

# SENTIMENT ANALYSIS

Sentiment Analysis is a text analysis method that detects polarity e.g. a positive or negative opinion within the text, whether a whole document, paragraph or a sentence

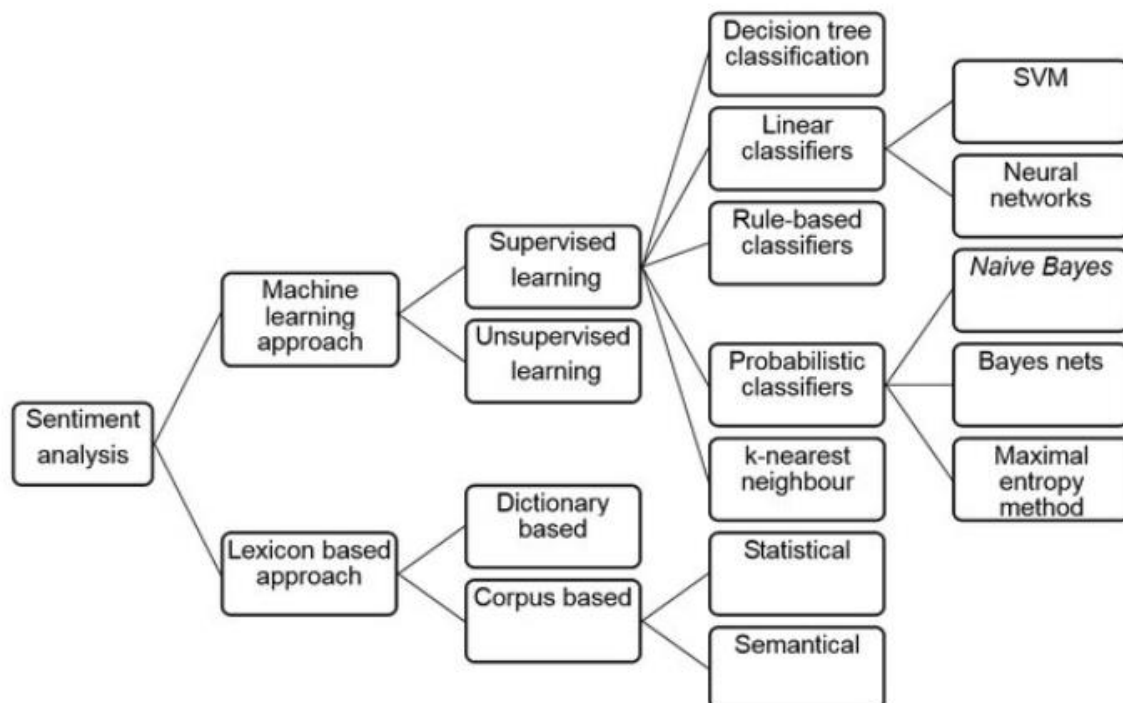
## Types of Sentiment Analysis

1. Fine-Grained Sentiment Analysis
  - a. Very positive
  - b. Positive
  - c. Neutral
  - d. Negative
  - e. Very Negative
2. Emotion Detection
  - a. Detecting emotions like happiness frustration, anger, sadness and so on.
  - b. Many emotion detection systems use lexicons
3. Aspect-based Sentiment Analysis
  - a. If one wants to know which aspects or feature of a product.
    - i. People are mentioning in a positive, neutral or negative way.
4. Multilingual sentiment analysis
  - a. Detecting the language in texts automatically.

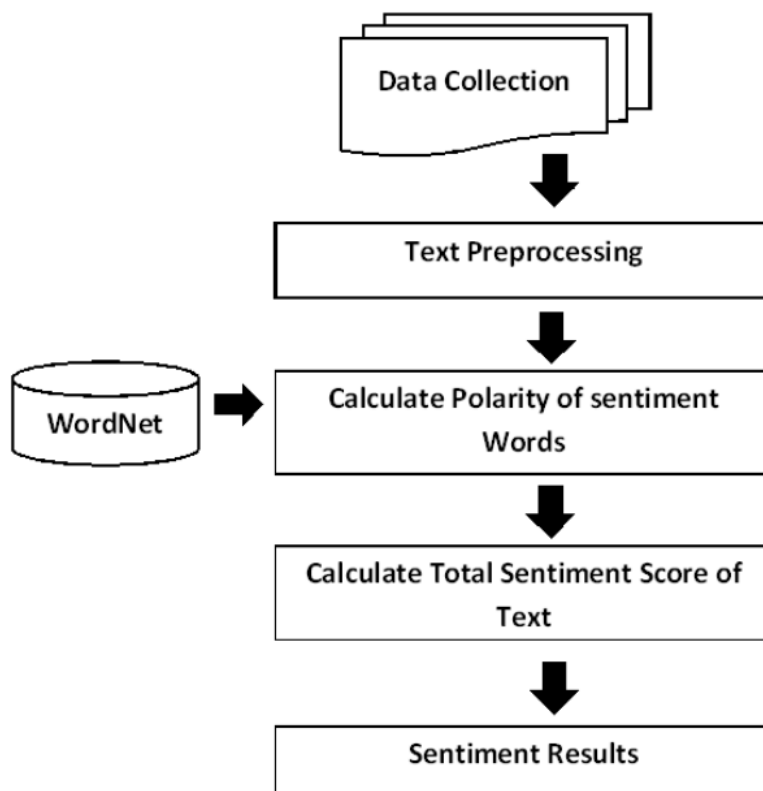
## Techniques and Algorithms

There are **three** type of sentiment analysis

1. **Rule based** – manually crafted rules
2. **Automatic** – rely on machine learning
3. **Hybrid** – combination of both



## 1) Lexicon Based Approach for Sentiment Analysis



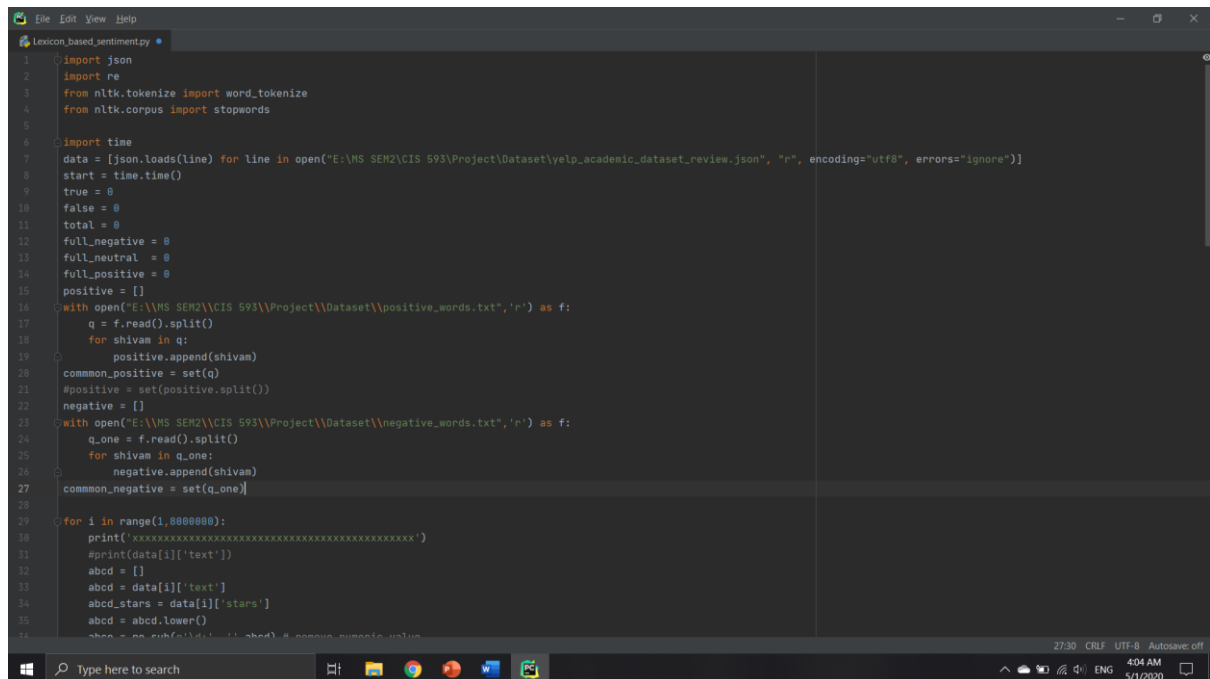
To do the lexical analysis of the data, first you must go through each NLP concepts i.e. removing of stop words, removing of white space etc.

After that you must do the text processing in which you have to tokenize the words in the review/text

After tokenization, you must check all the tokenized word to your word dictionary, in our case we have used the words which were provided by **Dr. Sunnie Chung** in our class website, there were 2 text file positive and negative words.

Now after doing all the checks, the last step is to check whether the text is positive or negative, by checking the words frequency if the positive words are more than that of negative words then we can make a judgement that the text is positive/negative.

## Screenshot of the screen for Lexicon based Sentiment Analysis



```
1 import json
2 import re
3 from nltk.tokenize import word_tokenize
4 from nltk.corpus import stopwords
5
6 import time
7 data = [json.loads(line) for line in open("E:\\MS SEM2\\CIS 593\\Project\\Dataset\\yelp_academic_dataset_review.json", "r", encoding="utf8", errors="ignore")]
8 start = time.time()
9 true = 0
10 false = 0
11 total = 0
12 full_negative = 0
13 full_neutral = 0
14 full_positive = 0
15 positive = []
16 with open("E:\\MS SEM2\\CIS 593\\Project\\Dataset\\positive_words.txt", "r") as f:
17     q = f.read().split()
18     for shivam in q:
19         positive.append(shivam)
20     common_positive = set(q)
21     #positive = set(positive.split())
22     negative = []
23 with open("E:\\MS SEM2\\CIS 593\\Project\\Dataset\\negative_words.txt", "r") as f:
24     q_one = f.read().split()
25     for shivam in q_one:
26         negative.append(shivam)
27     common_negative = set(q_one)
28
29 for i in range(1,800000):
30     print('xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx')
31     #print(data[i]['text'])
32     abcd = []
33     abcd = data[i]['text']
34     abcd_stars = data[i]['stars']
35     abcd = abcd.lower()
36     abcd = re.sub('[^a-zA-Z]', '', abcd) # remove numeric value
```

In this first we have imported the basic libraries so that we can further clean the generate text which will go under tokenization.

Apart from that we have loaded our 6GB JSON file and along with it we have loaded the positive and negative words.

After that we tried to get the 8Million data at once and did our lexicon analysis on all the data

## CODE for Lexicon based sentiment analysis

```
import json
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

data = [json.loads(line) for line in open("E:\MS SEM2\CIS
593\Project\Dataset\yelp_academic_dataset_review.json", "r",
encoding="utf8", errors="ignore")]

true = 0
false = 0
total = 0

full_negative = 0
full_neutral = 0
full_positive = 0

positive = []
with open("E:\\MS SEM2\\CIS 593\\Project\\Dataset\\positive_words.txt",'r')
as f:
    q = f.read().split()
    for shivam in q:
        positive.append(shivam)
        common_positive = set(q)

negative = []
with open("E:\\MS SEM2\\CIS 593\\Project\\Dataset\\negative_words.txt",'r')
as f:
    q_one = f.read().split()
    for shivam in q_one:
        negative.append(shivam)
        common_negative = set(q_one)

for i in range(1,10000):
    abcd = []
    abcd = data[i]['text']
    abcd_stars = data[i]['stars']
    abcd = abcd.lower()
    abce = re.sub(r'\d+', '', abcd) # remove numeric value
    abcd = abcd.strip() # removed whitespace
    abcd = re.sub(r'[^w\s]', '', abcd) #removed punctuations

    stop_words = set(stopwords.words('english')) # to remove english stop
words
    tokens = word_tokenize(abcd) #tokenizing whole file
    result = [i for i in tokens if not i in stop_words] # for stop words
removal
    check_one = set(result)

    positive_test = common_positive & check_one
    negative_test = common_negative & check_one

    a = 0
    for w1 in positive_test:
        word1 = result.count(w1)
        a = a + word1
```

```

b = 0
for w2 in negative_test:
    word2 = result.count(w2)
    b = b + word2

if b > a:
    #print("Review is Negative")
    #print("Real Rating",abcd_stars)
    full_negative = full_negative + 1
    if(abcd_stars == 1 or abcd_stars == 2):
        true = true + 1
        total = total + 1
    else:
        false =false + 1
        total = total + 1
elif a == b:
    #print("Review is Neutral")
    #print("Real Rating",abcd_stars)
    full_neutral = full_neutral + 1
    if(abcd_stars == 3):
        true = true + 1
        total = total + 1
    else:
        false =false + 1
        total = total + 1
else:
    #print("Review is Positive")
    #print("Real Rating",abcd_stars)
    full_positive = full_positive + 1
    if(abcd_stars == 4 or abcd_stars == 5):
        true = true + 1
        total = total + 1
    else:
        false =false + 1
        total = total + 1

print("True Positives- ",true)
print("False Positives- ",false)
print("Total Test Cases- ",total)
print("Accuracy- ",true/total)

```

## OUTPUT

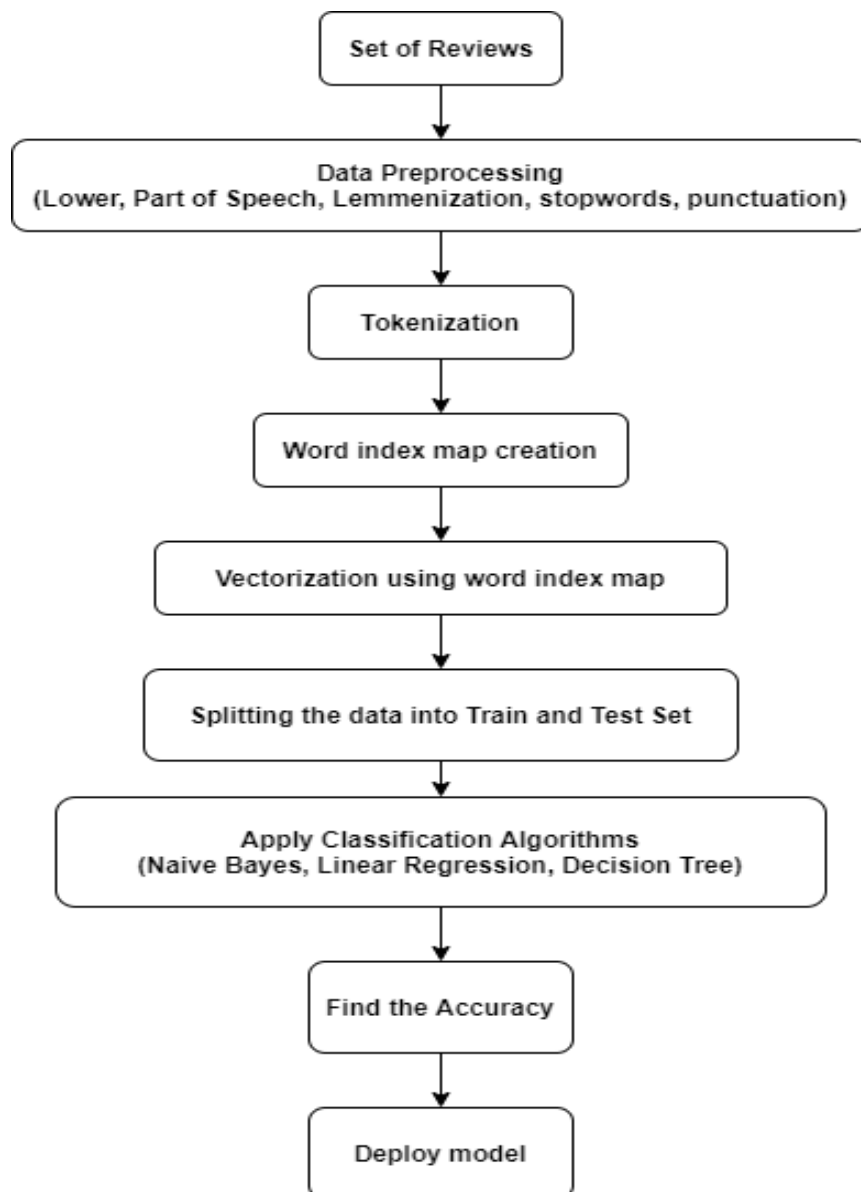
```

In [2]: runfile('E:/MS SEM2/CIS 593/Project/Code_sentiment_analysis/
Lexicon_based_sentiment.py', wdir='E:/MS SEM2/CIS 593/Project/
Code_sentiment_analysis')
True Positives- 7364
False Positives- 2635
Total Test Cases- 9999
Accuracy- 0.7364736473647365

```

This output is generated on initial 1-10000 data.

## 2) Machine Learning based Approach for Sentiment Analysis



Steps:

- In this method, we considered reviews as positive if it has 5 stars and reviews as negatives if it has star 1 or 2.
- Once we have positive reviews as well as negative review text in a separate list. Then our job is to create dictionary named `word_index_map`.

- To create word index map, each review should be tokenized in words and then it should be inserted into the word count dictionary where key/index is word and value is its frequency count.
- Before creation of the word count dictionary, each review should be tokenized and applied all the NLP preprocessing techniques like POS tagging, lemmatization, stop words removal, punctuation removal etc.
- Now its time to create vector matrix using the word\_index\_map which means if a word is positive then its row would have a rank value based on its occurrences.
- Once vectorization is done. We divided the data into Train and test set to apply classification algorithm

We have implemented four different ML based algorithm

1. Logistic Regression
2. SVC
3. Decision Tree
4. Naïve Bayes

## CODE for ML Approach

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()

#read the csv file and print the shape of the dataset
yelp = pd.read_csv("E:\MS SEM2\CIS 593\Project\Dataset\yelp.csv")
yelp.shape

#drop all the rows with NaN
yelp = yelp.dropna()
yelp.shape

#add one more column to find out the relation between length vs number of
reviews vs stars
yelp['text_len'] = yelp['text'].apply(len)
g = sns.FacetGrid(data=yelp, col='stars')
g.map(plt.hist, 'text_len', bins=50)

#This shuffles the data which help us to get the random tuple each time
from sklearn.utils import shuffle
yelp = shuffle(yelp)
```



```

#Defined two list for getting the postitives and negative reviews
positive_reviews = []
negative_reviews = []

#Get the 1600 positive reviews
p_counter = 0
index = 0
while p_counter != 1600:
    st = yelp['stars'][index]
    if (st == 5):
        positive_reviews.append(yelp['text'][index])
        p_counter = p_counter + 1
    index = index + 1
print(len(positive_reviews))

#Get the 1600 negative reviews
n_counter = 0
index = 0
while n_counter != 1600:
    st = yelp['stars'][index]
    if (st == 1 or st == 2):
        negative_reviews.append(yelp['text'][index])
        n_counter = n_counter + 1
    index = index + 1
print(len(negative_reviews))

#Tokenizer which tokenise the text into works and clean the data to prepare
for NLP
def my_tokenizer(text):
    text = text.lower() # downcase
    tokens = nltk.tokenize.word_tokenize(text) # split string into words
    (tokens)
    tokens = [t for t in tokens if len(t) > 2] # remove short words,
they're probably not useful (punctuation)
    tokens = [wordnet_lemmatizer.lemmatize(t) for t in tokens] # put words
into base form
    tokens = [t for t in tokens if t not in stopwords.words('english')] #
remove stopwords
    return tokens

#Dictionary and list defination for further use
word_index_map = {}
current_index = 0
positive_tokenized = []
negative_tokenized = []
orig_reviews = []

#Create dictionary for word count --> word_count_map from positive and
negative reviews
for review in positive_reviews:
    orig_reviews.append(review)
    tokens = my_tokenizer(review)
    positive_tokenized.append(tokens)
    for token in tokens:
        if token not in word_index_map:
            word_index_map[token] = current_index
            current_index += 1

for review in negative_reviews:
    orig_reviews.append(review)
    tokens = my_tokenizer(review)

```

```

        negative_tokenized.append(tokens)
    for token in tokens:
        if token not in word_index_map:
            word_index_map[token] = current_index
            current_index += 1

print("len(word_index_map):", len(word_index_map))

# Creation our input matrices by converting tokens to vector
def tokens_to_vector(tokens, label):
    x = np.zeros(len(word_index_map) + 1) # last element is for the label
    for t in tokens:
        i = word_index_map[t]
        x[i] += 1
    x = x / x.sum() # normalize it before setting label
    x[-1] = label
    return x

N = len(positive_tokenized) + len(negative_tokenized)
# (N x D+1 matrix - keeping them together for now so we can shuffle more
easily later
data = np.zeros((N, len(word_index_map) + 1))
i = 0
for tokens in positive_tokenized:
    xy = tokens_to_vector(tokens, 1)
    data[i,:] = xy
    i += 1

for tokens in negative_tokenized:
    xy = tokens_to_vector(tokens, 0)
    data[i,:] = xy
    i += 1

#train and test feature selection
X = data[:, :-1]
Y = data[:, -1]

#Apply machine learning algorithms for data mining
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.model_selection import train_test_split

#Classification using Logistic Regression
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(multi_class='multinomial', tol=1e-2,
solver='newton-cg', max_iter=15)
lr.fit(X_train, y_train)
y_predict = lr.predict(X_test)
print("\nClassification Algorithm - LOGISTIC REGRESSION")
print("Accuracy      - ", accuracy_score(y_test, y_predict))
print("Precesion     - ", precision_score(y_test, y_predict,
average='weighted'))

#Classification using SUPPORT VECTOR MACHINE
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)
from sklearn import svm
clf = svm.SVC(gamma=2, C=2)
clf.fit(X_train, y_train)

```

```

y_predict = clf.predict(X_test)
print("\nClassification Algotithm - SVC")
print("Accuracy      - ", accuracy_score(y_test, y_predict))
print("Precesion     - ", precision_score(y_test, y_predict,
average='weighted'))

#Classification using DECISION TREE
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)
from sklearn import tree
dt = tree.DecisionTreeClassifier(max_depth=25)
dt.fit(X_train, y_train)
y_predict = dt.predict(X_test)
print("\nClassification Algotithm - DECISION TREE")
print("Accuracy      - ", accuracy_score(y_test, y_predict))
print("Precesion     - ", precision_score(y_test, y_predict,
average='weighted'))

#Classification using NAIVE BAYES
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()
nb.fit(X_train, y_train)
y_predict = nb.predict(X_test)
print("\nClassification Algotithm - NAIVE BAYES")
print("Accuracy      - ", accuracy_score(y_test, y_predict))
print("Precesion     - ", precision_score(y_test, y_predict,
average='weighted'))

```

## Output for ML Based Approach

```

In [1]: runfile('E:/MS SEM2/CIS 593/Project/Code_sentiment_analysis/
machine_lerning_based_sentiment.py', wdir='E:/MS SEM2/CIS 593/Project/
Code_sentiment_analysis')
1600
1600
len(word_index_map): 17657

Classification Algotithm - LOGISTIC REGRESSION
Accuracy      -  0.796875
Precesion     -  0.7975804858294293

Classification Algotithm - SVC
Accuracy      -  0.8604166666666667
Precesion     -  0.8696435081443334

Classification Algotithm - DECISION TREE
Accuracy      -  0.7427083333333333
Precesion     -  0.7430256321874076

Classification Algotithm - NAIVE BAYES
Accuracy      -  0.8677083333333333
Precesion     -  0.8682962097306977

```

# TEXT SUMMARIZATION

Summarization can be defined as a task of producing a concise and fluent summary while preserving key information and overall meaning.

There are two kind of summarization:

- 1) Abstractive
- 2) Extractive

In this project we have used the Extractive Summarization.

It involves around the selection of words/phrases and sentences from the source document itself to make up the summary. There are different algorithm and techniques are used to define weights for the sentences and further rank them based on importance and similarity among each other.

To do Extractive Summarization: -

- 1) Input Document
- 2) Sentence Similarity/Ranking/Weight Sentences
- 3) Select sentences with higher rank

Automatic text summarization is a common problem in machine learning and natural language processing (NLP).

NLTK has been called “a wonderful tool for teaching and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

Requirements:

1. Python
2. NLTK Library of Python
3. IDE
4. Import all other necessities libraries
  - Pandas, Numpy

## Techniques and Algorithm

The techniques and algorithm used by us for text summarization are as follows:-

- 1) Sentence scoring using **Cosine Similarity** between sentences.
  - a. Steps
    - i. Input Article
    - ii. Split into sentences
    - iii. Remove stop words
    - iv. Build a similarity matrix
    - v. Generate rank based on matrix
    - vi. Pick top sentences for summary
  
- 2) Sentence Scoring based on **Word Frequency** in a sentence
  - a. Steps
    - i. Input article
    - ii. Remove stop words
    - iii. Create dictionary for word frequency
    - iv. Tokenize into the sentences
    - v. Generate rank based on term frequency
    - vi. Find threshold
    - vii. Generate summary
  
- 3) Sentence scoring based on **TF-IDF** score of words in a sentence
  - a. Steps
    - i. Input article
    - ii. Split into sentences
    - iii. Removing stop words
    - iv. Build a frequency matrix of the words in each sentence
    - v. Create TF matrix
    - vi. Create documents per words matrix
    - vii. Calculate IDF matrix
    - viii. Calculate TF-IDF matrix
    - ix. Score the sentence based on TF-IDF score
    - x. Find Threshold
    - xi. Generate Summary.

Here are the detailed steps of Text Summarization based on Word Count: -

- 1) Create the word Frequency Table
    - a. In this table we are going to make a dictionary for the word frequency from the text, here text is our whole paragraph.
  - 2) Generate clean sentences:
    - a. In this we are going to clean the sentences by removing
      - i. Stop words
      - ii. Removing whitespace
      - iii. Removing Numeric values
      - iv. Other unnecessary characters
  - 3) Tokenize the Sentences
    - a. In tokenization, we are separating the paragraph into couple of sentences and after that we are separating those sentences into tokens.
  - 4) Score the sentences: Term frequency
    - a. It is a method to score each sentence using the frequency of the each word in that sentence.
  - 5) Find the threshold
    - a. To find the threshold we are considering the average score of the sentences as a threshold.
  - 6) Generate the summary
    - a. To generate the summary
      - i. If the sentence score is above threshold then we will put that sentence in the summary
      - ii. If the sentence score is less than threshold score, then we are going to drop that sentence
- The same process needs to be done if you want to rank the sentence based on TF-IDF, but variation is, after getting term frequency table, you need to find the IDF and multiply that matrix with Term frequency matrix.

- For the cosine similarity-based approach, you tokenize the document sentences and between the sentences, you find the similarity and create one matrix and on the bases of those numbers, one can generate the summary based on similarities.

## **CODES and OUTPUT (Text Summarization)**

### **Cosine Similarity**

#### **Code**

```
import json
data = [json.loads(line) for line in open("E:\MS SEM2\CIS
593\Project\Dataset\yelp_academic_dataset_review.json", "r",
encoding="utf8", errors="ignore")]

from nltk.corpus import stopwords
from nltk.cluster.util import cosine_distance
import numpy as np
import networkx as nx

#This function reads the whole article and splite it into the sentences.
def read_article(file):
    article = file.split(". ")
    sentences = []
    for sentence in article:
        sentences.append(sentence.replace("[^a-zA-Z]", " ").split(" "))
    sentences.pop()
    return sentences

#This function finds the similarity between all the combination of
sentences
def sentence_similarity(sent1, sent2, stopwords=None):
    if stopwords is None:
        stopwords = []

    sent1 = [w.lower() for w in sent1]
    sent2 = [w.lower() for w in sent2]
    all_words = list(set(sent1 + sent2))

    vector1 = [0] * len(all_words)
    vector2 = [0] * len(all_words)

    # build the vector for the first sentence
    for w in sent1:
        if w in stopwords:
            continue
        vector1[all_words.index(w)] += 1

    # build the vector for the second sentence
    for w in sent2:
        if w in stopwords:
            continue
        vector2[all_words.index(w)] += 1

    return 1 - cosine_distance(vector1, vector2)
```

```

#This function builds similarity matrix based on the above function -
sentence_similarity
def build_similarity_matrix(sentences, stop_words):
    # Create an empty similarity matrix
    similarity_matrix = np.zeros((len(sentences), len(sentences)))

    for idx1 in range(len(sentences)):
        for idx2 in range(len(sentences)):
            if idx1 == idx2: #ignore if both are same sentences
                continue
            similarity_matrix[idx1][idx2] =
sentence_similarity(sentences[idx1], sentences[idx2], stop_words)

    return similarity_matrix

#this is the main function which calls all other required function to
generate summary
def generate_summary(file_name, top_n):
    stop_words = stopwords.words('english')
    summarize_text = []

    sentences = read_article(file_name)
    sentence_similarity_matrix = build_similarity_matrix(sentences,
stop_words)
    sentence_similarity_graph =
nx.from_numpy_array(sentence_similarity_matrix)
    scores = nx.pagerank(sentence_similarity_graph)
    ranked_sentence = sorted(((scores[i],s) for i,s in
enumerate(sentences)), reverse=True)

    for i in range(top_n):
        summarize_text.append(" ".join(ranked_sentence[i][1]))

    print("Summarize Text: \n", ". ".join(summarize_text))

# Call the function here to generate the summary
generate_summary( data[5502]['text'], 6)

```

## Output

```

In [3]: runfile('E:/MS SEM2/CIS 593/Project/Code_text_sumemrization/
text_analysis_tdscience_cosine_similarity.py', wdir='E:/MS SEM2/CIS 593/
Project/Code_text_sumemrization')

```

Summarize Text:

I also LOVE that they're open on Sunday - yay! A nice relaxing way to treat yourself at the end of the week. I have did that once with good results as they are very punctual. There are even a few male clients. Thursday, Friday and Sat. The color choice is huge. I think the prices are fair - not cheap, but not over-priced.

There is a man that is the manager or owner who is normally in the front of the shop giving manicures and he is very friendly and knowledgeable and keeps things organized and everyone on schedule



## Word Count/Word Frequency based approach

### Code

```
import json
data = [json.loads(line) for line in open("E:\MS SEM2\CIS
593\Project\Dataset\yelp_academic_dataset_review.json", "r",
encoding="utf8", errors="ignore")]

from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize, sent_tokenize

#This function gets the text and word frequency table
def frequency_table(text_string) -> dict:
    stopWords = set(stopwords.words("english"))
    words = word_tokenize(text_string)
    ps = PorterStemmer()

    freqTable = dict()
    for word in words:
        word = ps.stem(word)
        if word in stopWords:
            continue
        if word in freqTable:
            freqTable[word] += 1
        else:
            freqTable[word] = 1
    return freqTable

#This function score the sentence based on the frequency of the word in it
def sentences_score(sentences, freqTable) -> dict:
    sentenceValue = dict()
    for sentence in sentences:
        word_count_in_sentence = (len(word_tokenize(sentence)))
        for wordValue in freqTable:
            if wordValue in sentence.lower():
                if sentence[:10] in sentenceValue:
                    sentenceValue[sentence[:10]] += freqTable[wordValue]
                else:
                    sentenceValue[sentence[:10]] = freqTable[wordValue]
            sentenceValue[sentence[:10]] = sentenceValue[sentence[:10]] //
word_count_in_sentence
        return sentenceValue

#Get the average score - threshold
def get_average_score(sentenceValue) -> int:
    sumValues = 0
    for entry in sentenceValue:
        sumValues += sentenceValue[entry]
    # Average value of a sentence from original text
    average = int(sumValues / len(sentenceValue))
    return average
```

```

#This function finds the summary and return to the run_summarization
funciton
def summary(sentences, sentenceValue, threshold):
    sentence_count = 0
    summary = ''
    for sentence in sentences:
        if sentence[:10] in sentenceValue and sentenceValue[sentence[:10]]
> (threshold):
            summary += " " + sentence
            sentence_count += 1
    return summary

#this function does all the steps of Text summerization by callin each
funciton saperately
def run_summarization(text):
    freq_table = frequency_table(text)
    sentences = sent_tokenize(text)
    sentence_scores = sentences_score(sentences, freq_table)
    threshold = get_average_score(sentence_scores)
    summaryText = summary(sentences, sentence_scores, 1.3 * threshold)
    print(summaryText)

#if you wan't summary, run me
if __name__ == '__main__':
    run_summarization(data[5502]['text'])

```

## Output

```

In [6]: runfile('E:/MS SEM2/CIS 593/Project/Code_text_sumemrization/
text_analysis_becominghuman_TF_Matrix.py', wdir='E:/MS SEM2/CIS 593/Project/
Code_text_sumemrization')

```

I've gone to this nail salon about 5-6 times/year for several years now. The color choice is huge. There are even a few male clients. A nice relaxing way to treat yourself at the end of the week. Thursday, Friday and Sat. are their busiest days, but they have a full house of manicurists to handle the crowd. If you find someone you really like, you can always make an appnt. I think the prices are fair - not cheap, but not over-priced. They also have another person in the front of the shop that is the sceduler, answe the phones, restocks supplies, etc.

## TF-IDF matrix based approach

### Code

```
import json
data = [json.loads(line) for line in open("E:\MS SEM2\CIS
593\Project\Dataset\yelp_academic_dataset_review.json", "r",
encoding="utf8", errors="ignore")]

import math
from nltk import sent_tokenize, word_tokenize, PorterStemmer
from nltk.corpus import stopwords

#This function finds the count of each word and creates frequency matrix
for each word
def frequency_table(sentences):
    frequency_matrix = {}
    stopWords = set(stopwords.words("english"))
    ps = PorterStemmer()
    for sent in sentences:
        freq_table = {}
        words = word_tokenize(sent)
        for word in words:
            word = word.lower()
            word = ps.stem(word)
            if word in stopWords:
                continue

            if word in freq_table:
                freq_table[word] += 1
            else:
                freq_table[word] = 1
        frequency_matrix[sent[:15]] = freq_table
    return frequency_matrix

#This fucntion gets the word frequency matrix and make TF matrix
def get_tf_matrix(freq_matrix):
    tf_matrix = {}
    for sent, f_table in freq_matrix.items():
        tf_table = {}
        count_words_in_sentence = len(f_table)
        for word, count in f_table.items():
            tf_table[word] = count / count_words_in_sentence
        tf_matrix[sent] = tf_table
    return tf_matrix

#This function creates word per document table which will be used for IDF
matrix creation
def documents_per_words_count(freq_matrix):
    word_per_doc_table = {}
    for sent, f_table in freq_matrix.items():
        for word, count in f_table.items():
            if word in word_per_doc_table:
                word_per_doc_table[word] += 1
            else:
                word_per_doc_table[word] = 1
    return word_per_doc_table
```

```

#This function uses the word per document table and gets other two
parameters and finds IDF matrix
def get_idf_matrix(freq_matrix, count_doc_per_words, total_documents):
    idf_matrix = {}
    for sent, f_table in freq_matrix.items():
        idf_table = {}
        for word in f_table.keys():
            idf_table[word] = math.log10(total_documents /
float(count_doc_per_words[word]))
        idf_matrix[sent] = idf_table
    return idf_matrix

#This is simple multiplicaiton of TF matrix and IDF matrix to find TF-IDF
scores for words
def tf_idf_matrix_multiplication(tf_matrix, idf_matrix):
    tf_idf_matrix = {}
    for (sent1, f_table1), (sent2, f_table2) in zip(tf_matrix.items(),
idf_matrix.items()):
        tf_idf_table = {}
        for (word1, value1), (word2, value2) in zip(f_table1.items(),
f_table2.items()): # here, keys are the same in both the table
            tf_idf_table[word1] = float(value1 * value2)
        tf_idf_matrix[sent1] = tf_idf_table
    return tf_idf_matrix

#Based on the TF-IDF score, each sentence will be valued
def sentence_score(tf_idf_matrix) -> dict:
    sentenceValue = {}
    for sent, f_table in tf_idf_matrix.items():
        total_score_per_sentence = 0
        count_words_in_sentence = len(f_table)
        for word, score in f_table.items():
            total_score_per_sentence += score
        sentenceValue[sent] = total_score_per_sentence /
count_words_in_sentence
    return sentenceValue

#Find the average score - define as threashold
def _find_average_score(sentenceValue) -> int:
    sumValues = 0
    for entry in sentenceValue:
        sumValues += sentenceValue[entry]
    # Average value of a sentence from original summary_text
    average = (sumValues / len(sentenceValue))
    return average

#This function gets each sentence, its value and threashold and generate
the summary
def summary(sentences, sentenceValue, threshold):
    sentence_count = 0
    summary = ''
    for sentence in sentences:
        if sentence[:15] in sentenceValue and sentenceValue[sentence[:15]]
>= (threshold):
            summary += " " + sentence
            sentence_count += 1
    return summary

```

```

#This fucntion calls each supporting function to generate summary
def run_summarization(text):
    sentences = sent_tokenize(text)
    total_documents = len(sentences)
    freq_matrix = frequency_table(sentences)
    tf_matrix = get_tf_matrix(freq_matrix)
    count_doc_per_words = documents_per_words_count(freq_matrix)
    idf_matrix = get_idf_matrix(freq_matrix, count_doc_per_words,
total_documents)
    tf_idf_matrix = tf_idf_matrix_multiplication(tf_matrix, idf_matrix)
    sentence_scores = sentence_score(tf_idf_matrix)
    threshold = _find_average_score(sentence_scores)
    summaryValue = summary(sentences, sentence_scores, 1.0 * threshold)
    return summaryValue

#If you wanna get the summaried text, run me
if __name__ == '__main__':
    result = run_summarization(data[5502]['text'])
    print(result)

```

## Output

```

In [13]: runfile('E:/MS SEM2/CIS 593/Project/Code_text_sumemrization/
text_analysis_becominghuman_TF-IDF_Matrix.py', wdir='E:/MS SEM2/CIS 593/
Project/Code_text_sumemrization')

```

The color choice is huge. There are even a few male clients. I also LOVE that they're open on Sunday - yay! A nice relaxing way to treat yourself at the end of the week. Thursday, Friday and Sat. are their busiest days, but they have a full house of manicurists to handle the crowd. I have did that once with good results as they are very punctual. My only complaint is that the other side of being punctual is that sometimes I do feel a bit rushed. I think the prices are fair - not cheap, but not over-priced. They have a long table in the front so you can sit and dry your toes or finger nails.

**Here is the actual review for which you have read the summary from three different techniques**

```
In [16]: data[5502]['text']
```

```
Out[16]: "I've gone to this nail salon about 5-6 times/year for several years now. For me, it's very convenient with lots of stores, restaurants and a grocery minutes away and I love that you don't need an appointment. The color choice is huge. So in the summer when I'm running errands I can just pop in there and there's normally not more than a 5-10 minute wait, but they do take appnts and there are a lot of regular clients. There are even a few male clients. I also LOVE that they're open on Sunday - yay! A nice relaxing way to treat yourself at the end of the week. Thursday, Friday and Sat. are their busiest days, but they have a full house of manicurists to handle the crowd. I've had various levels of quality depending on who you get, but all seem to be well trained, perform their tasks very conscientiously with good hygiene practices and are pleasant to talk to. It also seems to be well ventilated as I don't get that overwhelming chemical smell when I walk in. If you find someone you really like, you can always make an appnt. I have did that once with good results as they are very punctual. Now this is not a luxury spa where you'll be given flavored coffee while you have a pedicure, its a moderately large, nails-only salon without a lot of fluff. My only complaint is that the other side of being punctual is that sometimes I do feel a bit rushed. I think the prices are fair - not cheap, but not over-priced.\n\nThere is a man that is the manager or owner who is normally in the front of the shop giving manicures and he is very friendly and knowledgeable and keeps things organized and everyone on schedule. They also have another person in the front of the shop that is the sceduler, answees the phones, restocks supplies, etc. I usually go for a pedicure without a massage every few months in the cold weather months to get rid of all that dried, dead skin and it feels sooo nice. I get pedicures more frequently in the warmer months with an occasional manicure to treat myself if there's a special event coming up. They have a long table in the front so you can sit and dry your toes or finger nails."
```

## Dataset

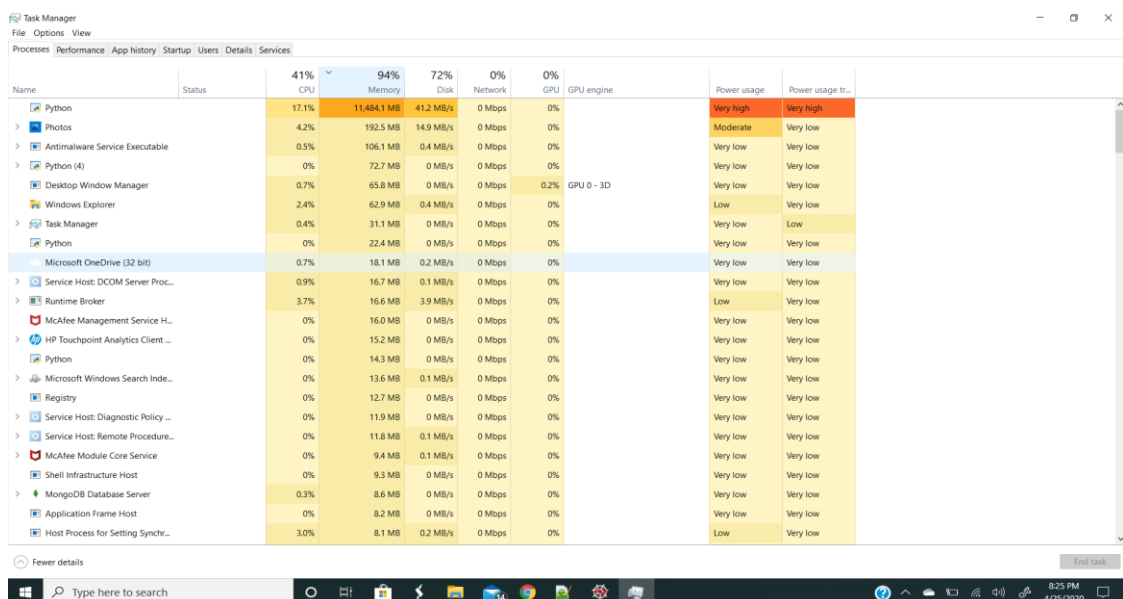
We have used Yelp academic review dataset for our analysis.

<https://www.yelp.com/dataset>

When we find the length of each review and plot the graph where text\_len on X axis and counts on Y axis, we come to know that data is skewed towards positive side. Hence, I choose equal amount of reviews for both the sides 1600 each out of 10000 total reviews to make word\_count\_map.

## Challenges faced

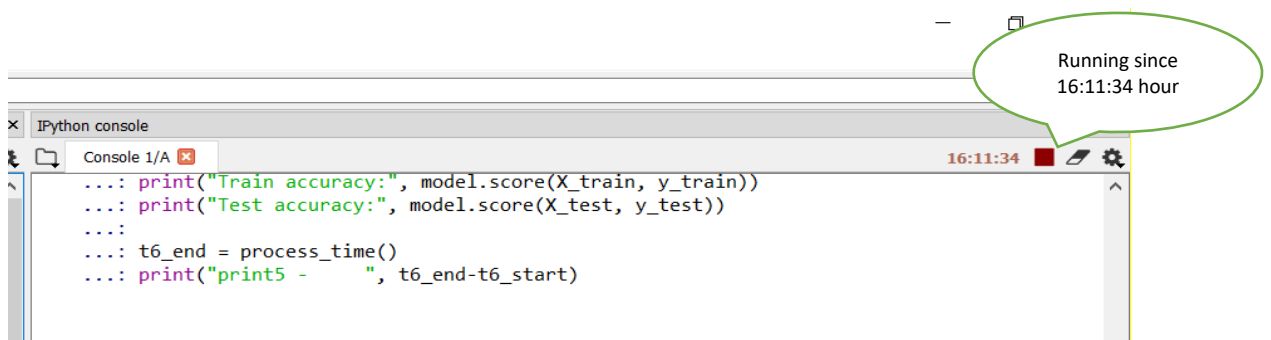
1. After analysis, our review we came to know that most of the review in our dataset was positive and because of that some of our ML Approach was not working good in initial phase.
2. Of course, the size of our data, its was more than 6GB and it was taking more time to just load the data. When we are tried to load those data directly by reading the whole json file, the CPU, Memory and Disk Utilization wen so high that none of the other services were responding. Then we read the json file row by row, means one json review data at a time. And that helped us to reduce time to 10 min.



The screenshot shows the Windows Task Manager 'Processes' tab. The 'Python' process is highlighted, showing 41% CPU usage, 94% memory usage, and 72% disk usage. Other processes like Photos, Antimalware Service Executable, and Desktop Window Manager are also visible with their respective resource usage.

Name	Status	CPU	Memory	Disk	Network	GPU	GPU engine	Power usage	Power usage tr...
Python		41%	94%	72%	0%	0%		Very high	Very high
Photos		4.2%	192.5 MB	14.9 MB/s	0 Mbps	0%		Moderate	Very low
Antimalware Service Executable		0.5%	106.1 MB	0.4 MB/s	0 Mbps	0%		Very low	Very low
Python (4)		0%	72.7 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Desktop Window Manager		0.7%	65.8 MB	0 MB/s	0 Mbps	0.2%	GPU 0 - 3D	Very low	Very low
Windows Explorer		2.4%	62.9 MB	0.4 MB/s	0 Mbps	0%		Low	Very low
Task Manager		0.4%	31.1 MB	0 MB/s	0 Mbps	0%		Very low	Low
Python		0%	22.4 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Microsoft OneDrive (32 bit)		0.7%	18.1 MB	0.2 MB/s	0 Mbps	0%		Very low	Very low
Service Host: DCOM Server Proc...		0.9%	16.7 MB	0.1 MB/s	0 Mbps	0%		Very low	Very low
Runtime Broker		3.7%	16.6 MB	3.9 MB/s	0 Mbps	0%		Low	Very low
McAfee Management Service H...		0%	16.0 MB	0 MB/s	0 Mbps	0%		Very low	Very low
HP Touchpoint Analytics Client ...		0%	15.2 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Python		0%	14.3 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Microsoft Windows Search Inde...		0%	13.6 MB	0.1 MB/s	0 Mbps	0%		Very low	Very low
Registry		0%	12.7 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Service Host: Diagnostic Policy ...		0%	11.9 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Service Host: Remote Procedure...		0%	11.8 MB	0.1 MB/s	0 Mbps	0%		Very low	Very low
McAfee Module Core Service		0%	9.4 MB	0.1 MB/s	0 Mbps	0%		Very low	Very low
Shell Infrastructure Host		0%	9.3 MB	0 MB/s	0 Mbps	0%		Very low	Very low
MongoDB Database Server		0.3%	8.6 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Application Frame Host		0%	8.2 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Host Process for Setting Synch...		3.0%	8.1 MB	0.2 MB/s	0 Mbps	0%		Low	Very low

3. At one time when we were trying to perform the sentiment analysis using Machine learning based approach, it took us almost 23.7 hours to make word\_index\_map for just positive tweets which was around 5 million and yet we didn't get output and hence we cancelled the program.



4. When we tried to run the lexicon-based sentiment analysis then the time duration for different range of dataset is shown in the graph below.

