Group 2

Depressive text classification.

7th May 2021

TEAM MEMBERS

- Davith Lon
- Bryan Khoo
- Ami Khalsa
- Ashish Pant
- Saurav Pawar

Project Description

Introduction

With a global pandemic in place, everyone is advised to be isolated to prevent the further spread of COVID. Along with schools, work places, and restaurants closing down, people have no choice but to stay sheltered with minimal interaction. This builds up frustration and depressive thoughts. Fortunately, with the advancements of technology, we are able to express our emotions on the internet. Twitter is one of the social media platforms that allows users to express their thoughts[1].

However, because of the popularity of Twitter, it may be easy for “tweets” that are seeking help to get lost in the sea of data. In this project, we will be building a text classifier that will take in a text and classify the text as depressive or not. We believe that with this classifier, it will be easier to identify those “tweets" who are depressive and provide appropriate help to the user.

Background

[1] https://esrc.ukri.org/research/impact-toolkit/social-media/twitter/what-is-twitter/

Twitter is widely used for research. One of the research done recently was on migraine tweets[2]. The methods used in this research, such as natural language processing, will be observed and examined to verify whether they are appropriate for our project.

A research named “Online suicide prevention through optimised text classification”[2]. Has already been done in the past; it focuses on classification of text that expresses suicidal thoughts. It works on a Dutch-language forum post to detect suicidality.

Another similar research named “Potential use of text classification tools as signatures of suicidal behavior: A proof-of-concept study using Virginia Woolf’s personal writings”[3]. This research conducted text analysis and model creation based on the diary of Virgina Woolf prior to her suicide. In this research they were able to create a Naive Bayes model that had an 80.45% accuracy.

This increment is an extension of the previous increment. It will include data extraction, data cleaning, analysis and visualization of the analysis.

GOALS AND OBJECTIVES

Motivation

Depression is “a common and serious medical illness that negatively affects how you feel, the way you think and how you act.”[4] The symptoms of depression ranges from changes in appetite, loss of interest or pleasure in activities once enjoyed, thoughts of death or suicide, and etc.[5] Leaving these symptoms unidentified and untreated has unfortunately claimed the lives of 800,000 people every year.[6] It also has lasting impacts on education. High school students with recent symptoms of depression are more than twice as likely as their peers to drop out.[5] This is troubling, because during the COVID-19 Pandemic there has been an elevated amount of adverse mental health conditions with depression being the lead condition. This is why our project wants to investigate the significance of variables that cause depression to create a better understanding of their impacts.

Significance

---

[2] https://journals.sagepub.com/doi/full/10.1177/2515816319898867
[3] https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0204820
[4] https://www.psychiatry.org/patients-families/depression/what-is-depression
[5] https://www.nimh.nih.gov/health/topics/depression/index.shtml
[6] https://www.who.int/news-room/fact-sheets/detail/depression
[5]https://pubmed.ncbi.nlm.nih.gov/29195763/

The following are the reasonings for the significance of this project.

1. The project can help health organizations to determine whether a particular text is depressed and take appropriate actions based on the result.
2. A text classification model that can score how depressed a piece of text could be used for monitoring service.
3. The project can help provide infographics that medical organizations can use to educate the public.
4. “Globally, more than 264 million people of all ages suffer from depression.”[3] So, with researching this topic we’d help understand an issue that impacts a significant amount of people globally.

Objectives

In this project, we are determined to identify a depressed text and give a probability that particular text is depressive. By using the different attributes provided in the datasets, we will be able to determine specific patterns and identifying words that may strongly suggest that a given text is depressive, Also, we’ll identify how depressive a text is based on the specific words and patterns in the text.

Features

With these findings, we can potentially identify different stages of depression a user might be in depending on their recent “tweets”. Additionally, this ML model could contribute to the growth research in depression and modern preventative measures.

Dataset

The dataset that we will be using in this project will be a dataset from Kaggle[7]. The dataset contains posts and threads they’re from, r/SuicideWatch or r/depression. This dataset was selected because the posts made on these subreddits were most likely from individuals who are depressed and may be suffering from depression. Therefore, there will be valuable information for us to mine.

To expand upon this dataset, we scraped the reddit groups r/SuicideWatch and r/Depression to get more recent data. The data we got covered all posts up to 4/1/2021. After the data was scraped, it had to be cleaned through HDFS and Hive to remove any posts that were deleted or

[7] https://www.kaggle.com/nikhileswarkomati/suicide-watch

removed or were blank. This would help us filter out any posts that were not relevant such as spam or empty threads.

This data set only contains the post and what thread it comes from. We will be interested in looking at the distribution of posts from the two reddit threads. We will also be interested in mining information and patterns from the posts themselves.

Additionally, we will also be pulling live data from Twitter using their Twitter API. We are scraping tweets based off of the hashtags included with the posts. We are currently only interested in three hashtags. Those being #depression, #suicidal, and #anxiety. We chose these specific hashtags as they would most likely yield the type of text we would want to analyze.

From the Twitter API, we will only be concerned with the full_text, created_at, followers_count, possibly_sensitive, verified, and location.

Detailed Design of Features

Our Model

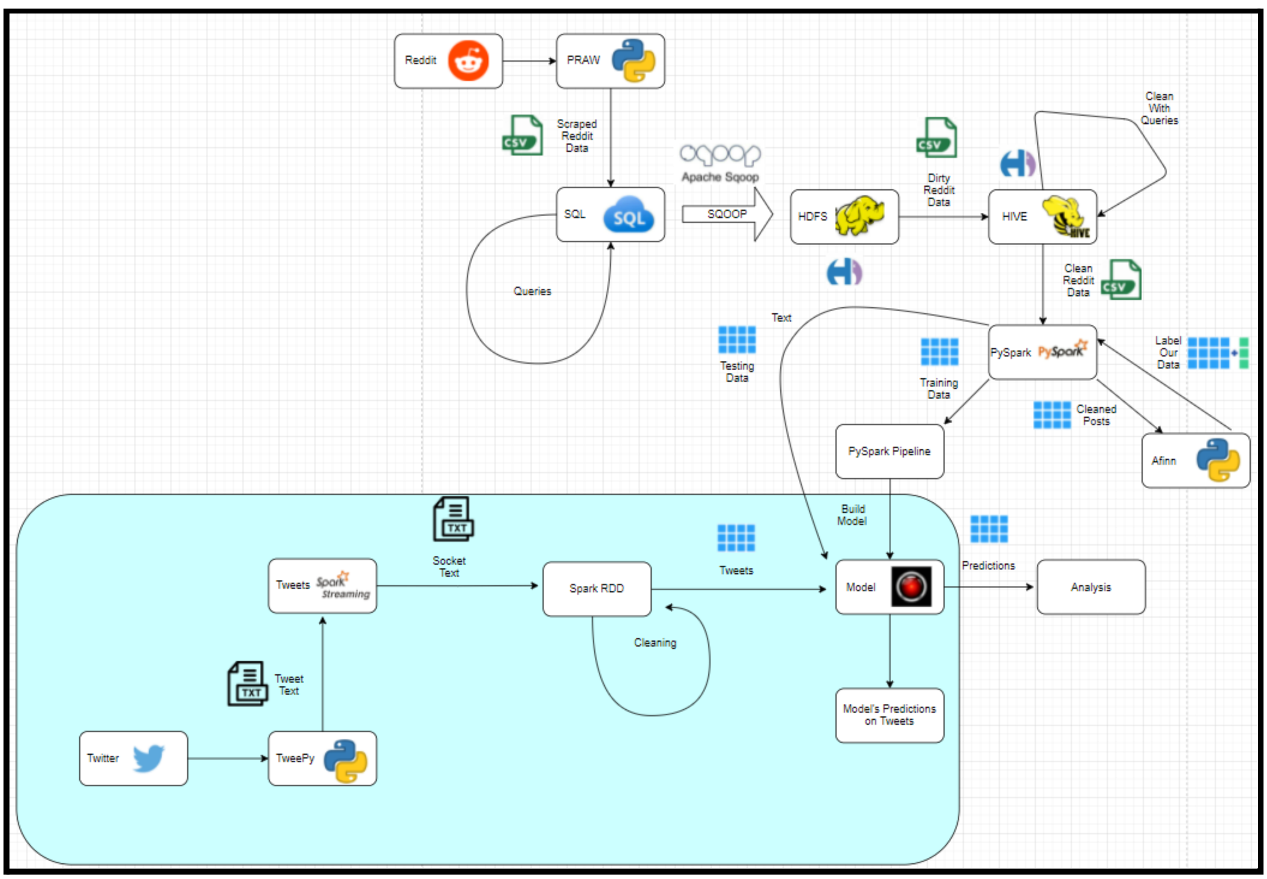


Fig 1. Workflow Diagram

Workflow

Our data starts with scraping reddit for posts that we find in Depression SubReddit. From there, we pull the CSV data into SQL and perform some basic analysis like post count, word count. Now that we have the data in SQL we use SQOOP to transfer it into HDFS from where we can load it into Hive. Once we get the data into Hive we perform data cleaning queries like removal of special characters, lowercase all the data, remove empty posts and remove posts that have subreddit as NULL.

Now that we have our clean data, we can perform analysis on it. We were able to run Hadoop's mapreduce on the clean data and get a word count of all the words in the posts. Next, we did Bigrams and Trigram around the word "depression". Then, we did a variety of SolR queries. Finally, from the word counts we create a Word Cloud.

We also use AFINN for sentiment analysis and provide labels to all the posts as a 0(Not Depressed) and 1(Depressed) and append it to the data. After we get our labelled data we can use it to train different models, In this case we are using Logistic Regression, Naive Bayes and Linear Support Vector Classifier. We train on 90% of the labeled data and perform validation on the other 10%.

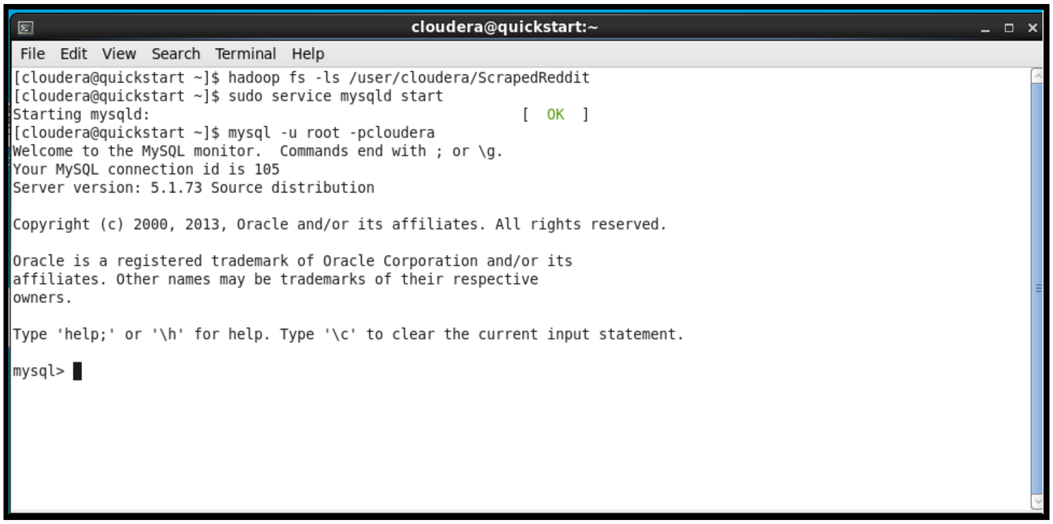
Now that we have our trained model we get our live stream of tweets using spark streaming and clean the tweets and pass them through our model to see if it was depressed or not.

Analysis of Data

Data Preprocessing:

A. SQL

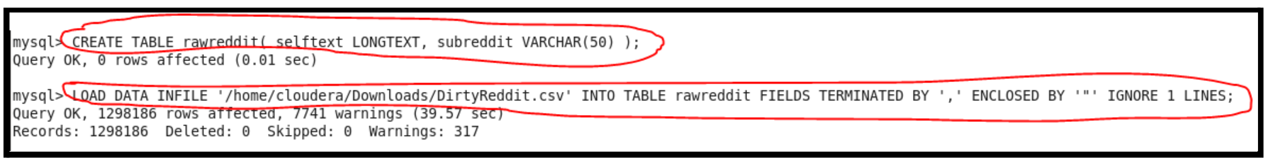
First, we started our mySQL server and entered the mySQL shell.



```
cloudera@quickstart:~
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart ~]$ hadoop fs -ls /user/cloudera/ScrapedReddit
[cloudera@quickstart ~]$ sudo service mysqld start
Starting mysqld:                                           [  OK  ]
[cloudera@quickstart ~]$ mysql -u root -pcloudera
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 105
Server version: 5.1.73 Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Fig 2. Launching mySQL on cloudera

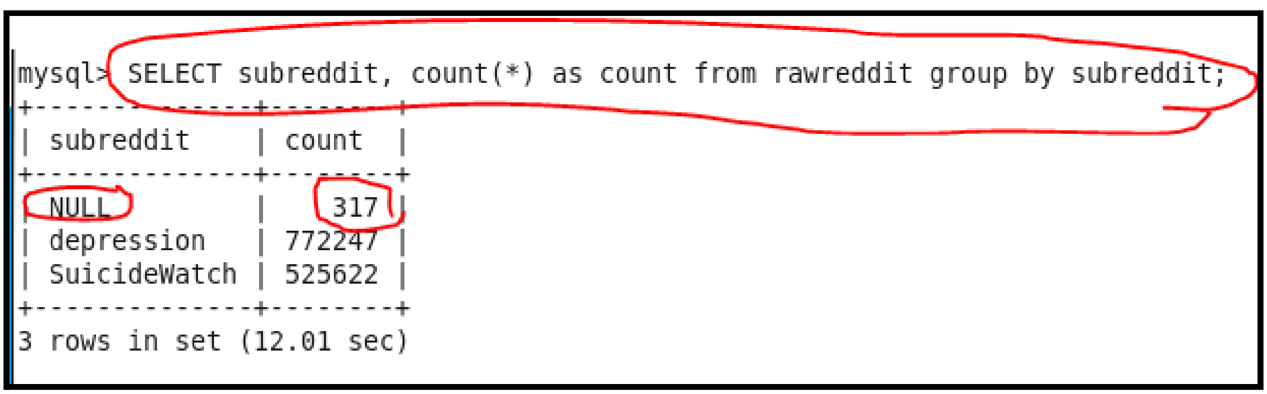
We created a table in our mySQL server to hold our scraped data. Then, we loaded the scraped data csv file into the table.



```
mysql> CREATE TABLE rawreddit( selftext LONGTEXT, subreddit VARCHAR(50) );
Query OK, 0 rows affected (0.01 sec)

mysql> LOAD DATA INFILE '/home/cloudera/Downloads/DirtyReddit.csv' INTO TABLE rawreddit FIELDS TERMINATED BY ',' ENCLOSED BY '"' IGNORE 1 LINES;
Query OK, 1298186 rows affected, 7741 warnings (39.57 sec)
Records: 1298186  Deleted: 0  Skipped: 0  Warnings: 317
```

Fig 3. Loading dataset into mySQL

We then queried to make sure everything loaded ok and we go the following results when looking at the amount of posts per subreddit.



```
mysql> SELECT subreddit, count(*) as count from rawreddit group by subreddit;
+--------------+--------+
| subreddit    | count  |
+--------------+--------+
| NULL         |    317 |
| depression   | 772247 |
| SuicideWatch | 525622 |
+--------------+--------+
3 rows in set (12.01 sec)
```

Fig 4. Counting subreddit posts in dataset

This informed us that we needed to remove these NULL subreddit entries in the HIVE preprocessing.

B. Sqoop

Since we had stored the scraped data in a relational database like mySQL we had to use Sqoop inorder to get the data into our Hadoop Distributed File System (HDFS).

```
[cloudera@quickstart ~]$ sqoop import --connect jdbc:mysql://localhost/490project --username root --password cloudera --table rawreddit --m 1 --target-dir /user/cloudera/ScrapedReddit/sqoop;
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
21/05/05 10:37:43 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.13.0
21/05/05 10:37:43 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
21/05/05 10:37:43 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
21/05/05 10:37:43 INFO tool.CodeGenTool: Beginning code generation
21/05/05 10:37:44 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `rawreddit` AS t LIMIT 1
21/05/05 10:37:44 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `rawreddit` AS t LIMIT 1
21/05/05 10:37:44 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/lib/hadoop-mapreduce
Note: /tmp/sqoop-cloudera/compile/4346d77b925f744dcd79c313fae6fe21/rawreddit.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
21/05/05 10:37:46 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-cloudera/compile/4346d77b925f744dcd79c313fae6fe21/rawreddit.jar
21/05/05 10:37:46 WARN manager.MySQLManager: It looks like you are importing from mysql.
21/05/05 10:37:46 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
```

Fig 5. Transfer dataset from mySQL to Hadoop using Sqoop

Looking at the transfer amount we know that the table was transferred to our HDFS just fine.

```
        Map-Reduce Framework
                Map input records=1298186
                Map output records=1298186
                Input split bytes=87
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=1528
                CPU time spent (ms)=12010
                Physical memory (bytes) snapshot=150040576
                Virtual memory (bytes) snapshot=1511235584
                Total committed heap usage (bytes)=60751872
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=1183714602
21/05/05 10:38:37 INFO mapreduce.ImportJobBase: Transferred 1.1024 GB in 49.9712 seconds (22.5906 MB/sec)
21/05/05 10:38:37 INFO mapreduce.ImportJobBase: Retrieved 1298186 records.
[cloudera@quickstart ~]$
```

Fig 6. Result of transferring tables from mySQL to Hadoop

C. HIVE

Similar to mySQL we had to create a table to hold our scraped data that now resides on our HDFS. Once we do that, we can load the scraped data into the HIVE table for pre processing. This was all done through HUE.

Hive   Add a name...   Add a description...

```
CREATE TABLE IF NOT EXISTS redditscrape(
    selftext STRING,
    subreddit STRING
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
TBLPROPERTIES ('skip.header.line.count' = '1');

LOAD DATA INPATH '/user/cloudera/ScrapedReddit/sqoop/part-m-00000' INTO TABLE redditscrape;
```

Fig 7. Creating a table and load data on HIVE

With the table filled, we ran four queries to clean our data. The first query was run to remove any special characters that are in the subreddit post.

Fig 8. Remove special characters in table

The second query, removed any posts that were empty, deleted, or removed from the dataset.

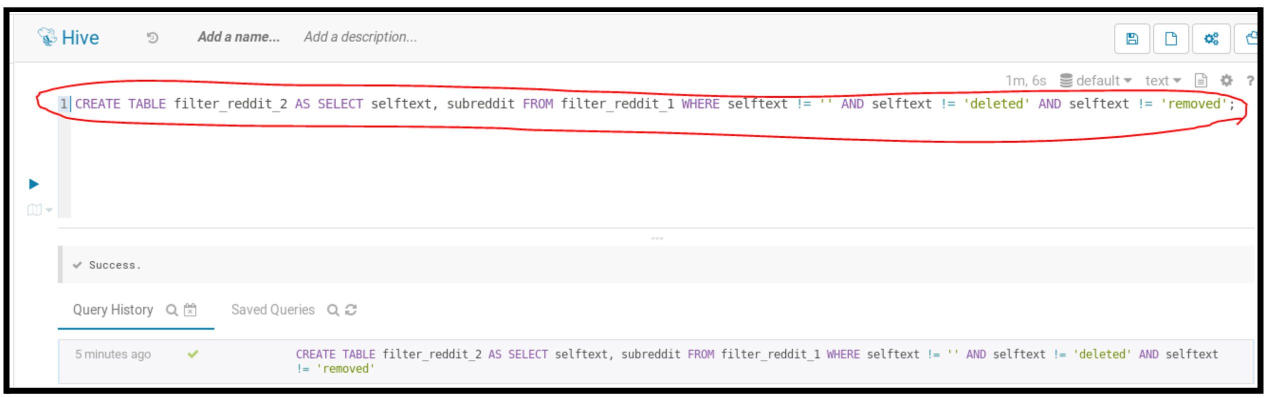


Fig 9. Remove empty posts, deleted or removed posts in table

The third query, brought all of the posts down to lowercase characters only so we won't have the issue of case sensitivity.

Fig 10. Lowercase every word in table

The fourth query, removed any entries that have a subreddit of NULL from our data.

Fig 11. Filter out NULL rows in table

To make sure that we had posts with valid subreddits we did a query to get the subreddit distribution of posts.

Fig 12. Table size after cleaning the dataset

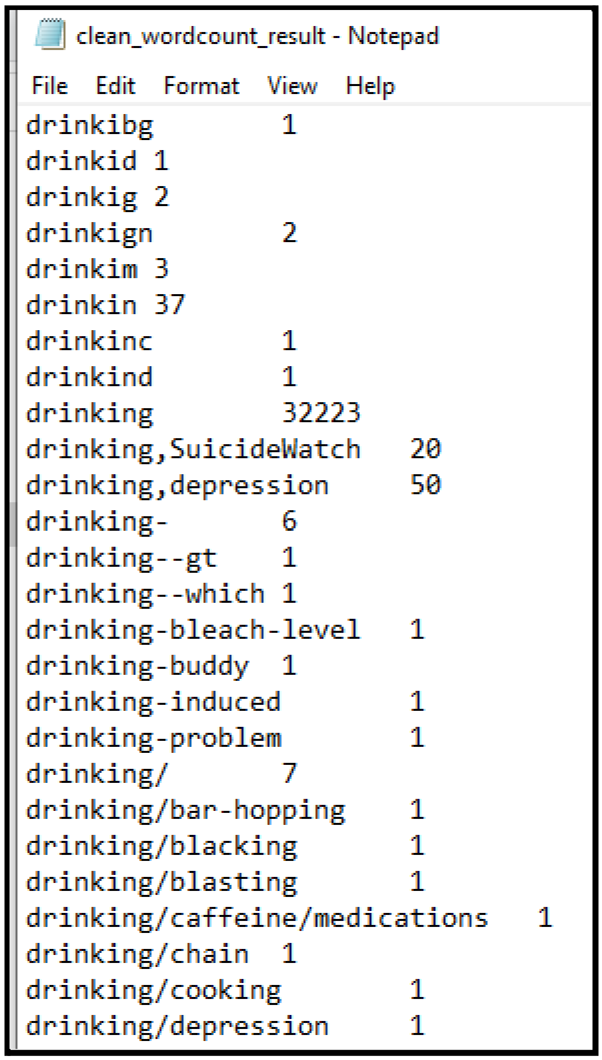
Finally, we saved the cleaned data table into a CSV file.

Fig 13. Exporting dataset as CSV from HIVE

Analysis:

MapReduce (Hadoop):

We used Hadoop's mapper and reducer to count each unique word's occurrence in the cleaned dataset. This was done by adding the cleaned data to HDFS and using the source code as provided in our lectures (Lesson 2). The word count could be useful to track what particular words come up frequently in posts discussing suicide or depression. Words with the highest occurrence may be red flags to a social media company that someone may need help. The output was a file that contained each word used in our dataset and the number of times it had been counted:



clean_wordcount_result - Notepad

File Edit Format View Help

```
drinkibg	1
drinkid 1
drinkig 2
drinkign	2
drinkim 3
drinkin 37
drinkinc	1
drinkind	1
drinking	32223
drinking,SuicideWatch	20
drinking,depression	50
drinking-	6
drinking--gt	1
drinking--which 1
drinking-bleach-level	1
drinking-buddy	1
drinking-induced	1
drinking-problem	1
drinking/	7
drinking/bar-hopping	1
drinking/blacking	1
drinking/blasting	1
drinking/caffeine/medications	1
drinking/chain	1
drinking/cooking	1
drinking/depression	1
```

Fig 14. Word count using Hadoop

Bigrams and Trigrams on "depression" (PySpark)

This analysis looked at pairs of words that were likely to come up together. We then calculated the likelihood ratio for the key word "depression". With this type of analysis we can see interesting patterns like how the likelihood ratio of “depression” coming before “anxiety” is higher than the vice versa. The following are the top results of performing a Bigram and Trigram on the key word “depression”.

```
(('depression', 'anxiety'), 131.70544652948615)
(('anxiety', 'depression'), 72.22559030989585)
(('severe', 'depression'), 45.797027239998144)
(('suffering', 'depression'), 38.95904130927086)
(('major', 'depression'), 30.13600332875144)
(('suffer', 'depression'), 27.362614897322388)
(('struggling', 'depression'), 26.779261833819106)
(('chronic', 'depression'), 22.07980264585341)
(('diagnosed', 'depression'), 21.879939849418278)
(('noticed', 'depression'), 21.32728644706868)
(('dealing', 'depression'), 20.763148455855518)
(('ptsd', 'depression'), 20.66112977381583)
(('depression', 'suicidal'), 18.334598430478266)
(('depression', 'excuse'), 18.147180861872506)
(('struggled', 'depression'), 17.75320355097563)
(('history', 'depression'), 17.038040829154518)
(('depression', 'comes'), 12.459294837020794)
(('depression', 'meds'), 11.763935979998784)
(('understand', 'depression'), 11.436903905808226)
(('due', 'depression'), 10.702910067258937)
(('depression', 'since'), 9.537978108930261)
(('back', 'depression'), 6.552105196987837)
(('living', 'depression'), 5.9822099935789845)
(('depression', 'doesnt'), 4.6062403734209925)
(('depression', 'started'), 4.440231845630621)
```

Fig 15. Bigrams for ‘depression’

```
(('feel', 'like', 'depression'), 9528.119876643184)
(('depression', 'feel', 'like'), 9524.726073164264)
(('dont', 'know', 'depression'), 7189.405642579027)
(('depression', 'dont', 'want'), 3945.9455799729335)
(('depression', 'suicidal', 'thoughts'), 1676.546969059987)
(('depression', 'feels', 'like'), 1621.1240780335927)
(('depression', 'long', 'time'), 1160.0917769141388)
(('depression', 'dont', 'think'), 1017.8117751787623)
(('social', 'anxiety', 'depression'), 923.6868735689229)
(('severe', 'depression', 'anxiety'), 739.8683300794883)
(('depression', 'social', 'anxiety'), 716.9740414879772)
(('diagnosed', 'depression', 'anxiety'), 699.5068469579847)
(('struggling', 'depression', 'anxiety'), 678.034050016812)
(('suffer', 'depression', 'anxiety'), 613.9660401327799)
(('due', 'depression', 'anxiety'), 557.0603968390519)
(('depression', 'anxiety', 'ptsd'), 552.3716514790202)
(('depression', 'anxiety', 'long'), 530.5193312587514)
(('pretty', 'sure', 'depression'), 467.60472509759904)
(('depression', 'bipolar', 'disorder'), 410.7831189223852)
(('worst', 'part', 'depression'), 406.3844335463595)
(('ive', 'struggling', 'depression'), 316.13682196086165)
(('diagnosed', 'clinical', 'depression'), 299.6137399335885)
(('depression', 'coming', 'back'), 268.52637204436166)
(('depression', 'long', 'remember'), 262.75191405049486)
(('struggling', 'depression', 'long'), 251.22579919545637)
(('depression', 'keeps', 'getting'), 235.9716322294812)
(('ive', 'suffered', 'depression'), 176.75338403513857)
(('severe', 'case', 'depression'), 118.57712762235414)
```

Fig 16. Trigrams for ‘depression’

Solr Queries (Solr)

This helped us run a wildcard search on various words associated with depression and suicide risk. We ran queries on the words depressed and hurt. These returned posts with the words depressed and hurt respectively included in the post.

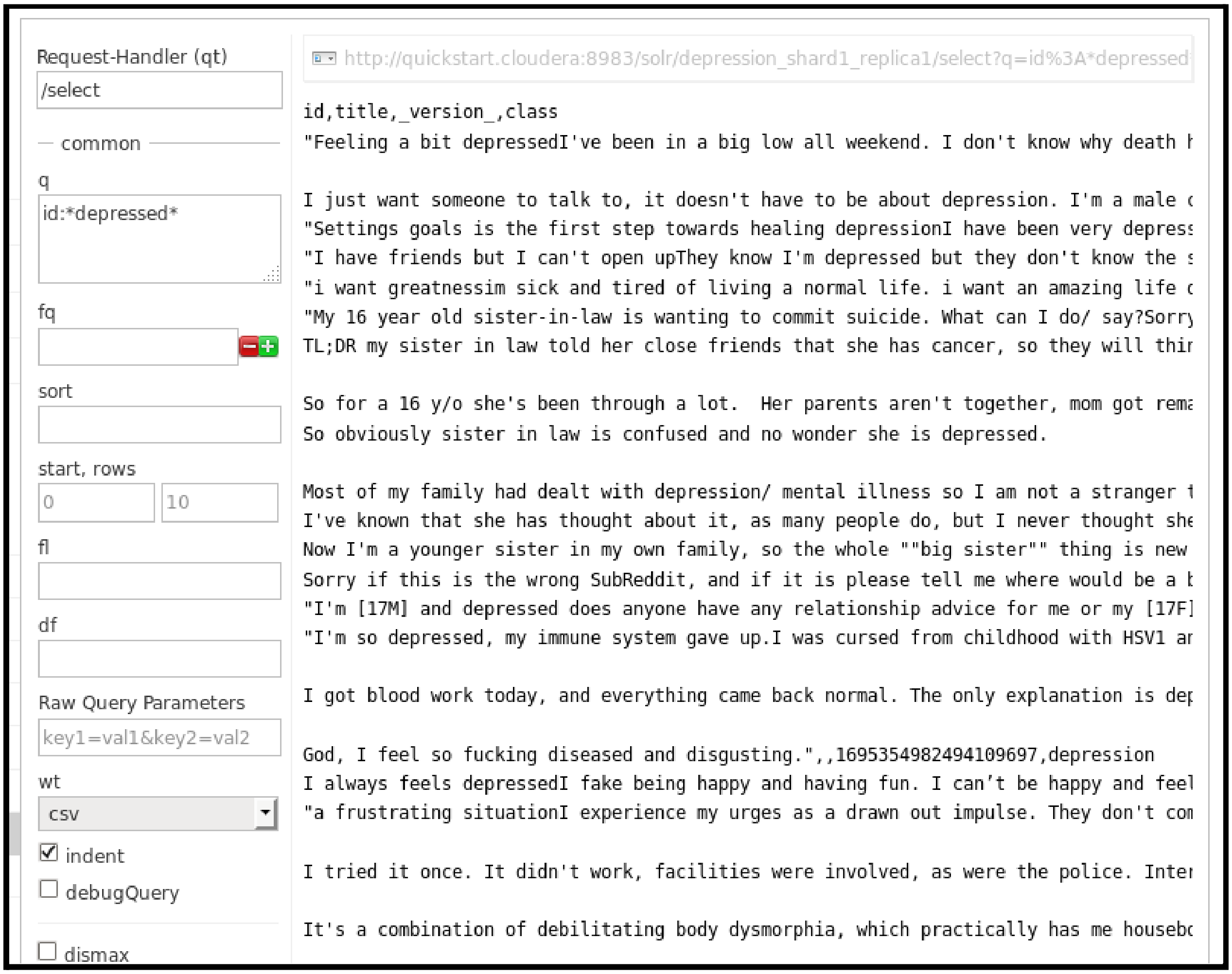


Fig 17. Wildcard search for ‘depressed’ ’on Solr

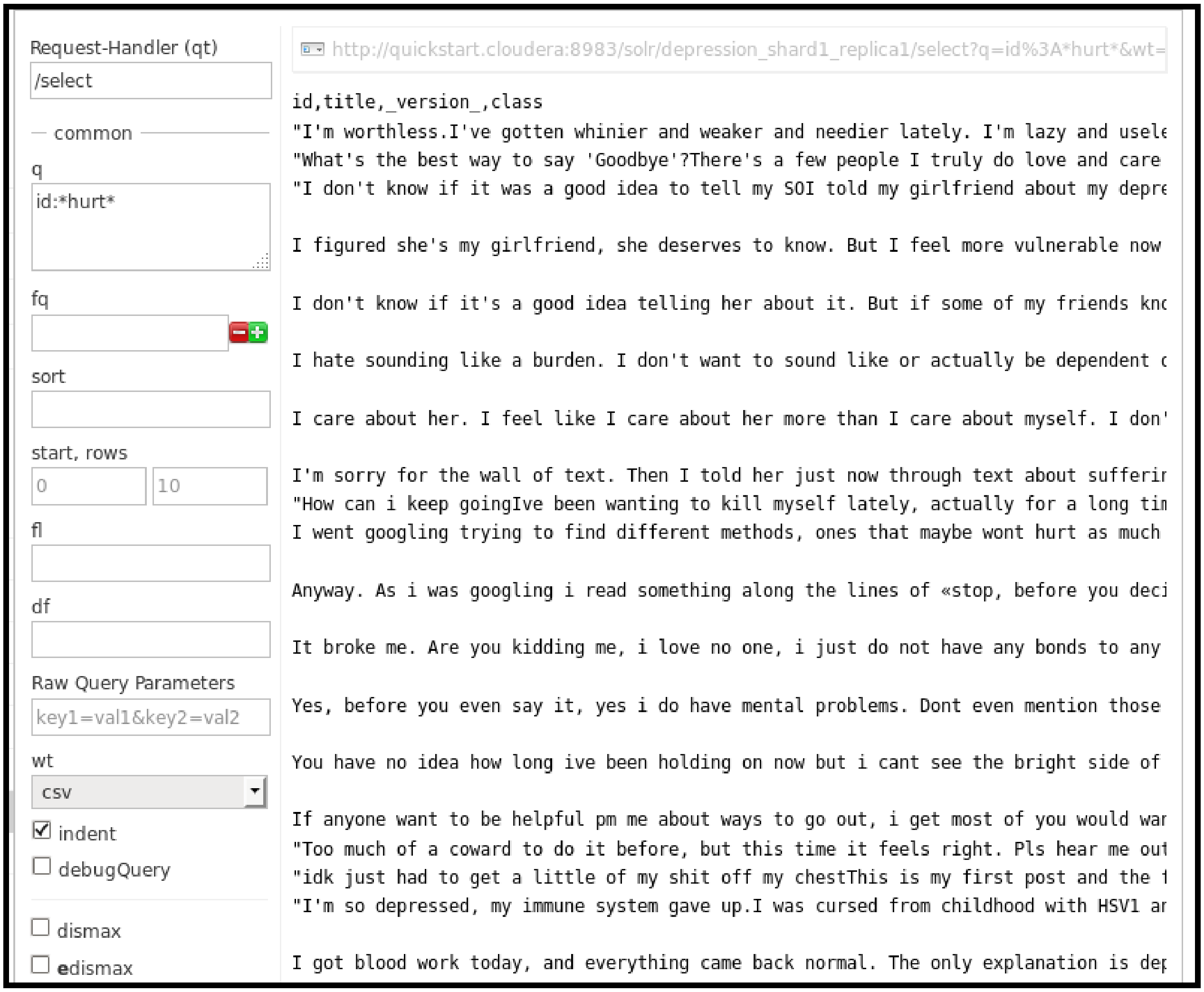


Fig 18. Wildcard search for ‘hurt’ on Solr

We did an additional wildcard search on “self harm” since it is common that people suffering from depression also struggle with self harm.

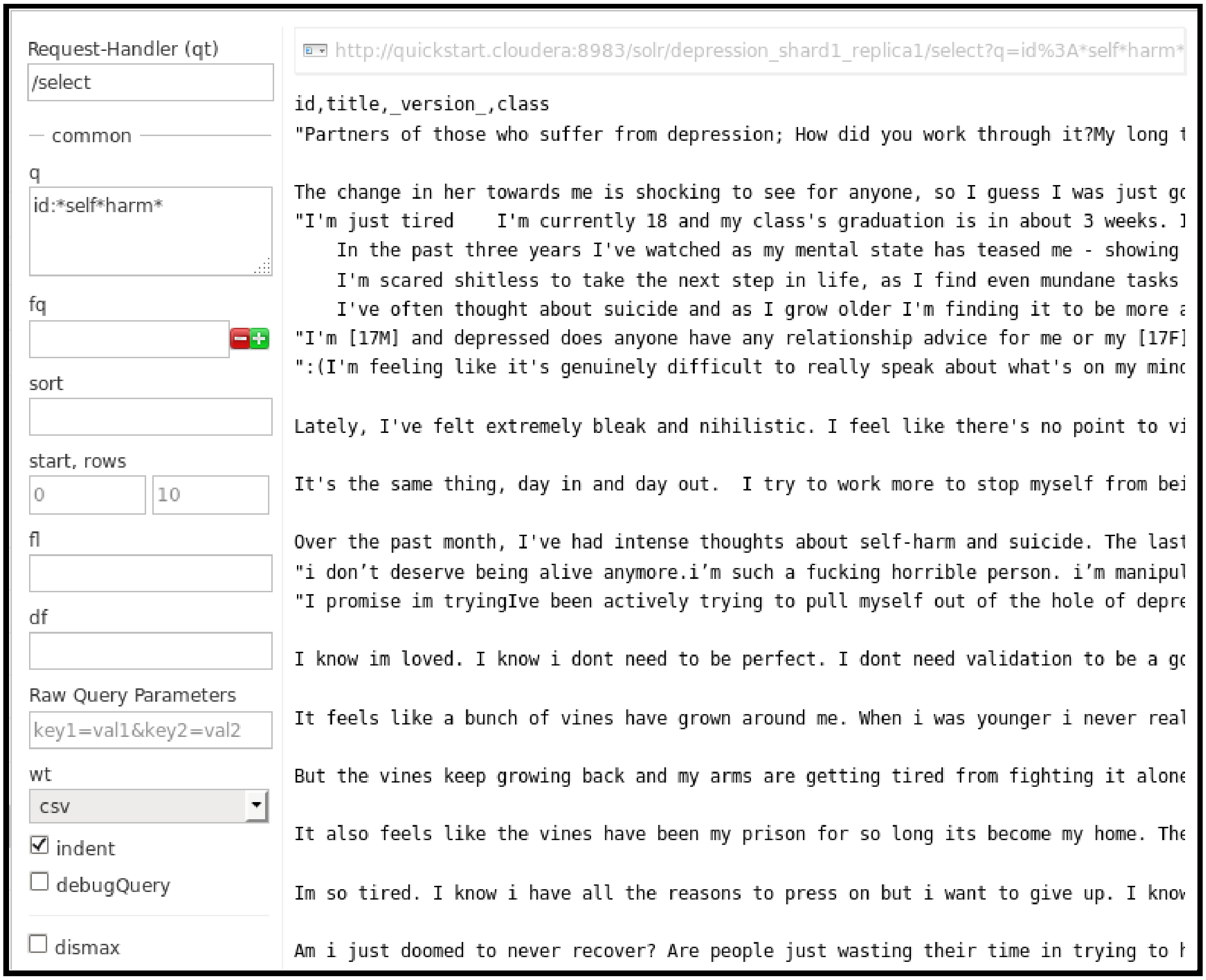


Fig 19. Wildcard search 'self harm' on Solr

We also did proximity searching for posts that talked about hurting themselves by searching for the words "hurt myself" within 10 words of each other:

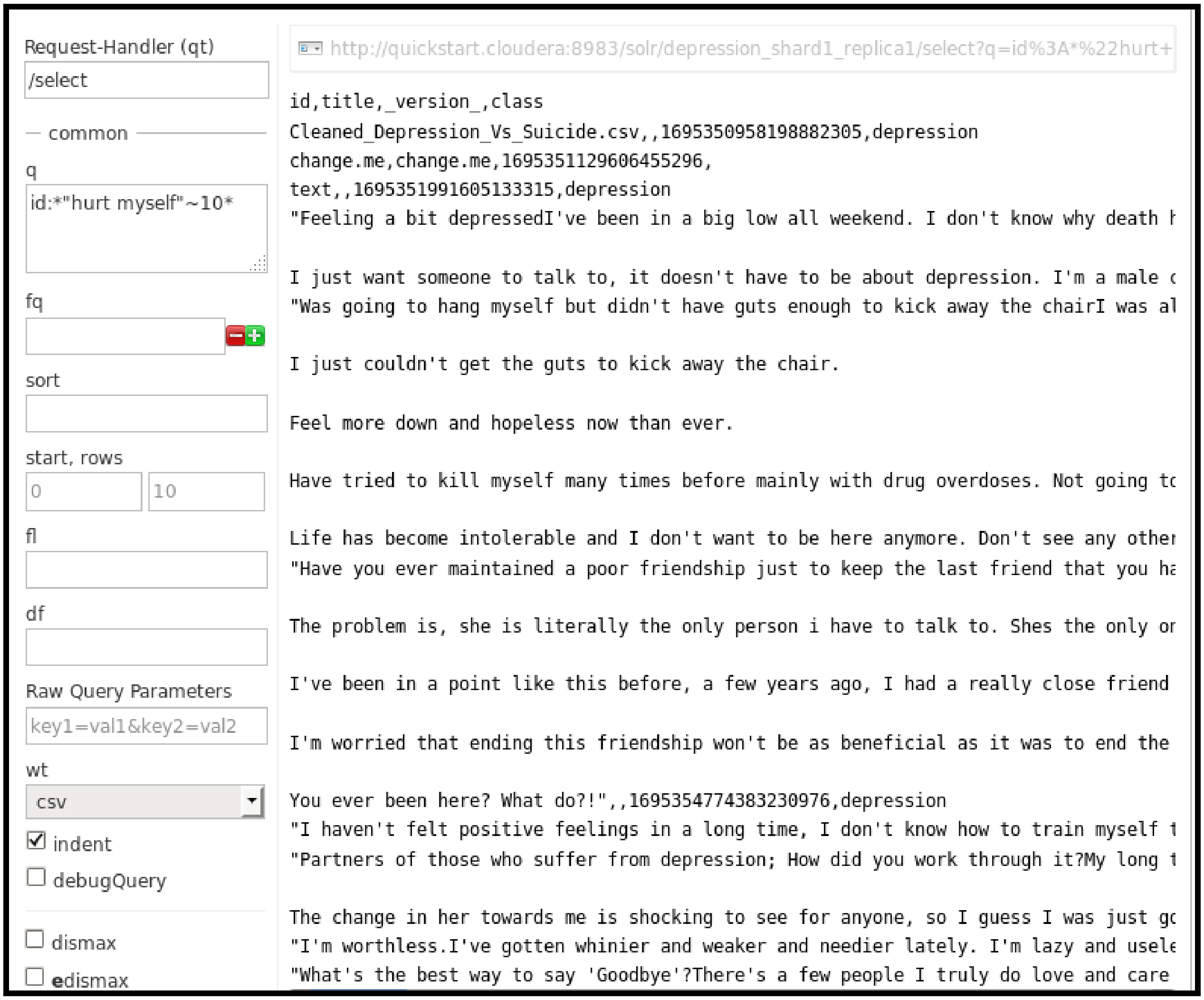


Fig 20. Proximity search on 'hurt myself' on Solr

Fuzzy search on the word hope:

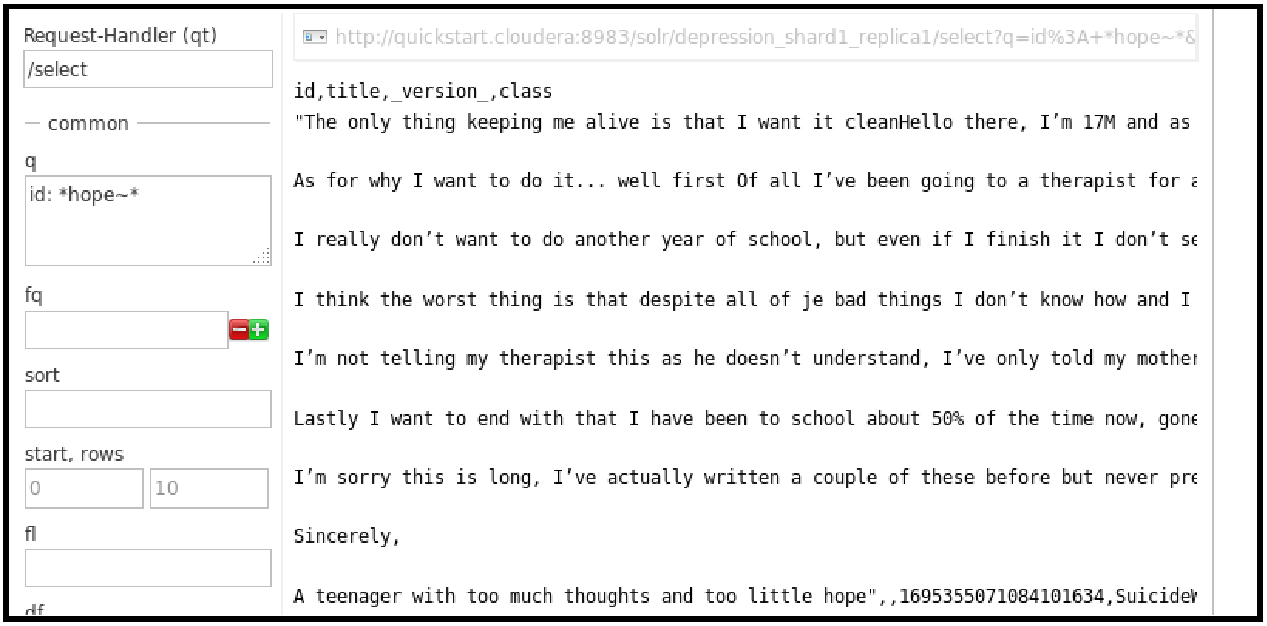


Fig 20. Fuzzy search ‘hope’ on Solr

Word Cloud (Pyspark):

For the Word Count of the Reddit posts we created the following Word Cloud graphic to show high frequency words.

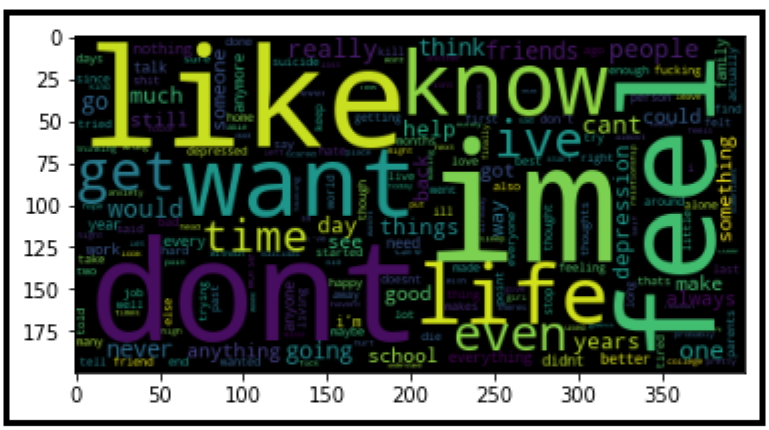


Fig 21. Diagram of most frequently used words

```
+------+-------+
|    _1|     _2|
+------+-------+
|    im|1202526|
|  dont| 833336|
|  like| 810509|
|  feel| 704298|
|  want| 627313|
|  know| 609032|
|  life| 547364|
|   get| 517916|
|   ive| 438814|
|  even| 432674|
|  time| 426501|
|really| 416692|
|people| 410211|
|  cant| 402223|
| would| 378815|
|   one| 373920|
| think| 319247|
| going| 316714|
| never| 314062|
|    go| 309134|
+------+-------+
only showing top 20 rows
```

Fig 22. Top 20 rows of word count from highest to lowest

Implementation

A. Labeling

After preprocessing our data, we now have a clean dataset which does not include any empty rows, removed posts nor special characters in the dataset. We decided to use PySpark to analyze our dataset to give us a more in-depth understanding of our dataset. The diagram below shows the process of importing our cleaned dataset into PySpark.

```
df = spark.read.option("header", "false").option("quote", "\"").option("escape", "\"").option("inferSchema", "true").csv("/content/drive/MyDrive/490/Data/cleanData.csv").toDF("selftext", "subreddit")
```

```
# Take look and make sure everything is ok
df.show()
```

```
+--------------------+----------+
|            selftext| subreddit|
+--------------------+----------+
|ive had depressio...|depression|
|i just need to ve...|depression|
|i see the world s...|depression|
|hey reddit i hope...|depression|
|ive sought advice...|depression|
|does anyone else ...|depression|
|so a while ago i ...|depression|
|i get so anxious ...|depression|
|i recently  came ...|depression|
|im a   36 male fr...|depression|
|well im here to a...|depression|
|ive always felt l...|depression|
|is it bad that if...|depression|
|let me preface th...|depression|
|i just cant i tri...|depression|
|ive been done for...|depression|
|im 26 and just we...|depression|
|a lot of the time...|depression|
|it feels like no ...|depression|
|so i wake up in a...|depression|
+--------------------+----------+
only showing top 20 rows
```

Fig 23. Importing data into PySpark

After the dataset has been successfully imported into PySpark, we are able to further process and analyze our data. We begin by performing a sentimental analysis on each reddit post by using lexicons called Afinn. As Afinn contains more than 3000 words and a score associated with each word, we will be able to rate how negative a particular text is by passing the text into the score() method of our Afinn object. Since we will be performing Afinn on all of the selftext column in our dataset, we created a User Defined Function in PySpark and applied it to the selftext column as shown in the diagram below.

```
# This Function will return 1 if the text is negative and 0 if the text is positive.
# This is based on the scoring from the Afinn object
udfNew = F.udf(lambda x: 1 if afin.score(x) < 0 else 0)

data = df.select(F.col('selftext'), udfNew(F.col('selftext')).alias('label'))
data = data.withColumn("label", F.col("label").cast("int"))
```

Fig 24. Creating UDF for AFINN method

B. Preventing Undersampling and Oversampling

To prevent undersample and over sampling, we selected 50,000 posts that had label 1 and 50,000 posts that had label 0.

```
data.registerTempTable("dataWithLabel")
```

Fig 25. Registering a new temporary SQL table on PySpark

```
temp1 = sqlContext.sql("SELECT * from dataWithLabel WHERE label = 1 LIMIT 50000")

temp2 = sqlContext.sql("SELECT * from dataWithLabel WHERE label = 0 LIMIT 50000")

data2 = temp1.union(temp2)
```

Fig 26. SQL queries on PySpark using SQL Context

C. Data Splitting and Pipeline Configurations

Before training our model, we want to split the dataset into 2 separate datasets where one would be dedicated to training a model and the other to test the trained model. The diagram below shows the process of splitting the dataset into 2. The train dataset would include 90% of the actual dataset where the test dataset would only include 10% of the dataset.

```
# splits[0] is my training set, splits[1] is my testing set
splits = data2.randomSplit([0.9, 0.1], 1234)
```

Fig 27. Splitting dataset into 2

In this project, we will be using multiple Machine Learning algorithms to perform analysis on our dataset. Thus, there will be 3 dedicated pipelines for each model. The pipelines will be shown in the diagram below.

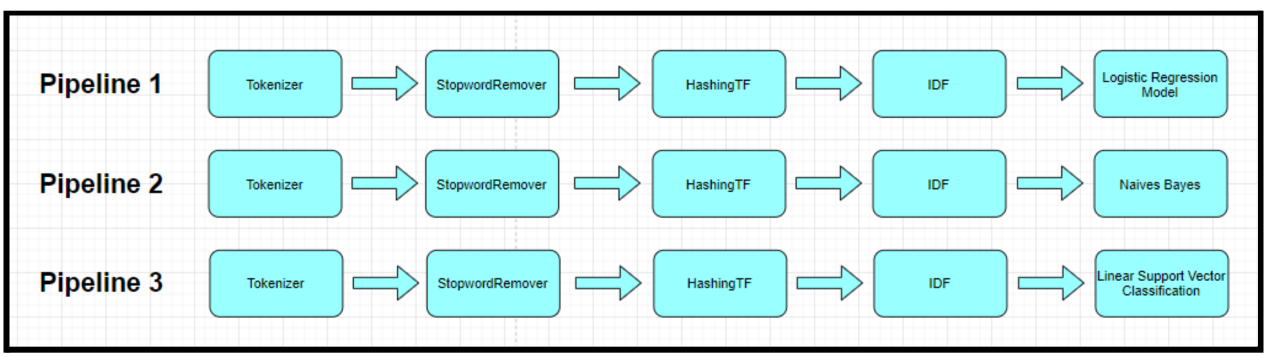


Fig 28. Workflow for each pipeline

In each pipeline, the dataset passed into the pipeline will be tokenized, processed to remove any stopwords, hashed to find the term frequency for each word (HashingTF), and be used to calculate the inverse document frequency (IDF). The diagram below shows the process of setup and configuration of each pipeline for each algorithm.

```
tokenizer = Tokenizer(inputCol="selftext", outputCol="words")

remover = StopWordsRemover(inputCol="words", outputCol="filtered", caseSensitive=False)

hashingTF = HashingTF(inputCol="filtered", outputCol="rawfeatures", numFeatures= 4096)

idf = IDF(inputCol="rawfeatures", outputCol="features", minDocFreq= 0)

lr = LogisticRegression(regParam=0.01, threshold=0.5)

nb = NaiveBayes()

lsvc = LinearSVC(regParam= 0.01, threshold=0.5)

pipeline1 = Pipeline(stages=[tokenizer, remover, hashingTF, idf, lr])

pipeline2 = Pipeline(stages=[tokenizer, remover, hashingTF, idf, nb])

 pipeline3 = Pipeline(stages=[tokenizer, remover, hashingTF, idf, lsvc])
```

Fig 29. Pipeline configurations

D. Building and Testing Models

After we have successfully created individual pipelines for each model, we will pass in the training dataset to build our models.

```
# Logistic Regression Model
model1 = pipeline1.fit(splits[0])

# Naive Bayes Model
model2 = pipeline2.fit(splits[0])

# Linear Support Vector Classification Model
model3 = pipeline3.fit(splits[0])
```

Fig 30. Passing training dataset into each model

Once each model has been trained, we will pass in the test dataset into the model to test our models using the models' transform function. Below are the results for each model.

Logistic Regression Model:

```
# Binary Classification Evaluator

eval1 = BinaryClassificationEvaluator(metricName="areaUnderROC")
print("Area Under the ROC Curve: {}".format(eval1.evaluate(predictions1)))

Area Under the ROC Curve: 0.927023485415756
```

Fig 31. Area under the ROC curve of the Logistic Regression model

```
# Multiclass Classification Evaluator

eval2 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
print("Accuracy: " + str(eval2.evaluate(predictions1)))

eval3 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="weightedPrecision")

print("Precision: " + str(eval2.evaluate(predictions1)))

Accuracy: 0.8582536419697918
Precision: 0.8582536419697918
```

Fig 32. Accuracy and precision of Logistic Regression model

Naives Bayes Model:

```
# Binary Classification Evaluator

eval4 = BinaryClassificationEvaluator(metricName="areaUnderROC")
print("Area Under the ROC Curve: {}".format(eval1.evaluate(predictions2)))

Area Under the ROC Curve: 0.40505961527085854
```

Fig 33. Area under the ROC curve of the Naives Bayes model

```
# Multiclass Classification Evaluator

eval5 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
print("Accuracy: " + str(eval5.evaluate(predictions2)))

eval6 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="weightedPrecision")

print("Precision: " + str(eval6.evaluate(predictions2)))

Accuracy: 0.7768343909196532
Precision: 0.7818824488930595
```

Fig 34. Accuracy and precision of Naives Bayes model

LSVC (Linear Support Vector Classification) Model:

```
# Binary Classification Evaluator

eval7 = BinaryClassificationEvaluator(metricName="areaUnderROC")
print("Area Under the ROC Curve: {}".format(eval7.evaluate(predictions3)))

Area Under the ROC Curve: 0.9283388771784026
```

Fig 35. Area under the ROC curve of the LSVC model

```
# Multiclass Classification Evaluator

eval8 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
print("Accuracy: " + str(eval8.evaluate(predictions3)))

eval9 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="weightedPrecision")

print("Precision: " + str(eval9.evaluate(predictions3)))

Accuracy: 0.8428814013763518
Precision: 0.8619162777668492
```

Fig 36. Accuracy and precision of LSVC model

After our analysis, the Logistic Regression model has the highest accuracy out of the 3 models. However, the LSVC model has a slightly higher precision compared to the Logistic Regression model. Overall, the Naives Bayes model has the lowest accuracy and precision out of the 3 models. Thus, we decided to use the Logistic Regression model for our live streamed tweets. Now, we must save our Logistic Regression Model.

```
# Save our Logistic Regression Model
model1.save('/content/drive/MyDrive/490/Model')
```

Fig 37. Saving Logistic Regression model to a directory

Socket Stream Generation (Tweets):

```python
def get_tweets():
    # Query formation
    url = 'https://stream.twitter.com/1.1/statuses/filter.json'
    query_data = [('language', 'en'), ('locations', '-130,-20,100,50'),('track','#')]
    query_url = url + '?' + '&'.join([str(t[0]) + '=' + str(t[1]) for t in query_data])
    # Query request as a stream of tweets
    response = requests.get(query_url, auth=my_auth, stream=True)
    print(query_url, response)
    return response
```

Fig 38. Getting data from twitter.

In order to create a Socket Stream for tweets. I am using the TCP localhost to trigger getting tweets when a reader is available, in this case it is Spark readStream. When someone is connected to listen to the stream, the script starts to retrieve the tweets from twitter and starts sending them to the socket.

```python
# Socket Setup
TCP_IP = "localhost"
TCP_PORT = 9009
conn = None
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
# Listening if anyone connected
s.listen(1)
print("Waiting for TCP connection...")
conn, addr = s.accept()
print("Connected...Starting getting tweets.")
resp = get_tweets()
parseSend(resp, conn)
```

Fig 39. Setting up a TCP socket stream.

Using the API keys I got from Twitter Developer account I am sending a request for retrieving tweets as a JSON response. Twitter provides a stream of tweets using *stream.twitter.com*. I then parse the JSON response and get the required tweet text from it and send it as an encoded response.

```python
def parseSend(http_resp, tcp_connection):
  for line in http_resp.iter_lines():
    try:
        # Parse JSON response from Twitter API
        tweet = json.loads(line)
        # Extract Text, Username and Name
        tweet_text = tweet['text']
        tweet_user = tweet['user']['screen_name']
        tweet_name = tweet['user']['name']
        print("Tweet Text: " + tweet_text)
        print("Tweet Username: " + tweet_user)
        print("Tweet Name: " + tweet_name)
        print ("-------------------------------------------")
        # Add line break and send to socket
        text = tweet_text + '\n'
        tcp_connection.send(text.encode('utf-8'))
    except:
        e = sys.exc_info()
        print(e)
```

Fig 40. Parsing the JSON response and sending tweetText to socket.

Now we run the stream generator in command prompt and it waits for a stream listener to start fetching tweets.

```
Select C:\Windows\System32\cmd.exe - python  streamingGen.py
C:\Users\Akkip\Documents>python streamingGen.py
Waiting for TCP connection...
Connected... Starting getting tweets.
https://stream.twitter.com/1.1/statuses/filter.json?language=en&locations=-130,-20,100,50&track=# <R
esponse [200]>
Tweet Text: @WhiteHouse Coming soon. https://t.co/JnDcrld5PX
Tweet Username: JoeShow68347212
Tweet Name: JoeShow
------------------------------------------
Tweet Text: @mmpadellan Very nice ▯  and true.
Tweet Username: Artistwhodreams
Tweet Name: David the Cancer Survivor.
------------------------------------------
Tweet Text: @nahteegi titties
Tweet Username: skydollabills
Tweet Name: skyl
------------------------------------------
Tweet Text: Just posted a photo @ Houston Real Estate https://t.co/J1x4S8TQKY
Tweet Username: corytheREguy
Tweet Name: Cory Grant
------------------------------------------
Tweet Text: Nice
@ITSNOTDIPONKOR
@KunduRishab
@THEabhirup1
Tweet Username: dasrupam40
Tweet Name: Rupam Das
------------------------------------------
```

Fig 41. Socket stream output of retrieved tweets.

Socket Stream reader and predictor (Tweets):

```
# Initializing spark session
sc = SparkContext(appName="PySparkShell")
sc.setLogLevel("ERROR")
spark = SparkSession(sc)

# Loading the pretrained model on Reddit data
model = PipelineModel.load('C:\\Users\\Akkip\\Documents\\Model')
```

Fig 42. Initialize SparkSession and load the pretrained model.

We got the model that was trained on Reddit data because we had labels for that data. I downloaded the trained model into my local machine from Google Colab and then loaded it into the model variable for prediction.

```
# initialize the streaming context
ssc = StreamingContext(sc, batchDuration= 3)
# Create a DStream that will connect to hostname:port, like localhost:9009
# localhost:9009 is the stream we are doing
tweetsline = ssc.socketTextStream("localhost",9009)
# split the tweet text by apliting at line break to get the list of tweets
tweets = tweetsline.flatMap(lambda line : line.split('\n'))
# send the rdd for prediction from trained model
tweets.foreachRDD(get_prediction)
# Start the computation
ssc.start()
# Wait for the computation to terminate
ssc.awaitTermination()
```

Fig 43. Initialize SparkSession and load the pretrained model.

Now that our SparkSession was initialized we create a Streaming Context with a *batchDuration* of 3. Here batchDuration is the time it will listen to the stream to create one RDD. Then we set up the *socketTextStream* to listen to the stream that we create for the twitter text from our *streamingGen.py*. The data is then flat mapped and split at every line break. Then we call the get prediction function for each RDD(Tweet) and see what the model predicts.

```
# Function to preprocess data and perform prediction on  received tweets
def get_prediction(tweet_text):
    try:
        # Filter out all the mentions and links
        tweet_text = tweet_text.map(lambda l: re.sub(r"(?:\@|https?\://)\S+", "", l))
        # Filter out all the other special characters
        tweet_text = tweet_text.map(lambda l: re.sub(r"[^a-zA-Z0-9]+", ' ', l).lower())
        # Filter the tweets to include those with length > 0
        tweet_text = tweet_text.filter(lambda x: len(x) > 0)
        # create a dataframe with column name 'selftext' and each row will contain the tweetText
        rowRdd = tweet_text.map(lambda w: Row(selftext=w))
        # create a spark dataframe
        wordsDataFrame = spark.createDataFrame(rowRdd)
        # transform the data using the pipeline and get the predicted sentiment
        model.transform(wordsDataFrame).select('selftext','prediction').show()
    except :
        e = sys.exc_info()
        print(e)
```

Fig 44. Function to preprocess data and perform prediction on received tweets.

The *get_prediction* function performs 4 tasks, 2 types of mapping, 1 filter and finally prediction. The first mapping is to remove all the hyperlinks and @ mentions from the tweets. Now taking that cleaned data and removing all the special characters from it and making it in lower case. Now that our data is cleaned of all the non necessary things we can check if there is any text left in the tweet if it has no text then drop that tweet using the filter. Now that we have the preprocessed data we make it into a Spark dataframe and pass it into the model to perform predictions and show them into the console.

```
Select C:\Windows\System32\cmd.exe - python  processor.py
C:\Users\Akkip\Documents>python processor.py
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
(<class 'ValueError'>, ValueError('RDD is empty'), <traceback object at 0x0000028D7C8107C0>)
+--------------------+----------+
|            selftext|prediction|
+--------------------+----------+
| yes i agree but ...|       0.0|
|test geo hierarch...|       1.0|
| the karkeh with ...|       0.0|
|                true |       0.0|
|   plz land at my job|       1.0|
| lastnightasamacc...|       1.0|
|i m ready for a r...|       0.0|
|           hi follow|       1.0|
|wahala no better ...|       1.0|
|      one na problem|       1.0|
|her loves dead th...|       1.0|
| and extreme and ...|       1.0|
|i ll miss this ha...|       0.0|
| looking for foot...|       0.0|
|      they charging |       0.0|
|this is called wo...|       0.0|
|i finally got aro...|       0.0|
|               damn |       1.0|
| this is criminal...|       0.0|
| take it back tak...|       1.0|
+--------------------+----------+
only showing top 20 rows

+--------------------+----------+
|            selftext|prediction|
+--------------------+----------+
| spotted an b429 ...|       0.0|
|           29th may |       0.0|
|the new maris ros...|       0.0|
| went through my ...|       0.0|
|                    |       0.0|
| yeah let s see t...|       1.0|
| had both shots w...|       0.0|
| happy day sweeth...|       0.0|
|see you later at ...|       0.0|
|no drive by bitch...|       1.0|
|             someday|       0.0|
|sooo i bought all...|       1.0|
|exhilarating mess...|       0.0|
|don t piss off th...|       1.0|
|okay lucas also n...|       1.0|
```

Fig 45. Spark model performing prediction on tweets and returning a dataframe with tweets and predictions.

Conclusion

By the end of this project, we had successfully created a classifying model that could decently classify a depressive piece of text. This model was then able to take in live tweets and label them as depressive and non depressive. However, the model created would need to through more rigorous testing and be built on a larger set of data.

Future Work

If we were to continue this project, we would like to outsource the data labeling process to a company that handles data labeling using humans. This would provide a better scoring system than relying on the Afinn module. With human labeling, it will also encompass the human mind in it's evaluation. Additionally, we would want to process the results of the live streamed tweets through or model to potentially set up a proper twitter user monitoring program. This program would establish and define the rules and ethics of monitoring Twitter users and when intervention is needed. Finally, we would want to take the time to start to optimize our ML configurations to get us the best possible model that we can. We would want to have a high performing model prior to any sort of actual deployment on any social media platform.

Project management

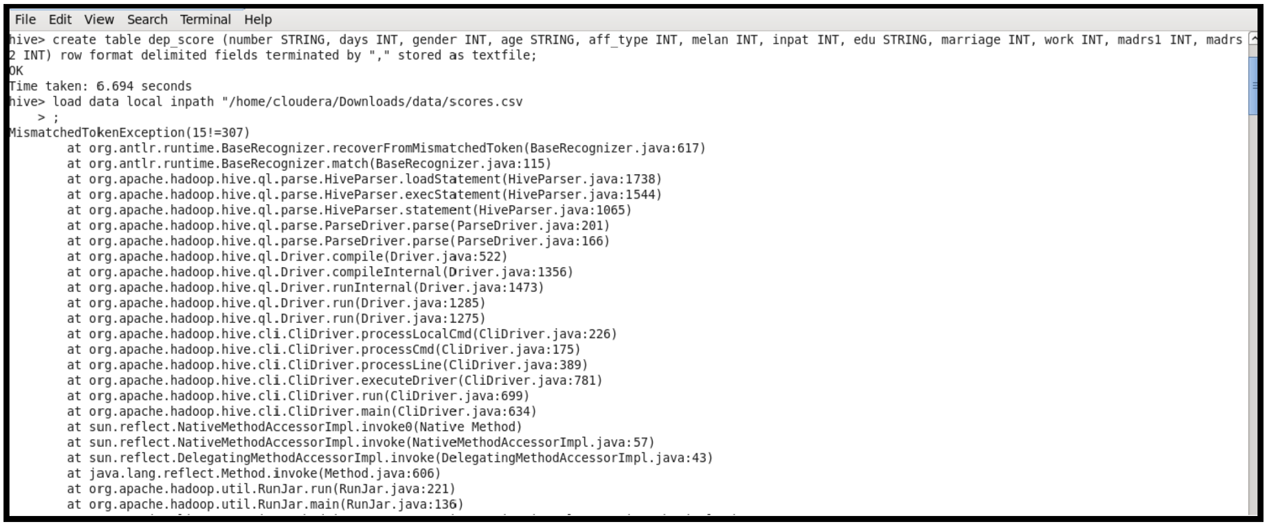
Task	Contributor
Model/Workflow Diagrams	**Davith Lon**
Scraping Reddit Posts	**Davith Lon**
Creating SQL table and Loading Data into Table	**Bryan Khoo**
Query the Data in SQL	**Bryan Khoo**
Sqoop the Data to HDFS	**Bryan Khoo**
Creating HIVE Table and Loading the Dataset	**Davith Lon**
Query the Data for Pre Processing in HIVE	**Davith Lon**
Saving the Cleaned Data to CSV	**Bryan Khoo**
Hadoop Mapreduce	**Ami Khalsa**
Bigrams and Trigrams on “depression”	**Ami Khalsa**

SolR Queries	**Ami Khalsa**
WordCloud	**Ashish Pant**
Loading the Data into PySpark	**Davith Lon**
Labeling Data (Afinn)	**Davith Lon**
Preventing Undersampling and Oversampling	**Bryan Khoo**
Train Test Split	**Bryan Khoo**
Pipeline Element and Pipeline Configuration	**Bryan Khoo**
Model Building	**Davith Lon**
Model Analysis	**Davith Lon**
Twitter API	**Ashish Pant**
Tweet(JSON) Parsing	**Ashish Pant**
Twitter Spark Streaming	**Ashish Pant**
Tweet Cleaning	**Ashish Pant**
Tweet Analysis with Logistic Regression Model	**Ashish Pant**

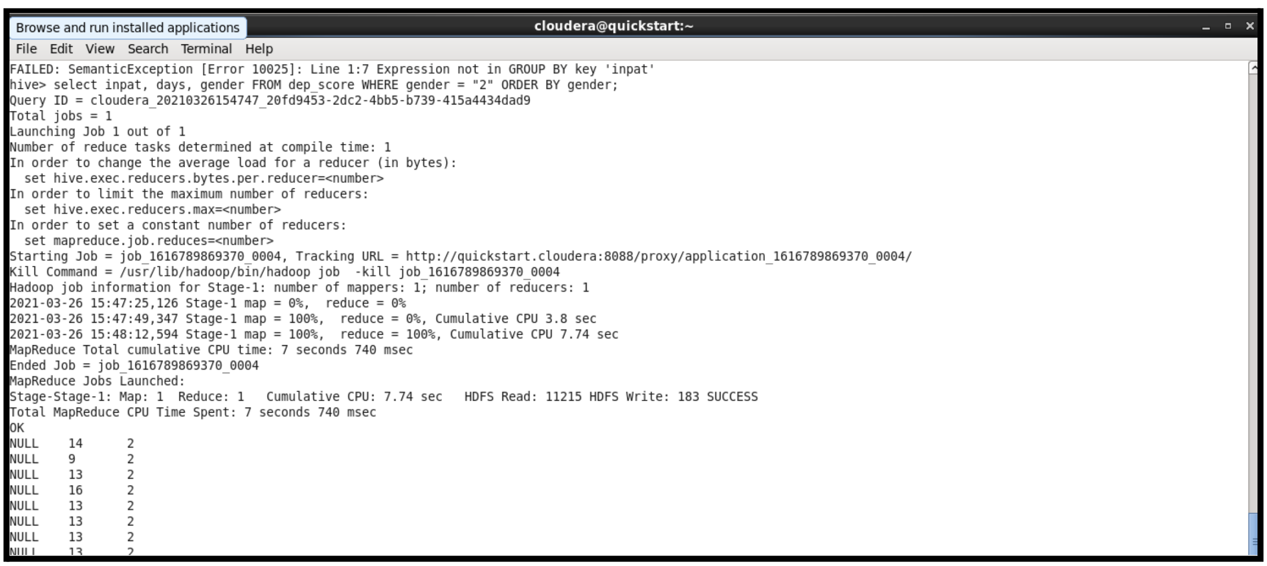
Saurav Pawar (Hive/ PySpark):

The following analysis and queries were done on our old dataset that we were basing our project on. After Saurav had done his analysis on this dataset we realized we needed to change the direction of our project and found other sources of data. We will provide his work down here to show his contributions to the project.

- Uploading dataset on Hive



- SELECT inpat, days, gender FROM dep_score WHERE gender = ‘2’ ORDER BY gender;
- Relationship of the female patients with their days of records from dataset.



- SELECT number, gender, inpat, madrs1 FROM dep_score WHERE (madrs1)>20 ORDER BY number;
- Relationship between patients, their gender and melancholic type from dataset.

```
hive> select number, gender, inpat, madrs1 FROM dep_score WHERE (madrs1)>20 ORDER BY number;
Query ID = cloudera_20210326154242_ecb31609-6421-480a-bd27-077b4c1a1828
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1616789869370_0003, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1616789869370_0003/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1616789869370_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2021-03-26 15:43:11,456 Stage-1 map = 0%,  reduce = 0%
2021-03-26 15:43:44,611 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.28 sec
2021-03-26 15:44:07,524 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.84 sec
MapReduce Total cumulative CPU time: 7 seconds 840 msec
Ended Job = job_1616789869370_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.84 sec   HDFS Read: 11331 HDFS Write: 275 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 840 msec
OK
condition_10    2       2       28
condition_11    1       2       24
condition_12    2       2       25
condition_14    1       2       28
condition_19    2       1       26
condition_2     2       2       24
condition_20    1       1       27
condition_21    2       1       26
condition_22    1       1       29
```

- SELECT melan, mariage, edu FROM dep_score ORDER BY edu desc limit 5;
- Relationship between melancholia, marriage, and education from dataset.

```
30-34   2
40-44   3
Time taken: 89.098 seconds, Fetched: 56 row(s)
hive> select melan, marriage, edu FROM dep_score order by edu desc limit 5;
Query ID = cloudera_20210326150909_0b92a1cf-fe73-4a8a-8e29-56ca03d8683a
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1616789869370_0002, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1616789869370_0002/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1616789869370_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2021-03-26 15:09:59,437 Stage-1 map = 0%,  reduce = 0%
2021-03-26 15:10:13,423 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.01 sec
2021-03-26 15:10:28,285 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.5 sec
MapReduce Total cumulative CPU time: 4 seconds 500 msec
Ended Job = job_1616789869370_0002
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.5 sec   HDFS Read: 10345 HDFS Write: 47 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 500 msec
OK
NULL    NULL    edu
NULL    1       6-10
2       1       6-10
2       2       6-10
```

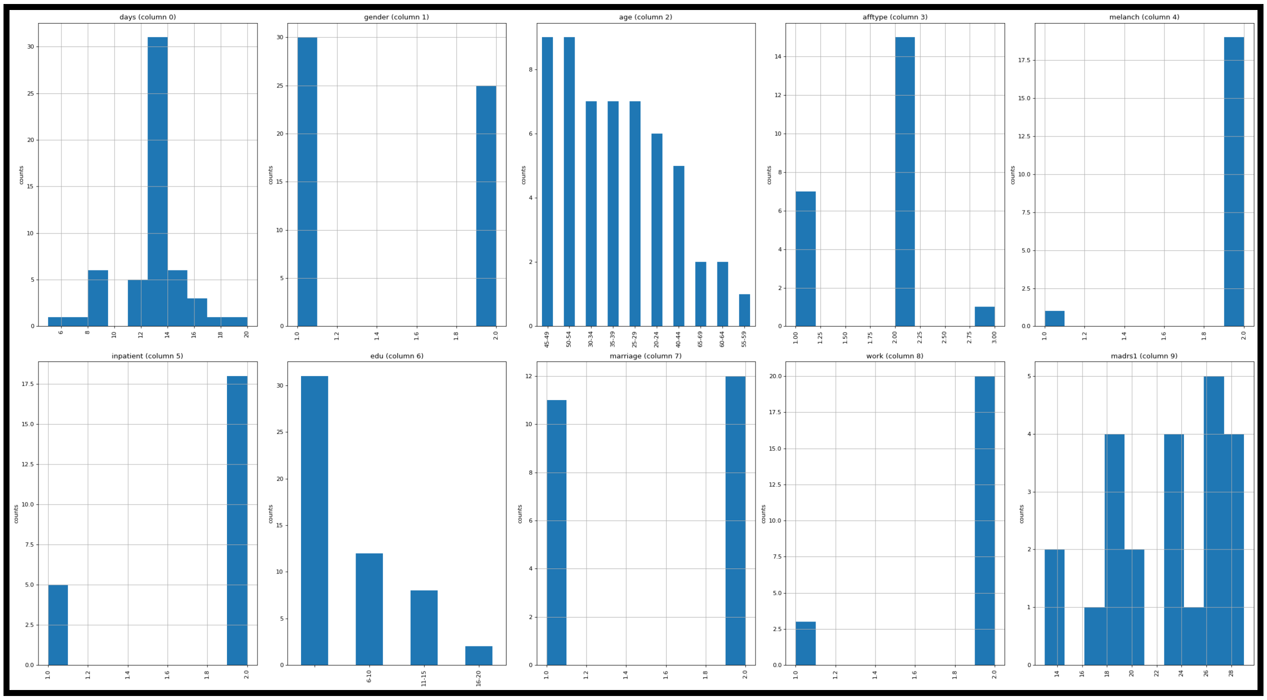
- SELECT age, affinity type FROM dep_score ORDER BY aff_type;
- Relationship between age and affinity type

```
Browse and run installed applications          cloudera@quickstart:~
File  Edit  View  Search  Terminal  Help
hive> select age, aff_type from dep_score ORDER BY aff_type;
Query ID = cloudera_20210326150505_5b520c74-e0d2-40ea-b5c5-167756c169f5
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1616789869370_0001, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1616789869370_0001/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1616789869370_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2021-03-26 15:06:12,957 Stage-1 map = 0%,  reduce = 0%
2021-03-26 15:06:34,822 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.55 sec
2021-03-26 15:06:52,837 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 5.34 sec
MapReduce Total cumulative CPU time: 5 seconds 340 msec
Ended Job = job_1616789869370_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.34 sec   HDFS Read: 9871 HDFS Write: 479 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 340 msec
OK
25-29   NULL
20-24   NULL
35-39   NULL
50-54   NULL
45-49   NULL
50-54   NULL
35-39   NULL
65-69   NULL
20-24   NULL
```

- SELECT age, gender, work FROM dep_score;

```
Access documents, folders and network places          cloudera@quickstart:~
File  Edit  View  Search  Terminal  Help
hive> select gender, age, work from dep_score;
OK
NULL    age     NULL
2       35-39   2
2       40-44   2
1       45-49   2
2       25-29   1
2       50-54   2
1       35-39   2
1       20-24   1
2       25-29   2
2       45-49   2
2       45-49   2
1       45-49   2
2       40-44   2
2       35-39   2
1       60-64   2
2       55-59   1
1       45-49   2
1       50-54   2
2       40-44   2
2       50-54   2
1       30-34   2
2       35-39   2
1       65-69   2
1       30-34   2
2       25-29   NULL
1       30-34   NULL
2       30-34   NULL
1       25-29   NULL
1       30-34   NULL
1       25-29   NULL
```

- (PySpark) Pandas library to visualize the relation between the patients and their contributing factors for depression that are given in the datasets. Such as Age, marital status, work, and so on.


days (column 0)
gender (column 1)
age (column 2)
afftype (column 3)
melanch (column 4)
inpatient (column 5)
edu (column 6)
marriage (column 7)
work (column 8)
madrs1 (column 9)
counts
45-49
50-54
30-34
35-39
25-29
20-24
40-44
65-69
60-64
55-59
6-10
11-15
16-20

Story Telling

Chapter 1: Life

Who?

As per the World Health Organisation more than 264 million people in all the age groups are affected by depression.[8] Even children who are joyful can be affected by depression. Depression is not always related to work or financial issues.

What?

Depression can cause many problems:

- Feeling Sadness or emptiness.
- Insomnia
- Memory or Decision troubles.
- Motivation to sucide.
- Heart Attack.
- Fatigue
- Weakened Immune System
- Overeating or appetite loss leading to weight fluctuations.

These symptoms can interfere with a person's life in many major ways. They can impact education, employment, and relationships. Mental health is a contributing factor in the likelihood of a student dropping out of school,[6] and the symptoms of depression are a significant influence of one's work status.[7]

When?

As of 2021, we are currently facing a worldwide pandemic where billions of people are potentially at risk. Leaders around the globe began advising people to stay isolated at home and be socially distant from each other. This call for separation caused a huge spike in the depression and suicide rate around everywhere. For example, Japan has recently announced, in February 2021, a minister of loneliness “to address matters of national importance ‘including the issue of increasing women’s suicide rate under the pandemic.’”[8]

Where?

Depression is a mental illness that has no constraints on who it can affect. People all around the globe have individuals who are affected by depression. However, what really makes an impact is where the people are. This is so, because societies around the world have different takes on mental illness. While some are more progressive, others still have stigmas about mental illness. With the current pandemic, this makes it even harder for people living in these stigmatized parts of the world.

Why?

Mental illness hasn’t always been an open topic for discussion like it is now. Mental illness has become more accepted in many Western cultures, however many Eastern cultures, for example, still see mental illness as a taboo. So, one part of moving towards removing the stigma on mental illness globally lies both on the societal level and scholarly level. We must be able to provide more clear and concise information about mental health to the public in order to normalize it.

We also have observed significant impacts from depression on an individual’s life. From education to employment and one’s personal life, depression causes numerous challenges to overcome. By examining this issue more closely, we hope to find information that can help reduce the challenges faced and improve lives not only on an individual level but a societal one.

---

[8] https://www.businesstoday.in/current/world/japan-appoints-loneliness-minister-to-tackle-suicide-rates/story/432226.html

[6] https://pubmed.ncbi.nlm.nih.gov/27627885/

[7] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4314052/

Chapter 2: Data

Who:

This dataset includes members of the world who are struggling with depression and/or suicidal thoughts. In our dataset, r/SuicideWatch and r/depression posts from Reddit, we had posts from members of the r/SuicideWatch and r/depression subreddit communities. Both communities were represented equally, meaning 50% of the posts came from r/SuicideWatch and 50% came from r/depression. Since depression is an illness that can affect anyone, we did not filter any demographics out of our dataset.

No identifying information other than what the user provided in their own post was collected. Users are anonymous from the dataset's end. Data about the posting user (age, legal name, gender, etc.) was not collected or used.

What:

The dataset acquired from Kaggle includes posts that were made on each subreddit: r/SuicideWatch and r/depression. The dataset that was pulled from Twitter included the date when the tweet was posted, tweet ID, tweet, name of the account, user ID, followers count for the account, the location of the user, status for the tweet, and whether the twitter account is verified.

When:

In our r/SuicideWatch and r/depression posts from Reddit dataset, the dataset was last updated on Jan. 20, 2021 and covers data from 2008-12-15 to 2021-01-01. It is not real time data. The posts are from a variety of times. Many people relate to old posts just as well as new ones. For this reason, we believe it is useful to examine this data regardless of time since the content of this data is the collection of thoughts of many individuals experiencing struggles with their mental health at the time of their posting. Those thoughts are valuable for us to examine so that we can better understand the people that are currently struggling.

Where:

The dataset acquired from Kaggle was collected from an online forum called Reddit, under the subreddits of r/SuicideWatch and r/depression. Reddit is a platform that is available worldwide which allows us to have the assumption that the dataset obtained is a global dataset. However, the dataset did not include the actual location for each post. Hence, we are not able to conclude where the majority of our dataset originates from.

The dataset acquired from Twitter was collected from social media that is available worldwide as well. Within our dataset, there is a field labeled location which helps us determine the origin for a particular tweet. A section of the data verifies that the majority of the tweets collected were from the United States while still having a few tweets from other countries as well. However, it is important to keep in mind that the location field might not be accurate as some accounts had inputs on the field that were not locatable.

Why:

Fortunately for us, the Kaggle dataset was collected for the exact same reason we're using it. The original poster of the dataset stated, "When I thought of building a text classifier to detect Suicide Ideation I couldn't find any public dataset. Hope this can be useful to anyone looking for suicide detection datasets and can save their time[9]". Additionally, the behaviour and mindset of the posters on these subreddits is exactly what we want fueling the text.

As for the Twitter scraping, the reasoning behind collecting this data is similar to the Reddit postings. The idea is that with the specific hashtags filtering our post collections, we should be able to get organic text live with the same mind set behind it. However, with the Twitter data there is a lot more to filter through. As we are collecting from the hashtags, we can't say for sure everyone is using them the way we think they would be. Also there is the problem of Twitter bots that can create junk for us.

[9] https://www.kaggle.com/nikhileswarkomati/suicide-watch

Chapter 3: The Scientist and AI

Who:

The data scientist understood the domain to be all the people who are depressed and are in need of help in any way possible and the dataset are posts and tweets done by users on Reddit and Twitter.

What:

The dataset in this project was scraped off from Reddit thread where the posts were made by the reddit users. The dataset is then loaded into SQL from the Scraped CSV which then, with the help of Sqoop, is sent into HDFS. The data is then loaded into Hive for cleaning. Once we have clean data, we use Afinn to label it based on sentiment analysis in PySpark. Here, we use three different ML models to see which one performs best. The models we used were Logistic Regression, Naive Bayes and Linear Support Vector Classifier. Out of all three we got the best performance using the Logistic Regression model. Data Scientist needs to have knowledge about what model to use for which kind of task, for example, a classifier can only be used for classification.

When:

When building the models, it took a couple iterations of experimentation in order to get decent results during our final increment. When it comes to the efficiency of the models, the Logistic Regression seemed to perform the best in regards to it's area under the ROC (92%) and its accuracy (85%) and precision (85%) were the same. The Linear Support Vector Classifier did the second best , with an area under the ROC (92%) and its accuracy (84%) and precision (86%) When it came to the Naives Bayes, it performed the worse with an area under the ROC (40%) and its accuracy (77%) and precision (78%). From these results, we decided to go with the Logistic Regression model to use on our live data as its accuracy was 1% higher than the Linear Support Vector Classifier. We preferred accuracy over precision when comparing the Logistic Regression compared to the Linear Support Vector Classifier.

Where:

The Experiment was a part of an Applied Programming Course at UMKC under guidance of Zeenat Tariq.

Why:

The Machine Learning models used in this project are:

- Logistic Regression
    - Logistic Regression is a type of predictive analysis[10]. By using this Machine Learning model, we are able to classify whether a particular text is depressive or

[10] https://www.statisticssolutions.com/what-is-logistic-regression/

not. The model obtains its features after the Inverse Document Frequency Stage of the Pipelines.

- Naives Bayes
    - Naive Bayes is a type of classification algorithm. Similarly with the Logistic Regression, we use this algorithm to predict whether a given text is depressive.
- Linear Support Vector Classification
    - LSVC is a classification algorithm, thus this is why we also built this model.

Chapter 4: Users

Who:

The user of this application can be used by different organisations like Substance Abuse and Mental Health Services Administration, National Institute of Mental Health and many social media sites like Facebook, Twitter and Reddit to keep a watch on those who make depressive posts.

What:

The Application can take a text and tell if the post indicates a person is at risk for suicide and depression or not. This can identify high risk users who may be contemplating ending their life, giving the social media companies the chance to send them a message or a notification, which may interrupt the suicidal thoughts or assist the users in getting help. Currently, it works on Live Twitter Stream.

When:

It can be used at all times. The streaming of the live tweets from Twitter is an example of a 24/7 use of the application.

Where:

The Application will be deployed on the web or server as a watcher for those who post on social media.

Why:

It can work on it’s own without any interference from the user and keep a monitor on the posts at all times. Considering big data, one of the largest forms of data is the amount of content social media users are putting out daily, worldwide. It is simply too much data for any one person to be expected to filter through. This type of automation eliminates the man power needed to determine if a post is depressive.

How:

The application can be used by SAMHSA, NIMH to locate depressed people on the internet. These organizations could then alter their programs and organization to cater to the results of the application.

Chapter 5: The Society

Who:

The people who are depressed will be impacted as they will be able to get help without even asking for it. Here we sampled 50,000 Depressed posts and 50,000 Normal posts. There was no over or under sampling. The Data Scientists who worked on this are Ami Khalsa, Ashish Pant, Bryan Khoo, Davith Lon and Saurav Pawar.

What:

The social impact is betterment of society by helping out those in need. There is no effect on privacy, security and fairness as the data that was taken was public and from social media platforms.

When:

The impacts can be seen when our application gets implemented on a large scale. The concern comes when this data is used for advertisement or spying on people. If in any way, shape or form this application is used maliciously the application or system should be suspended immediately.

Where:

The impact will happen all over the world but currently the application only has English support so the impact will only be seen in places that use english to communicate online. There can only be one issue, that is if the culture that it is used in is different from the one it was trained in then it can show some fairness issues.

Why:

The impacts are important because it will be for the wellbeing of the humanity race the mental health of all those that can be helped will get much better and they can live a much better life.

How:

Since people are getting to know more about how machine learning can help them, they are getting more accepting towards it. In the past decade we have seen the rise of smart devices and if we keep the community in the loop, we can educate them about how our app can also affect them positively.

REFERENCES

[1] https://www.psychiatry.org/patients-families/depression/what-is-depression
[2]https://www.sciencedirect.com/science/article/abs/pii/S002002551830094X
[3] https://www.nimh.nih.gov/health/topics/depression/index.shtml
[4] https://www.who.int/news-room/fact-sheets/detail/depression
[5]
https://www.businesstoday.in/current/world/japan-appoints-loneliness-minister-to-tackle-suicide-rates/story/432226.html
[6] https://pubmed.ncbi.nlm.nih.gov/29195763/
[7] https://pubmed.ncbi.nlm.nih.gov/27627885/
[8] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4314052/
[9]https://www.who.int/news-room/fact-sheets/detail/depression#:~:text=Depression%20is%20a%20common%20mental,affected%20by%20depression%20than%20men.