# ABSTRACT

Sentiment analysis or opinion mining is one of the major tasks of NLP (Natural Language Processing). Sentiment analysis has gained much attention in recent years. In this paper, we aim to tackle the problem of sentiment polarity categorization, which is one of the fundamental problems of sentiment analysis. A general process for sentiment polarity categorization is proposed with detailed process descriptions. Data used in this study are online product reviews collected from Amazon.com.

Experiments for both sentence-level categorization and review-level categorization are performed with promising outcomes. At last, we also give insight into our future work on sentiment analysis.

# INDEX

| **Chapter** | **Title** | **Page. No** |
| --- | --- | --- |
| 1 | Abstract | 4 |
| 2 | Introduction | 7 |
| 3 | Requirement Analysis | 8 |
| 4 | Real-World Example | 10 |
| 5 | Implementation | 12 |
| 6 | Bag of Words | 19 |
| 7 | Test Data Transformation | 23 |
| 8 | Conclusion | 24 |
| 9 | Result | 27 |
| 10 | References | 28 |

## Teamwork Plan & Execution

| **Team Member Name** | **Work Planned** | **Work Completed** | **Remarks** |
| --- | --- | --- | --- |
| Shourya Prasad | Individual project | 100% | - |
|  |  |  | - |
|  |  |  | - |
|  |  |  |  |

**Team Member 1 Sign Team Member 2 Sign Team Member 3 Sign Team Member 4 Sign**

# INTRODUCTION

Sentiment Analysis, as the name suggests, means to identify the view or emotion behind a situation. It basically means to analyze and find the emotion or intent behind a piece of text or speech or any mode of communication.

In this article, we will focus on the sentiment analysis of text data.

We, humans, communicate with each other in a variety of languages, and any language is just a mediator or a way in which we try to express ourselves. And, whatever we say has a sentiment associated with it. It might be positive or negative or it might be neutral as well.

# REQUIREMENT ANALYSIS

Suppose, there is a fast-food chain company and they sell a variety of different food items like burgers, pizza, sandwiches, milkshakes, etc. They have created a website to sell their food and now the customers can order any food item from their website and they can provide reviews as well, like whether they liked the food or hated it.

* User Review 1: I love this cheese sandwich, it’s so delicious.
* User Review 2: This chicken burger has a very bad taste.
* User Review 3: I ordered this pizza today.

So, as we can see that out of these above 3 reviews,

The first review is definitely a positive one and it signifies that the customer was really happy with the sandwich.

The second review is negative, and hence the company needs to look into their burger department.

And, the third one doesn’t signify whether that customer is happy or not, and hence we can consider this as a neutral statement.

By looking at the above reviews, the company can now conclude that it needs to focus more on the production and promotion of their sandwiches as well as improve the quality of their burgers if they want to increase their overall sales.

But, now a problem arises, that there will be hundreds and thousands of user reviews for their products and after a point of time it will become nearly impossible to scan through each user review and come to a conclusion.

Neither can they just come up with a conclusion by taking just 100 reviews or so, because maybe the first 100-200 customers were having similar taste and liked the sandwiches, but over time when the no. of reviews increases, there might be a situation where the positive reviews are overtaken by more no. of negative reviews.

Therefore, this is where the Sentiment Analysis Model comes into play, which takes in a huge corpus of data having user reviews and finds a pattern and comes up with a conclusion based on real evidence rather than assumptions made on a small sample of data.

(We will explore the working of a basic Sentiment Analysis model later in this article.)

We can even break these principal sentiments(positive and negative) into smaller sub sentiments such as “Happy”, “Love”, ”Surprise”, “Sad”, “Fear”, “Angry” etc. as per the needs or business requirement.

# REAL-WORLD EXAMPLE

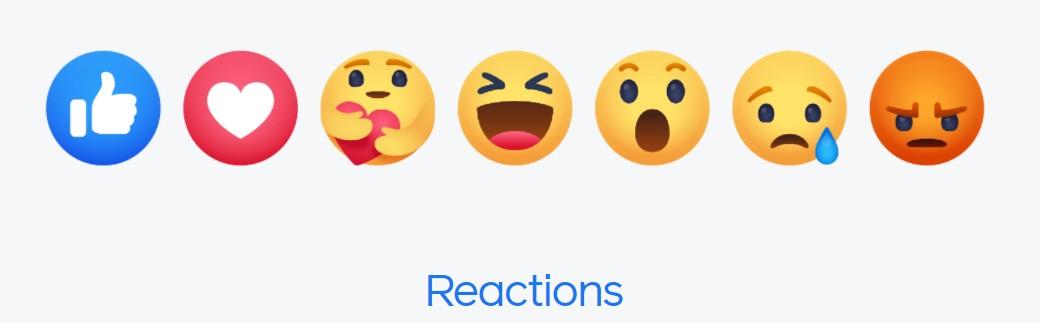
* There was a time when the social media

services like Facebook used to just have two emotions associated with each post, i.e You can like a post or you can leave the post without any reaction and that basically signifies that you didn’t like it.

* But, over time these reactions to post have

changed and grew into more granular sentiments which we see as of now, such as “like”, “love”, “sad”,

“angry” etc.



And, because of this upgrade, when any company promotes their products on Facebook, they receive more specific reviews which will help them to enhance the customer experience.

And because of that, they now have more granular control on how to handle their consumers, i.e. they can target the customers who are just “sad” in a different way as compared to customers who are “angry”, and come up with a business plan accordingly because nowadays, just doing the bare minimum is not enough.

Now, as we said we will be creating a Sentiment Analysis Model, but it’s easier said than done.

As we humans communicate with each other in a way that we call Natural Language which is easy for us to interpret but it’s much more complicated and messy if we really look into it.

Because there are billions of people and they have their own style of communicating, i.e. a lot of tiny variations are added to the language and a lot of sentiments are attached to it which is easy for us to interpret but it becomes a challenge for the machines.

This is why we need a process that makes the computers understand the Natural Language as we humans do, and this is what we call Natural Language Processing(NLP). And, as we know Sentiment Analysis is a sub-field of NLP and with the help of machine learning techniques, it tries to identify and extract the insights.

# IMPLEMENTATION

*#importing all the Librarires required*

!pip3 install seaborn  
**import** pandas **as** pd  
**import** matplotlib.pyplot **as** plt  
**import** seaborn **as** sns  
**from** wordcloud **import** WordCloud  
**import** re

Requirement already satisfied: seaborn in c:\users\shourya\anaconda3\lib\site-packages (0.12.2)  
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in c:\users\shourya\anaconda3\lib\site-packages (from seaborn) (3.7.0)  
Requirement already satisfied: pandas>=0.25 in c:\users\shourya\anaconda3\lib\site-packages (from seaborn) (1.5.3)  
Requirement already satisfied: numpy!=1.24.0,>=1.17 in c:\users\shourya\anaconda3\lib\site-packages (from seaborn) (1.23.5)  
Requirement already satisfied: fonttools>=4.22.0 in c:\users\shourya\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (4.25.0)  
Requirement already satisfied: python-dateutil>=2.7 in c:\users\shourya\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (2.8.2)  
Requirement already satisfied: pillow>=6.2.0 in c:\users\shourya\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (9.4.0)  
Requirement already satisfied: cycler>=0.10 in c:\users\shourya\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (0.11.0)  
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\shourya\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.4.4)  
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\shourya\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (3.0.9)  
Requirement already satisfied: contourpy>=1.0.1 in c:\users\shourya\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.0.5)  
Requirement already satisfied: packaging>=20.0 in c:\users\shourya\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (22.0)  
Requirement already satisfied: pytz>=2020.1 in c:\users\shourya\anaconda3\lib\site-packages (from pandas>=0.25->seaborn) (2022.7)  
Requirement already satisfied: six>=1.5 in c:\users\shourya\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.1->seaborn) (1.16.0)

*#installing wordcloud*

!pip3 install wordcloud

Requirement already satisfied: wordcloud in c:\users\shourya\anaconda3\lib\site-packages (1.9.2)  
Requirement already satisfied: matplotlib in c:\users\shourya\anaconda3\lib\site-packages (from wordcloud) (3.7.0)  
Requirement already satisfied: pillow in c:\users\shourya\anaconda3\lib\site-packages (from wordcloud) (9.4.0)  
Requirement already satisfied: numpy>=1.6.1 in c:\users\shourya\anaconda3\lib\site-packages (from wordcloud) (1.23.5)  
Requirement already satisfied: contourpy>=1.0.1 in c:\users\shourya\anaconda3\lib\site-packages (from matplotlib->wordcloud) (1.0.5)  
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\shourya\anaconda3\lib\site-packages (from matplotlib->wordcloud) (3.0.9)  
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\shourya\anaconda3\lib\site-packages (from matplotlib->wordcloud) (1.4.4)  
Requirement already satisfied: cycler>=0.10 in c:\users\shourya\anaconda3\lib\site-packages (from matplotlib->wordcloud) (0.11.0)  
Requirement already satisfied: packaging>=20.0 in c:\users\shourya\anaconda3\lib\site-packages (from matplotlib->wordcloud) (22.0)  
Requirement already satisfied: fonttools>=4.22.0 in c:\users\shourya\anaconda3\lib\site-packages (from matplotlib->wordcloud) (4.25.0)  
Requirement already satisfied: python-dateutil>=2.7 in c:\users\shourya\anaconda3\lib\site-packages (from matplotlib->wordcloud) (2.8.2)  
Requirement already satisfied: six>=1.5 in c:\users\shourya\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib->wordcloud) (1.16.0)

*#nltk – Natural Language Toolkit is a collection of libraries for natural language processing*  
*#stopwords – a collection of words that don’t provide any meaning to a sentence*  
*#WordNetLemmatizer – used to convert different forms of words into a single item but still keeping the*

**import** nltk  
nltk.download('stopwords')  
nltk.download('wordnet')  
nltk.download('omw-1.4')  
**from** nltk.corpus **import** stopwords  
**from** nltk.stem **import** WordNetLemmatizer

[nltk\_data] Downloading package stopwords to  
[nltk\_data] C:\Users\Shourya\AppData\Roaming\nltk\_data...  
[nltk\_data] Package stopwords is already up-to-date!  
[nltk\_data] Downloading package wordnet to  
[nltk\_data] C:\Users\Shourya\AppData\Roaming\nltk\_data...  
[nltk\_data] Package wordnet is already up-to-date!  
[nltk\_data] Downloading package omw-1.4 to  
[nltk\_data] C:\Users\Shourya\AppData\Roaming\nltk\_data...  
[nltk\_data] Package omw-1.4 is already up-to-date!

*#Scikit-Learn (Machine Learning Library for Python)*  
*#CountVectorizer – transform text to vectors*  
*#RandomizedSearchCV – for hyperparameter tuning*  
*#RandomForestClassifier – machine learning algorithm for classification*  
*#Accuracy Score – no. of correctly classified instances/total no. of instances*  
*#Precision Score – the ratio of correctly predicted instances over total positive instances*  
*#Recall Score – the ratio of correctly predicted instances over total instances in that class*  
*#Roc Curve – a plot of true positive rate against false positive rate*  
*#Classification Report – report of precision, recall and f1 score*  
*#Confusion Matrix – a table used to describe the classification models*

!pip install -U scikit-learn  
**from** sklearn.model\_selection **import** GridSearchCV  
**from** sklearn.ensemble **import** RandomForestClassifier  
**from** sklearn.metrics **import** roc\_curve  
**from** sklearn.metrics **import** auc  
**from** sklearn.metrics **import** ConfusionMatrixDisplay, accuracy\_score, precision\_score, recall\_score, classification\_report

Requirement already satisfied: scikit-learn in c:\users\shourya\anaconda3\lib\site-packages (1.2.2)  
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\shourya\anaconda3\lib\site-packages (from scikit-learn) (2.2.0)  
Requirement already satisfied: joblib>=1.1.1 in c:\users\shourya\anaconda3\lib\site-packages (from scikit-learn) (1.1.1)  
Requirement already satisfied: numpy>=1.17.3 in c:\users\shourya\anaconda3\lib\site-packages (from scikit-learn) (1.23.5)  
Requirement already satisfied: scipy>=1.3.2 in c:\users\shourya\anaconda3\lib\site-packages (from scikit-learn) (1.10.0)

*#emotion Dataset includes train.txt,test.txt,val.txt*

df\_train = pd.read\_csv("train.txt",delimiter=';',names=['text','label'])  
df\_val = pd.read\_csv("val.txt",delimiter=';',names=['text','label'])

df = pd.concat([df\_train,df\_val])  
df.reset\_index(inplace=True,drop=True)

*#Now, we will create a custom encoder to convert categorical target labels to numerical form, i.e. (0 and 1)*

**def** custom\_encoder(df):  
 df.replace(to\_replace ="surprise", value =1, inplace=True)  
 df.replace(to\_replace ="love", value =1, inplace=True)  
 df.replace(to\_replace ="joy", value =1, inplace=True)  
 df.replace(to\_replace ="fear", value =0, inplace=True)  
 df.replace(to\_replace ="anger", value =0, inplace=True)  
 df.replace(to\_replace ="sadness", value =0, inplace=True)  
custom\_encoder(df['label'])

*#Data Pre-Processing*

*#object of WordNetLemmatizer*  
lm = WordNetLemmatizer()  
**def** text\_transformation(df\_col):  
 corpus = []  
 **for** item **in** df\_col:  
 new\_item = re.sub('[^a-zA-Z]',' ',str(item))  
 new\_item = new\_item.lower()  
 new\_item = new\_item.split()  
 new\_item = [lm.lemmatize(word) **for** word **in** new\_item **if** word **not** **in** set(stopwords.words('english'))]  
 corpus.append(' '.join(str(x) **for** x **in** new\_item))  
 **return** corpus

corpus = text\_transformation(df['text'])

*#Bag Of Words(BOW)*

**from** pylab **import** rcParams  
rcParams['figure.figsize'] = 20,8  
word\_cloud = ""  
**for** row **in** corpus:  
 **for** word **in** row:  
 word\_cloud+=" ".join(word)  
wordcloud = WordCloud(width = 1000, height = 500,background\_color ='white',min\_font\_size = 10).generate(word\_cloud)  
plt.imshow(wordcloud)

<matplotlib.image.AxesImage at 0x20d88e5a200>



*#Estimator or model – RandomForestClassifier in our case*  
*#parameters – dictionary of hyperparameter names and their values*  
*#cv – signifies cross-validation folds*  
*#return\_train\_score – returns the training scores of the various models*  
*# n\_jobs – no. of jobs to run parallelly (“-1” signifies that all CPU cores will be used which reduces the training time drastically)*  
*#First, We will create a dictionary, “parameters” which will contain the values of different hyperparameters.*  
*#We will pass this as a parameter to RandomSearchCV to train our random forest classifier model using all possible*   
*#combinations of these parameters to find the best model.*

**from** sklearn.feature\_extraction.text **import** CountVectorizer  
cv = CountVectorizer(ngram\_range=(1,2))  
traindata = cv.fit\_transform(corpus)  
X = traindata  
y = df.label

parameters = {'max\_features': ('auto','sqrt'),  
 'n\_estimators': [50, 10, 15],  
 'max\_depth': [2, 5, None],  
 'min\_samples\_split': [2, 5, 10],  
 'min\_samples\_leaf': [1, 2, 5, 10],  
 'bootstrap': [True, False]}

**from** sklearn.model\_selection **import** RandomizedSearchCV  
rf\_RandomGrid = RandomizedSearchCV(RandomForestClassifier(),parameters,n\_iter=10,cv=5,return\_train\_score=True,n\_jobs=-1)  
rf\_RandomGrid.fit(X,y)  
rf\_RandomGrid.best\_params\_

{'n\_estimators': 15,  
 'min\_samples\_split': 10,  
 'min\_samples\_leaf': 1,  
 'max\_features': 'sqrt',  
 'max\_depth': None,  
 'bootstrap': True}

*#And then, we can view all the models and their respective parameters,*  
*#mean test score and rank as GridSearchCV stores all the results in the cv\_results\_ attribute.*

**for** i **in** range(10):  
 print('Parameters: ',rf\_RandomGrid.cv\_results\_['params'][i])  
 print('Mean Test Score: ',rf\_RandomGrid.cv\_results\_['mean\_test\_score'][i])  
 print('Rank: ',rf\_RandomGrid.cv\_results\_['rank\_test\_score'][i])

Parameters: {'n\_estimators': 10, 'min\_samples\_split': 10, 'min\_samples\_leaf': 5, 'max\_features': 'auto', 'max\_depth': None, 'bootstrap': False}  
Mean Test Score: 0.9354444444444443  
Rank: 3  
Parameters: {'n\_estimators': 50, 'min\_samples\_split': 5, 'min\_samples\_leaf': 10, 'max\_features': 'auto', 'max\_depth': None, 'bootstrap': True}  
Mean Test Score: 0.8012777777777778  
Rank: 4  
Parameters: {'n\_estimators': 50, 'min\_samples\_split': 5, 'min\_samples\_leaf': 2, 'max\_features': 'auto', 'max\_depth': 2, 'bootstrap': False}  
Mean Test Score: 0.5443888888888889  
Rank: 9  
Parameters: {'n\_estimators': 50, 'min\_samples\_split': 2, 'min\_samples\_leaf': 10, 'max\_features': 'auto', 'max\_depth': 5, 'bootstrap': False}  
Mean Test Score: 0.5451111111111111  
Rank: 6  
Parameters: {'n\_estimators': 50, 'min\_samples\_split': 2, 'min\_samples\_leaf': 5, 'max\_features': 'auto', 'max\_depth': 5, 'bootstrap': False}  
Mean Test Score: 0.5449999999999999  
Rank: 7  
Parameters: {'n\_estimators': 15, 'min\_samples\_split': 2, 'min\_samples\_leaf': 1, 'max\_features': 'auto', 'max\_depth': None, 'bootstrap': True}  
Mean Test Score: 0.9411111111111111  
Rank: 2  
Parameters: {'n\_estimators': 15, 'min\_samples\_split': 10, 'min\_samples\_leaf': 1, 'max\_features': 'sqrt', 'max\_depth': None, 'bootstrap': True}  
Mean Test Score: 0.9425555555555555  
Rank: 1  
Parameters: {'n\_estimators': 10, 'min\_samples\_split': 10, 'min\_samples\_leaf': 2, 'max\_features': 'sqrt', 'max\_depth': 5, 'bootstrap': False}  
Mean Test Score: 0.5895  
Rank: 5  
Parameters: {'n\_estimators': 50, 'min\_samples\_split': 10, 'min\_samples\_leaf': 2, 'max\_features': 'sqrt', 'max\_depth': 5, 'bootstrap': False}  
Mean Test Score: 0.5446666666666666  
Rank: 8  
Parameters: {'n\_estimators': 50, 'min\_samples\_split': 2, 'min\_samples\_leaf': 10, 'max\_features': 'sqrt', 'max\_depth': 2, 'bootstrap': True}  
Mean Test Score: 0.5443888888888889  
Rank: 9

*#Now, we will choose the best parameters obtained from GridSearchCV and create a final random forest classifier model and then train our new model.*

rfc = RandomForestClassifier(max\_features=rf\_RandomGrid.best\_params\_['max\_features'],  
 max\_depth=rf\_RandomGrid.best\_params\_['max\_depth'],  
 n\_estimators=rf\_RandomGrid.best\_params\_['n\_estimators'],  
 min\_samples\_split=rf\_RandomGrid.best\_params\_['min\_samples\_split'],  
 min\_samples\_leaf=rf\_RandomGrid.best\_params\_['min\_samples\_leaf'],  
 bootstrap=rf\_RandomGrid.best\_params\_['bootstrap'])  
rfc.fit(X,y)

RandomForestClassifier(min\_samples\_split=10, n\_estimators=15)

*#Test Data Transformation*

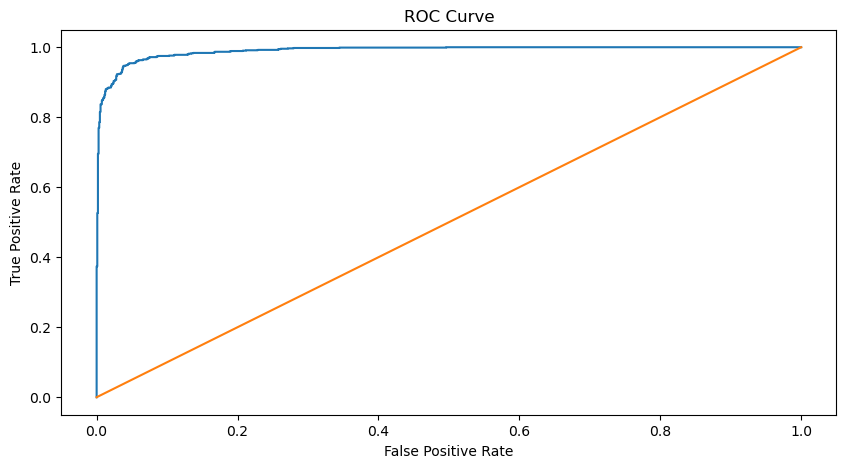
test\_df = pd.read\_csv('test.txt',delimiter=';',names=['text','label'])  
X\_test,y\_test = test\_df.text,test\_df.label  
*#encode the labels into two classes , 0 and 1*  
test\_df = custom\_encoder(y\_test)  
*#pre-processing of text*  
test\_corpus = text\_transformation(X\_test)  
*#convert text data into vectors*  
testdata = cv.transform(test\_corpus)  
*#predict the target*  
predictions = rfc.predict(testdata)

rcParams['figure.figsize'] = 10,5  
cm=ConfusionMatrixDisplay.\_\_init\_\_(y\_test,predictions)  
acc\_score = accuracy\_score(y\_test,predictions)  
pre\_score = precision\_score(y\_test,predictions)  
rec\_score = recall\_score(y\_test,predictions)  
print('Accuracy\_score: ',acc\_score)  
print('Precision\_score: ',pre\_score)  
print('Recall\_score: ',rec\_score)  
print("-"\*50)  
cr = classification\_report(y\_test,predictions)  
print(cr)

Accuracy\_score: 0.9475  
Precision\_score: 0.957351290684624  
Recall\_score: 0.9271739130434783  
--------------------------------------------------  
 precision recall f1-score support  
  
 0 0.94 0.96 0.95 1080  
 1 0.96 0.93 0.94 920  
  
 accuracy 0.95 2000  
 macro avg 0.95 0.95 0.95 2000  
weighted avg 0.95 0.95 0.95 2000

*#Model Evaluation*

predictions\_probability = rfc.predict\_proba(testdata)  
fpr,tpr,thresholds = roc\_curve(y\_test,predictions\_probability[:,1])  
plt.plot(fpr,tpr)  
plt.plot([0,1])  
plt.title('ROC Curve')  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.show()



*#Model Prediction*

**def** expression\_check(prediction\_input):  
 **if** prediction\_input == 0:  
 print("Input statement has Negative Sentiment.")  
 **elif** prediction\_input == 1:  
 print("Input statement has Positive Sentiment.")  
 **else**:  
 print("Invalid Statement.")  
*# function to take the input statement and perform the same transformations we did earlier*  
**def** sentiment\_predictor(input):  
 input = text\_transformation(input)  
 transformed\_input = cv.transform(input)  
 prediction = rfc.predict(transformed\_input)  
 expression\_check(prediction)  
input1 = ["i feel more virtuous than when i eat veggies dipped in hummus."]  
input2 = ["I bought a new phone and it's so bad."]  
sentiment\_predictor(input1)  
sentiment\_predictor(input2)

Input statement has Positive Sentiment.  
Input statement has Negative Sentiment.



# RESULT

* We have successfully created a Sentimental Analysis model using NLP.
* We Revised New Libraries for optimal runtime and ensured letter runtime.
* We use Random Forest Classifier and RandomizedSearch (i.e RandomizedSearchCV instead of GridSearchCV)

**REFERENCES**

* We have used the dataset that contains 3 separate files named train.txt, test.txt and val.txt.
* https://www.analyticsvidhya.com/blog/2021/06/nlp-sentiment-analysis/