**1.1 Introduction**

Opinion mining or Sentiment Analysis refers to the use of [natural language processing](https://en.wikipedia.org/wiki/Natural_language_processing), [text analysis](https://en.wikipedia.org/wiki/Text_analytics) and [computational linguistics](https://en.wikipedia.org/wiki/Computational_linguistics) to systematically identify, extract, quantify, and study affective states and subjective information. Sentiment analysis is widely applied to [voice of the customer](https://en.wikipedia.org/wiki/Voice_of_the_customer) materials such as reviews and survey responses, online and social media, and healthcare materials for applications that range from [marketing](https://en.wikipedia.org/wiki/Marketing) to [customer service](https://en.wikipedia.org/wiki/Customer_relationship_management) to clinical medicine.

Sentiment analysis aims to determine the attitude of a speaker, writer, or other subject with respect to some topic or the overall contextual polarity or emotional reaction to a document, interaction, or event. The attitude may be a judgment or evaluation, affective state, or the intended emotional communication. It can be considered as a classification process.

Nowadays when the microblogging platforms, such as Twitter, are commonly used, the task of sentiment analysis becomes even more interesting. Microblogging introduces a new way of communication, where people are forced to use short texts to deliver their messages, hence containing new acronyms, abbreviations, and grammatical mistakes that were generated intentionally. Although there are several known tasks related to sentiment analysis, in this project we will focus on the common problem of identifying the sentiments broadly classified into 3 categories. These are: happy, sad & neutral.

**1.2 Dataset**

The data set that we have used contains labels for the emotional content such as happiness, sadness etc of texts.

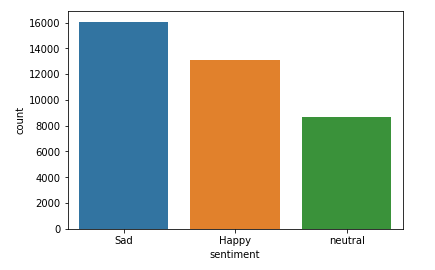


Fig.1 Plot showing the number of tweets of different sentiments in dataset

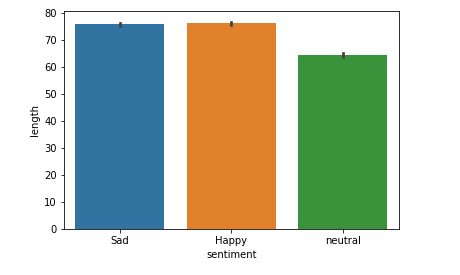


Fig.2 Plot showing the length of tweets of different sentiments

**1.3 Classifier**

The classifier that we have used is Naïve Bayes Classifier. The Naive Bayes classifier is based on Bayes’ theorem with independence assumptions between predictors. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naive Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods. Naive Bayes is a classification algorithm for binary (two-class) and multi-class classification problems. The technique is easiest to understand when described using binary or categorical input values. This method comes under the category of supervised learning. Supervised learning is the machine learning task of inferring a function from labelled training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object and a desired output value.

**1.4 Environment**

The entire program is written in Python. And various libraries are used. These are:

1.4.1 Libraries Used

* NumPy: NumPy is an acronym for "Numeric Python" or "Numerical Python". It is an open source extension module for Python, which provides fast precompiled functions for mathematical and numerical routines. Furthermore, NumPy enriches the programming language Python with powerful data structures for efficient computation of multi-dimensional arrays and matrices. The implementation is even aiming at huge matrices and arrays. Besides that the module supplies a large library of high-level mathematical functions to operate on these matrices and arrays.
* Matplotlib: Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and [IPython](http://ipython.org/) shell, the [jupyter](http://jupyter.org/index.html) notebook, web application servers. Matplotlib can be used to generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc.
* Pandas: **pandas** is a Python package providing fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real world** data analysis in Python.
* Seaborn: Seaborn is a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics.
* Pipeline: There are standard workflows in a machine learning project that can be automated. In Python scikit-learn, Pipelines help to clearly define and automate these workflows. Pipelines work by allowing for a linear sequence of data transforms to be chained together culminating in a modelling process that can be evaluated.

**1.5 Process of analysing the dataset**

1. Preprocessing: The dataset originally contains 13 different sentiments. But we have segregated the tweets only in 3 categories i.e. happy, sad and neutral & thereby reducing the emotion classes. Also in this step stopwords are removed, unnecessary punctuation marks are removed from the tweets .
2. Feature Extraction: In this step the CountVectorizer converts the collection of text into a matrix of token counts. This description can be vectorised into a sparse two-dimensional matrix suitable for feeding into a classifier (maybe after being piped into a [text.TfidfTransformer](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html" \l "sklearn.feature_extraction.text.TfidfTransformer" \o "sklearn.feature_extraction.text.TfidfTransformer) for normalization).
3. Analysis by the classifer: The vectorised form obtained after feature extraction is fed into the classifier and the precision is evaluated.

**1.6 Source Code of Sentiment Analyser**

import numpy as np

import seaborn as sns

import matplotlib as plt

import pandas as pd

import re

import string

from nltk.corpus import stopwords

%matplotlib inline

# Reducing the number of emotion classes

df = pd.read\_csv('text\_emotion.csv',index\_col=0)

df.loc[(df['sentiment']=='enthusiasm') | (df['sentiment']=='fun') | (df['sentiment']=='love') | (df['sentiment']=='happiness') | (df['sentiment']=='relief'),'sentiment'] = 'Happy'

df.loc[(df['sentiment']=='sadness') | (df['sentiment']=='empty') | (df['sentiment']=='hate') | (df['sentiment']=='anger')|(df['sentiment']=='worry') | (df['sentiment']=='boredom'),'sentiment'] ='Sad'

df.drop(df[df.sentiment=='surprise'].index,inplace = True)

sns.countplot(x='sentiment',data=df)

df['length'] = df['content'].str.len()

sns.barplot(x='sentiment',y='length',data=df)

sns.stripplot(x='sentiment',y='length',data=df,jitter=True)

# stop = stopwords.words('english')

# replaced\_words=['AT\_USER','URL']

# stop.extend(replaced\_words)

stop=['AT\_USER','URL']

def text\_process(tweet):

# process the tweets

#Convert to lower case

tweet = tweet.lower()

#Convert www.\* or https?://\* to URL

tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','URL',tweet)

#Convert @username to AT\_USER

tweet = re.sub('@[^\s]+','AT\_USER',tweet)

#Remove additional white spaces

tweet = re.sub('[\s]+', ' ', tweet)

#Replace #word with word

tweet = re.sub(r'#([^\s]+)', r'\1', tweet)

#trim

tweet = tweet.strip('\'"')

return [word for word in tweet.split() if word not in stop]

df['content'].head(10).apply(text\_process)

from sklearn.feature\_extraction.text import CountVectorizer #converts bag of words into vector

from sklearn.feature\_extraction.text import TfidfTransformer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.svm import SVC

from sklearn.cross\_validation import train\_test\_split

tweet\_train,tweet\_test,label\_train,label\_test=train\_test\_split(df['content'],df['sentiment'],test\_size=0.3)

from sklearn.pipeline import Pipeline

pipeline=Pipeline([('bow',CountVectorizer(analyzer=text\_process)),('tfidf',TfidfTransformer()),('classifier',MultinomialNB()),])

pipeline.fit(tweet\_train,label\_train)

predictions = pipeline.predict(tweet\_test)

from sklearn.metrics import classification\_report

print(classification\_report(label\_test,predictions))

**1.6 Experimental results and analysis**

Using Naïve Bayes Classifier we achieved the precision of 0.58 or 58%.

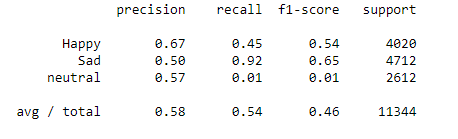


Fig.3 The results as achieved by using Naïve Bayes Classifier

**1.7 Other approaches**

Other than Naïve Bayes classifier, there are approaches like KNN and SVM classifiers. KNN is a non-parametric lazy learning algorithm. It is non parametric, it means that it does not make any assumptions on the underlying data distribution. It is also a lazy algorithm. What this means is that it does not use the training data points to do any generalization. In other words, there is no explicit training phaseor it is very minimal. This means the training phase is pretty fast.

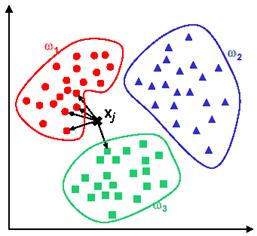


Fig. A plot showing graphic implementation of KNN classifier

Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well .

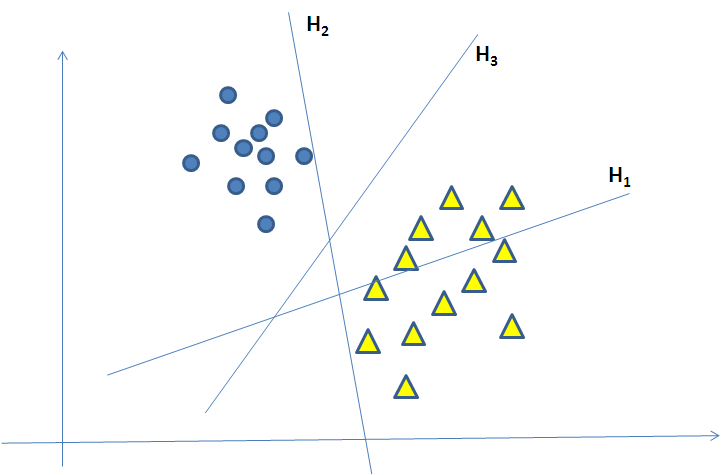


Fig. A plot showing hyper-planes (SVM)