**E- COMMERCE REVIEWS ANALYSIS WITH**

**MACHINE LEARNING**

**SYNOPSIS**

The most important after sales Empathizing session is done mainly through Reviews but Reading each and every review is not practically possible, So to overcome this pairwise ranking and sentiment analysis is conducted through automated Machine Learning based program, so E-Commerce applications provide an added advantage to customer to buy product with added suggestions in the form of reviews. Obviously, reviews are useful and impactful for customers those are going to a buy product. which will showcase only relevant reviews to the customers. This approach will sort reviews based on their relevance with the product and avoid showing irrelevant reviews. This work has been done in three phases- feature extraction, pairwise review ranking, and classification. The outcome is a sorted list of reviews, review ranking accuracy and classification accuracy. In Existing System, the dataset has to be manually analysed with individual scripts which is a very slow process to analyse the complete data set, also it is not cross-platform, Whereas in Proposed System it is completely GUI based, So it is easy to analyse any dataset in a single execution with multiple graphs and charts, also it is cross-platform.

**SOFTWARE SPECIFICATION**

|  |  |
| --- | --- |
| **COMPONENTS** | **REQUIREMENTS** |
| OPERATING SYSTEM | Windows 8 and Above |
| FRONT END | Python (TKinter) |
| ML DATA SET | Reviews Dataset in CSV |
| BACK END | Python (Seaborn) |
| NLP | NLTK |

**HARDWARE SPECIFICATION**

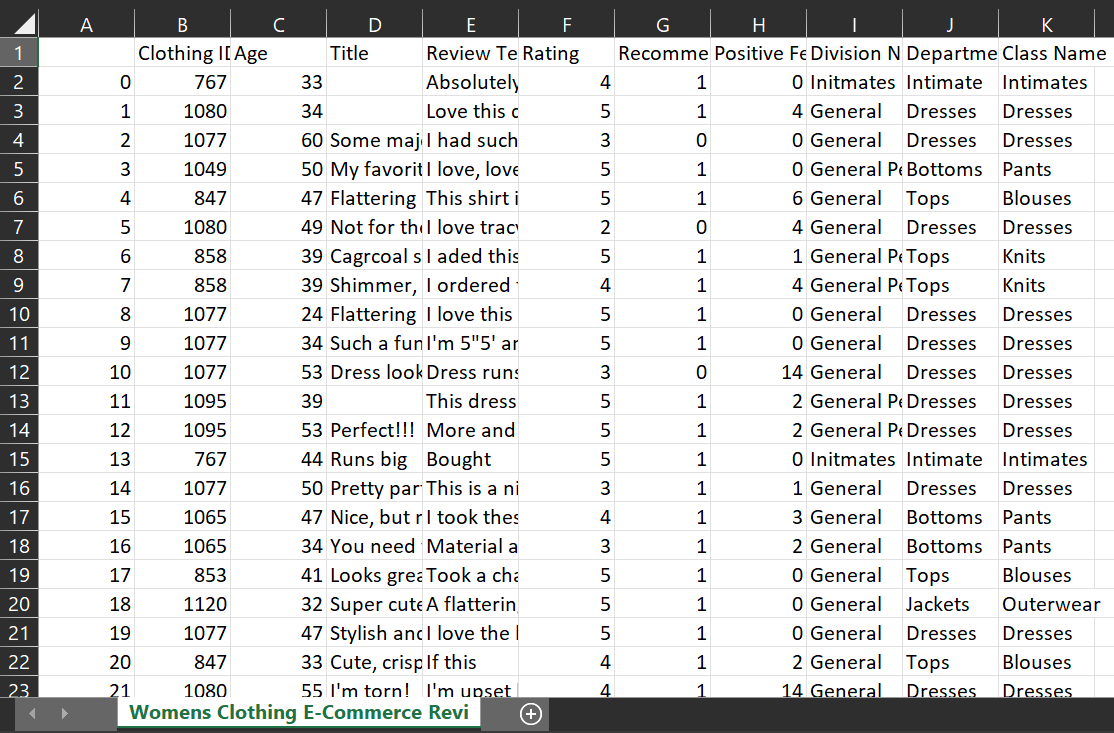
|  |  |
| --- | --- |
| **COMPONENTS** | **REQUIREMENTS** |
| CPU | x86 64-bit CPU (Intel / AMD architecture) |
| RAM | 2GB Minimum |
| STORAGE | 5GB Minimum disk space |
| PERIPHERALS | Common Peripherals (Mouse, Keyboard,..) |

**MODULE DESCRIPTION**

**Machine Learning – Data Set**

A dataset in machine learning is, quite simply, a collection of data pieces that can be treated by a computer as a single unit for analytic and prediction purposes. This means that the data collected should be made uniform and understandable for a machine that doesn’t see data the same way as humans do. For this, after collecting the data, it’s important to pre-process it by cleaning and completing it, as well as annotate the data by adding meaningful tags readable by a computer.

A tabular dataset can be understood as a database table or matrix, where each column corresponds to a particular variable, and each row corresponds to the fields of the dataset. The most supported file type for a tabular dataset is “Comma Separated File,” or CSV. But to store a “tree-like data,” we can use the JSON file more efficiently.

FIG 1 : DataSet – Reviews Set by Kaggle

The above dataset (FIG 1) includes 23486 rows and 10 feature variables. Each row corresponds to a customer review, and includes the variables:

* Clothing ID: Integer Categorical variable that refers to the specific piece.
* Age: Positive Integer variable of the reviewer’s age.
* Title: String variable for the title of the review.
* Review Text: String variable for the review body.
* Rating: Positive Ordinal Integer variable for the product score granted by the customer from 1 Worst, to 5 Best.
* Recommended IND: Binary variable stating where the customer recommends the product where 1 is recommended, 0 is not recommended.
* Positive Feedback Count: Positive Integer documenting the number of other customers who found this review positive.
* Division Name: Categorical name of the product high level division.
* Department Name: Categorical name of the product department name.
* Class Name: Categorical name of the product class name.

**NLP**

Natural language processing (NLP) refers to the branch of computer science and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

NLP combines computational linguistics—rule-based modelling of human language—with statistical, machine learning, and deep learning models. Together, these technologies enable computers to process human language in the form of text or voice data and to ‘understand’ its full meaning, complete with the speaker or writer’s intent and sentiment.

NLP drives computer programs that translate text from one language to another, respond to spoken commands, and summarize large volumes of text rapidly—even in real time. There’s a good chance you’ve interacted with NLP in the form of voice-operated GPS systems, digital assistants, speech-to-text dictation software, customer service chatbots, and other consumer conveniences. But NLP also plays a growing role in enterprise solutions that help streamline business operations, increase employee productivity, and simplify mission-critical business processes.

**Seaborn**

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures.

Seaborn helps to explore and understand the data. Its plotting functions operate on data frames and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

**Tkinter**

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

The tkinter package (“Tk interface”) is the standard Python interface to the Tcl/Tk GUI toolkit. Both Tk and tkinter are available on most Unix platforms, including macOS, as well as on Windows systems.

**Program**

Python program is used along with Seaborn, NLP, Tkinter and such packages to run the Software

# E-Commerce Reviews Analysis with Machine Learning

# By Sangeetha Priya RE

import matplotlib.pyplot as plt

import nltk

import numpy as np

import pandas as pd

import seaborn as sns

from wordcloud import WordCloud, STOPWORDS

import tkinter as tk

from tkinter import ttk

from tkinter import filedialog as fd

from tkinter.messagebox import showinfo

window = tk.Tk()

window.title('Sangeetha Priya RE - E-Commerce Review Processing By pairwise ranking and sentiment analysis')

Name = tk.Label(text="Select The DataSet")

Name1 = tk.Label(text="E-Commerce Review Processing By pairwise ranking and sentiment analysis")

Name2 = tk.Label(text="By Sangeetha Priya RE")

Name1.pack()

Name2.pack()

Name.pack()

def select\_file():

global filename

filename = 0

filetypes = (

('csv files', '\*.csv'),

)

filename = fd.askopenfilename(

title='Select a CSV File',

initialdir='/',

filetypes=filetypes)

showinfo(

title='Selected File',

message=filename

)

showinfo(

title='Processing...',

message='Close the "Select The DataSet" Dialog Box to Get Analytics'

)

open\_button = ttk.Button(

window,

text='Open a File',

command=select\_file

)

open\_button.pack(expand=True)

window.mainloop()

while (filename!=0):

df = pd.read\_csv(filename)

for column in ["Division Name","Department Name","Class Name","Review Text"]:

df = df[df[column].notnull()]

df.drop(df.columns[0], inplace=True, axis=1)

df['Label'] = 0

df.loc[df.Rating >= 3, ['Label']] = 1

df['Word Count'] = df['Review Text'].str.split().apply(len)

df.describe().T.drop('count', axis=1)

df[['Title', 'Division Name', 'Department Name', 'Class Name']].describe(include=['O']).T.drop('count', axis=1)

## Average Rating and Recommended IND by Class Name Correlation

key = 'Class Name'

temp = (df.groupby(key)[['Rating', 'Recommended IND', 'Age']]

.aggregate(['count', 'mean']))

temp.columns = ['Count', 'Rating Mean', 'Recommended Likelihood Count',

'Recommended Likelihood', 'Age Count', 'Age Mean']

temp.drop(['Recommended Likelihood Count', 'Age Count'], axis=1, inplace=True)

# Plot Correlation Matrix

f, ax = plt.subplots(figsize=[10, 7])

ax = sns.heatmap(temp.corr(),

annot=True, fmt='.2f', cbar\_kws={'label': 'Correlation Coefficient'})

ax.set\_title('Correlation Coefficient for Mean and Count for\nRating, Recommended Likelihood, and Age\nGrouped by {}'.format(key))

plt.xticks(rotation=45)

plt.tight\_layout()

plt.savefig('meanrating-recommended-classname-corr.png', format='png', dpi=300)

plt.show()

print('Class Categories:\n',df['Class Name'].unique())

g = sns.jointplot(y='Recommended Likelihood', x='Age Mean', data=temp,

kind='reg', color='b')

plt.subplots\_adjust(top=0.999)

g.fig.suptitle('Age Mean and Recommended Likelihood\nGrouped by Clothing Class')

plt.savefig('meanage-recommended-clothing.png', format='png', dpi=300)

plt.ylim(.7, 1.01)

# Working with Text

pd.set\_option('max\_colwidth', 500)

df[["Title","Review Text", "Rating"]].sample(7)

## Text Cleaning

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer

from nltk.tokenize import RegexpTokenizer

ps = PorterStemmer()

tokenizer = RegexpTokenizer(r'\w+')

stop\_words = set(stopwords.words('english'))

def preprocessing(data):

txt = data.str.lower().str.cat(sep=' ') #1

words = tokenizer.tokenize(txt) #2

words = [w for w in words if not w in stop\_words] #3

#words = [ps.stem(w) for w in words] #4

return words

import matplotlib as mpl

stopwords = set(STOPWORDS)

size = (20, 10)

def cloud(text, title, stopwords=stopwords, size=size):

mpl.rcParams['figure.figsize'] = (10.0, 10.0)

mpl.rcParams['font.size'] = 12

mpl.rcParams['savefig.dpi'] = 300

mpl.rcParams['figure.subplot.bottom'] = .1

wordcloud = WordCloud(width=1600, height=800,

background\_color='black',

stopwords=stopwords).generate(str(text))

fig = plt.figure(figsize=size, facecolor='k')

plt.imshow(wordcloud)

plt.axis('off')

plt.title(title, fontsize=50, color='y')

plt.tight\_layout()

plt.savefig('{}.png'.format(title), format='png', dpi=300)

plt.show()

def wordfreqviz(text, x):

word\_dist = nltk.FreqDist(text)

top\_N = x

rslt = pd.DataFrame(word\_dist.most\_common(top\_N),

columns=['Word', 'Frequency']).set\_index('Word')

mpl.style.use('ggplot')

rslt.plot.bar(rot=0)

def wordfreq(text, x):

word\_dist = nltk.FreqDist(text)

top\_N = x

rlst = pd.DataFrame(word\_dist.most\_common(top\_N),

columns=['Word', 'Frequency']).set\_index('Word')

return rlst

new\_stop = set(STOPWORDS)

new\_stop.update([x.lower() for x in list(df['Class Name'][df['Class Name'].notnull()].unique())]

+ ['dress', 'petite'])

# Cloud

cloud(text=df.Title[df.Title.notnull()].astype(str).values,

title='WordCloud for Titles',

stopwords=new\_stop,

size = (7,4))

title ='Most Frequent Words in Highly Rated Comments'

temp = df['Review Text'][df.Rating.astype(int) >= 3]

# Modify Stopwords to Exclude Class types, suchs as 'dress'

new\_stop = set(STOPWORDS)

new\_stop.update([x.lower() for x in list(df['Class Name'][df['Class Name'].notnull()].unique())]

+ ['dress', 'petite'])

# Cloud

cloud(text= temp.values, title=title,stopwords= new\_stop)

# Bar Chart

wordfreq(preprocessing(temp), 20).plot.bar(rot=45, legend=False, figsize=(15, 5), color='g',

title=title)

plt.ylabel('Occurrence Count')

plt.xlabel('Most Frequent Words')

plt.tight\_layout()

plt.savefig('most-freq-words-high-rate-comments.png', format='png', dpi=300)

plt.show()

# Low Raited

title ='Most Frequent Words in Low Rated Comments'

temp = df['Review Text'][df.Rating.astype(int) < 3]

# Modify Stopwords to Exclude Class types, suchs as 'dress'

new\_stop = set(STOPWORDS)

new\_stop.update([x.lower() for x in list(df['Class Name'][df['Class Name'].notnull()].unique())]

+ ['dress', 'petite', 'skirt', 'shirt'])

# Cloud

cloud(temp.values, title=title, stopwords=new\_stop)

reviews = df['Review Text'].astype(str).str.lower()

type(reviews)

features = reviews.tolist()

features

import re

from string import punctuation

for index in range(len(features)):

all\_text = ''.join([character for character in features[index] if character not in punctuation])

features[index] = re.split(r'\n|\r', all\_text)

features[index] = ' '.join([word for word in features[index]])

features

labels = np.array(df['Recommended IND'], np.int)

labels.shape

labels[labels == 1].shape[0]

labels[labels == 0].shape[0]

from keras.utils import to\_categorical

labels = to\_categorical(labels)

labels[:10]

from keras.preprocessing.sequence import pad\_sequences

from keras.preprocessing.text import Tokenizer

t = Tokenizer()

t.fit\_on\_texts(features)

vocabulary\_size = len(t.word\_index) + 1

print('Vocabulary size : {}'.format(vocabulary\_size))

encoded\_features = t.texts\_to\_sequences(features)

max\_length = 300

padded\_features = pad\_sequences(encoded\_features, maxlen=max\_length, padding='post')

embeddings\_index = dict()

with open('/home/darth/GitHub Projects/senti-internship-notes/abienagarap/word-vectors/glove.840B.300d.txt') as file:

data = file.readlines()

# store <key, value> pair of FastText vectors

for line in data[1:]:

word, vec = line.split(' ', 1)

embeddings\_index[word] = np.array([float(index) for index in vec.split()], dtype='float32')

print('Loaded {} word vectors.'.format(len(embeddings\_index)))

embedding\_matrix = np.zeros((vocabulary\_size, max\_length))

for word, i in t.word\_index.items():

embedding\_vector = embeddings\_index.get(word)

if embedding\_vector is not None:

embedding\_matrix[i] = embedding\_vector

words = []

for word, i in t.word\_index.items():

embedding\_vector = embeddings\_index.get(word)

if embedding\_vector is not None:

words.append(word)

print('{} words covered.'.format(len(words)))

percentage = (len(words) / vocabulary\_size) \* 100.00

print('{}% of {} words were covered'.format(percentage, vocabulary\_size))

def train\_test\_split(features, labels, \*\*kwargs):

# concatenate the features and labels array

dataset = np.c\_[features, labels]

# shuffle the dataset

np.random.shuffle(dataset)

# split the dataset into features, labels

features, labels = dataset[:, 0:max\_length], dataset[:, max\_length:]

# get the split size for training dataset

split\_index = int(kwargs['train\_size'] \* len(features))

# split the dataset into training/validation dataset

train\_features, validation\_features = features[:split\_index], features[split\_index:]

train\_labels, validation\_labels = labels[:split\_index], labels[split\_index:]

# get the split size for validation dataset

split\_index = int(kwargs['validation\_size'] \* len(validation\_features))

# split the validation dataset into validation/testing dataset

validation\_features, test\_features = validation\_features[:split\_index], validation\_features[split\_index:]

validation\_labels, test\_labels = validation\_labels[:split\_index], validation\_labels[split\_index:]

# return the partitioned dataset

return [train\_features, train\_labels], [validation\_features, validation\_labels], [test\_features, test\_labels]

train\_dataset, validation\_dataset, test\_dataset = train\_test\_split(features=padded\_features, labels=labels,

train\_size=0.60, validation\_size=0.50)

print('Dataset size : {}'.format(padded\_features.shape[0]))

print('Train dataset size : {}'.format(train\_dataset[0].shape[0]))

print('Validation dataset size : {}'.format(validation\_dataset[0].shape[0]))

print('Test dataset size : {}'.format(test\_dataset[0].shape[0]))

from keras import callbacks

from keras.layers import Bidirectional

from keras.layers import Dense

from keras.layers import Dropout

from keras.layers import Embedding

from keras.layers import LSTM

from keras.models import Sequential

from sklearn.model\_selection import StratifiedKFold

model = Sequential()

e = Embedding(vocabulary\_size, max\_length,

weights=[embedding\_matrix], input\_length=max\_length, trainable=False)

model.add(e)

model.add(Bidirectional(LSTM(256)))

model.add(Dropout(0.50))

model.add(Dense(2, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(train\_dataset[0], train\_dataset[1], epochs=32, batch\_size=256, verbose=1,

validation\_data=(validation\_dataset[0], validation\_dataset[1]))

score = model.evaluate(test\_dataset[0], test\_dataset[1], verbose=1)

print('loss : {}, acc : {}'.format(score[0], score[1]))

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

test\_predictions = model.predict(test\_dataset[0])

test\_predictions = np.argmax(test\_predictions, axis=1)

class\_names = ['(0) Not recommended class', '(1) Recommended class']

report = classification\_report(np.argmax(test\_dataset[1], axis=1), test\_predictions, target\_names=class\_names)

print(report)

conf\_matrix = confusion\_matrix(np.argmax(test\_dataset[1], axis=1), test\_predictions)

print(conf\_matrix)

plt.figure(figsize=(8, 8))

sns.heatmap(conf\_matrix, annot=True, annot\_kws={'size': 16}, cmap='coolwarm', fmt='.2f')

plt.savefig('conf\_matrix\_recommendation.png', format='png', dpi=300)

from sklearn.metrics import roc\_auc\_score

roc = roc\_auc\_score(y\_score=test\_predictions, y\_true=np.argmax(test\_dataset[1], 1))

print(roc)

from sklearn.metrics import auc

from sklearn.metrics import roc\_curve

fpr, tpr, \_ = roc\_curve(np.argmax(test\_dataset[1], 1), test\_predictions)

roc\_auc = auc(fpr, tpr)

print(roc\_auc)

plt.figure(figsize=(10, 7))

plt.plot(fpr, tpr, lw=2, color='darkorange', label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic example')

plt.legend(loc='lower right', fontsize=16)

plt.savefig('roc.png', format='png', dpi=300)

plt.show()

## Sentiment Classification

labels = np.array(df['Sentiment'])

labels

labels = np.array([2 if label == 'Positive' else (1 if label == 'Neutral' else 0) for label in labels],

np.int)

labels

positive\_class = int(labels[labels == 2].shape[0])

neutral\_class = int(labels[labels == 1].shape[0])

negative\_class = int(labels[labels == 0].shape[0])

df = pd.DataFrame.from\_dict({'positive': [positive\_class], 'negative': [negative\_class], 'neutral': [neutral\_class]})

plt.figure(figsize=(8, 8))

sns.set(font\_scale=2)

sns.set\_style('whitegrid')

ax = sns.barplot(data=df)

ax = ax.set\_xlabel('Frequency Distribution of Sentiment Classes')

labels = to\_categorical(labels)

train\_dataset, validation\_dataset, test\_dataset = train\_test\_split(features=padded\_features, labels=labels,

train\_size=0.60, validation\_size=0.50)

model = Sequential()

e = Embedding(vocabulary\_size, max\_length,

weights=[embedding\_matrix], input\_length=max\_length, trainable=False)

model.add(e)

model.add(Bidirectional(LSTM(256), merge\_mode='sum'))

model.add(Dropout(0.50))

model.add(Dense(3, activation='softmax'))

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(train\_dataset[0], train\_dataset[1], epochs=32, batch\_size=256, verbose=1,

validation\_data=(validation\_dataset[0], validation\_dataset[1]))

score = model.evaluate(test\_dataset[0], test\_dataset[1], verbose=1)

print('loss : {}, acc : {}'.format(score[0], score[1]))

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

test\_predictions = model.predict(test\_dataset[0])

test\_predictions = np.argmax(test\_predictions, axis=1)

class\_names = ['(0) Negative class', '(1) Neutral class', '(2) Positive class']

report = classification\_report(np.argmax(test\_dataset[1], axis=1), test\_predictions, target\_names=class\_names)

print(report)

conf\_matrix = confusion\_matrix(np.argmax(test\_dataset[1], axis=1), test\_predictions)

print(conf\_matrix)

plt.figure(figsize=(8, 8))

plt.savefig('conf\_matrix\_sentiment.png', format='png', dpi=300)

sns.heatmap(conf\_matrix, annot=True, annot\_kws={'size': 16}, cmap='coolwarm', fmt='.2f')

**TABLE DESIGN**

**ER Diagram**

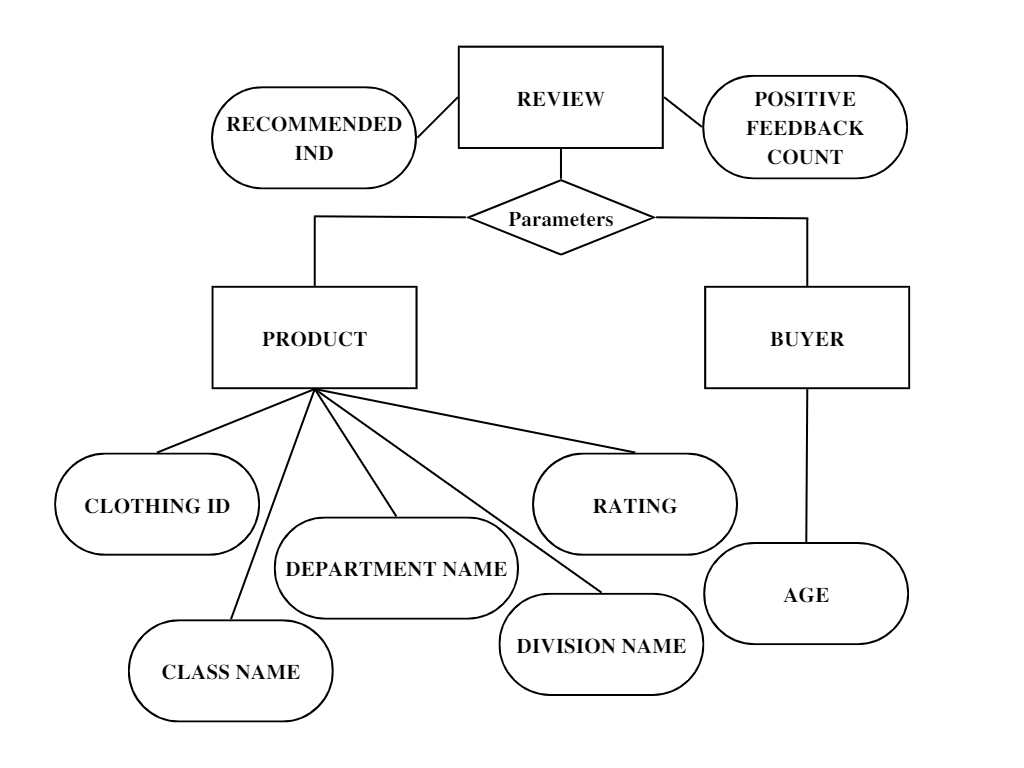
****

FIG 2 – Entity Relation Diagram of Dataset**.**

**DF Diagram**

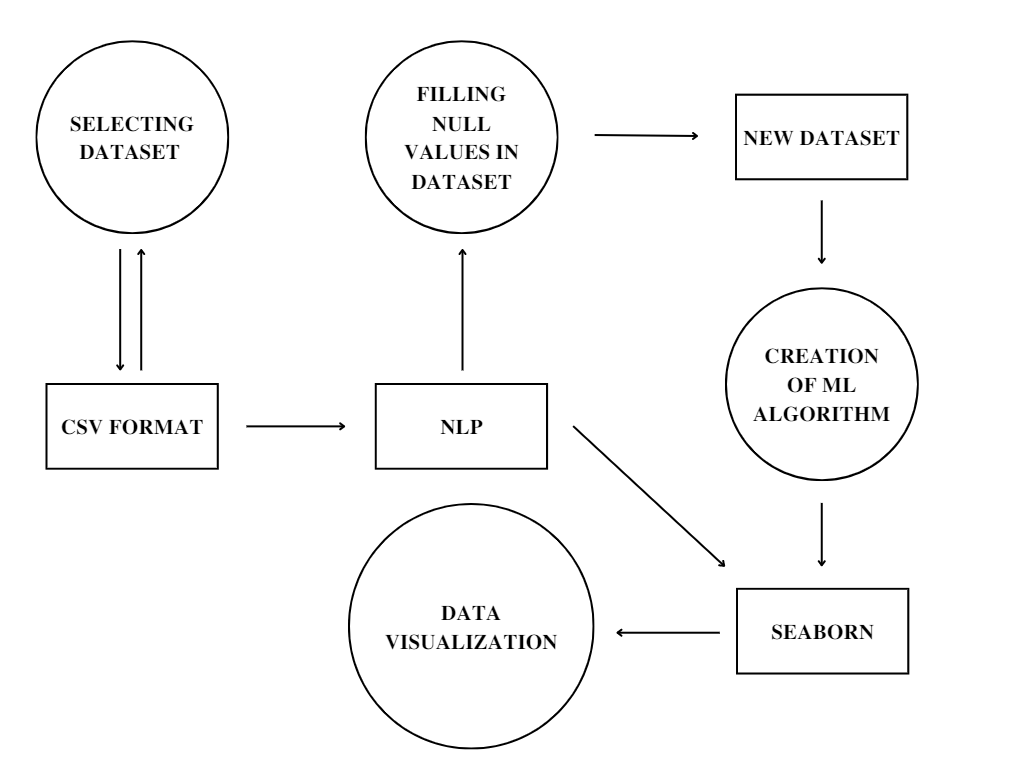
****

FIG 3 – Data Flow Diagram.

**OUTPUT**

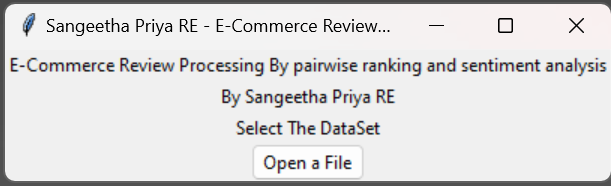
**** Using tkinter the GUI based prompt pops up to select the Dataset to process (Fig2)

FIG 2 : Dialog box to select dataset.

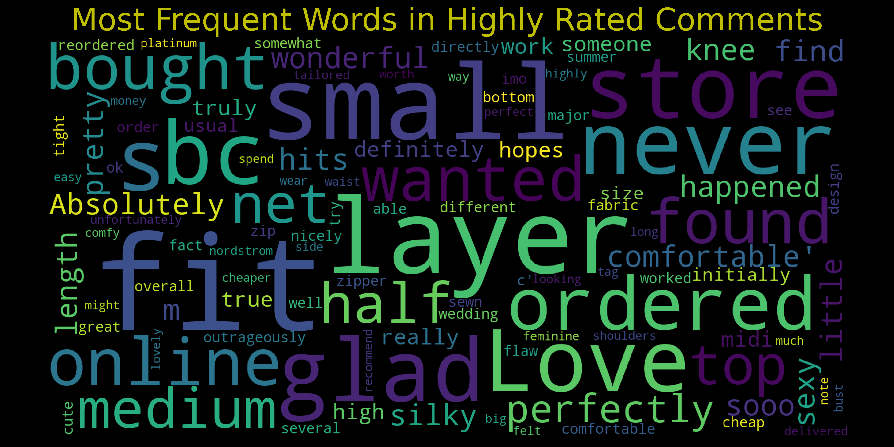
 Analysis models are produced as shown in Fig 3 & Fig 4

FIG 3 : Most Frequent words in Comments

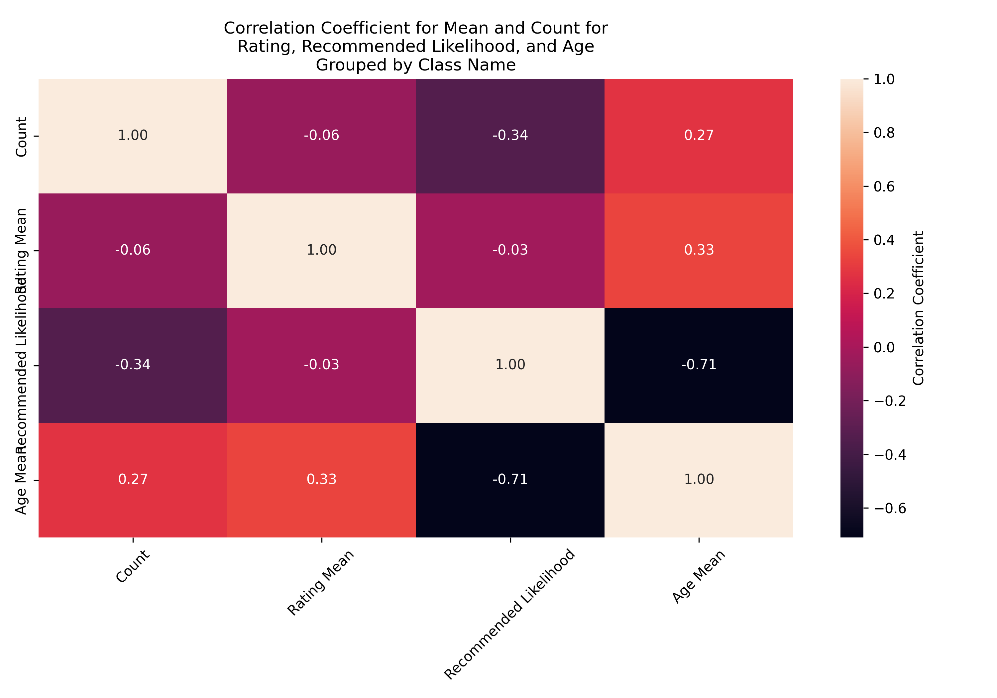


FIG 4 : Heatmap analysis for various factors of Dataset