# Cleveland State University



**CIS 660: Data mining**

**Final Report**

**Submitted By: Submitted To:**

Dhruv Patel (2815077) Dr. SS Chung

Vishwa Lakhnakiya (2834394)

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

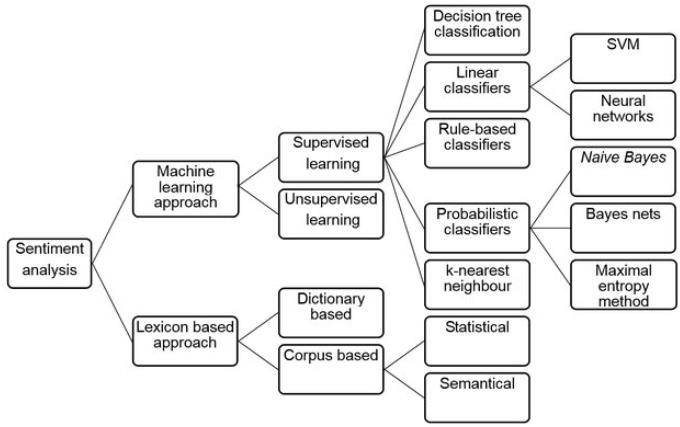
## Types of Sentiment Analysis

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

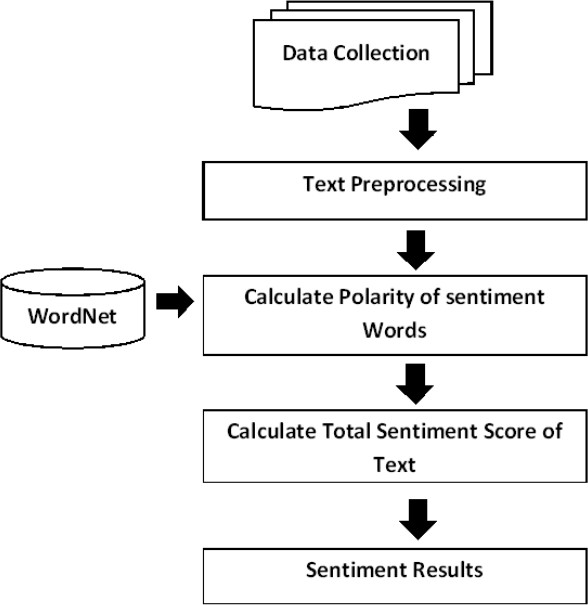
## Techniques and Algorithms

There are **three types** of sentiment analysis

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



## Lexicon-Based Approach for Sentiment Analysis



* To do the lexical analysis of the data, first, you must go through each NLP concept i.e. removing stop words, removing 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 words in your word dictionary, in our case we used the words which we downloaded from GitHub, there were 2 text files positive and negative words.
* Now after doing all the checks, the last step is to check whether the text is positive or negative using the popular formula, which minimizes the total number of negative words from the total number of positive words divided by the total number of words.

### 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("C:/Users/Rikul/Desktop/Data Mining/Data Mining Project/archive/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("C:/Users/Rikul/Desktop/Data Mining/Data Mining Project/positive\_words.txt",'r') as f:

q = f.read().split() for shivam in q:

positive.append(shivam) commmon\_positive = set(q)

negative = []

with open("C:/Users/Rikul/Desktop/Data Mining/Data Mining Project/archive/negative\_words.txt",'r') as f:

q\_one = f.read().split() for shivam in q\_one:

negative.append(shivam) commmon\_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 = commmon\_positive & check\_one negative\_test = commmon\_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

#StSc = (number of positive words - number of negative words) / total number of words

sentiment\_score = (true+false)/total

if(sentiment\_score>0):

print("Sentiment score is Positive.")

elif(sentiment\_score<0):

print("Sentiment score is Negative.")

elif(sentiment\_score == 0):

print("Sentiment score is Neutral.")

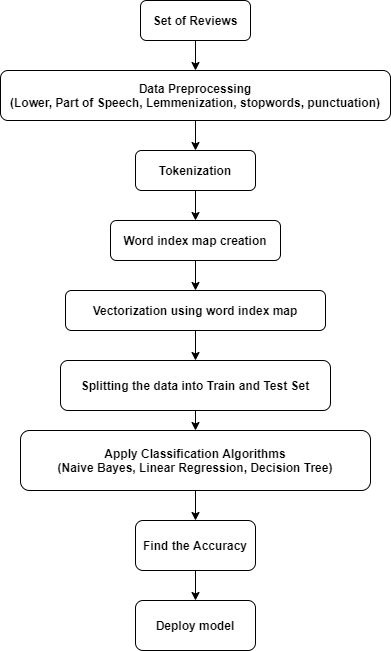
### OUTPUT

**Text

Description automatically generated**

* This output is generated on initial 1-10000 data.

## Machine Learning-based Approach for Sentiment Analysis



**Steps:**

* In this method, we considered reviews as positive if it has 5 stars and reviews as negative 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 a dictionary named word\_index\_map.
* To create a word index map, each review should be tokenized in words, and then it should be inserted into the word count dictionary where the key/index is the word and the value is its frequency count.
* Before the creation of the word count dictionary, each review should be tokenized and applied to all the NLP preprocessing techniques like POS tagging, lemmatization, stop words removal, punctuation removal, etc.
* Now it’s time to create a 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 sets to apply the 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("C:/Users/Rikul/Desktop/Data Mining/Data Mining Project/archive/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 = for | 0  tokens in positive\_tokenized: xy = tokens\_to\_vector(tokens, data[i,:] = xy  i += 1 | 1) |
| for | tokens in negative\_tokenized: xy = tokens\_to\_vector(tokens, data[i,:] = xy  i += 1 | 0) |

#train and test feature selection

X = data[:,:-1]

Y = data[:,-1]

#Apply machine learning algrithms for data mining from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import train\_test\_split

# Apply machine learning algrithms for data mining

# Function for evaluating the model

def evaluate(y\_test, y\_predict):

cf\_matrix = confusion\_matrix(y\_test,y\_predict)

sns.heatmap(cf\_matrix, annot = True, fmt = 'd',cmap="Blues")

plt.title('Heatmap of confusion matrix for Test data')

plt.ylabel('True label')

plt.xlabel('Predicted label')

print("Precision Score: ", precision\_score(y\_test,y\_predict)\*100)

print("Recall Score: ", recall\_score(y\_test,y\_predict)\*100)

print("Acuracy score: ",accuracy\_score(y\_test,y\_predict)\*100)

print("F1 score: ",f1\_score(y\_test,y\_predict)\*100)

#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 Algotithm - LOGISTIC REGRESSION")

evaluate(y\_test, y\_predict)

#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")

evaluate(y\_test, y\_predict)

#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")

evaluate(y\_test, y\_predict)

#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")

evaluate(y\_test, y\_predict)

**Output for ML-Based Approach**

* Logistic Regression

**Chart

Description automatically generated**

* SVC

**A picture containing application

Description automatically generated**

* Decision Tree

**Chart

Description automatically generated**

* Naïve Bayes

Chart, treemap chart

Description automatically generated

**Relation between length vs the number of reviews vs stars**

**Chart

Description automatically generated with low confidence**

**Comparison of 4 classifiers algorithm**

* After comparing between 4 classification algorithms, we can say that the Support Vector System (SVC) performed better with an accuracy of 86.67.

# TEXT SUMMARIZATION

* Summarization can be defined as the task of producing a concise and fluent summary while preserving key information and overall meaning.
* There are two kinds of summarization:

1. Abstractive
2. Extractive

* In this project, we have used Extractive Summarization.
* It involves the selection of words/phrases and sentences from the source document itself to make up the summary. There are different algorithms 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 a 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 necessary 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.
   1. Steps
      1. Input Article
      2. Split into sentences
      3. Remove stop words
      4. Build a similarity matrix
      5. Generate rank based on matrix
      6. Pick top sentences for summary
2. Sentence Scoring based on **Word Frequency** in a sentence
   1. Steps
      1. Input article
      2. Remove stop words
      3. Create a dictionary for word frequency
      4. Tokenize into the sentences
      5. Generate rank based on term frequency
      6. Find threshold
      7. Generate summary
3. Sentence scoring based on **TF-IDF** score of words in a sentence
   1. Steps
      1. Input article
      2. Split into sentences
      3. Removing stop words
      4. Build a frequency matrix of the words in each sentence
      5. Create TF matrix
      6. Create documents per words matrix
      7. Calculate IDF matrix
      8. Calculate TF-IDF matrix
      9. Score the sentence based on the TF-IDF score
      10. Find Threshold
      11. Generate Summary.

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

1. Create the word Frequency Table
   1. 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:
   1. In this, we are going to clean the sentences by removing
      1. Stop words
      2. Removing whitespace
      3. Removing Numeric values
      4. Other unnecessary characters
3. Tokenize the Sentences
   1. In tokenization, we are separating the paragraph into a couple of sentences and after that, we are separating those sentences into tokens.
4. Score the sentences: Term frequency
   1. It is a method to score each sentence using the frequency of each word in that sentence.
5. Find the threshold
   1. To find the threshold we are considering the average score of the sentences as a threshold.
6. Generate the summary
   1. To generate the summary
      1. If the sentence score is above the threshold then we will put that sentence in the summary
      2. If the sentence score is less than the 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 the variation is, after getting the term frequency table, you need to find the IDF and multiply that matrix with the 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("C:/Users/Rikul/Desktop/Data Mining/Data Mining Project/archive/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 sentenses. 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 sentenses

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 fucntion builds similarity matrix based on the above funtion - 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 funtion to generate summary

def generate\_summary(file\_name, top\_n): stop\_words = stopwords.words('english') summarize\_text = []

sentences = read\_article(file\_name)

sentence\_similarity\_martix = build\_similarity\_matrix(sentences, stop\_words)

sentence\_similarity\_graph = nx.from\_numpy\_array(sentence\_similarity\_martix)

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 functoin here to generate the summary generate\_summary( data[55]['text'], 6)

**Output**

Text

Description automatically generated

### Word Count/Word Frequency-based approach

**Code**

import json

data = [json.loads(line) for line in open("C:/Users/Rikul/Desktop/Data Mining/Data Mining Project/archive/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 fucntion score the sentence based on teh 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[55]['text'])

**Output**

A screenshot of a computer

Description automatically generated

### TF-IDF matrix-based approach

**Code**

import json

data = [json.loads(line) for line in open("C:/Users/Rikul/Desktop/Data Mining/Data Mining Project/archive/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 fequency 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[55]['text']) print(result)

**Output**

A screenshot of a computer

Description automatically generated

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

**Text

Description automatically generated**

# Dataset

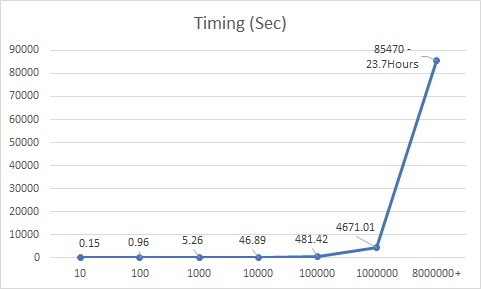
* We have used Yelp academic review dataset for our analysis. <https://www.yelp.com/dataset>.
* When we find the length of each review and plot the graph where text\_len is on X-axis and counts on the Y-axis, we come to know that data is skewed towards the positive side. Hence, I choose an equal amount of reviews for both sides 1600 each out of 10000 total reviews to make a word\_count\_map.

# Challenges faced

1. After analysis, and 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 well in the initial phase.
2. Of course, the size of our data was more than 4GB and it was taking more time to just load the data. When we 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, which means one JSON review data at a time. And that helped us to reduce time to 10 min.

Graphical user interface, application, table, Excel

Description automatically generated

1. At one time when we were trying to perform the sentiment analysis using a Machine learning-based approach, it took us almost 23.7 hours to make word\_index\_map for just positive tweets which were around 3 million, and yet we didn’t get output and hence we canceled the program.
2. When we tried to run the lexicon-based sentiment analysis, the time duration for different ranges of the dataset is shown in the graph below.