CS6109 – Compiler Design – Mini Project

Feature Extraction from Comments

Reuel Samuel Sam – 2018103053
V.S. Suryaa -2018103610
Srihari S - 2018103601

College of Engineering – Guindy, Anna University, Sardar Patel Road, Chennai – 600025

---

1. Introduction

In this day and age, where the internet has taken control of most people's lives, social media exists as one of the most powerful platforms to share and voice one's opinions. People put up content looking for appreciation and constructive criticism from the general public to help improve their skills in their art. In these scenarios, users can speak freely on any relevant topic. The high volume of user-generated content makes a manual analysis of this discourse unviable. Consequently, automatic analysis techniques are needed to extract the opinions expressed in users' comments, given that these opinions are an implicit parameter of unquestionable interest for a vast section of the society. These posts and the corresponding comments left on one's handle, assist people in decision making under various situations based on the opinions of other people. Many a times these comments influence one's thought process while making decisions due to higher number of likes and comments viewed on such posts. However, these responses usually in the form of comments, are at times more negative and positive and thereby may cause some form of emotional pain to the original content poster. Our project aims to automate this task of analysing the reactions on the posts and generate a report based on the outcome. This project tries to tackle this situation head on by segregating the comments left on posts into three main categories - positive, neutral and negative. The comments are analysed using a Lexicon based sentiment analysis approach that breaks down the comment to its constituent parts and provides the user an output with desirable filtered comments. Socially concise individuals use this platform to impact lives. Many comments on a post tend to be interpreted is a number of ways that contrast with what was intended. In order to overcome this ambiguity

in expression, the viewers of the post are provided a feature to predict the number of likes their comment would yield based on the post and existing comments. Decision to express or not is in their own hands.

As in today's world, everyone shares their emotions online, through social media platforms and social networking has left no section of the society untouched. Thousands of profiles are created every day on these websites. It has rapidly become the best platform for not only connecting with people but also to form and maintain relationships. With their tremendous outreach, these websites have become the most convenient way to share media as well as important information. These days Facebook, twitter, LinkedIn, Pinterest and Instagram are the most eminent social networking websites all across the globe. These social networking websites generate enormous amount of data in the form of posts, comments and likes. Rather than keeping this data in the data warehouse, this data can be analyzed to optimize the social networking posts. Sentiment analysis is one such technique that can be used for this purpose. It refers to the analysis of the text in order to identify the emotions conveyed through the text by the writer. Sentiment analysis can be done using two approaches, that is, the Lexicon based approach and Machine Learning approach. This project integrates both the approaches to satisfy the user's needs. People express their opinion on the social networking websites through likes and comments. Users may either like/comment on a post or do both. Thus, the data generated by these platforms can be used for the analysis of the sentiments expressed by the users on various posts. Existing social media platforms don't give us the privilege to track the activities of the users and analyze the user's behaviour for future predictions such as what to post when to post and whom to target. Reports can be generated by the analyses done on the data of the users on the platform. According to the posts and the actions of the users, the data that is generated is analyzed by the sentiment model. In a corporate world or a university, it can be considered important to keep track of the related population's sentimental behaviour towards the institution as it gives a great amount of detail on how a user feels about being a part of that institution and in what way should the institution engage further with the concerned user. A social media platform with the ability to perform sentiment analysis and produce a report for the 'high level' users is not a mere content sharing platform anymore. It becomes a full-fledged authoritative tool that would facilitate the decision-making process.

Keywords or phrases: Compiler Design, Python, Comment analysis, Neural Network, Database, Lexical Analysis

2. Related Works

1) Used a Lexicon based sentiment analysis to analyse the emotion in the comments and provide statistics of likes and dislikes. Proposed a methodology to identify the polarity of popularity without much hassle and effort. The work also took into consideration the overall impact of different parts of speeches in grammar. But the approach is very rudimentary to analyse the sentiment by taking same polarity for a number of words and thus not indulging the intricacies. Assumed that the user interacts with proper grammar and avoids spelling check.

2) Mainly composed of two modules – Focus Detection and Sentiment Analysis. It analyzed each sentence in the comments to create a set of tuples that abstractly represents users' opinions by assigning sentiment expressions to opinion focuses. Contained colloquial expressions that are of crucial importance to analysis and a hierarchical structure to improve results of focus detection. But was unable to detect hints of sarcasm and satire in comments. Also, the requirement of sufficient information is a must.

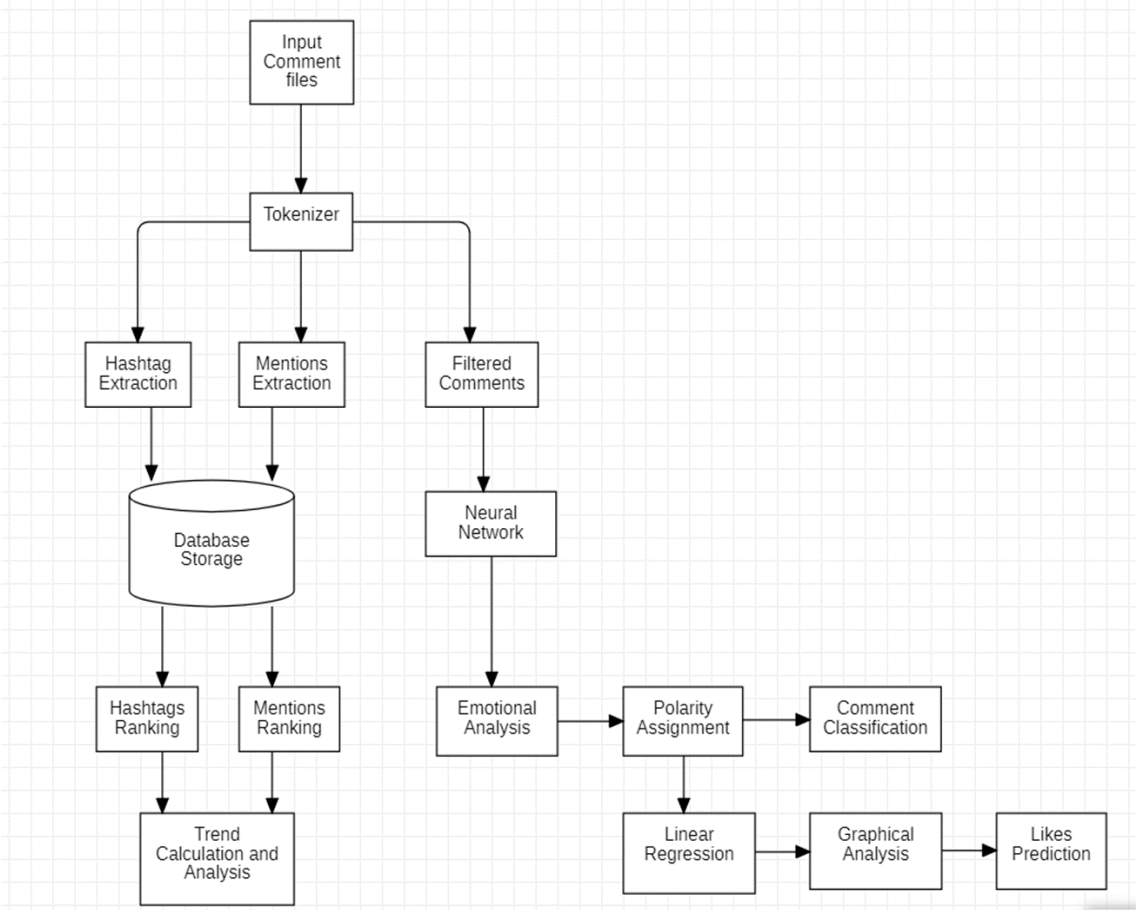
3) Adopted LICW – text analysing software along with SVM, maximum entropy and Naïve Bayes models. Notified close circle and doctors about the extreme emotions of the user based on his reactions to posts thereby ensuring that improper decisions aren't taken. The users are individually rated based on their response to other posts. The impact of an individual post and its response hasn't been taken care of.

4) Analyzed the different cultural variations, take out emotions, and obtained the sentiment behind ML techniques, comparing them on sentence-level for depression measurement. Proposed a multi-class emotion classifier that determined emotions with greater precision. Meta Learning tends to be in-accurate and is problematic with timing and maintenance with complex models.

5) Dealt with Surveying opportunities and challenges in Online Social Networking with emphasis on the most popular users. Receiving personalized posts based on liked ones. Focus on semantic security and coping up phishing attacks with celebrities. Combining the views of the close circle general public, makes the media uninviting. Ways to filter responses from comments not discussed.

---

3. System Architecture



---

4. Modules

Module 1: Tokenization of the comments:

The comments left on a post of a user are entered in a file x.txt for a post "x". The likes obtained by each of the comment are also entered in the same file. This module is composed of three different phases each having the same input file x.txt, returning an output file each accomplishing different tasks.

Phase 1: Extraction of Mentions

An account is considered mentioned if it is of the form @account_name. An account name can be one or more letters or digits or a combination of the two. Underscore '_' and period '.' is also permissible. A mention is extracted if it doesn't start with a letter or a digit, followed by "@" which is then followed by an account name and any delimiter. The following code snippet accomplishes this functionality. A file x_mentions.txt is created which contains the list of accounts that have been mentioned in a comment of post "x". The details in x_mentions.txt are stored in the format:

PostID;account_name;CommentNumber

```
letter [a-zA-Z_.]
digit [0-9]
delim [ \t]
tag ({letter}|{digit})+
%%
[^{letter}|^{digit}]"@"{tag}{delim} {
bzero(res,sizeof(res));
yyless(yyleng-1);
strncpy(res,yytext+2,yyleng-2);
fprintf(yyout,"%s;%s;%d\n",file_name,res,line_count);
REJECT;
}
"\n".*"\n" {line_count++;REJECT;}
%%
```

Phase 2: Extraction of Hashtags

Any word is considered as a hashtag if it is of the form #word. A word can be one or more letters or digits or a combination of the two. Underscore '_' and period '.' is also permissible. An hashtag is extracted if it starts with a delimiter,

followed by "#" which is then followed by a word and any delimiter. The following code snippet accomplishes this functionality. A file x_hashes.txt is created which contains the list of hashtags in a comment of post "x". The details in x_hashes.txt are stored in the format:

PostID;word;CommentNumber

```
letter [a-zA-Z_]
digit [0-9]
tag ({letter}|{digit})+
delim [ \t]

%%
[ \n]"#"{tag}{delim} {
        bzero(res,sizeof(res));
        yyless(yyleng-1);
        strncpy(res,yytext+2,yyleng-2);
        fprintf(yyout,"%s;%s;%d\n",file_name,res,line_count);
        REJECT;
}
"\n".*"\n" {line_count++;REJECT;}
%%
```

Phase 3: Extraction of Filtered Comments

A file x_filtered_comments.txt is created which contains the comments of post "x" filtered by removing mentions, punctuations, proper nouns and stores the likes obtained by that comment. A word is considered as a proper noun, if it begins with an UpperCase letter and doesn't occur at the beginning of a sentence (or) if it is all UpperCase letters. The details in x_ filtered_comments.txt are stored in the format:

LikesObtained;filteredComment;CommentNumber

```
digit [0-9]
number {digit}+
punctuation [.!?`,;:#@]+
caps [ ][A-Z]+[ ]
first [ ][A-Z][a-z]+{punctuation}[ ]

%%
^{number}"LIKES " {
        int i=0;
        while(yytext[i]!='L'){
                fprintf(yyout,"%d",yytext[i++]-48);
        }
        fprintf(yyout,";");
        yyless(yyleng-1);
}
{caps} {
        fprintf(yyout," ");
        yyless(yyleng-1);
}
{first} {
        yyless(yyleng-1);
}
{punctuation} {
        fprintf(yyout," ");
}
```

Module 2: Database Storage

Once the comments are filtered by *Module-1*, it is necessary to store the results in a database for easier access and purpose of organization. Furthermore, this allows for easier implementation of numerous other features – finding max count, counting specific occurrence etc.

To do so, the input files are the output files taken from *Module-1*.

Input:

- x_mentions.txt
- x_hashtags.txt

Phase 1: Population of Database

The input files need to be separated in required fashion and then stored in database. To do so, each input of the file is split using ';' as the delimiter. The result is of the form *postno;tag;commentno.*

This is then split into a class and then tossed into a table present in the database. However, duplicate tags need to be avoided. This has been accounted for by considering duplicate tags as those with the same comment number, tag and post number.

Code: Population of database

```
def updatedatabase(conn, List, table):
    c = conn.cursor()
    for i in range(len(List)):
        cursor = conn.execute("Select * from %s where
content = '%s'"%(table, List[i].content))
        flag = False
        for row in cursor:
            if row[0].lower() == List[i].content.lower()
and row[1] == List[i].cno and row[2] == List[i].pno:
                flag = True
                break
        else:
            flag = False

        if flag == True:
            pass
            #print("Duplicate comment: ",row)
```

```
            else:
                c.execute("INSERT INTO %s
    VALUES('%s',%d,%d)"%(table,List[i].content.lower(),int(Lis
    t[i].cno),int(List[i].pno)))
                #print("Record inserted")

            conn.commit()
```

Phase 2: Finding Count

One of the features we have implemented involves the need to find the count of tagged mention or comment on a particular post as specified by the user. To do so, the database helps enormously to simplify the search and display of results.

Code: Finding Count of a key

```
def countoccurance(conn, key, table):
    c = conn.cursor()
    res = c.execute("SELECT COUNT(content) FROM %s
WHERE content='%s'"%(table, key.lower()))
    for row in res:
        return row[0]
    else:
        return 0
```

Phase 3: Displaying maximum occurring tag

Another feature we have implemented in this project is the addition of t. To do so, the database helps enormously to simplify the search and display of results and in finding the most tagged hashtag or mention in a particular post. Using of a database highly simplifies this search.

Code: Displaying Maximum occurring tag

```
def findmax(conn,table):
    c = conn.cursor()
    res = c.execute("SELECT DISTINCT content,
COUNT(content) as count FROM %s GROUP BY content
ORDER BY count DESC"%table)
    count_list = list()
    flag = False
    for row in res:
        flag = True
        count_list.append(row)
    if flag == False:
        count_list.append(None)
        return -1

    table = tabulate(count_list,
headers=["Title","Count"])
    print(table)
    return count_list[0], table
```

To note: Our database is not necessarily smart. It is the SQL statements that make it easier to process such data.

Module 3: Emotion Calculation

Input:

A text file containing the filtered comment with number of likes and serial number of comments

Goal:

- Phase-1 goal:

  Emotional analysis of the filtered comments using neural network to extract crux emotion of the post

- Phase-2 goal:

  Predicting the number of likes for a comment with features observed and extracted in previous phase

AFINN:

AFINN sentiment analysis in Python: Wordlist-based approach for sentiment analysis. Developed and curated by Finn Nielsen, you can find more details on this lexicon in the paper, "A new ANEW: evaluation of a word list for sentiment analysis in microblogs", proceedings of the ESWC 2011 Workshop. The current version of the lexicon is AFINN-en-165. txt and it contains over 3,300+ words with a polarity score associated with each word.

Each word has a label in the range of [−4,4], where −4is the most **negative** and 4 is the most **positive**. Most of the positive words were labelled with +2 and most of the negative words with –2, see the histogram in Figure 1. I typically rated strong obscene words, e.g., as listed in, with either –4 or –5. The word list has a bias towards negative words (1598, corresponding to 65%) compared to positive words (878). A single phrase was labelled with valence 0.

Code: Generating score for comments:

```
from afinn import Afinn
import csv

# text file containing filtered comments
file1 = open('comments.txt', 'r')
Lines = file1.readlines()
# neural network used for emotion analysis
analyze = Afinn()

row = []

for line in Lines:
    # extracting data required for analysis
    output = line.split(';')
    # polarity of each word is analyzed to find the
overall emotion of comment
    score = analyze.score(output[1])
```

Input File: comments.txt

1; The best day in Karans show ;1
2; had better plans with har n12 founders23 ;2

Intermediate results:

```
print("Comment:" + output[1].strip() + "\nScore:" +
str(score))
```

Output:

- Comment: The best day in Karans show
  Score:3.0
- Comment: design rules the world isn't it sammer undou bted come on agree
  Score:1.0
- Comment: fun fact it is necessary to have a space endl after each comment
  Score:4.0

Code: Preparing data for phase 2

```
l1 = [output[0], output[1], output[2].strip(),
score]
    # compiling data for all comment
    row.append(l1)
# output file for phase 1
filename = "output.csv"
# categorizing and writing data
fields = ['no-of-likes', 'comments', 'serial',
'score']
# writing to csv file
with open(filename, 'w') as csvfile:
    # creating a csv writer object
    csvwriter = csv.writer(csvfile)

    # writing the fields
    csvwriter.writerow(fields)

    # writing the data rows
    csvwriter.writerows(row)
```

Code: Pre-process data for phase 2

```
import numpy as np
import matplotlib.pyplot as plt  # To visualize
import pandas as pd  # To read data
from sklearn.linear_model import LinearRegression

# load data set
data = pd.read_csv('output.csv')

# features to fit the linear regression
Y = data.iloc[1:, 0].values.reshape(-1, 1)  # values
converts it into a numpy array
X = data.iloc[1:, 3].values.reshape(-1, 1)  # -1
means that calculate the dimension of rows, but have
1 column
```

Output File: output.csv

no-of-likes,comments,serial,score
1, The best day in Karans show ,1,3.0
2, had better plans with har n12 founders23 ,2,2.0

Linear Regression:

The representation is a linear equation that combines a specific set of input values (x) the solution to which is the predicted output for that set of input values (y). As such, both the input values (x) and the output value are numeric.

The linear equation assigns one scale factor to each input value or column, called a coefficient and represented by the capital Greek letter Beta (B). One additional coefficient is also added, giving the line an additional degree of freedom (e.g. moving up and down on a two-dimensional plot) and is often called the intercept or the bias coefficient.

For example, in a simple regression problem (a single x and a single y), the form of the model would be:

$$y = B0 + B1*x$$

In higher dimensions when we have more than one input (x), the line is called a plane or a hyper-plane. Linear regression assumes that the relationship between your input and output is linear. It does not support anything else. We may need to transform data to make the relationship linear. Linear regression will make more reliable predictions if the input and output variables have a Gaussian distribution. Linear regression assumes that your input and output variables are not noisy. Consider using data cleaning operations that let better exposure and clarify the signal in your data.

Code: Fitting a linear regressor as a predictor

```
linear_regressor = LinearRegression()  # create
object for the class Linear Regression loaded
using sklearn

# training using linear regression
linear_regressor.fit(X, Y)  # perform linear
regression
Y_pred = linear_regressor.predict(X)  # make
predictions

# visualization of training data and prediction
plt.scatter(X, Y)
plt.plot(X, Y_pred, color='red')
plt.show()
```

Visualization of the prediction

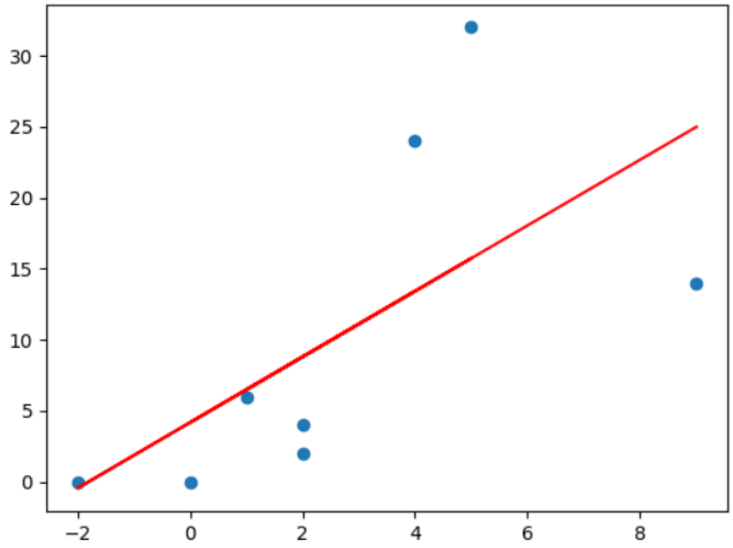


Fig 1: Graphical Representation - Likes vs Score

Score	Actual Likes	Predicted Likes
2	2	8.80438
1	6	6.49139
4	24	13.4304
-2	0	-0.447574
5	32	15.7433
0	0	4.1784
2	4	8.80438
9	14	24.9953

Fig 2: Tabular Representation - Likes vs Score

Module 4: Graphical User Interface

The final module in this project is the one that ties everything all together – An Interface. The main challenge here is the inclusion of different language files. *Module-1* being in C/C++ and *Module 2 and 3* being in Python did cause issues when it came to combining them all in one. To overcome this hurdle, the C code was cross compiled to make it compatible for Windows. The executable .exe file was then executed as a child process of the GUI which was written in Python.

For the interface creation, Python's PyQt5 package has been used. The functionality has been described later in this description.

There are mainly 5 windows (excluding message boxes):

- Opening Main Window
- Features Window
- Count Occurrence Window
- Most Tagged Window
- Post Emotion Window

Phase 1: Main Window

The user is welcomed to our project via this window. Here, they are prompted to enter a text file containing comments in the format specified in *Module-1*. On submission, the input file is then provided as an input to *Module-1* which then executes as a child process and produces the required output file. Once the child processes terminate, *Module-2* is called and the database is updated.

Code: Button definitions

```
        def selectFile(self, Window1):
            #self.filepath
            fp = QFileDialog.getOpenFileName()
            self.filepath = fp[0].split("/")[-1]
            global postno
            postno = self.filepath
            print(self.filepath)

        def submit_button_pressed(self, window):
            tmp=subprocess.Popen("hashes.exe " +
    self.filepath,shell=True,stdout = subprocess.PIPE)
            tmp.wait()
            result = []
            for line in tmp.stdout:
                result.append(line)
```

```
        tmp=subprocess.Popen("mentions.exe " +
self.filepath,shell=True,stdout = subprocess.PIPE)
        result = []
        for line in tmp.stdout:
            result.append(line)
        tmp.wait()

        tmp=subprocess.Popen("filter_comments.exe " +
self.filepath,shell=True,stdout = subprocess.PIPE)
        result = []
        for line in tmp.stdout:
            result.append(line)
        tmp.wait()

        fd = open("post_no.txt","w")
        fd.write(self.filepath.split(".")[0])
        fd.close()

        import emotion_analysis

        print("Done")
        self.update_db()
        self.openWindow(2, window)


    def update_db(self):
        global conn
        post = self.filepath.split(".")[0]
        mfile = post+"_mentions.txt"
        hfile = post+"_hashtags.txt"
        mList = openfile(mfile)
        hList = openfile(hfile)
        updatedatabase(conn, mList, TABLES[0])
        print("MENTIONS updated")
        updatedatabase(conn, hList, TABLES[1])
        print("HAHSTAGS updated")
```

Phase 2: Features Window

The user is prompted to pick between three of our features. This acts as a gateway window to the other windows that do the actual computation. The features are:

- Count Occurrence
- Find Max tagged
- Check Post Emotion

Phase 3: Count Occurrence Window

In this phase, a key is entered by the user and a tag-type is chosen. *Module-2* is then called based on the tag-type and the count of the entered key is then returned.

Code: Find Button Definition

```
        def find_button_pressed(self):
            print("FIND PRESSED")
            global conn
            global TABLES
            choice = self.type_list.currentText()
            index = self.type_list.findText(choice,
    QtCore.Qt.MatchFixedString)
            key = self.key_input.text()
            count = countoccurance(conn, key, TABLES[index])
            msg = QMessageBox()
            msg.setWindowTitle("Count Occurance Result: ")
            msg.setText("Finished Searching Database:")
            msg.setIcon(QMessageBox.Information)
            msg.setDefaultButton(QMessageBox.Ok)
            msg.setInformativeText("Count of \'"+key+"\':
    "+str(count))
            msg.exec_()
            print("Count of \'%s\': %d"%(key,count))
```

Phase 4: Most Tagged Window

In this phase, the user is prompted to choose between the two tag types. *Module-2* is then called and the most tagged mention/hashtag is presented in a message box.

Code: Find Button Definition

```
        def confirm_button_pressed(self):
            print("CONFIRM PRESSED")
            global conn
            global TABLES
            choice = self.type_list.currentText()
            index = self.type_list.findText(choice,
    QtCore.Qt.MatchFixedString)
            mcontent, table = findmax(conn,TABLES[index])
            msg = QMessageBox()
            msg.setWindowTitle("Most Tagged Result: ")
            msg.setText("Finished Searching Database:")
            msg.setIcon(QMessageBox.Information)
            msg.setDefaultButton(QMessageBox.Ok)
            msg.setInformativeText("Maximum Occuring
    "+TABLES[index]+" is \'"+mcontent[0]+"\' with count of
    "+str(mcontent[1]) )
            msg.setDetailedText(table)
            msg.exec_()
```

Phase 5: Post Emotion Window

This phase involves the inclusion of *Module-3* and its features. The user is allowed to view the comments filtered as neutral, optimistic or pessimistic. Another added feature allows the user to enter a new comment and get a response with the predicted number of likes the comment may fetch after studying the trend on the specific post.

Code: Displaying of post emotion

```
        def confirm_button_pressed(self):
            global postno

            choice = self.type_list.currentText()
            index = self.type_list.findText(choice,
    QtCore.Qt.MatchFixedString)

            print(postno, choice, index)
            self.display_comments(index)


        def display_comments(self,flag):
            with open('output.csv', newline='\n') as csvfile:
                rec = []
                spamreader = csv.reader(csvfile, delimiter=',',
    quotechar='|')
                for row in spamreader:
                    temp = row[2]
                    row[2] = row[0]
                    row[0] = temp
                    rec.append(row)
            new_rec = []
            for i in range(1,len(rec)):
                if flag == 1:
                    if int(rec[i][3]) > 0:
                        new_rec.append(rec[i][:-1])
                elif flag == 0:
                    if int(rec[i][3]) == 0:
                        new_rec.append(rec[i][:-1])
                else:
                    if int(rec[i][3]) < 0:
                        new_rec.append(rec[i][:-1])
            table = tabulate(new_rec)
            msg = QMessageBox()
            msg.setStyleSheet("QLabel{min-width: 600px;}");
            msg.setWindowTitle("Result: ")
            msg.setText("Finished Searching Database:")
            msg.setInformativeText(table)
            msg.exec_()
```

Code: Predicting likes of a new comment

```
        def predict_button_pressed(self):
            print("PREDICT PRESSED")
            comment = self.comment_input.text()
            print(comment)
            fd = open("new_comment.txt","w")
            fd.write(comment)
            fd.close()
            import likeprediction
            import importlib
            importlib.reload(likeprediction)
            file1 = open("new_comment_likes.txt", 'r')
            Lines = file1.readlines()
            file1.close()
            likes = Lines[0]
            print(likes)
            msg = QMessageBox()
            msg.setWindowTitle("Result: ")
            msg.setText("Finished Calculating Predicted number of
    likes: "+likes)
            msg.setDefaultButton(QMessageBox.Ok)
            msg.exec_()
```

5. Result and Discussion

On combining all the modules, we arrived at a fully operational project that crosses off the goals we had set out to reach. As described in the previous sections, the user first inputs a text file which is then tokenized and the results are then stored in a database. As per the choice of the user, the required information is displayed while maintaining a reasonable level of abstraction. Furthermore, the post emotion is then calculated and the comments are segregated based on a score. User can then input a new comment and predict the number of likes this comment would fetch after observing the trend of the existing comments on the mentioned post. The result and implementation screenshots are provided in our presentation document breaking down the code and each module in detail.

6. Conclusion, Future Scope and Final Summary

Further analysis to find dislikes to remove ambiguity of likes (Users who make a negative comment and like the post) can be carried out. A deeper approach can be taken by finding adjective, adverb and other part of speeches and normalizing the polarity of the word accordingly. This is taken into consideration because some part of speeches tends to make a stronger impact on meaning than others. We also wish to be able to make a more easy-to-use interface that is flexible. Our longer vision ahead is to build a small-scale social media platform and let users interact and extract further details to make an in-depth analysis.

In conclusion, this project is widely applicable in today's world as this helps tackle important issues regarding mental health caused by social media influence. For further details, contact the authors mentioned at the start.

7. References

[1] R. Singh, R. Bagla and H. Kaur, "Text analytics of web posts' comments using sentiment analysis," 2015 International Conference and Workshop on Computing and Communication (IEMCON), Vancouver, BC, 2015, pp. 1-5, doi: 10.1109/IEMCON.2015.7344534.

[2] A.Moreo, M.Romero, J.L.Castro, J.M.Zurita – 2012, "Lexicon-based Comments-oriented News Sentiment Analyzer system".

[3] D. Tanna, M. Dudhane, A. Sardar, K. Deshpande and N. Deshmukh, "Sentiment Analysis on Social Media for Emotion Classification," 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2020, pp. 911-915, doi: 10.1109/ICICCS48265.2020.9121057.

[4] A. U. Hassan, J. Hussain, M. Hussain, M. Sadiq and S. Lee, "Sentiment analysis of social networking sites (SNS) data using machine learning approach for the measurement of depression," 2017 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, 2017, pp. 138-140, doi: 10.1109/ICTC.2017.8190959.

[5] Persia, F., & D'Auria, D. (2017). A Survey of Online Social Networks: Challenges and Opportunities. 2017 IEEE International Conference on Information Reuse and Integration (IRI).

[6] arXiv:1103.2903 - A new ANEW: Evaluation of a word list for sentiment analysis in microblogs - Finn Årup Nielsen

[7] K. M. Anitha, V. Saraswathy, K. Gayathri and S. S. Priya, "An Approach To Comment Analysis In Online Social Media," 2019 3rd International Conference on Computing and Communications Technologies (ICCCT), Chennai, India, 2019, pp. 332-335, doi: 10.1109/ICCCT2.2019.8824949.

[8] X. Lin, T. Bu, Q. He and Y. Liang, "A study on the impact of positive and negative comments on consumers' perceived usefulness," 2019 16th International Conference on Service Systems and Service Management (ICSSSM), Shenzhen, China, 2019, pp. 1-6, doi: 10.1109/ICSSSM.2019.8887617.