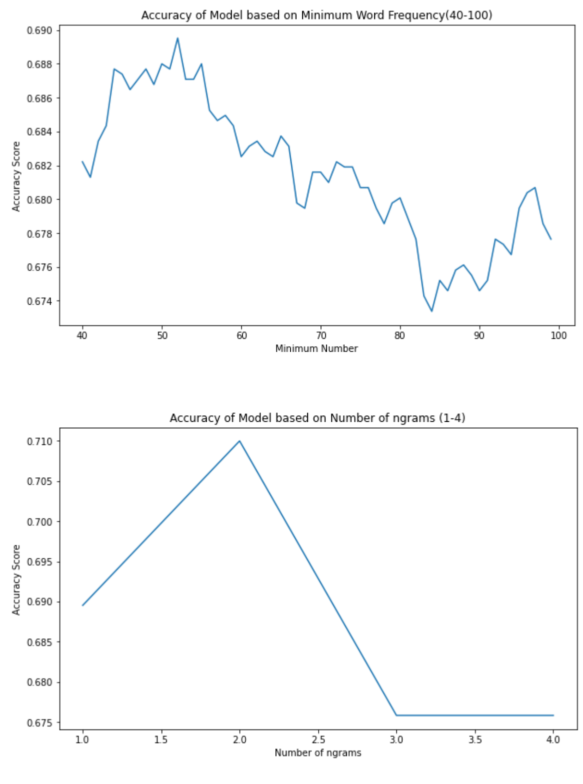


Fig 7. Text-Based Feature Selection Performance

In the end, the best result was a TFIDF Vectorizer with a minimum document frequency of 52 and 2 ngrams for each token. This means that the model will only take words or 2-word pairs that have appeared 52 or more times throughout all 10,000+ reviews. This helps with overfitting and saves a lot of time and processing power. With this in mind, we can export a processed dataset that has the percentage of positive words and a TFIDF vector for each review. We can finally move on to creating a model.

5. MODEL SELECTION

At this point in the analysis, the baseline logistic regression with ideal feature parameters has an accuracy of 71%. This is a good place to start, however, there is room for improvement using other modeling techniques.

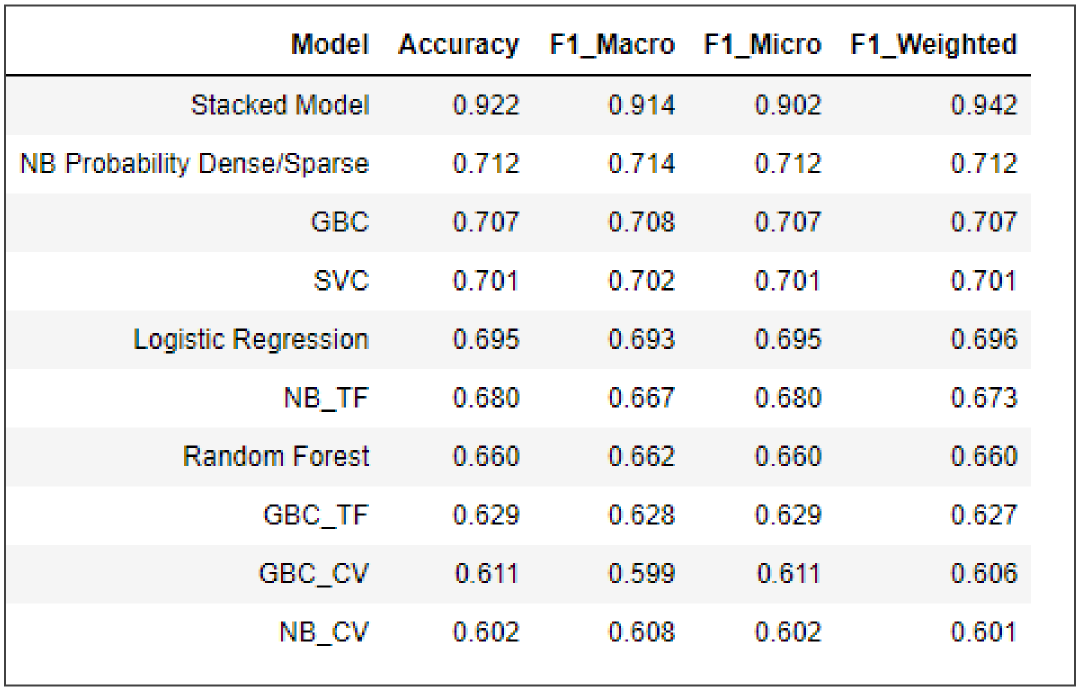
For this analysis, the model testing process will look like this:

- Select a model type
- Define a pipeline and use GridSearch to find the best parameters
- Save the model, accuracy, and f1 scores
- Compare the results with confusion matrices

In order to get a good variety of results, we tested 10 different types of models including Logistic Regression, Random Forest, Support Vector Classification, Gradient Boosted Classifier, two text-based Naive Bayes, two text-based Gradient Boosted Classifiers, one Naive Bayes with dense and sparse text matrices, and one stacked model. The stacked model was the most accurate, with an accuracy of 92%, this was achieved by combining the probability of the Naive Bayes model with the feature performance of the Gradient Boosted Classifier.

This process involved converting the full review text into a sparse matrix and then into a dense matrix. Matrices that contain mostly zero values are called sparse, distinct from matrices where most of the values are non-zero, called dense[4]. Using the dense matrix as an input and the classifications as an output, the Naive Bayes model outputs a probability value based on the content of each review. This number can then be added to the training dataset that was created in pre-processing to make an improved training dataset.

This new combined training dataset was then tested using a Gradient Boosted Classifier because that is the model that performed best using the numerical and text-based training data. This stacked model combines the best text-based model with the best numerical model to improve our baseline accuracy by over 20%.



Model	Accuracy	F1_Macro	F1_Micro	F1_Weighted
Stacked Model	0.922	0.914	0.902	0.942
NB Probability Dense/Sparse	0.712	0.714	0.712	0.712
GBC	0.707	0.708	0.707	0.707
SVC	0.701	0.702	0.701	0.701
Logistic Regression	0.695	0.693	0.695	0.696
NB_TF	0.680	0.667	0.680	0.673
Random Forest	0.660	0.662	0.660	0.660
GBC_TF	0.629	0.628	0.629	0.627
GBC_CV	0.611	0.599	0.611	0.606
NB_CV	0.602	0.608	0.602	0.601

Fig 8. Accuracy and F1 Scores for Each Model

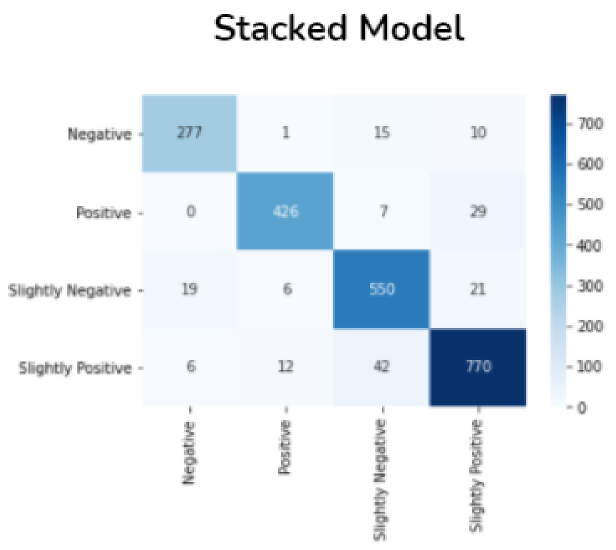
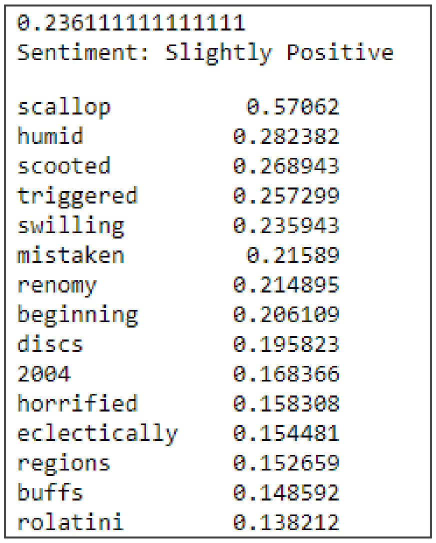


Fig 9. Final Confusion Matrix for Stacked Model

The confusion matrix shows the model's outputs for each of the four classes, and as you can see the distribution in Fig 9 is very similar to the distribution in Fig 3. The main drawback of the stacked model is that the data will have to be processed two times in order to first calculate the probability and then the classification result. For an accuracy boost of over 20% however, this is well worth it.

6. CONCLUSION

Now that this model has been trained, it is ready to be deployed and to predict new review data. The benefit of using a sentiment analysis model over a five-star rating system is that the model can output useful data in addition to the sentiment classification number. Take this output for example:



```
0.2361111111111111
Sentiment: Slightly Positive

scallop          0.57062
humid           0.282382
scooted         0.268943
triggered       0.257299
swilling        0.235943
mistaken         0.21589
renomy          0.214895
beginning       0.206109
discs           0.195823
2004            0.168366
horrified       0.158308
eclectically    0.154481
regions         0.152659
buffs           0.148592
rolatini        0.138212
```

Fig 10. Model Output from Italian Restaurant Review

Here we have the sentiment score, the classification, and the 15 words that had the largest impact on the score. You can obtain meaningful info from this result as opposed to just knowing the classification. In this case, the customer must have liked the scallops to cause such a high score, but you also notice words like ‘triggered’ and ‘horrified’ that may be cause for concern. The business owner might be inclined to look into these words and see how they can improve the positivity score by fixing those aspects.

The model’s accuracy will only improve over time as more data is tested. This has major business implications and could be very useful for any service that receives mostly text-based reviews. There is also room for growth when it comes to classification. What are the classification groups? What is being classified? These can all change to answer different questions using similar NLP methods. The potential for natural language processing is incredibly high, this analysis shows just one of the many applications of utilizing text-based data.

References

[1] B. Bitext, *The Bulk of Data Wandering on the Net Is Unstructured Data*, 25-Nov-2018. [Online]. Available: https://blog.bitext.com/the-bulk-of-data-wandering-on-the-net-is-unsructured-data#:~:text=Did%20you%20know%20that%20up,not%20asily%20understandable%20for%20computers.

[2] H. Jabreen, “Stemming and Lemmatization in Python,” *DataCamp Community*, 23-Oct-2018. [Online]. Available: https://www.datacamp.com/community/tutorials/stemming-lemmatiztion-python.

[3] Bing Liu, Minqing Hu and Junsheng Cheng. "Opinion Observer: Analyzing and Comparing Opinions on the Web." Proceedings of th 14th International World Wide Web conference (WWW-2005), May 10-14, 2005, Chiba, Japan.

[4] J. Brownlee, “A Gentle Introduction to Sparse Matrices for Machine Learning,” *Machine Learning Mastery*, 09-Aug-2019. [Online]. Available:https://machinelearningmastery.com/sparse-matrices-for-mchine-learning/#:~:text=Matrices%20that%20contain%20mostly%2zero,non%2Dzero%2C%20called%20dense.&text=That%20sparse 20matrices%20contain%20mostly,are%20distinct%20from%20den %20matrices.

[5] Haghighi, S., Jasemi, M., Hessabi, S. and Zolanvari, A. (2018). PyCM:Multiclass confusion matrix library in Python. Journal of Open Source Software, 3(25), p.729.

[6] H. Lane, C. Howard, and H. M. Hapke, *Natural language processing in action understanding, analyzing, and generating text with Python*. Shelter Island, NY: Manning, 2019.