# Topic: Text Mining (NLP)

**Instructions:**

Please share your answers filled in-line in the word document. Submit code separately wherever applicable.

Please ensure you update all the details:

**Name: Anandakrishnan k v ;;;;;;; Batch ID:** 19042021

**Topic: Text Mining and NLP**

**Grading Guidelines:**

**1. An assignment submission is considered complete only when correct and executable code(s) are submitted along with the documentation explaining the method and results. Failing to submit either of those will be considered an invalid submission and will not be considered for evaluation.**

**2. Assignments submitted after the deadline will affect your grades.**

**Grading Criteria:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Submission** | **Date** |  |  | **Submission** | **Date** |
| Correct | On time | A | 100 |  |  |
| 80% & above | On time | B | 85 | Correct | Late |
| 50% & above | On time | C | 75 | 80% & above | Late |
| 50% & below | On time | D | 65 | 50% & above | Late |
|  |  | E | 55 | 50% & below |  |
| Copied/No Submission |  | F | 45 |  |  |

* **Grade A: (>= 90):** When all assignments are submitted on or before the given deadline.
* **Grade B: (>= 80 and < 90):** 
  + When assignments are submitted on time but less than 80% of problems are completed.

(OR)

* + All assignments are submitted after the deadline.
* **Grade C: (>= 70 and < 80):** 
  + When assignments are submitted on time but less than 50% of the problems are completed.

(OR)

* + Less than 80% of problems in the assignments are submitted after the deadline.
* **Grade D: (>= 60 and < 70):**
  + Assignments submitted after the deadline and with 50% or less problems.
* **Grade E: (>= 50 and < 60):** 
  + Less than 30% of problems in the assignments are submitted after the deadline.

(OR)

* + Less than 30% of problems in the assignments are submitted before the deadline.
* **Grade F: (< 50):** No submission (or) malpractice.

**Hints:**

1. **Business Problem**

Task 1:

1. Extract reviews of any product from e-commerce website Amazon.
2. Perform sentiment analysis on this extracted data and build a unigram and bigram word cloud.

**R code:-**

######### Amazon Reviews Extraction ###########

#install.packages("rvest")

#install.packages("XML")

#install.packages("magrittr")

#install.packages("wordcloud")

library(tm)

library(wordcloud)

library(rvest)

library(XML)

library(magrittr)

######### Amazon URL ###########

aurl <- "https://www.amazon.in/OnePlus-Midnight-Black-128GB-Storage/product-reviews/B07DJHY82F/ref=cm\_cr\_getr\_d\_paging\_btm\_prev\_1?showViewpoints=1&pageNumber="

amazon\_reviews <- NULL

for (i in 1:30){

murl <- read\_html(as.character(paste(aurl,i,sep="")))

rev <- murl %>% html\_nodes(".review-text") %>% html\_text()

amazon\_reviews <- c(amazon\_reviews,rev)

}

?html\_nodes

?html\_text

?read\_html

write.table(amazon\_reviews,"Oneplus6T.txt")

getwd()

############################

#### Sentiment Analysis ####

txt <- amazon\_reviews

str(txt)

length(txt)

View(txt)

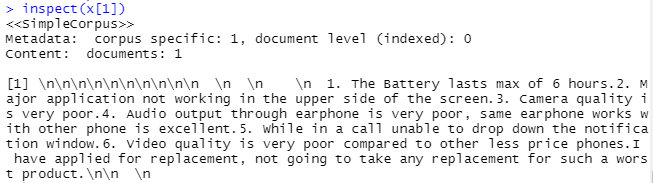
#install.packages("tm")

library(tm)

# Convert the character data to corpus type

x <- Corpus(VectorSource(txt))

inspect(x[1])



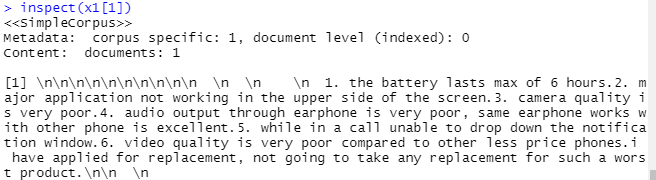
x <- tm\_map(x, function(x) iconv(enc2utf8(x), sub='byte'))

?tm\_map

# Data Cleansing

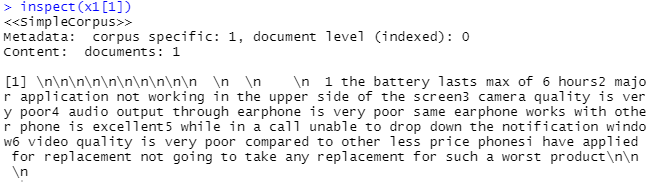
x1 <- tm\_map(x, tolower)

inspect(x1[1])



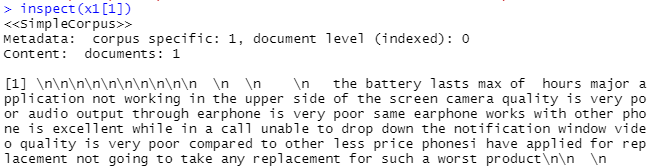
x1 <- tm\_map(x1, removePunctuation)

inspect(x1[1])



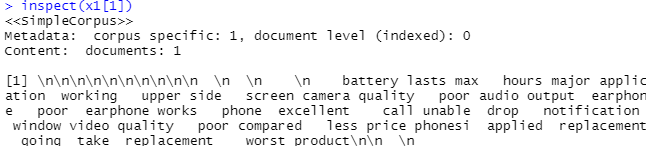
inspect(x1[5])

x1 <- tm\_map(x1, removeNumbers)

inspect(x1[1])

x1 <- tm\_map(x1, removeWords, stopwords('english'))

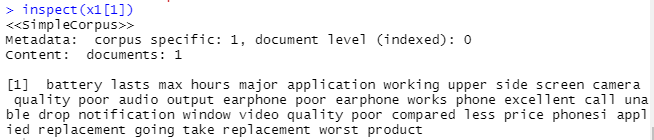
inspect(x1[1])



# striping white spaces

x1 <- tm\_map(x1, stripWhitespace)

inspect(x1[1])



# Term document matrix

# converting unstructured data to structured format using TDM

tdm <- TermDocumentMatrix(x1)

dtm <- t(tdm) # transpose ## creatm dtm from tdm

dtm <- DocumentTermMatrix(x1) ## directly creating tdm

# To remove sparse entries upon a specific value

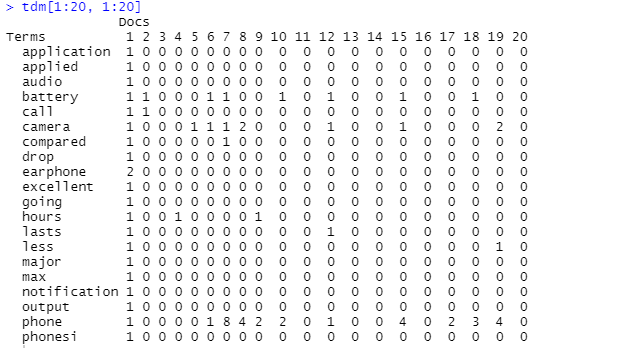
corpus.dtm.frequent <- removeSparseTerms(tdm, 0.99)

?removeSparseTerms

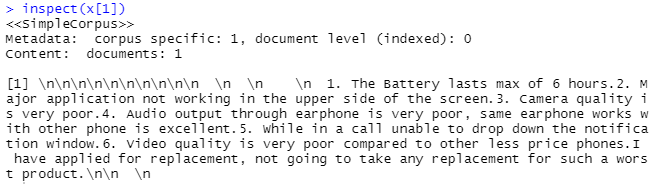
tdm <- as.matrix(tdm)

dim(tdm)

tdm[1:20, 1:20]



inspect(x[1])



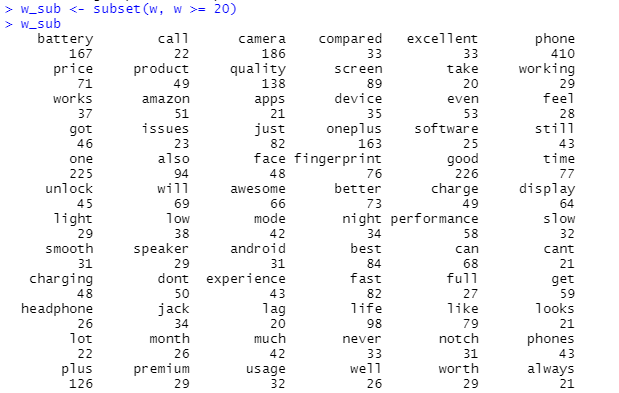
# Bar plot

w <- rowSums(tdm)

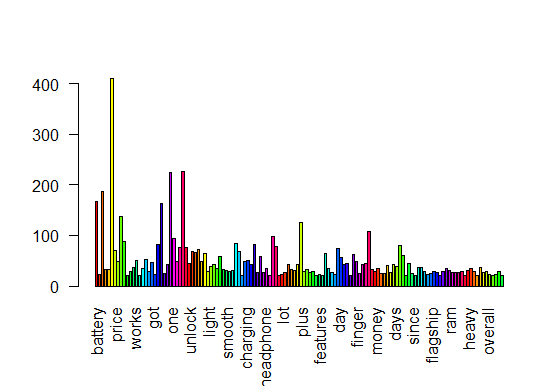
w

w\_sub <- subset(w, w >= 20)

w\_sub



barplot(w\_sub, las=2, col = rainbow(30))



# Term repeats maximum number of times

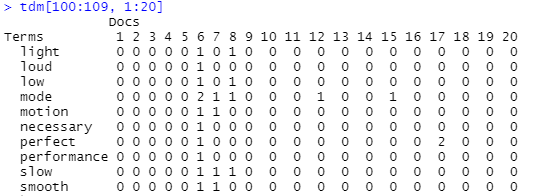
x1 <- tm\_map(x1, removeWords, c('phone','day','print','phones','product','amazon','buy','android','unlock','compared','month','thought','bought','months','night','back','days','thought','time','much','used','doesnt','review','feel','full','looks','takes','new','oneplus','will','even','plus','goes','like','still','got','dont','get','one','use','ised','phonesi','yrs','cant','just','also','using','since','overall','say','can','really','now'))

x1 <- tm\_map(x1, stripWhitespace)

tdm <- TermDocumentMatrix(x1)

tdm <- as.matrix(tdm)

tdm[100:109, 1:20]



# Bar plot after removal of the term 'phone','oneplus','will','goes','like',,still','got','ised','phonesi','yrs','cant','just','also','using','since','overall','say'

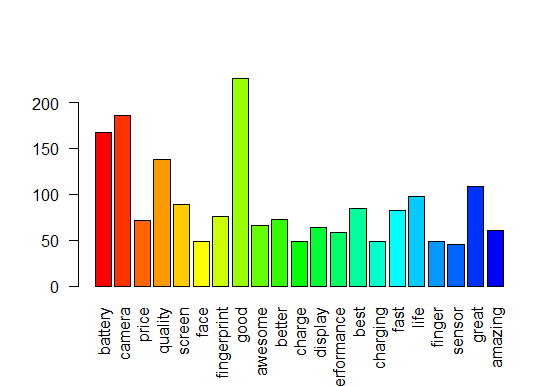
w <- rowSums(tdm)

w

w\_sub <- subset(w, w >= 45)

w\_sub

barplot(w\_sub, las=2, col = rainbow(30))



##### Word cloud #####

#install.packages("wordcloud")

library(wordcloud)

wordcloud(words = names(w\_sub), freq = w\_sub)

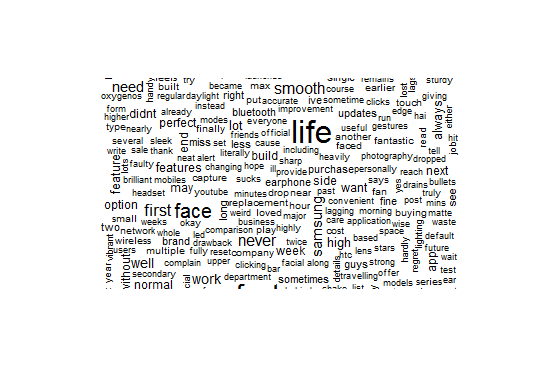


w\_sub1 <- sort(rowSums(tdm), decreasing = TRUE)

head(w\_sub1)

G:\txt mng & nlp\paint.png

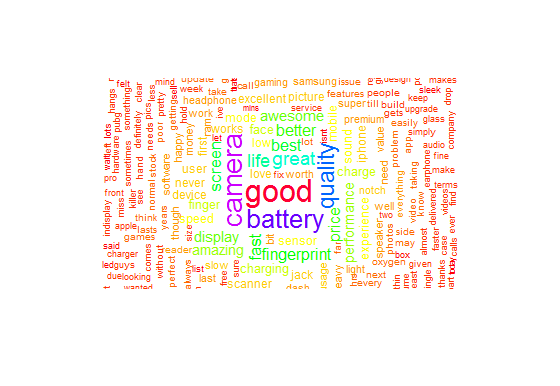
wordcloud(words = names(w\_sub1), freq = w\_sub1) # all words are considered



# better visualization

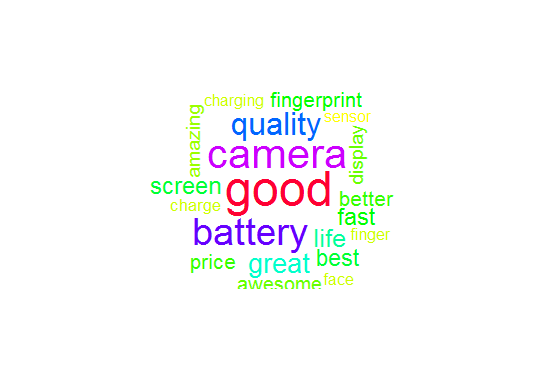
wordcloud(words = names(w\_sub1), freq = w\_sub1, random.order=F, colors=rainbow(30), scale = c(2,0.5), rot.per = 0.4)

windows()



wordcloud(words = names(w\_sub), freq = w\_sub, random.order=F, colors= rainbow(30),scale=c(3,0.5),rot.per=0.3)

windows()



?wordcloud

############# Wordcloud2 ###############

### for different visualization effect

#installed.packages("wordcloud2")

library(wordcloud2)

w1 <- data.frame(names(w\_sub), w\_sub)

colnames(w1) <- c('word', 'freq')

wordcloud2(w1, size=0.3, shape='circle')

?wordcloud2

wordcloud2(w1, size=0.3, shape = 'triangle')



wordcloud2(w1, size=0.3, shape = 'star')



#### Bigram ####

#install RWeka & rjava

##### since having package issues couldnt do bigram workcloud ###

#library(RWeka) # for bigram option

#library(rjava) # for bigram option

#library(wordcloud)

#minfreq\_bigram <- 2

#bitoken <- NGramTokenizer(x1, Weka\_control(min = 2, max = 2))

#two\_word <- data.frame(table(bitoken))

#sort\_two <- two\_word[order(two\_word$Freq, decreasing = TRUE), ]

#wordcloud(sort\_two$bitoken, sort\_two$Freq, random.order = F, scale = c(2, 0.35), min.freq = minfreq\_bigram, colors = brewer.pal(8, "Dark2"), max.words = 150)

#####################################

# lOADING Positive and Negative words

pos.words <- readLines(file.choose()) # read-in positive-words.txt

neg.words <- readLines(file.choose()) # read-in negative-words.txt

neg.words <- readLines(file.choose()) # read-in negative-words.txt

### Positive word cloud ###

pos.matches <- match(names(w\_sub1), pos.words)

pos.matches <- !is.na(pos.matches)

freq\_pos <- w\_sub1[pos.matches]

names <- names(freq\_pos)

windows()

wordcloud(names, freq\_pos, scale=c(4,1), colors = brewer.pal(8,"Dark2"))



### Matching Negative words ###

neg.matches <- match(names(w\_sub1), neg.words)

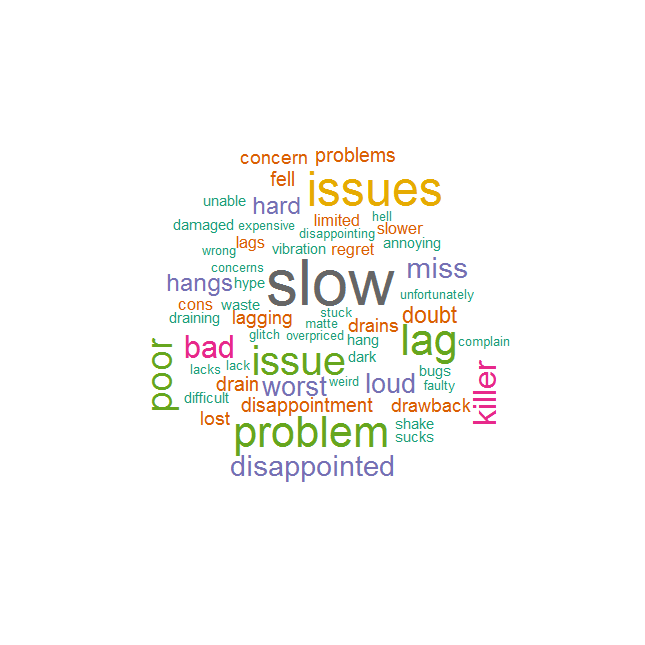
neg.matches <- !is.na(neg.matches)

freq\_neg <- w\_sub1[neg.matches]

names <- names(freq\_neg)

windows()

wordcloud(names, freq\_neg, scale=c(4,.5), colors = brewer.pal(8, "Dark2"))



**Python Code:-**

Simply pasting Python code

##### A mazone one plus mobile review #####

import requests # Importing requests to extract content from a url

from bs4 import BeautifulSoup as bs # Beautifulsoup is for web scrapping...used to scrap specific content

import re

import matplotlib.pyplot as plt

from wordcloud import WordCloud

# creating empty reviews list

oneplus\_reviews=[]

for i in range(1,21):

ip=[]

url="https://www.amazon.in/OnePlus-Display-Storage-4000mAH-Battery/product-reviews/B07HGJK535/ref=cm\_cr\_getr\_d\_paging\_btm\_next\_3?ie=UTF8&reviewerType=all\_reviews&pageNumber=1"+str(i)

response = requests.get(url)

soup = bs(response.content,"html.parser")# creating soup object to iterate over the extracted content

reviews = soup.find\_all("span",attrs={"class","a-size-base review-text review-text-content"})# Extracting the content under specific tags

for i in range(len(reviews)):

ip.append(reviews[i].text)

oneplus\_reviews=oneplus\_reviews+ip # adding the reviews of one page to empty list which in future contains all the reviews

# writng reviews in a text file

with open("oneplus.txt","w",encoding='utf8') as output:

output.write(str(oneplus\_reviews))

# Joinining all the reviews into single paragraph

ip\_rev\_string = " ".join(oneplus\_reviews)

import nltk

from nltk.corpus import stopwords

# Removing unwanted symbols incase if exists

ip\_rev\_string = re.sub("[^A-Za-z" "]+"," ", ip\_rev\_string).lower()

ip\_rev\_string = re.sub("[0-9" "]+"," ", ip\_rev\_string)

# words that contained in iphone XR reviews

ip\_reviews\_words = ip\_rev\_string.split(" ")

#TFIDF

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(ip\_reviews\_words, use\_idf=True,ngram\_range=(1, 3))

X = vectorizer.fit\_transform(ip\_reviews\_words)

with open("G://txt mng & nlp//Datasets NLP//stopwords\_en.txt","r") as sw:

stop\_words = sw.read()

stop\_words = stop\_words.split("\n")

stop\_words.extend(["oneplus","mobile","time","android","phone","device","screen","battery","product","good","day","price"])

ip\_reviews\_words = [w for w in ip\_reviews\_words if not w in stop\_words]

# Joinining all the reviews into single paragraph

ip\_rev\_string = " ".join(ip\_reviews\_words)

# WordCloud can be performed on the string inputs.

# Corpus level word cloud

wordcloud\_ip = WordCloud(

background\_color='White',

width=1800,

height=1400

).generate(ip\_rev\_string)

plt.imshow(wordcloud\_ip)

# positive words # Choose the path for +ve words stored in system

with open("G://txt mng & nlp//Datasets NLP//positive-words.txt","r") as pos:

poswords = pos.read().split("\n")

# Positive word cloud

# Choosing the only words which are present in positive words

ip\_pos\_in\_pos = " ".join ([w for w in ip\_reviews\_words if w in poswords])

wordcloud\_pos\_in\_pos = WordCloud(

background\_color='White',

width=1800,

height=1400

).generate(ip\_pos\_in\_pos)

plt.figure(2)

plt.imshow(wordcloud\_pos\_in\_pos)

# negative words Choose path for -ve words stored in system

with open("G://txt mng & nlp//Datasets NLP//negative-words.txt", "r") as neg:

negwords = neg.read().split("\n")

# negative word cloud

# Choosing the only words which are present in negwords

ip\_neg\_in\_neg = " ".join ([w for w in ip\_reviews\_words if w in negwords])

wordcloud\_neg\_in\_neg = WordCloud(

background\_color='black',

width=1800,

height=1400

).generate(ip\_neg\_in\_neg)

plt.figure(3)

plt.imshow(wordcloud\_neg\_in\_neg)

# wordcloud with bigram

import nltk

nltk.download('punkt')

nltk.download('wordnet')

from wordcloud import WordCloud, STOPWORDS

WNL = nltk.WordNetLemmatizer()

# Lowercase and tokenize

text = ip\_rev\_string.lower()

# Remove single quote early since it causes problems with the tokenizer.

text = text.replace("'", "")

tokens = nltk.word\_tokenize(text)

text1 = nltk.Text(tokens)

# Remove extra chars and remove stop words.

text\_content = [''.join(re.split("[ .,;:!?‘’``''@#$%^\_&\*()<>{}~\n\t\\\-]", word)) for word in text1]

# Create a set of stopwords

stopwords\_wc = set(STOPWORDS)

customised\_words = ['price', 'great','phone','day','print','phones','product','amazon','buy','android','unlock','compared','month','thought','bought','months','night','back','days','thought','time','much','used','doesnt','review','feel','full','looks','takes','new','oneplus','will','even','plus','goes','like','still','got','dont','get','one','use','ised','phonesi','yrs','cant','just','also','using','since','overall','say','can','really','now',] # If you want to remove any particular word form text which does not contribute much in meaning

new\_stopwords = stopwords\_wc.union(customised\_words)

# Remove stop words

text\_content = [word for word in text\_content if word not in new\_stopwords]

# Take only non-empty entries

text\_content = [s for s in text\_content if len(s) != 0]

# Best to get the lemmas of each word to reduce the number of similar words

text\_content = [WNL.lemmatize(t) for t in text\_content]

nltk\_tokens = nltk.word\_tokenize(text)

bigrams\_list = list(nltk.bigrams(text\_content))

print(bigrams\_list)

dictionary2 = [' '.join(tup) for tup in bigrams\_list]

print (dictionary2)

# Using count vectoriser to view the frequency of bigrams

from sklearn.feature\_extraction.text import CountVectorizer

vectorizer = CountVectorizer(ngram\_range=(2, 2))

bag\_of\_words = vectorizer.fit\_transform(dictionary2)

vectorizer.vocabulary\_

sum\_words = bag\_of\_words.sum(axis=0)

words\_freq = [(word, sum\_words[0, idx]) for word, idx in vectorizer.vocabulary\_.items()]

words\_freq =sorted(words\_freq, key = lambda x: x[1], reverse=True)

print(words\_freq[:100])

# Generating wordcloud

words\_dict = dict(words\_freq)

WC\_height = 1000

WC\_width = 1500

WC\_max\_words = 200

wordCloud = WordCloud(max\_words=WC\_max\_words, height=WC\_height, width=WC\_width, stopwords=new\_stopwords)

wordCloud.generate\_from\_frequencies(words\_dict)

plt.figure(4)

plt.title('Most frequently occurring bigrams connected by same colour and font size')

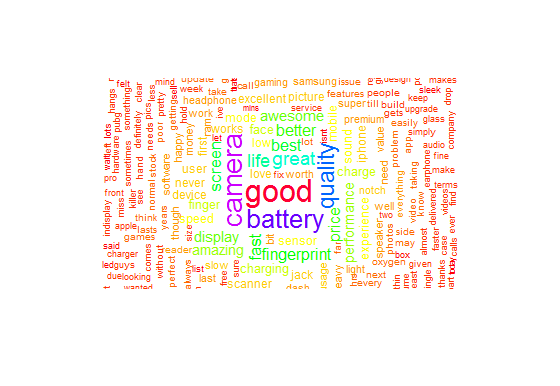
plt.imshow(wordCloud, interpolation='bilinear')

plt.axis("off")

plt.show()

**Summary:-**

Overall wordcloud:-



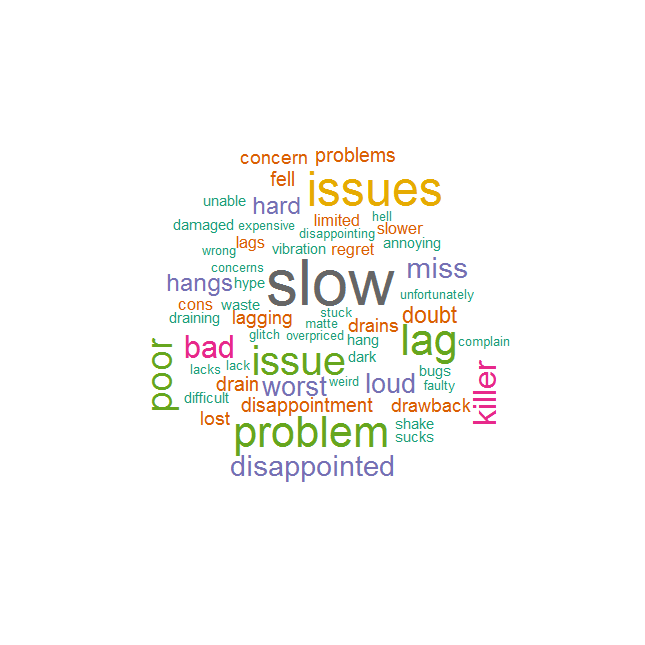
Positive wordcloud:-

Most repeating positive words in reviews



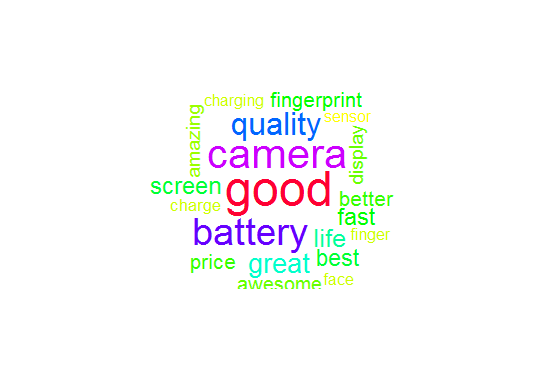
Negative wordcloud:-

Most repeating negative words in reviews



Wordcloud of most repeating words:-

Most of the most repeating words in users reviews are positive. So the product is succesful



**Task 2:**

1. **Extract reviews for any movie from IMDB and perform sentiment analysis.**

**Python Code:-**

##### imdb kaidhi film review #####

import requests # Importing requests to extract content from a url

from bs4 import BeautifulSoup as bs # Beautifulsoup is for web scrapping...used to scrap specific content

import re

import matplotlib.pyplot as plt

from wordcloud import WordCloud

# creating empty reviews list

oneplus\_reviews=[]

## all the reviews selected from a single page

ip=[]

url="https://www.imdb.com/title/tt9900782/reviews?ref\_=tt\_urv"

response = requests.get(url)

soup = bs(response.content,"html.parser")# creating soup object to iterate over the extracted content

reviews = soup.find\_all("div",attrs={"class":"text show-more\_\_control"})

for i in range(len(reviews)):

ip.append(reviews[i].text)

oneplus\_reviews=oneplus\_reviews+ip # adding the reviews of one page to empty list which in future contains all the reviews

# but here all the reviews are from same page: even though step is added but doesnt go for the loop

# writng reviews in a text file

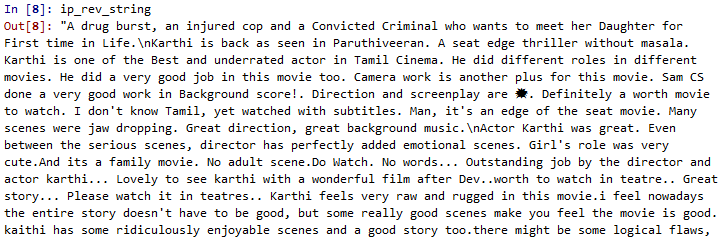
with open("oneplus.txt","w",encoding='utf8') as output:

output.write(str(oneplus\_reviews))

# Joinining all the reviews into single paragraph

ip\_rev\_string = " ".join(oneplus\_reviews)

ip\_rev\_string



import nltk

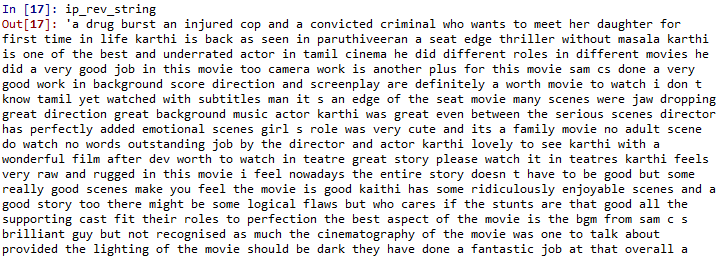
from nltk.corpus import stopwords

# Removing unwanted symbols incase if exists

ip\_rev\_string = re.sub("[^A-Za-z" "]+"," ", ip\_rev\_string).lower()

ip\_rev\_string = re.sub("[0-9" "]+"," ", ip\_rev\_string)

ip\_rev\_string



# words that contained in iphone XR reviews

ip\_reviews\_words = ip\_rev\_string.split(" ")

ip\_reviews\_words

#TFIDF

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(ip\_reviews\_words, use\_idf=True,ngram\_range=(1, 3))

X = vectorizer.fit\_transform(ip\_reviews\_words)

with open("G://txt mng & nlp//Datasets NLP//stopwords\_en.txt","r") as sw:

stop\_words = sw.read()

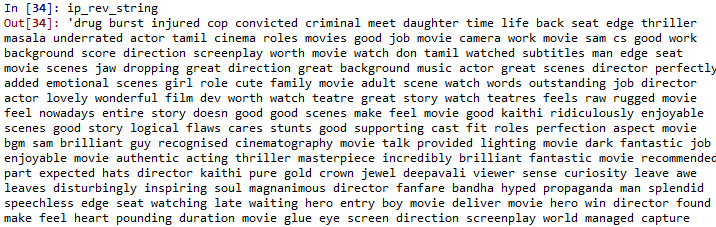
stop\_words = stop\_words.split("\n")

stop\_words.extend(["karthi","lokesh","paruthiveeran","kanagaraj"])

ip\_reviews\_words = [w for w in ip\_reviews\_words if not w in stop\_words]

# Joinining all the reviews into single paragraph without stop words

ip\_rev\_string = " ".join(ip\_reviews\_words)



# WordCloud can be performed on the string inputs.

# Corpus level word cloud

wordcloud\_ip = WordCloud(

background\_color='White',

width=1800,

height=1400

).generate(ip\_rev\_string)

plt.imshow(wordcloud\_ip)



# positive words # Choose the path for +ve words stored in system

with open("G://txt mng & nlp//Datasets NLP//positive-words.txt","r") as pos:

poswords = pos.read().split("\n")

# Positive word cloud

# Choosing the only words which are present in positive words

ip\_pos\_in\_pos = " ".join ([w for w in ip\_reviews\_words if w in poswords])

wordcloud\_pos\_in\_pos = WordCloud(

background\_color='White',

width=1800,

height=1400

).generate(ip\_pos\_in\_pos)

plt.figure(2)

plt.imshow(wordcloud\_pos\_in\_pos)



# negative words Choose path for -ve words stored in system

with open("G://txt mng & nlp//Datasets NLP//negative-words.txt", "r") as neg:

negwords = neg.read().split("\n")

# negative word cloud

# Choosing the only words which are present in negwords

ip\_neg\_in\_neg = " ".join ([w for w in ip\_reviews\_words if w in negwords])

wordcloud\_neg\_in\_neg = WordCloud(

background\_color='black',

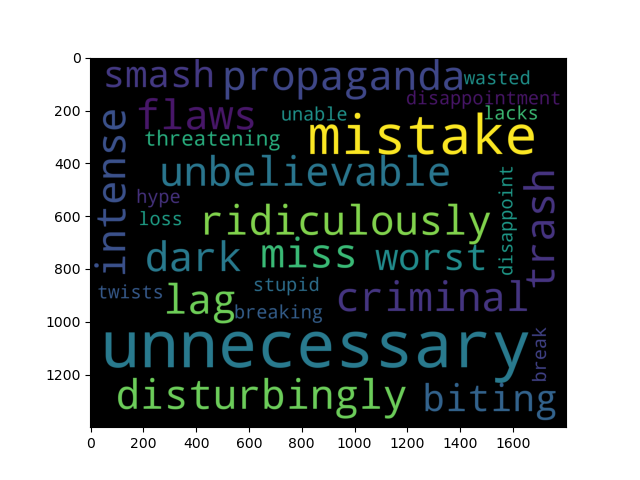
width=1800,

height=1400

).generate(ip\_neg\_in\_neg)

plt.figure(3)

plt.imshow(wordcloud\_neg\_in\_neg)



# wordcloud with bigram

import nltk

nltk.download('punkt')

nltk.download('wordnet')

from wordcloud import WordCloud, STOPWORDS

WNL = nltk.WordNetLemmatizer()

# Lowercase and tokenize

text = ip\_rev\_string.lower()

# Remove single quote early since it causes problems with the tokenizer.

text = text.replace("'", "")

tokens = nltk.word\_tokenize(text)

text1 = nltk.Text(tokens)

# Remove extra chars and remove stop words.

text\_content = [''.join(re.split("[ .,;:!?‘’``''@#$%^\_&\*()<>{}~\n\t\\\-]", word)) for word in text1]

# Create a set of stopwords

stopwords\_wc = set(STOPWORDS)

customised\_words = ["karthi","cinema","role","edge","seat","lokesh","paruthiveeran","kanagaraj","police station","actor","movie","director","daughter","film","make","life","tamil","injured","cop","doesnt","doesn", "meet","police","released","attempt","shooter",'month','thought','bought','months','night','back','days','thought','time','much','used','doesnt','review','feel','full','looks','takes','new','oneplus','will','even','plus','goes','like','still','got','dont','get','one','use','ised','phonesi','yrs','cant','just','also','using','since','overall','say','can','really','now',] # If you want to remove any particular word form text which does not contribute much in meaning

new\_stopwords = stopwords\_wc.union(customised\_words)

# Remove stop words

text\_content = [word for word in text\_content if word not in new\_stopwords]

# Take only non-empty entries

text\_content = [s for s in text\_content if len(s) != 0]

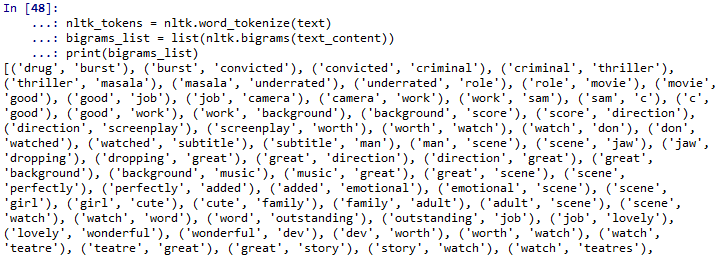
# Best to get the lemmas of each word to reduce the number of similar words

text\_content = [WNL.lemmatize(t) for t in text\_content]

nltk\_tokens = nltk.word\_tokenize(text)

bigrams\_list = list(nltk.bigrams(text\_content))

print(bigrams\_list)



dictionary2 = [' '.join(tup) for tup in bigrams\_list]

print (dictionary2)

# Using count vectoriser to view the frequency of bigrams

from sklearn.feature\_extraction.text import CountVectorizer

vectorizer = CountVectorizer(ngram\_range=(2, 2))

bag\_of\_words = vectorizer.fit\_transform(dictionary2)

vectorizer.vocabulary\_

sum\_words = bag\_of\_words.sum(axis=0)

words\_freq = [(word, sum\_words[0, idx]) for word, idx in vectorizer.vocabulary\_.items()]

words\_freq =sorted(words\_freq, key = lambda x: x[1], reverse=True)

print(words\_freq[:100])

# Generating wordcloud

words\_dict = dict(words\_freq)

WC\_height = 1000

WC\_width = 1500

WC\_max\_words = 200

wordCloud = WordCloud(max\_words=WC\_max\_words, height=WC\_height, width=WC\_width, stopwords=new\_stopwords)

wordCloud.generate\_from\_frequencies(words\_dict)

plt.figure(4)

plt.title('Most frequently occurring bigrams connected by same colour and font size')

plt.imshow(wordCloud, interpolation='bilinear')

plt.axis("off")

plt.show()

**Summary:-**

Overall unigram wordcloud :-



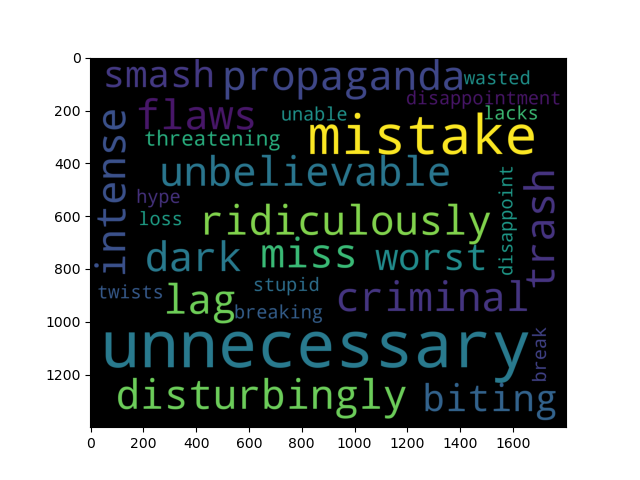
Positive unigram wordcloud:-

Most repeating positive words in reviews



Negative unigram wordcloud:-

Most repeating negative words in reviews



Bigram Wordcloud:-

Most of the most repeating bigram words in users reviews are positive. So the film is good one from users side.



1. **Write about the benefits/impact of the solution - in what way does the business (client) benefit from the solution provided?**

**Ans:-**

**The sentimental analysis on text data will helps the clients, even customers to know about the perticular products over all review . Thereby could analyse the product is gonna be succesful in market or not. And by analysing negative word cloud the clients can recognize what are the changes they have add in their product to win customers satisfaction.**

**Problem Statement: -**

In the era of widespread internet use, it is necessary for businesses to understand what the consumers think of their products. If they can understand what the consumers like or dislike about their products, they can improve them and thereby increase their profits by keeping their customers happy. For this reason, they analyze the reviews of their products on websites such as Amazon or Snapdeal by using text mining and sentiment analysis techniques.

Task 1:

1. Extract reviews of any product from e-commerce website Amazon.
2. Perform sentiment analysis on this extracted data and build a unigram and bigram word cloud.

Task 2:

1. Extract reviews for any movie from IMDB and perform sentiment analysis.

Task 3:

1. Choose any other website on the internet and do some research on how to extract text and perform sentiment analysis