

PROJECT REPORT
for
**DATA MINING AND DATA
WAREHOUSING
CSPC-328**



SUBMITTED TO:

**DR. NONITA SHARMA
ASSISTANT PROFESSOR
CSE DEPARTMENT**

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

**DR. B.R. AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY
JALANDHAR-144011**

MACHINE LEARNING APPROACH FOR HIDDEN EMOTIONS ANALYSIS

CONTENTS

i Team Members' Details.....	3
ii Links to Code.....	3
1. Abstract.....	4
2. Keywords.....	4
3. Introduction:	
3.1 Real-life Application.....	4
3.2 Dataset.....	4
3.3 Classification.....	5
3.4 Structure of the Project Report.....	5 – 6
4. Classification Methods:	
4.1 Information Gain.....	6
4.2 Gini Index.....	6
4.3 Naïve Bayes.....	6 – 7
4.4 KNN.....	7 – 8
4.5 Random Forest.....	8
4.6 Gradient Boost.....	9 – 10
5. Experimentation.....	10 – 24
6. Results and Discussion.....	24 – 29
7. Conclusion and Future Scope.....	29

TEAM MEMBER'S DETAILS

Roll No.	Name	Course Year/Semester	Email-id
18103030	Divyansha Sharma	B.Tech/CSE/6 Sem	divyanshas.cs.18@nitj.ac.in
18103037	Gurmeet	B.Tech/CSE/6 Sem	gurmeet.cs.18@nitj.ac.in
18103043	Hrishita	B.Tech/CSE/6 Sem	hrishita.cs.18@nitj.ac.in

LINK TO CODE:

[https://raw.githubusercontent.com/Dataminingproject/dataminingassignment/main/smileannotationsfinal%20\(1\).csv](https://raw.githubusercontent.com/Dataminingproject/dataminingassignment/main/smileannotationsfinal%20(1).csv)

1. Abstract

It is the contextual study & analysis of emotions induced by reading a tweet. By sentimental analysis we find impact of a text and the attitude it induces in a person. It comes under the domain of natural language processing along with text mining. Sentimental analysis helps to find how a statement is going to change the emotions of readers and based on that one can find suitable words to use for making the desired impact. We can use this analysis in political activities, making a campaign popular, in protest, popularity of individuals, business improvement, etc. In our project, we have operated on a real-world dataset after getting our hands on a smaller pool of it to explore how machine learning algorithms; supervised and unsupervised, can be used to find certain patterns and study chronological responses of algorithms on certain dataset.

After applying a given set of classification algorithms, herein lies our final report.

2. Keywords

Machine learning, Supervised Learning, Classification, Data Pre-processing, Bag of words

3. Introduction

3.1 Real Life Application

Sentimental analysis is used in various domains to extract the contextual information like:

Business: In understanding the sentiments associated with a brand, render services according to the feedback from the customers & provide a boom to the business.

Politics: In analysing the social influence of standing political bodies, from the writings of the public.

Opinion Building: In structuring opinion of an individual from inferred emotions associated with the writing.

Security Purposes: In searching for anti-social elements of the society who are spreading hatred provoking protests & social disorder.

3.2 Dataset

Source of dataset: This dataset is collected from twitter and feedback submitted by users for a comment. To collect the data replies of users also stored and categorized into emotions.

Attributes

1. Key: it is a nominal attribute which tells the id of the writer like 611511161

2. Tweet: it is a nominal attribute which tells the text reader to read like @rohit congo on your promotion. it is our independent or predictor variable.

3. Status: it is a categorical attribute which tells about the status of emotion of the reader like sad, happy, surprise, disgust, mix emotion etc. It is out dependent or response variable.

4. Nature: this dataset is of discrete nature with nominal and categorical nature attributes. This dataset has tweets in natural text form like (congo and all the best for your new job) which are not very useful in analysis. So, they need to be processed into a numerical form using natural **language processing**.

3.3 Classification

Classification as the name suggests is “to classify” something or in a lay man’s language assigning classes to a pool of data according to a patterned response.

For example, if one wants to study the nature of a person going out to play badminton according to the weather outside then we have two class labels namely: Yes/No (i.e either he will play or not) for a particular weather report like Temperature, wind, Rain, Humidity etc as the features of our data.

Advantages: - It is most suited for discrete dataset i.e in predicting categorical data because classification means assigning a particular category to our data. Hence, best suited when the predictor variable is categorical.

For example, Spam email detection, sentiment analysis, predicting if a middle-class customer goes to buy milk from a local nearby shop daily or not provided the training set containing daily-quality of milk, his salary etc.

Scope: Scope of classification algos is limited to classifying the algorithms only. Binary, Multiclass, Multi-label classification is an aid in the classification algorithm. For example:- Image classifier, Sentiment classifier and Survey Feedback classifier.

Limitations: - When we choose any algorithm, we are inclined towards the complexity of that particular algorithm. Therefore, choosing an algo with greater complexity on a larger dataset always provides the edge **BUT** here it is the disadvantage as it requires a large amount of time to process for some classification algorithms.

Methods: - Information Gain, Gini Index, Naïve Bayes, KNN, Random Forest, Gradient Boost to name a few.

3.4 Structure of the Project Report

Our Report covers up the following areas in the same order as mentioned below step by step

➤ **Analysis:**

Using methods like-

- i. Information Gain
- ii. Gini Index
- iii. Naïve Bayes
- iv. KNN
- v. Random Forest
- vi. Gradient Boost

➤ **Data Cleaning:**

- i. Removal of Null values/Unnecessary data
- ii. Structuring data
- iii. Removing irrelevant data
- iv. Filtering of text that indicates same emotions (Synonyms)
- v. Normalization
- vi. Train & Test Splitting

➤ **Classification:**

- i. Naïve Bayes Classification
- ii. Decision Tree
- iii. KNN
- iv. Random Forest

v. Gradient Boost

➤ **Performance Measures:**

- i. Accuracy
- ii. Confusion Matrix
- iii. F-score
- iv. Precision
- v. Recall

➤ **Visualization:**

- i. Pie Chart
- ii. Bar Graph

4. Classification Methods**4.1 Information Gain**

Explanation: Information gain is clear as the name suggests “gain of information” i.e. extracting important information to improve our training model. It is the reduction in randomness/entropy by transforming a dataset and is often used in training decision trees.

$$H = -(p_i \log_2 p_i)$$

$$Gain(S, D) = H(S) - \sum_{V \in D} \frac{|V|}{|S|} H(V)$$

4.2 Gini Index

Explanation: -The Gini Index is basically used to measure the impurity in our dataset and is calculated by subtracting the sum of the squared probabilities of each feature from one in **CART** algorithm.

$$Gini = 1 - \sum_{i=1}^C (p_i)^2$$

Lower Gini Index=High Purity.

4.3 Naïve Baye

Explanation: Naive Bayes Classifiers work on Bayes theorem of conditional probability. It assumes that each feature is independent and has the same weightage in calculation of target attribute.

Let X and Y are two events then according to bayes theorem:

$$P(X/Y) = (P(Y/X) * P(X)) / P(Y)$$

P(X/Y) = probability of X given that Y is done

$P(Y/X)$ = probability of Y given that X is done

$P(X)$ = probability of X

$P(Y)$ = probability of Y

Algorithm

- i) Calculate mean and standard deviation for the targeted variable.
- ii) Calculate the probability class wise using the *Gaussian density equation*.
- iii) Repeat the steps for each variable.
- iv) For each class calculate the greatest likelihood.

Working

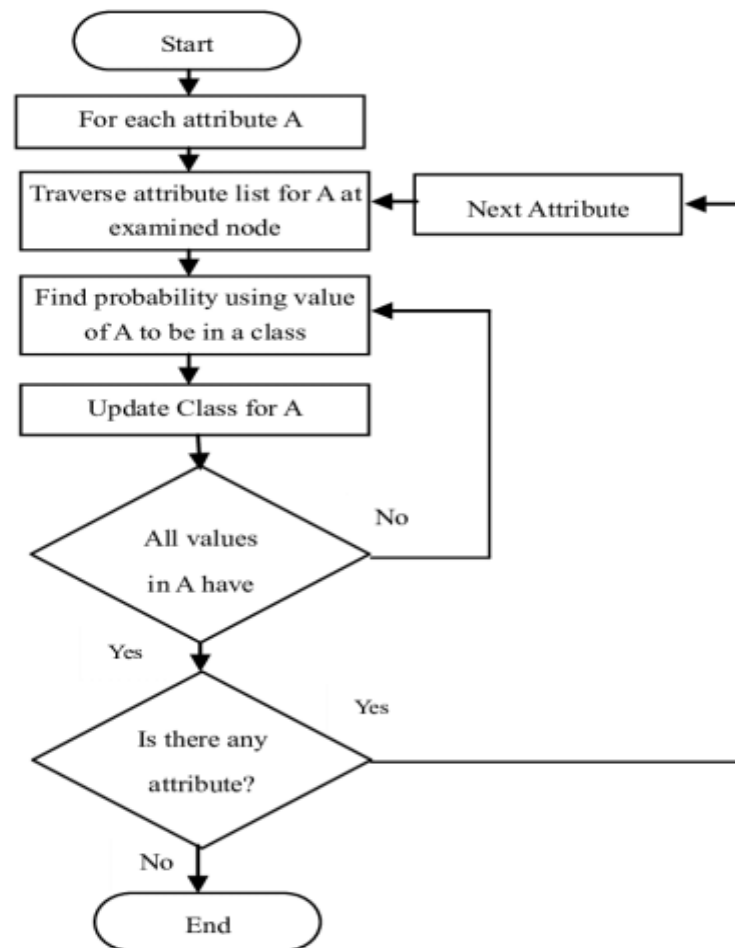


Figure 1 Naive Bayes Flow chart

4.4 KNN:

Explanation: KNN is a **non-parametric** classification algorithm which uses nearest k neighbours of a data point to predict the response variable. In this we use various distance measures to get k neighbours, and then we predict the most frequent neighbour as the target value. Various distance measures are euclidean distance, manhattan distance etc.

Euclidean distance = $\sqrt{\sum((x_{i1}-x_{i2})^2)}$

Manhattan distance = $\sum((\text{abs}(x_{i1}-x_{i2}))$

Algorithm

- i. Load the training and testing data & select an appropriate value of K.
- ii. Select K number neighbours, then calculate **Euclidean distance** of these neighbours.
- iii. As per the calculated Euclidean distance take K nearest neighbours.
- iv. For each category count number of data points, out of these K neighbours & assign the labels.
- v. The category for which the number of neighbours is maximum assign it new data points.

Working

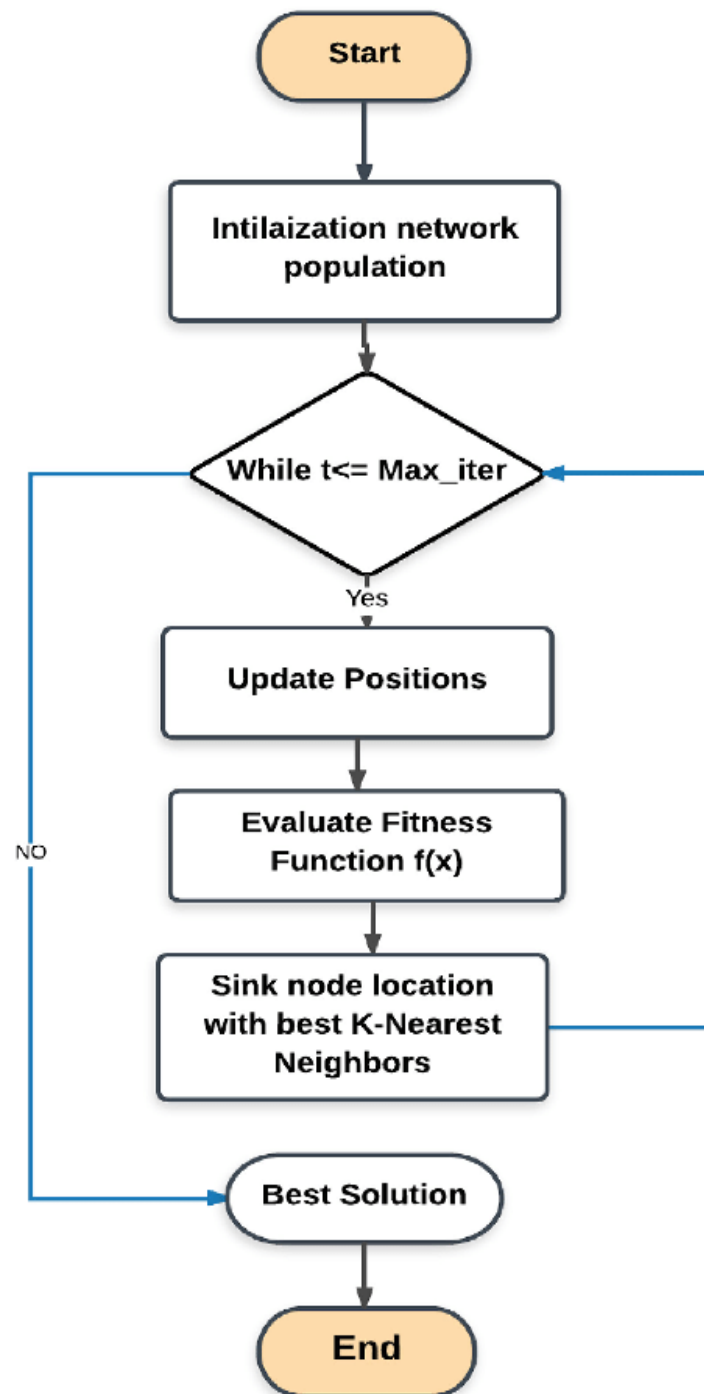


Figure 2 KNN Flowchart

4.5 Random Forest

A supervised learning algorithm that builds a more accurate & stable prediction model multiple decision by merging multiple decision tree, Random forest. It is a building block of that can be used for both regression & classification.

By keeping an account of how much the tree nodes that use that feature reduce impurity across all trees in the forest, we can measure a feature's importance.

Algorithm:

Condition:

Input Training set $A = (a_1, b_1), (a_2, b_2), \dots, (a_n, b_n)$, features F and number of tree N

Function $\text{RandomForest}(A, F)$:

- i) Initialize K as a Null set.
- ii) for i in $1, 2, \dots, N$ do A
 - (i) A^i is a bootstrap sample from A
 - k_i RandomizedTreeLearn ($A(i), F$)
 - K is $K \cup \{k_i\}$
- end for
- iii) return K
- iv) end function

Working

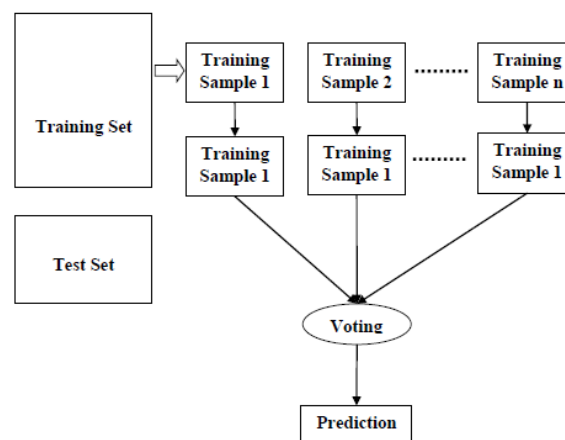


Figure 3 Working on Random Forest

4.6 Gradient Boosting

It is a technique that works on the principle of 'boosting' to develop a strong predictive model. It combines many weak algorithms, optimizing loss functions to build an additive model. The model improves its performance by minimizing the predecessor's prediction error.

Feature importance associated with Gradient Boosting Model are gain, coverage & frequency, these are constructed similar to Random Forest.

Algorithm:

1. Inputs
 $\{(x_i, y_i)\}_{i=1}^n$, Loss function $L(y, F(x))$,
2. For $m = 1$ to M
 - a. Pseudo-residuals

$$F_0(x) = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma).$$
 - b. Compute for optimization

$$\gamma_m = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)).$$
 - c. Model Updation

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$
3. Output $F_M(x)$.

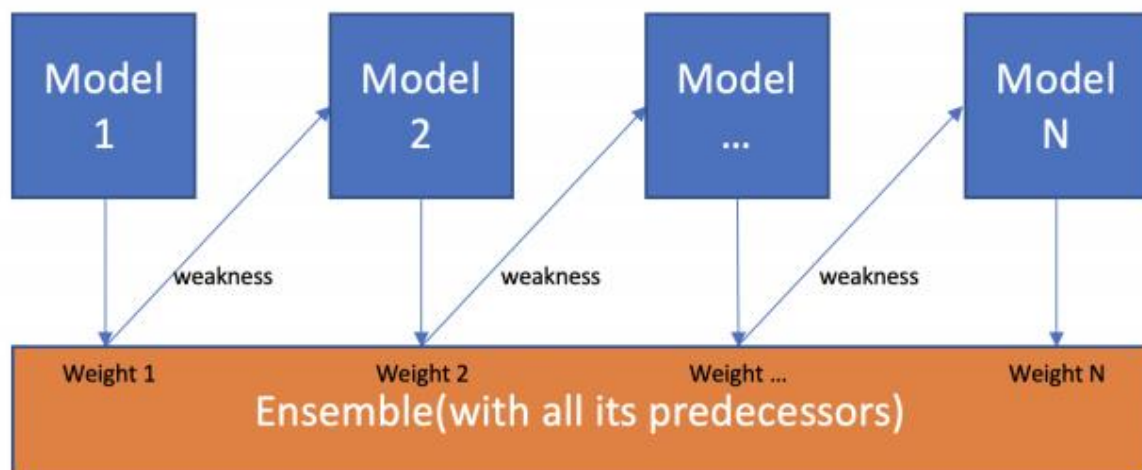
Working

Figure 4 Working of Gradient Boost

5. Experimentation

Just like we extract important information using KDD (Knowledge Discovery in Databases) process, we have done similar experimentation on our imported dataset.

5.1 Data Importing and Selection

It includes importing a dataset to analysing for the further work upon it using various classification algorithms.

- **Importing:** We have imported smile-annotations-final.csv for analysing it further.

Input:

```
import pandas as pd
data = pd.read_csv("https://raw.githubusercontent.com/Dataminingproject/dataminingassignment/main/smileannotationsfinal%20(1).csv",
                  names=["key", "tweet", "status"])

data
```

Figure 5

Output:

	key	tweet	status
0	611857364396965889	@aandraous @britishmuseum @AndrewsAntonio Merc...	nocode
1	614484565059596288	Dorian Gray with Rainbow Scarf #LoveWins (from...	happy
2	614746522043973632	@SelectShowcase @Tate_Stlves ... Replace with ...	happy
3	614877582664835073	@Sofabsports thank you for following me back. ...	happy
4	611932373039644672	@britishmuseum @TudorHistory What a beautiful ...	happy
...
3080	613678555935973376	MT @AliHaggett: Looking forward to our public ...	happy
3081	613294681225621504	@britishmuseum Upper arm guard?	nocode
3082	615246897670922240	@MrStuchbery @britishmuseum Mesmerising.	happy
3083	613016084371914753	@NationalGallery The 2nd GENOCIDE against #Bia...	not-relevant
3084	611566876762640384	@britishmuseum Experience #battlewaterloo from...	nocode

3085 rows × 3 columns

Figure 6 Dataset

- **Data Selection:** The data gathered needs to be effectively selected and segregated into meaningful sets based on its necessity. These parameters are crucial for data analysis because they form the base for further processing and will affect what kinds of data models are formed using different algorithms

Input:

```
data=data.iloc[:,1:]
data
```

Figure 7

Output:

	tweet	status
0	@aandraous @britishmuseum @AndrewsAntonio Merc...	nocode
1	Dorian Gray with Rainbow Scarf #LoveWins (from...	happy
2	@SelectShowcase @Tate_Stlves ... Replace with ...	happy
3	@Sofabsports thank you for following me back. ...	happy
4	@britishmuseum @TudorHistory What a beautiful ...	happy
...
3080	MT @AliHaggett: Looking forward to our public ...	happy
3081	@britishmuseum Upper arm guard?	nocode
3082	@MrStuchbery @britishmuseum Mesmerising.	happy
3083	@NationalGallery The 2nd GENOCIDE against #Bia...	not-relevant
3084	@britishmuseum Experience #battlewaterloo from...	nocode

3085 rows × 2 columns

Figure 8

5.2 Data Cleaning

In this step, we have removed some irrelevant data from our dataset which is creating noise and unrequired for the effective implementation of algorithms.

For Example, in our dataset collected from twitter-tweets, the feature “status” had values “nocode” and “not-relevant” that are basically referred as “null Values” in terms of analysis.

Null Values:

```
[ ] data.status.value_counts()

nocode          1572
happy           1137
not-relevant     214
angry            57
surprise         35
sad              32
happy|surprise   11
happy|sad        9
disgust|angry    7
disgust          6
sad|angry        2
sad|disgust      2
sad|disgust|angry 1
Name: status, dtype: int64
```

Figure 9 Null values

Here, as shown in the above snippet, we had 1572 and 214 null values creating noise in our dataset.

Input:

```
data.drop(data[data['status']=='nocode'].index,inplace=True)
data.drop(data[data['status']=='not-relevant'].index,inplace=True)
data
```

Figure 10 dropping null values

Therefore, removing them and cleaning our data, dataset looks like as shown below with 1299 rows and 2 columns.

Output:

```
data.drop(data[data['status']=='nocode'].index,inplace=True)
data.drop(data[data['status']=='not-relevant'].index,inplace=True)
data
```

	tweet	status
1	Dorian Gray with Rainbow Scarf #LoveWins (from...	happy
2	@SelectShowcase @Tate_Stlves ... Replace with ...	happy
3	@Sofabsports thank you for following me back. ...	happy
4	@britishmuseum @TudorHistory What a beautiful ...	happy
5	@NationalGallery @ThePoldarkian I have always ...	happy
...
3076	Good to see @liveatlica's art collection @Leed...	happy
3077	@RAMMuseum thanks, we'll have a look next week...	happy
3079	"@britishmuseum: Thanks for ranking us #1 in @...	happy
3080	MT @AliHaggett: Looking forward to our public ...	happy
3082	@MrStuchbery @britishmuseum Mesmerising.	happy

1299 rows x 2 columns

Figure 11

After removing irrelevant Values

```
[4] data.status.value_counts()

happy          1137
angry           57
surprise       35
sad            32
happy|surprise  11
happy|sad       9
disgust|angry   7
disgust         6
sad|angry       2
sad|disgust     2
sad|disgust|angry 1
Name: status, dtype: int64
```

Figure 12

5.3 Data Visualization

We have visualised data using pie chart and bar graph for better understanding and getting the hold of our dataset.

Pie Chart:

In this visualization, data is represented as a share of an entire pie (circle), giving the percentage share of each feature as shown:

Input:

```
[ ] import matplotlib.pyplot as plt

data.status.value_counts().plot(kind='pie', autopct='%1.0f%%')

<matplotlib.axes._subplots.AxesSubplot at 0x7fe8b5540810>
```

Figure 13

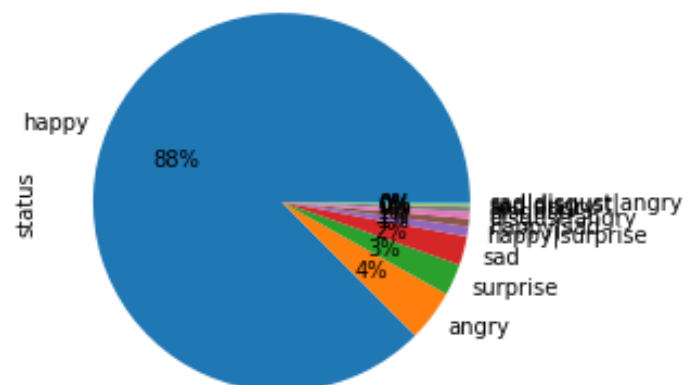
Output:

Figure 14

Bar Graph:

In this graph, Count v/s features is shown for better enhancement, height of the bar giving the number of values:

Input:

```
[ ] import matplotlib.pyplot as plt

data.status.value_counts().plot(kind='bar')

<matplotlib.axes._subplots.AxesSubplot at 0x7fe8b47e8c90>
```

Figure 15

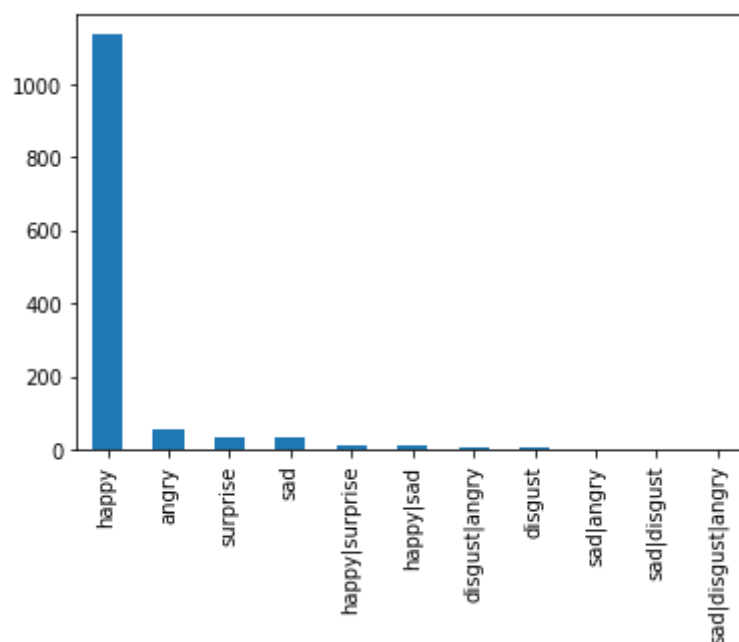
Output:

Figure 16 Bar graph

5.4 Data Processing

In this step, we have used the Natural Language Processing algorithm for converting text to numbers.

➤ Importing libraries:

Stopwords: It creates a collection of those words which are most frequently used and unimportant helping words like a, an, the, is etc. for the mentioned language. E.g., English here.

Tokenizer: It creates tokens of words from an entire sentence.

E.g.:- Today is sunny weather --- Today | is | sunny | weather

WordNet Lemmatizer: It converts the different forms of a single word into a single unit of words.

E.g.:- Doing, Do, Did, Done etc will be counted as a single unit of words.

Also, good, better and best will come under

```

import numpy as np
import pandas as pd
import re
import nltk
import matplotlib.pyplot as plt
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('stopwords')
stop_words=set(stopwords.words('english'))

[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!

```

Figure 17 Importing Libraries

➤ Processing Functions:

In this step, we have removed the unnecessary punctuation marks, special symbols, url inks, numbers from our text and capitalisation.

Input:

```

[8] def twitter_process(tweet):
    tweet = tweet.lower()
    #for removing numbers
    tweet = re.sub(r'\d+', '', tweet)
    #for removing urls
    tweet = re.sub(r"http\S+|https\S+", " ", tweet, flags = re.MULTILINE)
    #maintain the punctuation
    tweet = tweet.translate(str.maketrans(" ", " ", string.punctuation))
    #for removing special symbols
    tweet = re.sub(r'\@|\#|$', " ", tweet)
    tweet_tokens = word_tokenize(tweet)
    filtered_words = [word for word in tweet_tokens if word not in stop_words and len(word)>2]
    lemmatizer = WordNetLemmatizer()
    lemma_words = [lemmatizer.lemmatize(w, pos = 'a') for w in filtered_words]
    return " ".join(lemma_words)

data["tweet"] = data.tweet.apply(twitter_process)
data

```

Figure 18

Output:

	tweet	status
1	dorian gray rainbow scarf lovewins britishmuseum	happy
2	selectshowcase tatestives replace wish artist ...	happy
3	sofabsports thank following back great hear di...	happy
4	britishmuseum tudorhistory beautiful jewel por...	happy
5	nationalgallery thepoldarkian always loved pai...	happy
...
3076	good see liveatlicas art collection leedsartga...	happy
3077	rammuseum thanks well look next week friday done	happy
3079	britishmuseum thanks ranking tripadvisor thing...	happy
3080	alihaggett looking forward public engagement e...	happy
3082	mrstuchbery britishmuseum mesmerising	happy

1299 rows × 2 columns

Figure 19

➤ **Bag of Words Extraction:**

In this part, a bag of words for each document is created, and then a complete bag of the entire bag is created in a similar manner just like a vector irrespective of grammar and keeping all the words and their multiplicity intact in a bag.

```

tweets=data['tweet'].tolist()
from sklearn.feature_extraction.text import CountVectorizer
BOW_Vector=CountVectorizer(tweets)
print(BOW_Vector)

CountVectorizer(analyzer='word', binary=False, decode_error='strict',
dtype=<class 'numpy.int64'>, encoding='utf-8',
input=['dorian gray rainbow scarf lovewins britishmuseum',
'selectshowcase tatestives replace wish artist uses '
'next installation entralling',
'sofabsports thank following back great hear diverse '
'amp interesting panel defeatingdepression rammuseum',
'britishmuseu...
'castlehill kettlesyard explorearchives',
'week writing reports reports suddenly brightened '
'thought definingbeauty britishmuseum tomorrow '
'littlemissmoo',
'samba time britishmuseum camdentalking discoveryday', ...],
lowercase=True, max_df=1.0, max_features=None, min_df=1,
ngram_range=(1, 1), preprocessor=None, stop_words=None,
strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
tokenizer=None, vocabulary=None)

```

Figure 20

Input:

```

▶ BOW_Vector.fit(tweets)
  BOW_Vector.get_feature_names()
  #we will get bag of word

```

Figure 21

Output:

```

▶ ['aaroo',
  'abducted',
  'able',
  'aboard',
  'aboriginal',
  'aboutlondon',
  'abroadbrush',
  'absolutely',
  'absorbing',
  'abt',
  'abu',
  'accessibility',
  'accessible',
  'acclimatize',
  'accounts',
  'ace',
  'acenational',
  'acenterprises',
  'achieve',
  'achieved',
  'achievement',
  'acquire',
  'acquired',
  'acquires',
  'acropolismuseum',
  'act',
  'action',
  'activities',

```

Figure 22



Multiplicity of words in the entire dataset:

```

[23] array=BOW_Vector.transform(tweets).toarray()
      array

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

```

Figure 23

➤ **TFIDF Vectorization:**

It consists of two parts: - ‘term frequency’ and ‘inverse document frequency’. It focuses on the words which are more interesting. **Term Frequency:** frequency of how often a given word appears within a document. **Inverse Document Frequency:** frequency of words that appear a lot across documents

It is a score of how a given word is ‘frequent in a tweet wrt whole dataset’ in our case.

Input:

TF-IDF vectorization

```
[ ] from sklearn.feature_extraction.text import TfidfVectorizer

[ ] vectorizer = TfidfVectorizer (max_features=3000, min_df=6, max_df=0.75, stop_words=stopwords.words('english'))
    processed_features=vectorizer.fit_transform(tweets).toarray()
    processed_features
```

Figure 24

Output:

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

Figure 25

- Encoding: Here we have encoded target variables.
E.g:- Merging of happy|surprise label into happy one i.e. giving preference to the first word (word before |)

Before Encoding:

```
##make label likehappy|surprise as happy
data.status.value_counts()
```

```
happy          1137
angry           57
surprise        35
sad             32
happy|surprise  11
happy|sad        9
disgust|angry    7
disgust          6
sad|angry         2
sad|disgust       2
sad|disgust|angry 1
Name: status, dtype: int64
```

Figure 26

Encoding code:

```
[30] data=data.replace('happy|surprise','happy')
data=data.replace('happy|sad','happy')
data=data.replace('sad|disgust','sad')
data=data.replace('sad|disgust|angry','sad')
data=data.replace('sad|angry','sad')
data=data.replace('disgust|angry','sad')
data.status.value_counts()
status=data.iloc[:,1].values
status

array(['happy', 'happy', 'happy', ..., 'happy', 'happy', 'happy'],
      dtype=object)
```

Figure 27

After Encoding:

```
data.status.value_counts()

happy      1157
angry       57
sad         44
surprise    35
disgust      6
Name: status, dtype: int64
```

Figure 28

5.5 Dataset Splitting

Here, we have split our dataset into training and testing set.
Testing set size: 20%

Input:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(processed_features, status, test_size=0.2, random_state=0)
X_train
```

Figure 29

Output:

```
array([[0., 0., 0., ..., 0., 0., ],
       [0., 0., 0., ..., 0., 0., ],
       [0., 0., 0., ..., 0., 0., ],
       ...,
       [0., 0., 0., ..., 0., 0., ],
       [0., 0., 0.44888323, ..., 0., 0., ],
       [0., 0., 0., ..., 0., 0., ],
       [0., 0., 0., ..., 0., 0., ]])
```

Figure 30

Output:

```
array(['happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'sad',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'surprise', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'sad', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'angry', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'angry', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'angry', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'surprise',
      'angry', 'happy', 'happy', 'angry', 'happy', 'happy', 'happy',
      'angry', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'disgust',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'angry',
      'angry', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'angry', 'angry', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'sad', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'angry', 'angry', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'angry', 'happy', 'happy', 'happy', 'sad',
```

Figure 34

➤ Gini Index:**Input:**

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
dt1=DecisionTreeClassifier(criterion='gini',max_depth=5)
dt1.fit(X_train,y_train)
predictions=dt1.predict(X_test)
predictions
```

Figure 35

Output:

[illegible]

Figure 36

➤ **KNN:**

Input:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=20)
knn.fit(X_train, y_train)

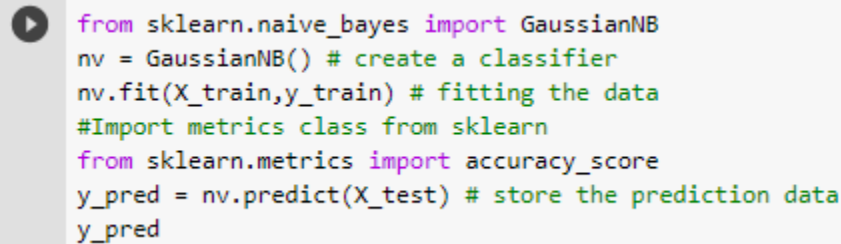
# Predict on dataset which model has not seen before
print(knn.predict(X_test))
```

Figure 37

Output:

[illegible]

Figure 38

➤ **Naive Bayes:****Input:**


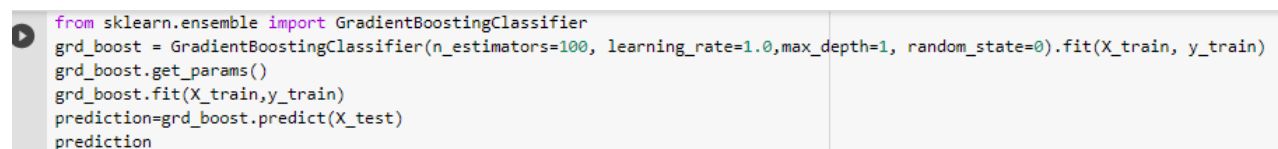
```
from sklearn.naive_bayes import GaussianNB
nv = GaussianNB() # create a classifier
nv.fit(X_train,y_train) # fitting the data
#Import metrics class from sklearn
from sklearn.metrics import accuracy_score
y_pred = nv.predict(X_test) # store the prediction data
y_pred
```

Figure 39

Output:


```
array(['happy', 'sad', 'happy', 'happy', 'happy', 'happy', 'happy',
       'happy', 'surprise', 'happy', 'disgust', 'happy', 'happy', 'happy',
       'happy', 'happy', 'happy', 'disgust', 'happy', 'happy', 'happy',
       'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'sad',
       'happy', 'happy', 'happy', 'happy', 'disgust', 'disgust', 'happy',
       'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
       'happy', 'happy', 'happy', 'angry', 'happy', 'happy', 'disgust',
       'disgust', 'sad', 'happy', 'happy', 'happy', 'happy', 'happy',
       'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
       'happy', 'happy', 'happy', 'happy', 'happy', 'angry', 'disgust',
       'happy', 'angry', 'happy', 'happy', 'happy', 'sad', 'disgust', 'happy',
       'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
       'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
       'happy', 'happy', 'disgust', 'happy', 'angry', 'happy', 'happy',
       'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
       'happy', 'happy', 'happy', 'angry', 'happy', 'happy', 'sad',
       'happy', 'happy', 'happy', 'happy', 'angry', 'happy', 'disgust',
       'sad', 'happy', 'sad', 'happy', 'happy', 'happy', 'happy', 'happy',
       'happy', 'happy', 'happy', 'happy', 'happy', 'sad', 'happy',
       'happy', 'happy', 'happy', 'happy', 'surprise', 'happy', 'happy',
       'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy']
```

Figure 40

➤ **Gradient Boost:****Input:**


```
from sklearn.ensemble import GradientBoostingClassifier
grd_boost = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,max_depth=1, random_state=0).fit(X_train, y_train)
grd_boost.get_params()
grd_boost.fit(X_train,y_train)
prediction=grd_boost.predict(X_test)
prediction
```

Figure 41

Output:

```

array(['happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'angry', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'angry', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'sad', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'disgust',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'angry', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'angry', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'sad', 'happy',
      'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
      'happy', 'angry', 'sad', 'happy', 'happy', 'happy', 'happy',

```

Figure 42

6. Results and Discussion**6.1 Confusion Matrix**

It is a tabular representation of prediction outcomes & classification results for the sake of visualizing outcome.

It identifies correct predictions of a model for different errors & classes.

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

Figure 43 Confusion Matrix

- **Random forest:** Confusion matrix for Random forest is as shown below. All the performance measures like Precision, recall, f-score and support are as shown

	[[6 0 8 1 0]				
	[0 0 1 0 0]				
	[0 0 222 1 0]				
	[0 0 6 0 0]				
	[0 1 14 0 0]]				
	precision	recall	f1-score	support	
angry	1.00	0.40	0.57	15	
disgust	0.00	0.00	0.00	1	
happy	0.88	1.00	0.94	223	
sad	0.00	0.00	0.00	6	
surprise	0.00	0.00	0.00	15	
accuracy			0.88	260	
macro avg	0.38	0.28	0.30	260	
weighted avg	0.82	0.88	0.84	260	

0.8769230769230769

Figure 44

- **Information gain:** Confusion matrix for Information Gain is as shown below. All the performance measures like Precision, recall, f-score and support are as shown

	[[8 0 7 0 0]				
	[0 0 1 0 0]				
	[7 0 209 6 1]				
	[0 0 6 0 0]				
	[0 1 14 0 0]]				
	precision	recall	f1-score	support	
angry	0.53	0.53	0.53	15	
disgust	0.00	0.00	0.00	1	
happy	0.88	0.94	0.91	223	
sad	0.00	0.00	0.00	6	
surprise	0.00	0.00	0.00	15	
accuracy			0.83	260	
macro avg	0.28	0.29	0.29	260	
weighted avg	0.79	0.83	0.81	260	

0.8346153846153846

Figure 45

- **Gini index:** Confusion matrix for Gini Index is as shown below. All the performance measures like Precision, recall, f-score and support are as shown

	[[6 0 9 0 0]				
	[0 0 1 0 0]				
	[1 0 221 1 0]				
	[0 0 6 0 0]				
	[0 0 15 0 0]]				
	precision	recall	f1-score	support	
angry	0.86	0.40	0.55	15	
disgust	0.00	0.00	0.00	1	
happy	0.88	0.99	0.93	223	
sad	0.00	0.00	0.00	6	
surprise	0.00	0.00	0.00	15	
accuracy			0.87	260	
macro avg	0.35	0.28	0.30	260	
weighted avg	0.80	0.87	0.83	260	

0.8730769230769231

Figure 46

- **KNN:** Confusion matrix for KNN is as shown below. All the performance measures like Precision, recall, f-score and support are as shown

[[7 0 8 0 0]				
[0 0 1 0 0]				
[0 0 223 0 0]				
[0 0 6 0 0]				
[0 0 15 0 0]]				
	precision	recall	f1-score	support
angry	1.00	0.47	0.64	15
disgust	0.00	0.00	0.00	1
happy	0.88	1.00	0.94	223
sad	0.00	0.00	0.00	6
surprise	0.00	0.00	0.00	15
accuracy			0.88	260
macro avg	0.38	0.29	0.31	260
weighted avg	0.81	0.88	0.84	260

0.8846153846153846

Figure 47

- **Naive Bayes:** Confusion matrix for Naive Bayes is as shown below. All the performance measures like Precision, recall, f-score and support are as shown

[[7 1 6 1 0]				
[0 1 0 0 0]				
[5 7 201 9 1]				
[0 1 4 1 0]				
[0 4 9 0 2]]				
	precision	recall	f1-score	support
angry	0.58	0.47	0.52	15
disgust	0.07	1.00	0.13	1
happy	0.91	0.90	0.91	223
sad	0.09	0.17	0.12	6
surprise	0.67	0.13	0.22	15
accuracy			0.82	260
macro avg	0.47	0.53	0.38	260
weighted avg	0.86	0.82	0.82	260

0.8153846153846154

Figure 48

- **Gradient Boost:** Confusion matrix for Gradient Boost is as shown below. All the performance measures like Precision, recall, f-score and support are as shown

	precision	recall	f1-score	support
angry	0.67	0.27	0.38	15
disgust	0.00	0.00	0.00	1
happy	0.87	0.96	0.92	223
sad	0.00	0.00	0.00	6
surprise	0.00	0.00	0.00	15
accuracy			0.84	260
macro avg	0.31	0.25	0.26	260
weighted avg	0.79	0.84	0.81	260

Figure 49

6.2 Performance Metrics Comparisons

It plays an important role in organisation of data and selecting the chief performance metrics is of utmost importance and focusing on these areas because these metrics bring about certain changes in hitting the bull's eye

Performance of implemented algorithms is given by:

S.No.	Algorithm	Accuracy
1.	Information Gain	84%
2.	Gini Index	87%
3.	Naive Bayes	81%
4.	KNN	88%
5.	Random Forest	87%
6.	Gradient Boost	85%

Comparison of classification algorithms in terms of their accuracy using visualisation:

	Name	Accuracy
3	KNeighborsClassifier	0.881
1	RandomForestClassifier	0.877
4	DecisionTreeClassifier	0.869
0	GradientBoostingClassifier	0.858
5	DecisionTreeClassifier	0.846
2	GaussianNB	0.815

Figure 51

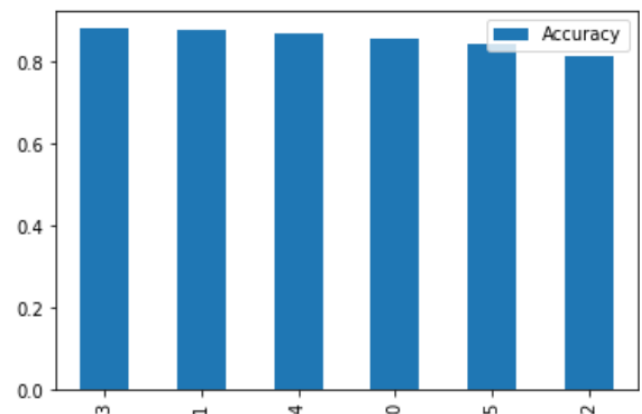


Figure 50



From the above plotting of accuracy of algorithms in the descending order, we can clearly see that **KNN Classifier** works **best** for the “smile” dataset or sentiment analysis of the tweets collected from twitter whereas **Naïve Bayes** works the **worst** of them all for it.

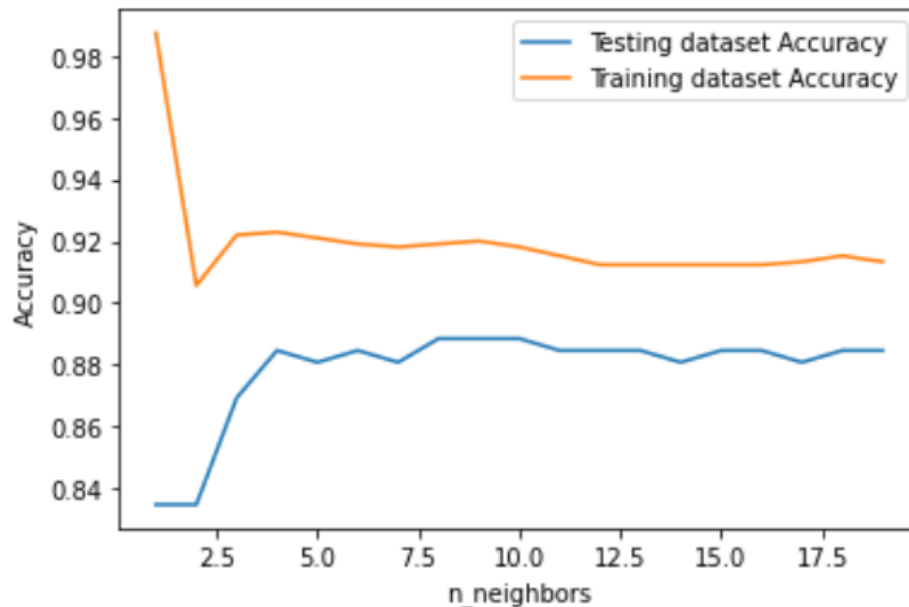
➤ **KNN:**

Figure 54 KNN plotting of accuracy v/s n_neighbours

7. Conclusion & Future Scope



Conclusion

After comparing the performance of different classifiers, we observe that KNN has the highest accuracy (88%) as compared to others for analysing '*HIDDEN EMOTIONS*'. The performance is calculated on the basis of classification report & confusion matrix.

Thus, for the available 'SMILE' dataset we can predict the emotions associated with the data by training out model with KNN classifier.



Future Scope

We can use better techniques available, that could provide better accuracy.

Better results of our model can also be obtained by using better defined dataset. Currently we have irregular dataset, as we can say it is more inclined towards "happy" label and therefore more efficient model can be built for regular dataset.

Some available systems are like Emotex, that was built using Twitter's emotion-labelled texts and the SVM, NB & DT classifiers, scope of improvements are available in them.