Investigating the Effectiveness of Machine Learning Techniques over Rule Based Systems in Sentiment Analysis

Z.X (maze508)

1.0 Introduction

1.1 Purpose

Sentiment Analysis can be understood as a knowledge-based classification problem where either a model, or a rule based system attempts to interpret the overall sentiment of a block of text, whether it is positive, negative or neutral. The most common application of this would be in businesses. Companies turn to Sentiment Analysis to try to track overall consumer confidence in a product or industry, allowing for them to adjust their business model in order to better plan for the future.[1]

The popularity of Sentiment Analysis stretches beyond its business applications. This type of analysis is also widely seen in the Finance Industry.[2] It allows one to gauge the general sentiment of a particular sector, industry or financial instrument and thus make trading decisions accordingly. However, for retail traders, the idea of Sentiment Analysis generally focuses it on a result of a Technical Analysis. For example, a large increase in trading volume as price moves higher may be attributed to an increase in investor sentiment.

Sentiment Analysis is generally approached in two major ways. A rule based approach or a machine learning approach. However, because of its higher barrier of entry, many generally flood towards a rule based system to perform these kinds of analysis. However with the advent of machine learning algorithms, Sentiment Analysis through the machine learning approach has become more popular. This leads one to question which of the two techniques to employ when one wants to design a Sentiment Analysis system.

---

[1] Mäntylä, M. V., Graziotin, D., & Kuutila, M. (2018). The evolution of sentiment analysis—A review of research topics, venues, and top cited papers. *Computer Science Review*, *27*, 16–32. https://doi.org/10.1016/j.cosrev.2017.10.002

[2] Sohangir, S., Wang, D., Pomeranets, A., & Khoshgoftaar, T. M. (2018). Big Data: Deep Learning for financial sentiment analysis. *Journal of Big Data*, *5*(1), 1–3. https://doi.org/10.1186/s40537-017-0111-6

1.2 Research Question

As aforementioned, in order to find the optimal way to perform Sentiment Analysis, this paper aims to investigate :

The Effectiveness of Machine Learning Based Systems over Rule Based Systems in Sentiment Analysis

2.0 Methodology and Framework

2.1 Machine Learning Based Sentiment Analysis

The machine learning based systems in this paper will focus on pre-trained models in **scikit-learn**[3], a machine learning library for Python. As one of the go to machine learning libraries, scikit-learn remains one of the most popular choices for machine learning related projects alongside **keras**[4]. Scikit-learn allows users to easily create a machine learning pipeline [5]based on their pre-built models by simply initialising fit and transform methods and a classifier. This allows for quick access to machine learning techniques, at the same time providing users with the ability to make small changes to the specifics of the models used by using the fit and transform methods.

In order to pick a suitable model for our Sentiment Analysis, different vectorizers and classifiers were tested.

Vectorizers[6] are methods for converting textual data into vectors which is the preferred form of data by models. Sklearn provides 2 pre-defined Vectorizers, the CountVectorizer and the TFIDFVectorizer[7].

---

[3] Scikit-learn. (n.d.). Retrieved February 20, 2021, from https://scikit-learn.org/stable/

[4] Team, K. (n.d.). Keras. Retrieved February 20, 2021, from https://keras.io/

[5] Sklearn.pipeline.pipeline¶. (n.d.). Retrieved February 20, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

[6] Nlp text pre-processing: Text vectorization. (2020, June 15). Retrieved February 20, 2021, from https://www.einfochips.com/blog/nlp-text-vectorization/

[7] 6.2. feature extraction¶. (n.d.). Retrieved February 20, 2021, from https://scikit-learn.org/stable/modules/feature_extraction.html

Initialising the CountVectorizer means the model only counts the number of times a word appears in a document. Thus, it is susceptible to being easily misguided by skewed data, resulting in a heavy bias to more frequently occurring words in the data. The TFIDFVectorizer on the other hand considers the overall document weightage of a word. This particular Vectorizer helps us deal with the flaw in the CountVectorizer by penalising the more frequent word occurrences.

Classifiers[8] are the predictors of the models. They utilise training data to understand how given input variables relate to the class and in the case where a classifier is trained accurately, it is able to predict future input values accurately. Different classifiers[9] are preferred when dealing with different classifications. For binary classification, we only wish to group an outcome into one of two groups while for multi-class-classification, we want to group an outcome into one of multiple groups (more than two). In our case, we desire to train a model that is able to determine whether a block of text is positive, negative or neutral. Thus, we want to use classifiers that specifically deal with multi-class-classification.

Having taken both Vectorisation and Classifiers into account, a simple test to determine the preferred model is performed. The CountVectorizer and TFIDFVectorizer are both tested with eight similar classifiers each : LinearSVC, SGDClassifier, LogisticRegression. MultinomialNB, DecisionTreeClassifier, RandomForestClassifier, AdaBoostClassifier, KNeighborsClassifier. The results are as shown :

[8] Shaikh, J. (2017, October 30). Machine learning, nlp: Text classification using SCIKIT-LEARN, Python and NLTK. Retrieved February 20, 2021, from https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a

[9] Classifier COMPARISON¶. (n.d.). Retrieved February 17, 2021, from https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html (Date Accessed : 17/02/2021)

CountVectorizer	TFIDFVectorizer
LinearSVC : Test Dataset Accuracy: 0.7810792349726776 Training Dataset Accuracy: 0.9778005464480874 SUM of Accuracies : 1.7589 SGDClassifier : Test Dataset Accuracy: 0.787568306010929 Training Dataset Accuracy: 0.95525956284153 SUM of Accuracies : 1.7428 LogisticRegression : Test Dataset Accuracy: 0.8012295081967213 Training Dataset Accuracy: 0.9456113387978142 SUM of Accuracies : 1.7468 MultinomialNB : Test Dataset Accuracy: 0.7793715846994536 Training Dataset Accuracy: 0.8528005464480874 SUM of Accuracies : 1.6322 DecisionTreeClassifier : Test Dataset Accuracy: 0.7021857923497268 Training Dataset Accuracy: 0.9959016393442623 SUM of Accuracies : 1.6981 RandomForestClassifier : Test Dataset Accuracy: 0.7609289617486339 Training Dataset Accuracy: 0.9959016393442623 SUM of Accuracies : 1.7568 AdaBoostClassifier : Test Dataset Accuracy: 0.7260928961748634 Training Dataset Accuracy: 0.706113387978142 SUM of Accuracies : 1.4322 KNeighborsClassifier : Test Dataset Accuracy: 0.4849726775956284 Training Dataset Accuracy: 0.6362704918032787 SUM of Accuracies : 1.1212	LinearSVC : Test Dataset Accuracy: 0.8049863387978142 Training Dataset Accuracy: 0.8868681693989071 SUM of Accuracies : 1.6919 SGDClassifier : Test Dataset Accuracy: 0.8131830601092896 Training Dataset Accuracy: 0.8613387978142076 SUM of Accuracies : 1.6745 LogisticRegression : Test Dataset Accuracy: 0.8077185792349727 Training Dataset Accuracy: 0.8557889344262295 SUM of Accuracies : 1.6635 MultinomialNB : Test Dataset Accuracy: 0.7657103825136612 Training Dataset Accuracy: 0.7755293715846995 SUM of Accuracies : 1.5412 DecisionTreeClassifier : Test Dataset Accuracy: 0.6752049180327869 Training Dataset Accuracy: 0.995816256830601 SUM of Accuracies : 1.671 RandomForestClassifier : Test Dataset Accuracy: 0.7704918032786885 Training Dataset Accuracy: 0.995816256830601 SUM of Accuracies : 1.7663 AdaBoostClassifier : Test Dataset Accuracy: 0.7551229508196722 Training Dataset Accuracy: 0.7446209016393442 SUM of Accuracies : 1.4997 KNeighborsClassifier : Test Dataset Accuracy: 0.7131147540983607 Training Dataset Accuracy: 0.8036202185792349 SUM of Accuracies : 1.5167

When picking a desired combination of Vectorizer and Classifiers, there are some things to take note of. As previously explained, the TFIDFVectorizer is preferred over the CountVectorizer.

Regarding the Classifiers, an extremely high accuracy in the training dataset but mediocre performance in the test dataset may prove worrisome. This is likely attributed to overfitting[10], when a model learns the detail and noise in the training data to the extent that it impairs the performance of the model on new data (test dataset).

Even with these considerations, there are a number of combinations that still fit our criteria : LinearSVC, SGDClassifier, LogisticRegression and RandomForestClassifier. Thus, without getting into the complications of each model, we have decided to use LinearSVC as our Classifier and the TFIDFVectorizer as our Vectorizer.

2.2 Rule Based Sentiment Analysis

For the rule based system, we will employ **vaderSentiment**[11]. VADER (Valence Aware Dictionary and Sentiment Reasoner) is a lexicon rule-based sentiment analysis tool specifically attuned to sentiments expressed in social media. VADER's sentiment score attached to each value is based on the average score of 10 separate people, rating the sentiment of a particular word or emoji from -4 to 4. This value is then averaged and normalized resulting in a final compound score of between -1 and 1. An example of VADER as Sentiment Analysis can be seen as follows :

[10] Brownlee, J. (2019, August 06). How to avoid overfitting in deep learning neural networks. Retrieved February 20, 2021, from https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/

[11] Cjhutto. (n.d.). Cjhutto/vadersentiment. Retrieved February 20, 2021, from https://github.com/cjhutto/vaderSentiment

```python
sentiment_test.py > ...
from vaderSentiment.vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import pandas as pd
import numpy as np

analyzer = SentimentIntensityAnalyzer()

sentence = "The food there is quite literally the worst I've ever tasted"

#! Checking Sentiment Score
def sentiment_analyser(sentence):
    score = analyzer.polarity_scores(sentence)
    print(f"{sentence} : {str(score)}")
    return score

score = sentiment_analyser(sentence)

print(score)
print(score['compound'])
```

```
The food there is quite literally the worst I've ever tasted : {'neg': 0.304, 'neu': 0.696, 'pos': 0.0, 'compound': -0.6557}
{'neg': 0.304, 'neu': 0.696, 'pos': 0.0, 'compound': -0.6557}
-0.6557
```

Vader also has a predefined threshold value to group texts into positive, negative and neutral sentiments.

> 1. **positive sentiment:** `compound` score >= 0.05
> 2. **neutral sentiment:** (`compound` score > -0.05) and (`compound` score < 0.05)
> 3. **negative sentiment:** `compound` score <= -0.05

Thus, according to VADER, the sentence : " *The food there is quite literally the worst I've ever tasted*" has a strong negative sentiment attached to it.

2.3 Data Engineering and Experimental Procedures

In order to determine the better alternative, both systems are tested on the same portion of the 2015 US Airline Tweets in the month of February. The sentiment allocation of the data is as shown :

```
Total Count : 14640
Positive Count : 2363 / 14640 (16.1%)
Negative Count : 9178 / 14640 (62.7%)
Neutral Count : 3099 / 14640 (21.2%)
```

Data is first arranged into a pandas dataframe and numerical values are attached to each different sentiment : Positive → 1, Negative → 0, Neutral → -1. The textual representations (e.g positive, negative

and neutral) are then removed alongside with other inessential information, leaving us with only two columns :

1. The Text
2. The Numerical Representation of Sentiments

Before passing the data straight into the machine learning pipeline/model, some data engineering processes are required. A tokenizer is first initialised and later passed as an argument into the model pipeline process. The tokenizer does a few things :

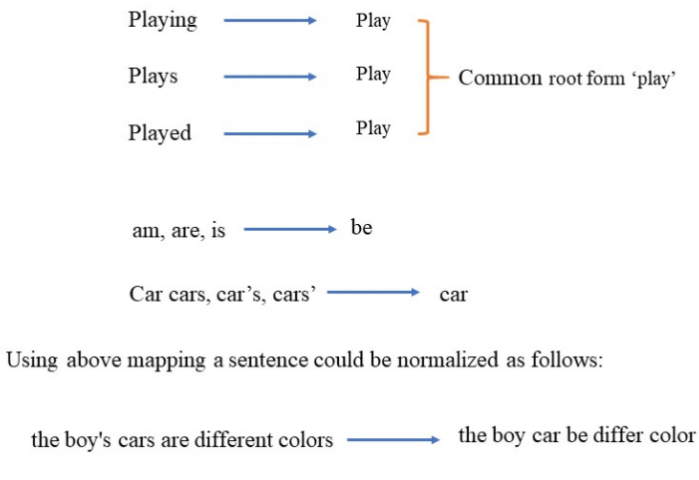
1. Defines a parser of a user-defined language
2. Remove stop words and punctuations
3. Performs Lemmatization [12]on the remaining words

The parser allows one to define the language of the data in which one desires to train the model on. In this case, it would just be English, but spacy provides support to other popular languages for example, Chinese, German and French.

Stop words and punctuations are removed as we do not want the model taking into (most of the time) redundant words/texts and building links around them. There are some, albeit rare, occasions where the presence of stop words may cause a difference in the sentiment conveyed of a sentence. These cases are however **NOT** taken into account in this experiment.

Lemmatization is a text normalization technique that resolves words to their canonical form.

[12] Heidenreich, H. (2018, December 21). Stemming? Lemmatization? WHAT? Retrieved February 20, 2021, from https://towardsdatascience.com/stemming-lemmatization-what-ba782b7c0bd8



[13]

The Process of Lemmatization allows one to standardise the words that will pass through the model, allowing the model to learn quickly and often, more accurately.

Since sklearn is a high level library, much of the data engineering portion can be compressed and this process can easily be done in a few lines :

```
def my_tokenizer(sentence):
    mytokens = parser(sentence)
    mytokens = [ word for word in mytokens if word not in stopwords and word not in punctuations ]
    mytokens = [ word.lemma_.lower().strip() if word.lemma_ != "-PRON-" else word.lower_ for word in mytokens ]
    return mytokens
```

NOTE : stopwords and punctuations are pre defined lists containing the respective elements

After these simple Data Engineering processes, the data is then split into a 80 : 20 ratio of test and training dataset respectively through sklearn's predefined functions `train_test_split`. Then, the model can be initialised with the TFIDFVectorizer and LinearSVC Classifier as suggested before. The training dataset is then passed into the model for training before obtaining the results of the model's performance on the test and training dataset.

The same set of test data is then passed through the rule based system to obtain an accuracy value there as well.

This process is then repeated ten times for ten different random states.

---

[13] Stemming and lemmatization in python. (n.d.). Retrieved February 20, 2021, from https://www.datacamp.com/community/tutorials/stemming-lemmatization-python

3.0 Analysis and Evaluation

3.1 Raw Data and Analysis

	RS* 1	RS 2	RS 3	RS 4	RS 5	RS 6	RS 7	RS 8	RS 9	RS 10	Avg
Model Test Accuracy (%)	**80**	**80**	**79**	**79**	**80**	**80**	**80**	**82**	**81**	**79**	**80**
Model Training Accuracy (%)	**89**	**89**	**89**	**89**	**89**	**89**	**89**	**89**	**89**	**89**	**89**
Rule Based Test Accuracy (%)	**21**	**22**	**22**	**21**	**21**	**21**	**21**	**20**	**22**	**23**	**21**
Rule Based Training Accuracy (%)	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**

Random State (RS)

Table Values are rounded to nearest integer

It is evident that the Machine Learning based system performed significantly better than the Rule Based system in Sentiment Analysis in our experiment, with a Model Test Accuracy of 80%, almost four times of the 21% accuracy achieved in the Rule Based System. The results are consistent in each Random State, suggesting that there have not been any issues regarding the model training.

3.2 Evaluation

To validate the experimental results, it is compared to literature studies that have been done in this area. Devika M. D[14] did a brief comparison of different Sentiment Analysis approaches. In his paper, he states that the machine learning approach yielded the best results in terms of performance, efficiency and accuracy, in line with our experimental conclusions. However, the rule based approach he took was able to achieve 86% performance accuracy at a sentence level, significantly more accurate as compared to our experiment.

This could be attributed to many factors, one of which would be Data Labelling to Rule Based System consistency. Since this experiment employed the use of open source data, the process of tagging the Sentiment of the data will be inconsistent to how each word/text is scored in the Rule Based System, resulting in a large deviation in the Sentiment scores. This is evidence of the superiority of the Machine Learning Based Approach over the Rule Based Approach as it is able to adapt to the data in a way where any one Rule Based System cannot.

It is also to note that the Machine Learning Based Approach in this paper is not perfectly optimised, thus the unreasonably low accuracy as compared to the other study discussed above. During the Lemmatization process, parts of speech[15] of each text/sentence were not defined which may have caused a slight drop in accuracy although it was a huge time saver. There are many more pre-processing techniques that could have been performed to raise the accuracy of the Machine Learning Based Approach but for simplicity's sake, these processes have been omitted.

3.3 Conclusion

In answering our research question. The essay has proven the effectiveness of Machine Learning Based Approaches of Sentiment Analysis to be more accurate than Rule Based Approaches.

---

[14] Devika, M., Sunitha, C., & Ganesh, A. (2016). Sentiment analysis: A comparative study on different approaches. *Procedia Computer Science, 87*, 44-49. doi:10.1016/j.procs.2016.05.124

[15] Stemming and lemmatization in python. (n.d.). Retrieved February 20, 2021, from https://www.datacamp.com/community/tutorials/stemming-lemmatization-python

3.4 Further Research

As suggested in the evaluation section, we could try to perform a Parts of Speech tag (POS) before lemmatization to try to find the main subject of the sentence. This not just increases the accuracy of the system, but could allow us to tag the sentiments of different subjects in the sentence, providing us with a more detailed idea and look into the desired sentiment aimed to be conveyed in each individual subject.

Aside from optimisation of the systems, changing the dataset type could provide us insights on how these two approaches fare against each other in varying conditions.

References :

1. Classifier COMPARISON¶. (n.d.). Retrieved February 17, 2021, from https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html (Date Accessed : 17/02/2021)
2. Devika, M., Sunitha, C., & Ganesh, A. (2016). Sentiment analysis: A comparative study on different approaches. *Procedia Computer Science, 87*, 44-49. doi:10.1016/j.procs.2016.05.124
3. Mäntylä, M. V., Graziotin, D., & Kuutila, M. (2018). The evolution of sentiment analysis—A review of research topics, venues, and top cited papers. *Computer Science Review*, *27*, 16–32. https://doi.org/10.1016/j.cosrev.2017.10.002
4. Sohangir, S., Wang, D., Pomeranets, A., & Khoshgoftaar, T. M. (2018). Big Data: Deep Learning for financial sentiment analysis. *Journal of Big Data*, *5*(1), 1–3. https://doi.org/10.1186/s40537-017-0111-6
5. Scikit-learn. (n.d.). Retrieved February 20, 2021, from https://scikit-learn.org/stable/
6. Team, K. (n.d.). Keras. Retrieved February 20, 2021, from https://keras.io/
7. Sklearn.pipeline.pipeline¶. (n.d.). Retrieved February 20, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
8. Nlp text pre-processing: Text vectorization. (2020, June 15). Retrieved February 20, 2021, from https://www.einfochips.com/blog/nlp-text-vectorization/
9. 6.2. feature extraction¶. (n.d.). Retrieved February 20, 2021, from https://scikit-learn.org/stable/modules/feature_extraction.html
10. Shaikh, J. (2017, October 30). Machine learning, nlp: Text classification using SCIKIT-LEARN, Python and NLTK. Retrieved February 20, 2021, from https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a
11. Brownlee, J. (2019, August 06). How to avoid overfitting in deep learning neural networks. Retrieved February 20, 2021, from https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/
12. Cjhutto. (n.d.). Cjhutto/vadersentiment. Retrieved February 20, 2021, from https://github.com/cjhutto/vaderSentiment
13. Heidenreich, H. (2018, December 21). Stemming? Lemmatization? WHAT? Retrieved February 20, 2021, from https://towardsdatascience.com/stemming-lemmatization-what-ba782b7c0bd8
14. Stemming and lemmatization in python. (n.d.). Retrieved February 20, 2021, from https://www.datacamp.com/community/tutorials/stemming-lemmatization-python

Appendix

Script A - Rule Based Sentiment Analysis

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import pandas as pd
import numpy as np

analyzer = SentimentIntensityAnalyzer()

sentence = "The food there is quite literally the worst I've ever tasted"

#! Checking Sentiment Score
def sentiment_analyser(sentence):
    score = analyzer.polarity_scores(sentence)
    print(f"{sentence} : {str(score)}")
    return score

score = sentiment_analyser(sentence)

print(score)
print(score['compound'])
```

Script B - Comparing Classifiers and Vectorizers

```
import pandas as pd
import spacy
import numpy as np
import sklearn as sk
import en_core_web_sm
from spacy.lang.en.stop_words import STOP_WORDS
import string
from spacy.lang.en import English
# Vectorizers
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
# Sklearn Pipeline
from sklearn.pipeline import Pipeline
# Transformer Mixin



#* Data Engineering
path = r"C:\Users\Admin\Downloads\archive\Tweets.csv"

raw_df = pd.read_csv(path)
# print(df)
col_name = []
for cols in raw_df.columns:
    col_name.append(cols)
print("Column Names :", col_name)
print()

airline_sentiment_list = raw_df["airline_sentiment"].tolist()
text_list = raw_df["text"].tolist()

df = pd.DataFrame()
df["Text"] = text_list
df["Airline Sentiment"] = airline_sentiment_list

#* Checking Dataset
pos_count = 0
neg_count = 0
neu_count =0
```

```
for i in airline_sentiment_list:
    if i == "positive":
        pos_count += 1
    elif i == "negative":
        neg_count += 1
    else:
        neu_count += 1

total_count = pos_count + neg_count + neu_count

print("Total Count :", total_count)
print("Positive Count :", pos_count, "/", total_count, f"({round(pos_count
/ total_count * 100, 1)}%)")
print("Negative Count :", neg_count, "/", total_count, f"({round(neg_count
/ total_count * 100, 1)}%)")
print("Neutral Count :", neu_count, "/", total_count, f"({round(neu_count
/ total_count * 100, 1)}%)")
print()

#* Adding Labels to data
df["Label"] = np.nan

pos_index = df[df["Airline Sentiment"] == "positive"].index.tolist()
neg_index = df[df["Airline Sentiment"] == "negative"].index.tolist()
neu_index = df[df["Airline Sentiment"] == "neutral"].index.tolist()

df["Label"].loc[pos_index] = 1
df["Label"].loc[neg_index] = 0
df["Label"].loc[neu_index] = -1

#* Setting up ML Model

#! Definitely an easier way to change this
nlp = en_core_web_sm.load()

#* Stop Words Initialising
stopwords = list(STOP_WORDS)
# print(stopwords)
#* Punctuation Initialising
punctuations = string.punctuation
```

```
# print(punctuations)
#* Initialising Parser
parser = English()

#* Tokenizer
#! Dealing with OOV Words Using BPE in tokenization -->
https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/
def my_tokenizer(sentence):
    mytokens = parser(sentence)
    mytokens = [ word.lemma_.lower().strip() if word.lemma_ != "-PRON-"
else word.lower_ for word in mytokens ]
    mytokens = [ word for word in mytokens if word not in stopwords and
word not in punctuations ]
    return mytokens


# Basic function to clean the text
def clean_text(text):
    return text.strip().lower()

# Vectorization
#? By Using CountVectorizer function we can convert the text to a matrix
#? By Using CountVectorizer we produce a spare matrix, but take note that
it is sometimes not suited for some ML models and should be converted to a
dense matrix first
#? -->
https://medium.com/@paritosh_30025/natural-language-processing-text-data-v
ectorization-af2520529cf7

vectorizer = CountVectorizer(tokenizer = my_tokenizer, ngram_range=(1,1))
tfidf_vecotirzer = TfidfVectorizer (max_features=2500, min_df=7,
max_df=0.8)


# Classifiers
from sklearn.svm import LinearSVC
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
```

```
from sklearn.neighbors import KNeighborsClassifier

#! Simple Explanation on ML :
#! --> https://www.youtube.com/watch?v=84gqSbLcBFE

# Splitting Data Set
from sklearn.model_selection import train_test_split

# Features and Labels
X = df['Text']
ylabels = df['Label']

#* Splits the data sets into training and test data set
X_train, X_test, y_train, y_test = train_test_split(X, ylabels,
test_size=0.2, random_state=42)

# Create the  pipeline to clean, tokenize, vectorize, and classify
using"Count Vectorizor"
#! Videos on Pipeline : https://www.youtube.com/watch?v=w9IGkBfOoic
classifier_svc = LinearSVC()
classifier_sgd = SGDClassifier()
#! Need to add max_iter=greater than no. of data passed through because
logisticregression defaults it to 100. Else if the error of solving is
varying noticeable, the ALgorithm will fail to converge. (NOTE : When
algorithm fails to convert, it doesn't always mean different accuracy)
#!
https://stackoverflow.com/questions/62658215/convergencewarning-lbfgs-fail
ed-to-converge-status-1-stop-total-no-of-iter
classifier_log = LogisticRegression(max_iter=15000)
classifier_MNB = MultinomialNB()
classifier_DTC = DecisionTreeClassifier()
classifier_RFC = RandomForestClassifier()
classifier_ada = AdaBoostClassifier()
classifier_knc = KNeighborsClassifier()

model_svc = Pipeline([
                ('vectorizer', tfidf_vecotirzer),
                ('classifier', classifier_svc)])
model_sgd = Pipeline([
                ('vectorizer', tfidf_vecotirzer),
```

```
                ('classifier', classifier_sgd)])
model_log = Pipeline([
                ('vectorizer', tfidf_vecotirzer),
                ('classifier', classifier_log)])
model_mnb = Pipeline([
                ('vectorizer', tfidf_vecotirzer),
                ('classifier', classifier_MNB)])
model_dtc = Pipeline([
                ('vectorizer', tfidf_vecotirzer),
                ('classifier', classifier_DTC)])
model_rfc = Pipeline([
                ('vectorizer', tfidf_vecotirzer),
                ('classifier', classifier_RFC)])
model_ada = Pipeline([
                ('vectorizer', tfidf_vecotirzer),
                ('classifier', classifier_ada)])
model_knc = Pipeline([
                ('vectorizer', tfidf_vecotirzer),
                ('classifier', classifier_knc)])

print()
print()
pipelines = [model_svc, model_sgd, model_log, model_mnb, model_dtc,
model_rfc, model_ada, model_knc]
pipeline_names = ["LinearSVC", "SGDClassifier", "LogisticRegression",
"MultinomialNB", "DecisionTreeClassifier", "RandomForestClassifier",
"AdaBoostClassifier", "KNeighborsClassifier"]

for i, pipe in enumerate(pipelines):
    pipe.fit(X_train,y_train)
    print(f"{pipeline_names[i]} :")
    print()
    print("Test Dataset Accuracy: ",pipe.score(X_test,y_test))
    print("Training Dataset Accuracy: ",pipe.score(X_train,y_train))
    print("SUM of Accuracies :", round(pipe.score(X_test,y_test) +
pipe.score(X_train,y_train), 4))
    print('\n')
```

Script C - Comparing Machine Learning Based Approaches with Rules Based Approaches

```
import pandas as pd
import pickle
import spacy
import numpy as np
import sklearn as sk
import en_core_web_sm
from spacy.lang.en.stop_words import STOP_WORDS
import string
from spacy.lang.en import English
# Vectorizers
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
# Sklearn Pipeline
from sklearn.pipeline import Pipeline
# Vader (Rule Based)
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer


#* Data Engineering
path = r"C:\Users\Admin\Downloads\archive\Tweets.csv"

raw_df = pd.read_csv(path)
# print(df)
col_name = []
for cols in raw_df.columns:
    col_name.append(cols)
print("Column Names :", col_name)
print()

airline_sentiment_list = raw_df["airline_sentiment"].tolist()
text_list = raw_df["text"].tolist()

df = pd.DataFrame()
df["Text"] = text_list
df["Airline Sentiment"] = airline_sentiment_list

#* Checking Dataset
```

```
pos_count = 0
neg_count = 0
neu_count = 0
for i in airline_sentiment_list:
    if i == "positive":
        pos_count += 1
    elif i == "negative":
        neg_count += 1
    else:
        neu_count += 1

total_count = pos_count + neg_count + neu_count

print("Total Count :", total_count)
print("Positive Count :", pos_count, "/", total_count, f"({round(pos_count
/ total_count * 100, 1)}%)")
print("Negative Count :", neg_count, "/", total_count, f"({round(neg_count
/ total_count * 100, 1)}%)")
print("Neutral Count :", neu_count, "/", total_count, f"({round(neu_count
/ total_count * 100, 1)}%)")
print()

#* Adding Labels to data
df["Label"] = np.nan

pos_index = df[df["Airline Sentiment"] == "positive"].index.tolist()
neg_index = df[df["Airline Sentiment"] == "negative"].index.tolist()
neu_index = df[df["Airline Sentiment"] == "neutral"].index.tolist()

df["Label"].loc[pos_index] = 1
df["Label"].loc[neg_index] = 0
df["Label"].loc[neu_index] = -1

#* Setting up ML Model

#! Definitely an easier way to change this
nlp = en_core_web_sm.load()

#* Stop Words Initialising
stopwords = list(STOP_WORDS)
```

```
# print(stopwords)
#* Punctuation Initialising
punctuations = string.punctuation
# print(punctuations)
#* Initialising Parser
parser = English()

#* Tokenizer
#! Dealing with OOV Words Using BPE in tokenization -->
https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/
def my_tokenizer(sentence):
    mytokens = parser(sentence)
    mytokens = [ word for word in mytokens if word not in stopwords and
word not in punctuations ]
    mytokens = [ word.lemma_.lower().strip() if word.lemma_ != "-PRON-"
else word.lower_ for word in mytokens ]
    return mytokens


# Vectorization
#? By Using CountVectorizer function we can convert the text to a matrix
#? By Using CountVectorizer we produce a spare matrix, but take note that
it is sometimes not suited for some ML models and should be converted to a
dense matrix first
#? -->
https://medium.com/@paritosh_30025/natural-language-processing-text-data-v
ectorization-af2520529cf7

vectorizer = CountVectorizer(tokenizer = my_tokenizer, ngram_range=(1,1))
tfidf_vectorizer = TfidfVectorizer (max_features=2500, min_df=7,
max_df=0.8)

# Classifiers
from sklearn.svm import LinearSVC


#! More About Classifiers :
#! --> https://www.youtube.com/watch?v=84gqSbLcBFE

# Splitting Data Set
```

```
from sklearn.model_selection import train_test_split

# Features and Labels
X = df['Text']
ylabels = df['Label']

#* Splits the data sets into training and test data set
X_train, X_test, y_train, y_test = train_test_split(X, ylabels,
test_size=0.2, random_state=10)

# Create the  pipeline to clean, tokenize, vectorize, and classify
using"Count Vectorizor"
#? Videos on Pipeline : https://www.youtube.com/watch?v=w9IGkBfOoic

classifier_svc = LinearSVC()

model_svc = Pipeline([
                ('vectorizer', tfidf_vectorizer),
                ('classifier', classifier_svc)])

#! Pipeline Details

    #! The first initial steps are all the data processing one wishes to
do while the last one is the classification algorithm
    #! Last Method ONLY Fit will apply while for the others Fit and
Transform will apply



# Fit our data
model_svc.fit(X_train,y_train)
# Predicting with a test dataset
sample_prediction = model_svc.predict(X_test)

# Prediction Results
# for (sample, pred) in zip(X_test, sample_prediction):
#     print(sample,"Prediction :",pred)

# Accuracy
print("MLT Test Dataset Accuracy: ",
str(round(model_svc.score(X_test,y_test)*100, 2))+"%" )
```

```
# print("Accuracy: ",model_svc.score(X_test,sample_prediction))
# Accuracy
print("MLT Training Dataset Accuracy: ",
str(round(model_svc.score(X_train,y_train)*100, 2))+"%" )
print()


X_test_list = X_test.tolist()
X_train_list = X_train.tolist()
Y_test_list = y_test.tolist()
Y_train_list = y_train.tolist()

analyzer = SentimentIntensityAnalyzer()

#! Checking Sentiment Score
def sentiment_analyser(sentence):
    score = analyzer.polarity_scores(sentence)
    # print(f"{sentence} : {str(score)}")
    return score

X_test_result = []
X_train_result = []


for i, sentence in enumerate(X_test_list):
    score = sentiment_analyser(sentence)
    score_total = score['compound']
    if -0.1 < int(score_total) < 0.1:
        X_test_result.append(-1)
    elif score_total > 0:
        X_test_result.append(1)
    else:
        X_test_result.append(0)

for i, sentence in enumerate(X_train_list):
    score = sentiment_analyser(sentence)
    score_total = score['compound']
    if -0.05 < int(score_total) < 0.05:
        X_train_result.append(-1)
    elif score_total > 0:
```

```
        X_train_result.append(1)
    else:
        X_train_result.append(0)

counter_train = 0
counter_test = 0

for i, x in enumerate(X_test_result):
    if x == Y_test_list[i]:
        counter_test += 1

for i, x in enumerate(X_train_result):
    if x == Y_train_list[i]:
        counter_train += 1

print("Rule Based Test Dataset Accuracy :", str(counter_test) + " / " +
str(len(Y_test_list)) + " (" +
str(round(counter_test/len(Y_test_list)*100, 2)) + "%)" )
print("Rule Based Training Dataset Accuracy :", str(counter_train) + " / "
+ str(len(Y_train_list)) + " (" +
str(round(counter_train/len(Y_train_list)*100, 2)) + "%)" )


#* Saving Trained Model
filename = 'finalized_model.sav'
pickle.dump(model_svc, open(filename, 'wb'))

#* Loading Back Trained Model
# loaded_model = pickle.load(open(filename, 'rb'))
```