FINAL REPORT

Sentiment Analysis

KUNWAR RANANJAY SINGH
2017B4A70504G
f20170504@goa.bits-pilani.ac.in

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
GOA CAMPUS

CONTENTS

Problem

Aim is to classify social media content using data science into positive, negative and neutral categories. It's also known as sentiment analysis.
For example : "enjoyed travelling today. Will visit again" can be classified as positive whereas "lot of rain today. Depressing !!!" can be classified as negative.

Social media has a lot of impact on the human brain and using technology which can classify posts into various categories can be used to filter out those which have a negative impact. It can be also used by companies to only present posts of a certain category that the user has selected(for example filtering out negative posts). Customer posts on websites can be used to improve products. It can be used by governments to reduce spread of misinformation and can also be used for fake news detection [1].

Related Work

Let's see some of the important points from the work done by other researchers.

Majority of the work in the field of sentiment analysis has been done on twitter data. In previous works it has been observed that an algorithm provides more accurate results on short tweets. In one of the studies done on COVID-19 tweets, Naive Bayes is better compared to Logistic Regression in terms of accuracy for both short and long tweets. Each model has its own strengths and weaknesses. Linear Regression is much easier to understand conceptually however it's prediction accuracy can be affected by outliers. Logistic Regression can be used to avoid overfitting compared to Polynomial Regression however it depends on assumption that data is linear. For example in tweets like "I am feeling sad today", "happy to visit Goa", we can see words like "sad", "happy" which would be present in a lot of negatively and positively classified tweets respectively and thus it becomes easier to classify them using logistic regression. However in tweets like "Movie was bad even though all the cast and crew members were great", we can see it has words like "bad", "great" which are the only words impacting sentiment of this tweet. Overall it has negative sentiment however by just going with the meaning of words it would be neutral. The Naive Bayes classifier has an assumption that features are independent. K-Nearest neighbour can be used to classify data into multiple categories like fear, sadness, anger, disgust however it's computationally inefficient for large data sets. According to some of the word clouds (where the size of a word is proportional to its frequency) it was observed that "trump", "virus", "stock" were present in large frequency in a lot of covid-19 tweets. This observation can be utilised to make more sophisticated preprocessing techniques. For example if the majority of tweets containing word "stock" have negative sentiment then while classifying we can have additional negative weightage to tweets having word "stock" provided we are classifying tweets during COVID-19 period [2].

Some of the common pre-processing techniques used are removing URLs, hash tags, usernames, correction of spelling, removing emoticons with their sentiment, removing punctuation, stop words, removing non-english words in tweets. Most work didn't use unsupervised machine learning techniques because we need some kind of mapping of tweets to some quantity which can be utilised for sentiment analysis. Just putting an English word won't give any meaning to the unsupervised algorithms. Lexicon based approaches in which semantic orientation of words and phrases are used seem to have less accuracy compared to machine learning techniques. As expected, accuracy of any ML algorithm is better when the training set is large. It has been observed that bigram which refers to a combination of two words can increase the accuracy since putting bigrams like "not sad", "not good" (they correspond to positive and negative sentiment, respectively) in calculation would reduce the effect of unigrams "sad", "good" (they correspond to negative and positive sentiment, respectively). In one of the experiments accuracy increased to 76.44 from 74.56 percent by using bigrams [3].

Lexicon based approaches can be used for unsupervised learning, but it has its own challenges. Polarity of the words in the sentence can be used to determine the sentiment but natural languages are complex and it won't consider things like negation.
Taking account of slang and emoticons in the sentences can improve the sentiment classification accuracy of the tweets. One study has shown that the presence of emoticons in tweets increased the accuracy from 79% to 85%[4].

Approach and Solution

Focus will be on tweets since they have a character limit of 280 [5]. This is because as the number of characters increases the post becomes more vague and less objective in terms of classifying into positive or negative whereas in tweets it's mostly inclined toward some emotion due to character limit.

I started with the most naive solution. Following steps were followed while coding the naive solution:

1. Splitting the tweets in tokens using space as a delimiter.
2. Converting the words into lower case.
3. Ignoring the words containing characters other than alpha-numeric.
4. Using a dictionary to map a token to count of sentiment (-1 is added if the token belongs to a tweet which has negative sentiment else +1 is added) . In other words we are creating a frequency table for each token where the value added while iterating the tweets is negative if the tweet belongs to negative sentiment else positive.
5. Splitting the dataset into 80-20 for training and testing purposes respectively. Two vectors are created, the first one contains the sum of the dictionary value of tokens in the tweet and the second contains corresponding sentiment (0 for negative, 1 for positive).
6. Logistic Regression and Support Vector Machine are used individually for training and then predicting the sentiment for test data.
7. Confusion matrix and accuracy is printed.

After that I thought of ways (on paper) to clean and preprocess the data in a better way, better and different ways of representing the data mathematically and using more algorithms and techniques. I implemented them in python.

There were four major steps :

1. Choosing the datasets.
2. Preprocessing and cleaning of the data.
3. Representing data mathematically.
4. Using algorithms for training and testing.

Choosing the datasets

Two different datasets were chosen to test the performance. First dataset contains random tweets (they don't belong to some specific domain). From now on we will refer to the first dataset as dataset-1. Second dataset contains tweets related to US airlines and we will refer to it as dataset-2.

Preprocessing and cleaning of the data

Following steps represents the whole process in preprocessing and cleaning of the data :

1. Dataframe rows are shuffled. It's possible that a lot of negative or positive tweets are concentrated at some consecutive set of rows in the dataset which can affect training process and test results. To avoid such a situation rows were shuffled.
2. Columns other than those having tweets or sentiment are removed. For example, a column having the date of tweet has no role in determining the sentiment.
3. Sentiment column values are mapped to +1 if tweet was having positive sentiment else -1 if tweet was having negative sentiment. Algorithms work with numbers and thus it's required to map sentiment to +1 or -1.
4. Emojis are replaced with their meaning. For example, happy emoji is replaced with the word "happy".
5. Contractions are expanded. For example "won't" is converted to "will not". This is helpful because there are sentences with the word "will not" and it would be helpful to consider their meaning similar to "won't".
6. Hashtag from words is removed. For example "#summer" is replaced with "summer". Words attached with hash convey meaning and thus it's important to not remove them completely.
7. "RT" representing retweet is removed. "RT" doesn't have any effect on sentiment and thus it should be removed.
8. Words in tweets having double forward slashes "//" are removed. Usually words in tweet with '#' or having underscores convey some meaning. However words with "//" are rare and they usually represent some website which has no role in determining the sentiment.
9. Positions of words having '!' (exclamation mark) are noted and '!' are removed. Finally Words having characters other than lower case, upper case alphabet, underscore are removed. Words beginning with '@' for example '@daniel' represent some other profile and thus it is not useful for determining the sentiment. Exclamation marks usually represent the extreme end of emotion and thus keeping track of words attached with an exclamation mark at the end can be useful for final sentiment calculation.
10. Tweets are changed to lowercase. "Not" and "not" convey the same meaning and thus we need to change all words into either lowercase or uppercase.
11. Words having underscores are splitted. For example "good_day_today" is changed to "good day today". Words having underscore are found frequently in tweets (mostly with hash at the front like #good_day_today).
12. Spell corrections are done. For example "wowwwwww" can be changed to "woww" (i.e removing multiple repeating characters) using regex and then further corrected to "wow" with spell correction packages.
13. Stop words other than "not" are removed. "is", "a" are some examples of stop words. They usually don't convey any meaning from sentiment analysis perspective. However if they are not removed from data then they can skew the results. For example if "a" is present in a lot of negative tweets in training data and in testing data if it's present in some tweet having positive sentiment, then it will try to push the result toward negative sentiment. "not" is not removed because it acts as negation and useful in cases like "not happy" where it changes the overall meaning.
14. Lemmatization of tweets is done. Stemming and Lemmatization can be used to shorten the words and group together similar words. For example in "Saurabh is Intern", "Saurabh is doing internship", both "intern" and "internship" convey the same thing from sentiment

analysis perspective and thus changing “internship” to “intern” will give better results. Lemmatization was used because it takes care of “part of speech tagging” i.e it considers the context and then makes the word short accordingly. For example it would be bad if both “international” and “internship” is reduced to “intern”. If we take another example to understand the importance of part of speech tagging : in “The man fans the flame”, “fans” is a verb whereas in “The fans watch the race” “fans” is a noun.

15. Those words which are having ‘!’ at the end of them, they are replaced with twice repetition of themselves. For example ‘happy!!’ replaced with ‘happy happy’.
16. Tuples are appended at the end of the tweets. For example, for the tweet : “not good today” and n=2, the tweet will be changed to “not good today _not_good _good_today”. There are words which often act as negation and thus it becomes important to consider the meaning of consecutive words as a group. For example in “not happy today”, “not” is acting as negation and we don’t need to see “not” and “happy” separately. If we have made tuples, then “_not_happy” will be there in the final modified tweet and thus it might prevent from displaying false sentiment. If we take another example : “This comedy movie wasn’t that great. It didn’t make the audience happy”. Here we can see that it's a negative sentiment. However it has words like “great”, “happy” which would convey positive sentiment. Words that decided the sentiment are “wasn’t” and “didn’t”. They act as negation. However “wasn’t”, “didn’t” are two word distance apart from “great”, “happy” and bigrams won’t help here. We need to consider 3 to 4 words together in a lot of cases to get true sentiment and thus I have considered upto 4-grams(tuples of size upto 4).
17. Dataframe having sentiment column and tweet column is saved as csv file which can be used further for testing with different models. All the previous process takes a lot of time (around 4 hours on my local computer for 20000 tweets and tuples size 4) and thus it would save time if we keep the modified data frame saved so that we don’t need to create it every time we use it in our model.

Representing data mathematically

Following three different techniques were used to represent the data mathematically so that it can be used by algorithms :

1. Positive-Negative frequency. We create a dictionary where key is a word and value is a pair. First value in the pair represents the number of tweets in training datasets which are positive and that word is present in it. Second value in the pair represents the number of tweets in training datasets which are negative and that word is present in it. In the final dataframe there are three columns. First column represents the sum of positive values of all words in the corresponding tweet at that row index. Second column represents the sum of negative values of all words in the corresponding tweet at that row index. Third column is the corresponding sentiment. Let’s understand it with example of three tweets where we choose 2 tweets for training and 1 tweet for testing :

tweet	sentiment

“i am happy today”	1
“i am sad today”	-1
”i am feeling happy”	1

Following dictionary would be created from first two rows :

word	positive sum	negative sum
i	1	-1
am	1	-1
happy	1	0
today	1	-1
sad	0	-1

Final dataframe where tweet column is replaced by two columns :

positive sum	negative sum	sentiment
4	-3	1
3	-4	-1
3	-2	1

Now we can use the first two rows for training and the final row for testing.

2. Bag of Words. Here columns are words (number of columns is equal to number of unique words). In each row and for each word we write the count of the number of times that word occurs in the corresponding tweet at the same row. An example will make this clear :

tweet	sentiment
“i am happy today”	1
“i am sad today”	-1
”i am feeling happy”	1

Corresponding data frame representing bag of words and last column as sentiment :

i	am	happy	today	sad	feeling	sentiment
1	1	1	1	0	0	1
1	1	0	1	1	0	-1
1	1	1	0	0	1	1

3. TF-IDF. Term Frequency (TF) of a word in a sentence is defined as $\frac{c}{L}$ where c is frequency of that word in the sentence and L is the total number of words in the sentence. For example in sentence "i like to travel and i like to listen music", TF of word "like" is $\frac{2}{10}$. Inverse Document Frequency (IDF) is defined as $log(\frac{N}{n})$ where N is total number of sentences and n is the number of sentences in which that word occurs. For example if we are given two sentences : "i like to travel and i like to listen music", "i am sad because we lost". Then the IDF of "like" is $log(\frac{2}{1})$ since it occurs only in the first sentence and there are a total two sentences. TF-IDF of a word = TF*IDF. For example in previous two sentences, TF-IDF of "like" in first sentence is $\frac{2}{10}*log(\frac{2}{1}) = 0.0602$. TF-IDF of "i" in first sentence is $\frac{2}{10}*log(\frac{2}{2}) = 0$. High TF-IDF of a word in a sentence implies that it occurs in that sentence more times compared to other sentences. Word "like" occurs only in the first sentence whereas word "i" occurred in both of them and thus word "like" has more TF-IDF value in the first sentence compared to "i". Our intuition also says to give "like" more importance compared to "i" while finding the sentiment.
   In our dataframe we treat the tweets as sentences and columns are words. Then in each row we write TF-IDF value for each word for the corresponding tweet.
   Let's understand it with help of an example :

tweet	sentiment
"i am happy today"	1
"i am sad today"	-1
"i am feeling happy"	1

   The corresponding data frame having TF values is :

i	am	happy	today	sad	feeling	sentiment
$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{0}{4}$	$\frac{0}{4}$	1

$\frac{1}{4}$	$\frac{1}{4}$	$\frac{0}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{0}{4}$	-1
$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{0}{4}$	$\frac{0}{4}$	$\frac{1}{4}$	1

The corresponding data frame having IDF values is :

i	am	happy	today	sad	feeling	sentiment
$log(\frac{3}{3})$	$log(\frac{3}{3})$	$log(\frac{3}{2})$	$log(\frac{3}{2})$	$log(\frac{3}{1})$	$log(\frac{3}{1})$	1
$log(\frac{3}{3})$	$log(\frac{3}{3})$	$log(\frac{3}{2})$	$log(\frac{3}{2})$	$log(\frac{3}{1})$	$log(\frac{3}{1})$	-1
$log(\frac{3}{3})$	$log(\frac{3}{3})$	$log(\frac{3}{2})$	$log(\frac{3}{2})$	$log(\frac{3}{1})$	$log(\frac{3}{1})$	1

The final dataframe with TF-IDF values is :

i	am	happy	today	sad	feeling	sentiment
0	0	0.044	0.044	0	0	1
0	0	0	0.044	0.119	0	-1
0	0	0.044	0	0	0.119	1

We can see that words like “happy”, “sad”, “feeling” have non-zero tf-idf value in the sentences in which they appear whereas “i”, ”am” have zero tf-idf value. Intuitively also we can say that “sad” must be given more importance than “i” in determining the sentiment. TF-IDF gives more importance to words which occur frequently only in a few sentences and less importance to words which occur in a lot of sentences.

Using algorithms for training and testing

Logistic Regression, Naive Bayes, Support Vector Machine (SVM), Random Forest, K-Nearest Neighbors were used for classifying independently. Ensemble learning techniques like XGBoost, Voting Classifier were also used.
Some of the algorithms used (for example Support Vector Machine) took more than a day for training and testing. Thus I used cuDF and cuML (since they can utilize GPU and thus can utilize google colab facility) in place of pandas and sklearn respectively for running large datasets with some of the algorithms.

Each algorithm was runned with different n-grams (i.e dataset with tweets having no tuples and different datasets having 2 to 4 tuples).

For every training and testing purpose, the dataset was divided into 80-20, respectively.

Code and Output

Complete source code can be found on my github page (link).

Format of the confusion matrix printed in output is :

True Negatives	False Positives
False Negatives	True Positives

Negative refers to negative sentiment and Positive refers to positive sentiment.

Dataset was divided into 80-20 for training and testing every time.

Following are the screenshots of naive solution for different dataset sizes:

```
Confusion Matrix for LogisticRegression :
[[18 34]
 [ 6 41]]
Accuracy :
0.5959595959595959
Confusion Matrix for SVM :
[[25 27]
 [ 9 38]]
Accuracy :
0.6363636363636364
Total test data (around 20%) for both cases :
99
```

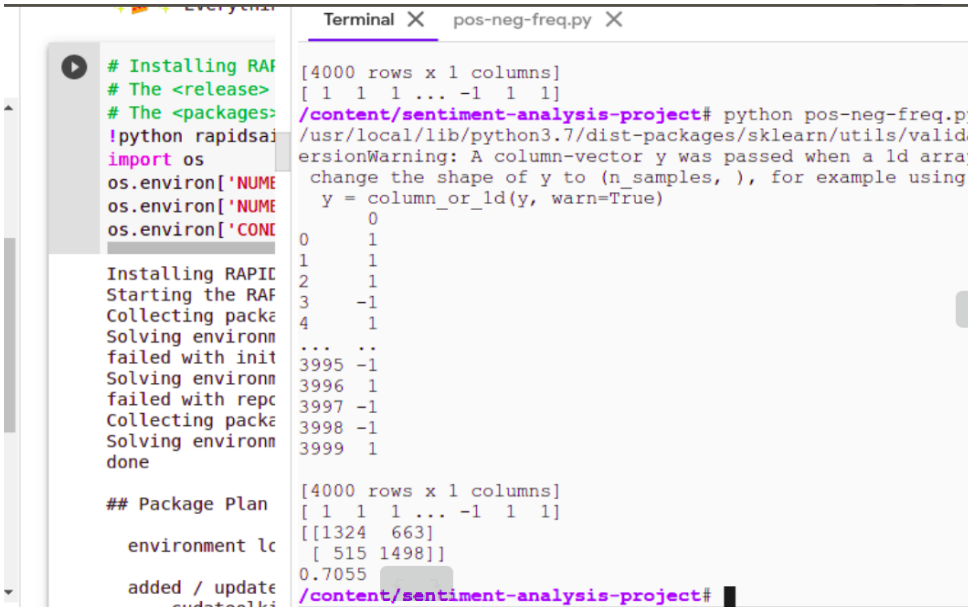
```
Confusion Matrix for LogisticRegression :
[[238 722]
 [111 928]]
Accuracy :
0.5832916458229115
Confusion Matrix for SVM :
[[507 453]
 [291 748]]
Accuracy :
0.6278139069534767
Total test data (around 20%) for both cases :
1999
```

Following is the screenshot of preprocessing of an example tweet :

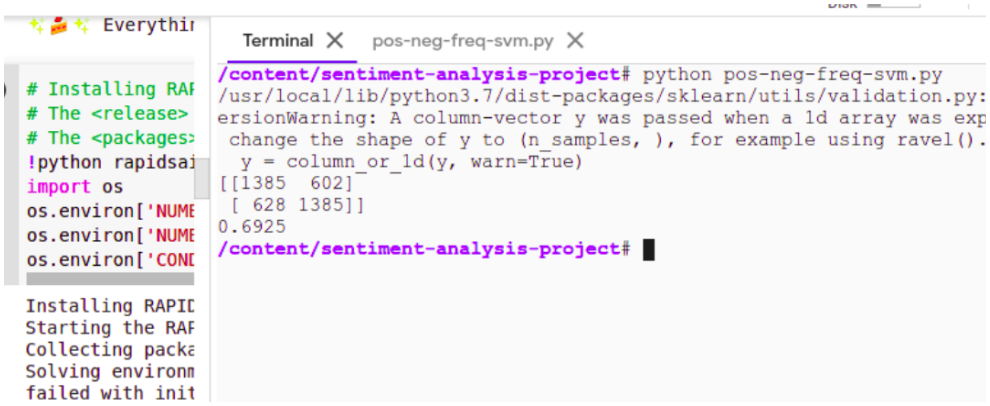
Let's see the screenshots of all the experiments. Note that different screenshots have different train and test size (it can be seen from the sum of the values in the confusion matrix). This is because some models required more memory size and thus small dataset size was used for them. Evaluation of the output will be discussed in the next section.

Following are the screenshots of confusion matrix and accuracy obtained for dataset-1 and positive-negative frequency :

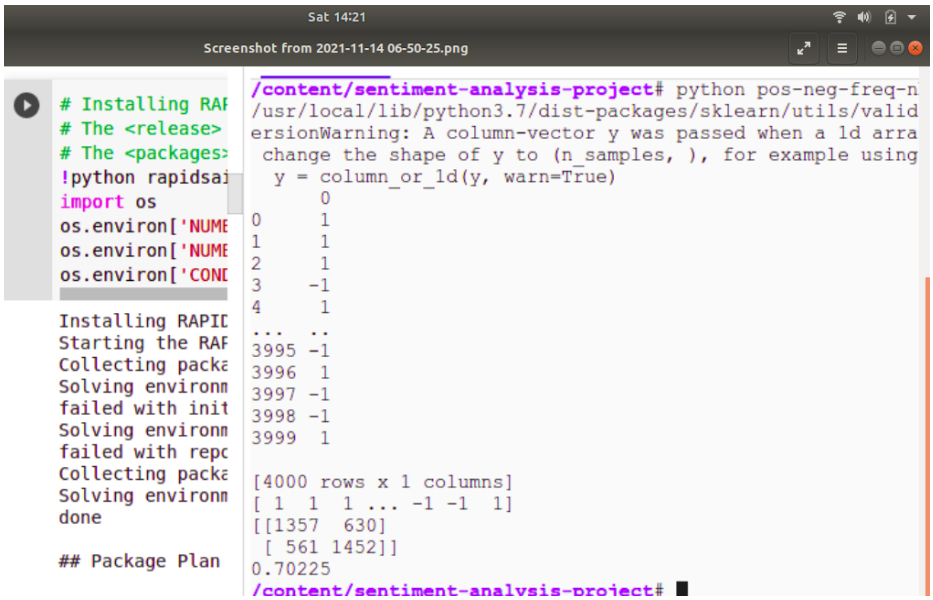
1. positive-negative frequency, 1-gram, logistic regression.



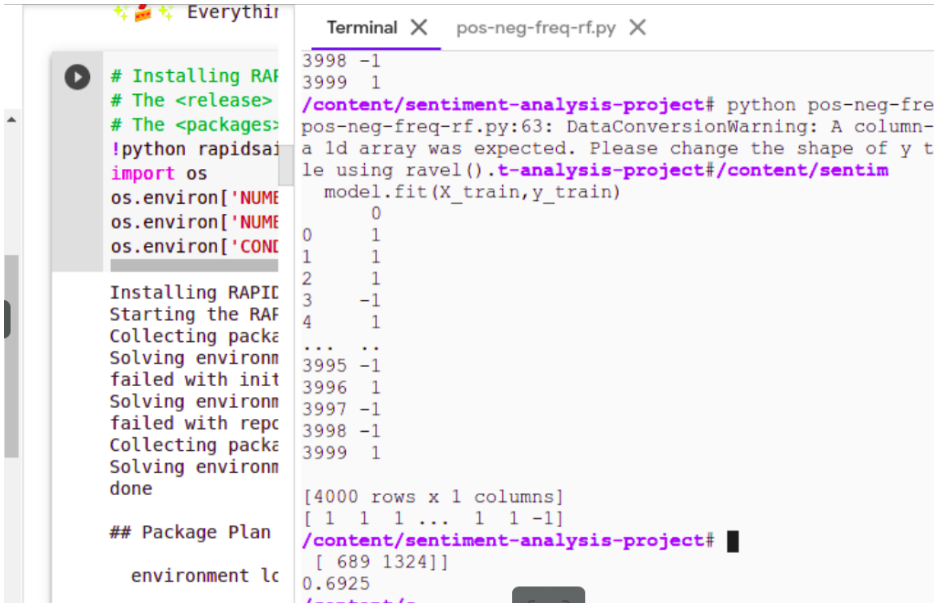
2. positive-negative frequency, 1-gram, SVM.



3. positive-negative frequency, 1-gram, multinomial naive bayes.



4. positive-negative frequency, 1-gram, random forest.



5. positive-negative frequency, 1-gram, KNN (K-Nearest Neighbors).

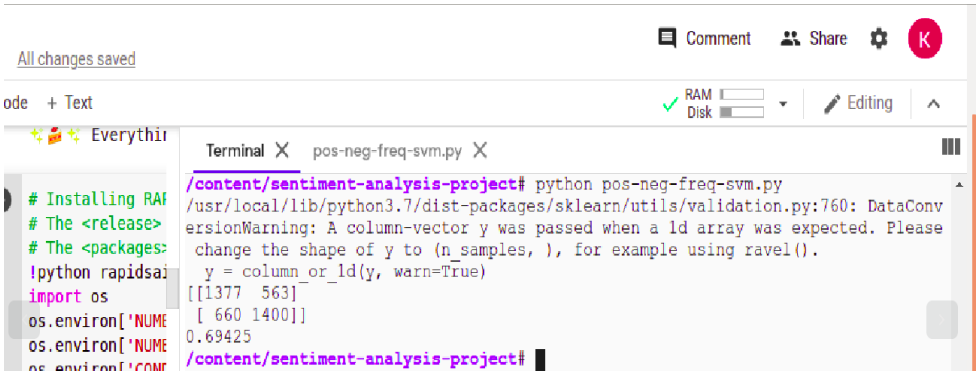
```
Terminal  pos-neg-freq-kn.py
/content/sentiment-analysis-project/sentiment-analysis-of-dataset-1# python pos-neg-freq-kn.py
/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198: DataConversior
rning: A column-vector y was passed when a 1d array was expected. Please change the shape of y
 (n_samples,), for example using ravel().
  return self._fit(X, y)
[[1449  538]
 [ 678 1335]]
0.696
/content/sentiment-analysis-project/sentiment-analysis-of-dataset-1#
```

6. positive-negative frequency, 2-grams, logistic regression.

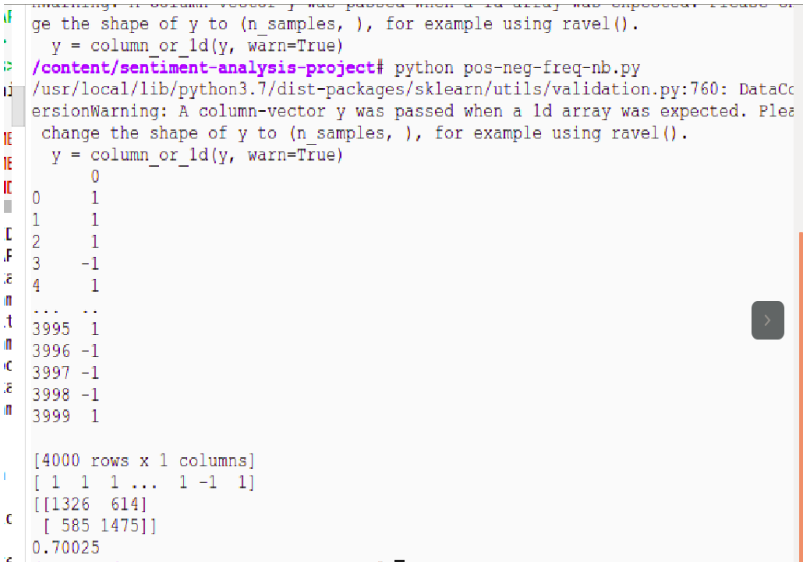
```
# Installing RAF
# The <release>
# The <packages>
!python rapidsai
import os
os.environ['NUME
os.environ['NUME
os.environ['CONI

Installing RAPID
Starting the RAF
Collecting packa
Solving environm
failed with init
Solving environm
failed with repo
Collecting packa
Solving environm
done

## Package Plan

  environment lo
```

```
/content/sentiment-analysis-project# passed when a 1d array w
t/sentim
/content/sentiment-analysis-project# python pos-neg-freq.py
^H/usr/local/lib/python3.7/dist-packages/sklearn/utils/valida
nversionWarning: A column-vector y was passed when a 1d array
se change the shape of y to (n_samples, ), for example using
  y = column_or_1d(y, warn=True)
      0
0     1
1     1
2     1
3    -1
4     1
...  ..
3995  1
3996 -1
3997 -1
3998 -1
3999  1

[4000 rows x 1 columns]
[ 1  1  1 ...  1 -1  1]
/content/sentiment-analysis-project#
 [ 543 1517]]
0.70275
```

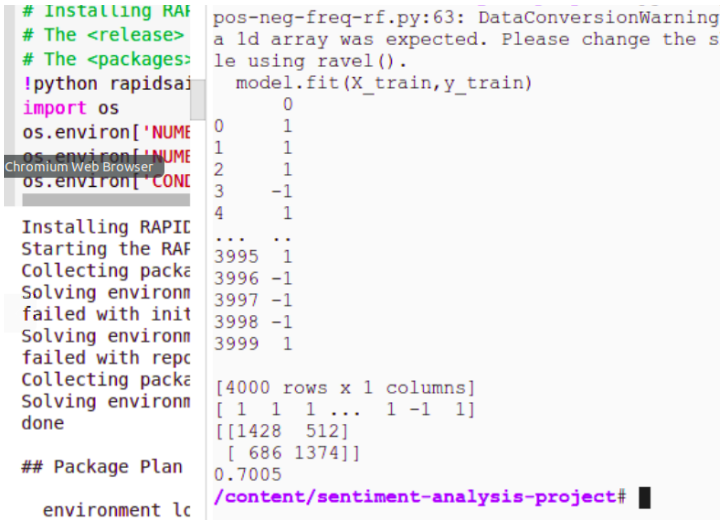
7. positive-negative frequency, 2-grams, SVM.



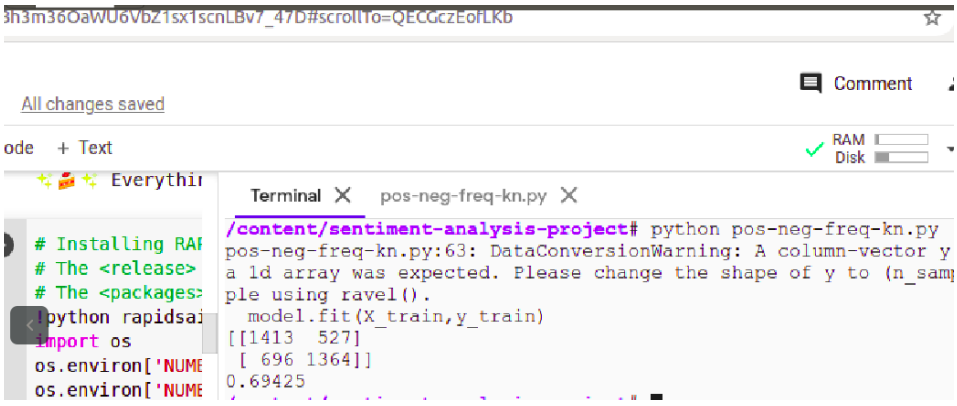
8. positive-negative frequency, 2-grams, multinomial naive bayes.



9. positive-negative frequency, 2-grams, random forest.



10. positive-negative frequency, 2-grams, KNN.



11. positive-negative frequency, 3-grams, logistic regression.

```
# Installing RAF
# The <release>
# The <packages>
!python rapidsai
import os
os.environ['NUME
os.environ['NUME
os.environ['COND

Installing RAPID
Starting the RAF
Collecting packa
Solving environm
failed with init
Solving environm
failed with repo
Collecting packa
Solving environm
done

## Package Plan

  environment lo
```

```
t/sentim
/content/sentiment-analysis-project# python pos-neg-freq.py
^H/usr/local/lib/python3.7/dist-packages/sklearn/utils/valid
nversionWarning: A column-vector y was passed when a 1d arra
se change the shape of y to (n_samples, ), for example using
  y = column_or_1d(y, warn=True)
      0
0     1
1     1
2     1
3    -1
4     1
...  ..
3995  1
3996 -1
3997 -1
3998 -1
3999  1

[4000 rows x 1 columns]
[ 1  1  1 ...  1 -1  1]
/content/sentiment-analysis-project#
 [ 543 1517]]
0.70275
/content/sentiment-analysis-project#/content/sentim
```

12. positive-negative frequency, 3-grams, SVM.

```
Everythir

# Installing RAF
# The <release>
# The <packages>
!python rapidsai
import os
os.environ['NUME
os.environ['NUME
os.environ['COND

Installing RAPID
Starting the RAF
Collecting packa
```

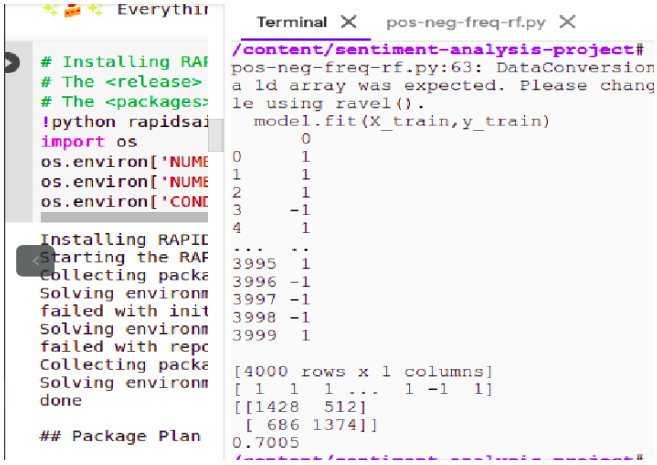
```
Terminal   pos-neg-freq-svm.py
/content/sentiment-analysis-project# python pos-neg-freq-svm.py
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py
ersionWarning: A column-vector y was passed when a 1d array was ex
 change the shape of y to (n_samples, ), for example using ravel()
  y = column_or_1d(y, warn=True)
[[1377  563]
 [ 660 1400]]
0.69425
/content/sentiment-analysis-project#
```

13. positive-negative frequency, 3-grams, naive bayes.

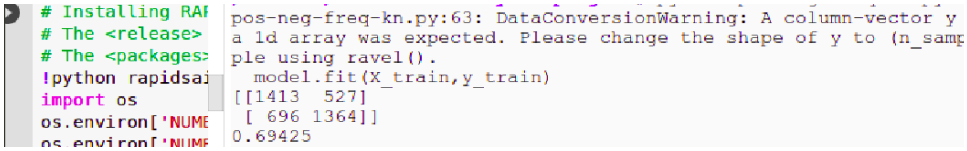
```
# The <packages>
!python rapidsai
import os
os.environ['NUME
os.environ['NUME
os.environ['COND

Installing RAPID
Starting the RAF
Collecting packa
Solving environm
failed with init
Solving environm
failed with repo
Collecting packa
Solving environm
done

## Package Plan

  environment lo

  added / update
```

```
/content/sentiment-analysis-project# python pos-neg-freq-nb.py
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation
ersionWarning: A column-vector y was passed when a 1d array was
 change the shape of y to (n_samples, ), for example using rave
  y = column_or_1d(y, warn=True)
      0
0     1
1     1
2     1
3    -1
4     1
...  ..
3995  1
3996 -1
3997 -1
3998 -1
3999  1

[4000 rows x 1 columns]
[ 1  1  1 ...  1 -1  1]
[[1326  614]
 [ 585 1475]]
0.70025
```

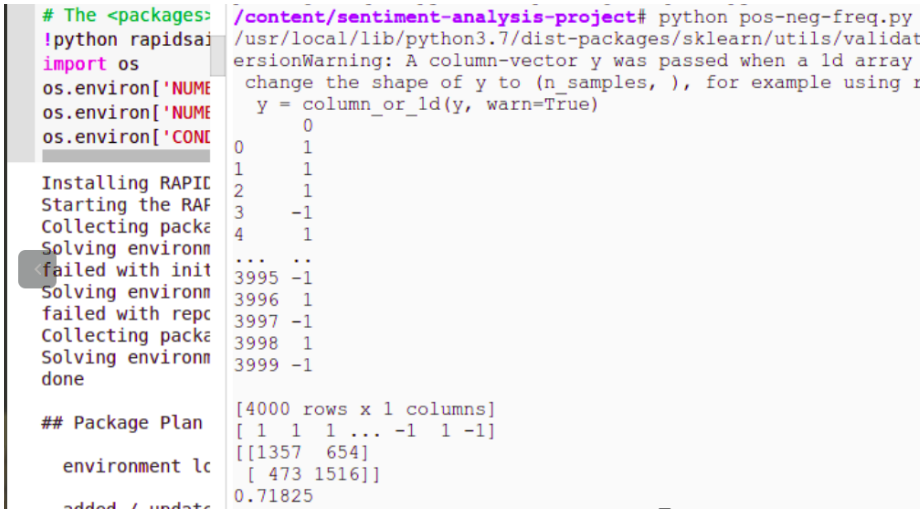
14. positive-negative frequency, 3-grams, random forest.



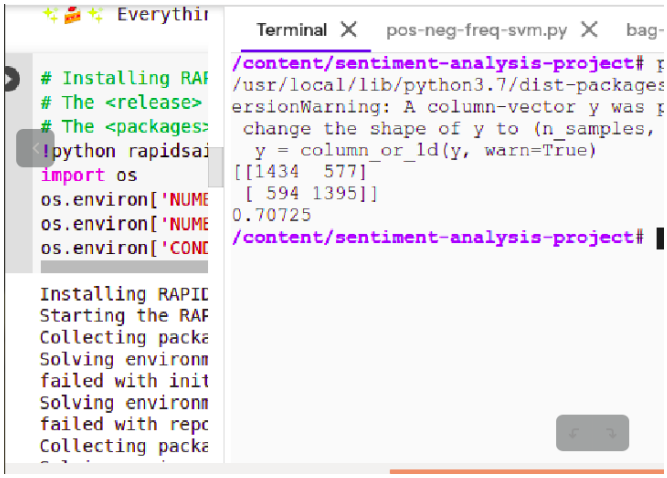
15. positive-negative frequency, 3-grams, KNN.



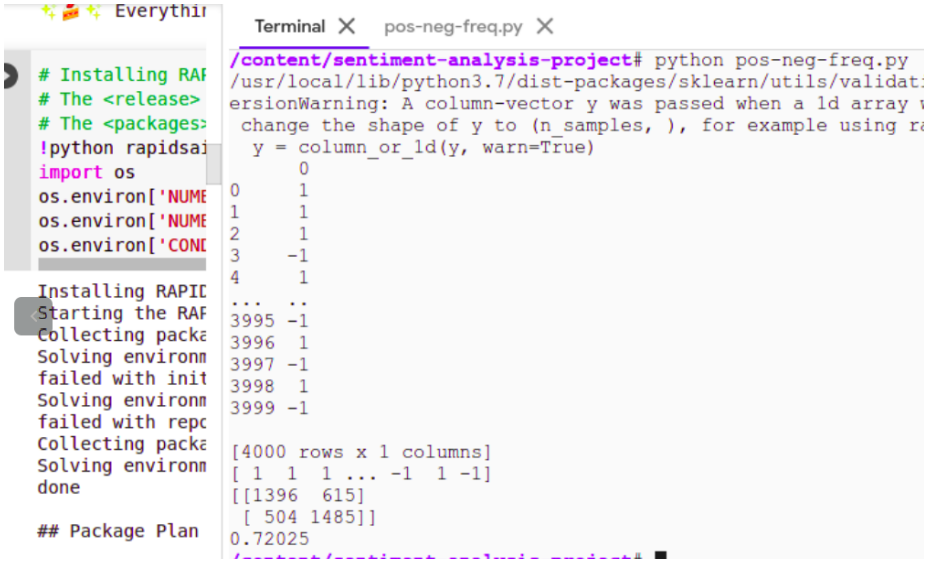
16. positive-negative frequency, 4-grams, logistic regression.



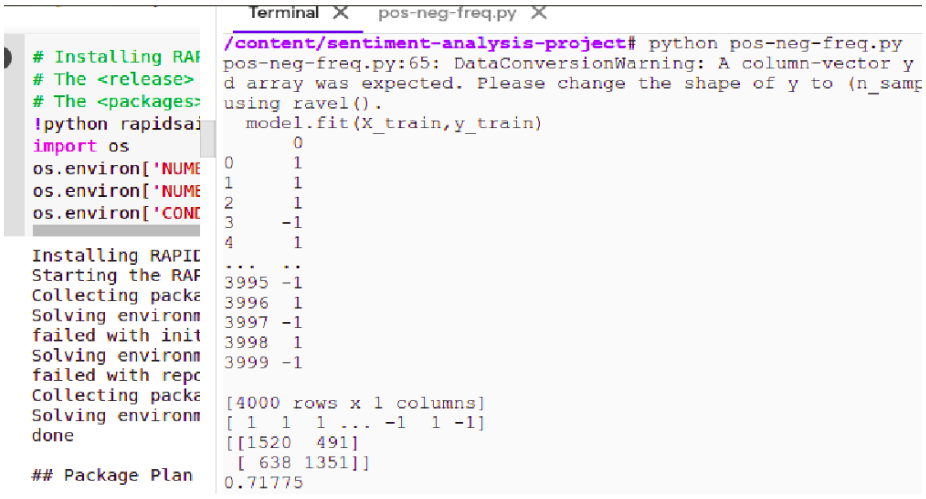
17. positive-negative frequency, 4-grams, SVM.



18. positive-negative frequency, 4-grams, naive bayes.



19. positive-negative frequency, 4-grams, random forest.



20. positive-negative frequency, 4-grams, KNN.

```
Solving environm
done

## Package Plan

  environment lo

  added / update
    - cudatoolki

/content/sentiment-analysis-project# python pos-neg-freq-kn.py
pos-neg-freq-kn.py:63: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_samples, ), for exam
ple using ravel().-analysis-project#/content/sentim
  model.fit(X_train,y_train)
[[1518  493]
 [ 640 1349]]
0.71675
/content/sentiment-analysis-project#
[0] 0:python3*                                  "297a110bf8f4" 00:21 14-Nov-21
```

Following are the screenshots of confusion matrix and accuracy obtained for dataset-1 and bag of words :

1. bag of words, 1-gram, logistic regression.

```
# Installing RAPI
# The <release> c
# The <packages>
!python rapidsai-
import os
os.environ['NUMBA
os.environ['NUMBA
os.environ['CONDA

Installing RAPIDS
Starting the RAPI
Collecting packag
Solving environme
failed with initi
Solving environme
failed with repod
Collecting packag
Solving environme
done

tcmalloc: large alloc 1216774144 bytes == 0x5639b9b980
1bb780354f 0x7f1bb7853b58 0x7f1bb7857b17 0x7f1bb78f620
d87240 0x5639b4dfb627 0x5639b4d88afa 0x5639b4dfad00 0x
da 0x5639b4df7737 0x5639b4df59ee 0x5639b4d88bda 0x5639
x5639b4df56f3 0x5639b4ebf4c2 0x5639b4ebf83d 0x5639b4eb
9b4e96e0c 0x7f1bb938dbf7 0x5639b4e96cea
/usr/local/lib/python3.7/dist-packages/sklearn/linear_
onvergenceWarning: lbfgs failed to converge (status=1)
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale
    https://scikit-learn.org/stable/modules/preprocess
Please also refer to the documentation for alternative
    https://scikit-learn.org/stable/modules/linear_mod
on
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
[[1008  414]
 [ 357 1021]]
0.7246428571428571
/content/sentiment-analysis-project#
```

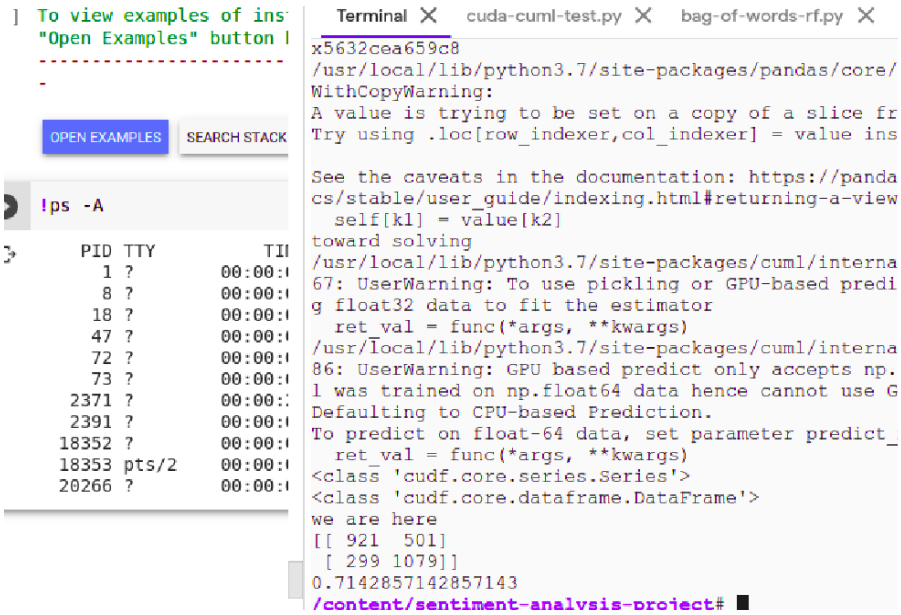
2. bag of words, 1-gram, SVM.

```
import os
os._exit(00)

Updating your Colab environ
Hit:1 https://cloud.r-proje
Ign:2 https://developer.dow
Hit:3 http://security.ubunt
Ign:4 https://developer.dow
Hit:5 https://developer.dow
Hit:6 https://developer.dow
Hit:7 http://ppa.launchpad.
Hit:8 http://archive.ubuntu

/content/sentiment-analysis-project# python bag-of-words-svm.py
tcmalloc: large alloc 1216774144 bytes == 0x557b14ca6000 @  0x7f
a001 0x7f154a15ae16 0x7f154a15aee6 0x7f154a19d585 0x7f154a1a1a97
54a233d65 0x557b108f8300 0x557b108f9ad8 0x557b109211d9 0x557b108
0x557b108f99c8 0x557b1092474a 0x557b10867af2 0x557b10896030 0x55
99c8 0x557b109211d9 0x557b10867af2 0x557b10896030 0x557b108f99c8
b1092474a 0x557b10867af2 0x557b10868d09 0x557b109438ab 0x557b109
0x557b109b1fd7 0x557b109b21ac 0x557b109b2709 0x557b109b285c 0x7f
9bf7 0x557b1092f901
[[ 967  455]
 [ 309 1069]]
0.7271428571428571
/content/sentiment-analysis-project#
```

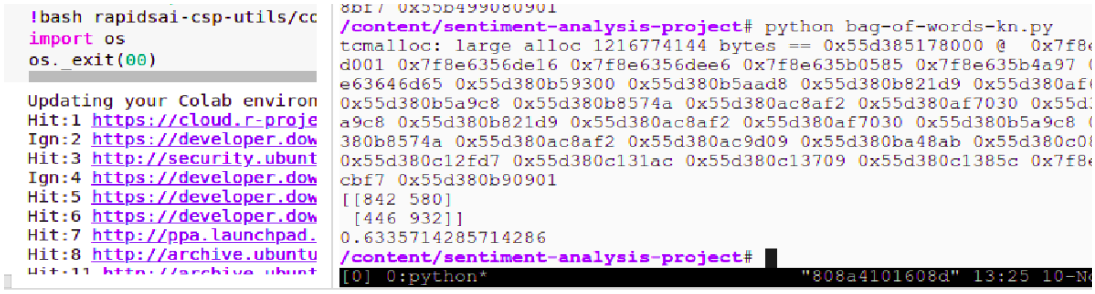
3. bag of words, 1-gram, naive bayes.

```
/content/sentiment-analysis-project/sentiment-analysis-of-dataset-1# python bag-of-words-nb.py
tcmalloc: large alloc 1216774144 bytes == 0x55f7df304000 @  0x7fcfbe6c4001 0x7fcfbb24854f 0x7fcf
8 0x7fcfbb29cb17 0x7fcfbb33b203 0x55f7dbac4544 0x55f7dbac4240 0x55f7dbb38627 0x55f7dbac5afa 0x55
d00 0x55f7dbb329ee 0x55f7dbac5bda 0x55f7dbb34737 0x55f7dbb329ee 0x55f7dbac5bda 0x55f7dbb37d00 0x
329ee 0x55f7dbb326f3 0x55f7dbbfc4c2 0x55f7dbbfc83d 0x55f7dbbfc6e6 0x55f7dbbd4163 0x55f7dbbd3e0c
d4acbf7 0x55f7dbbd3cea
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function
ture_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Ple
 get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
[[1065  357]
 [ 449  929]]
0.7121428571428572
```

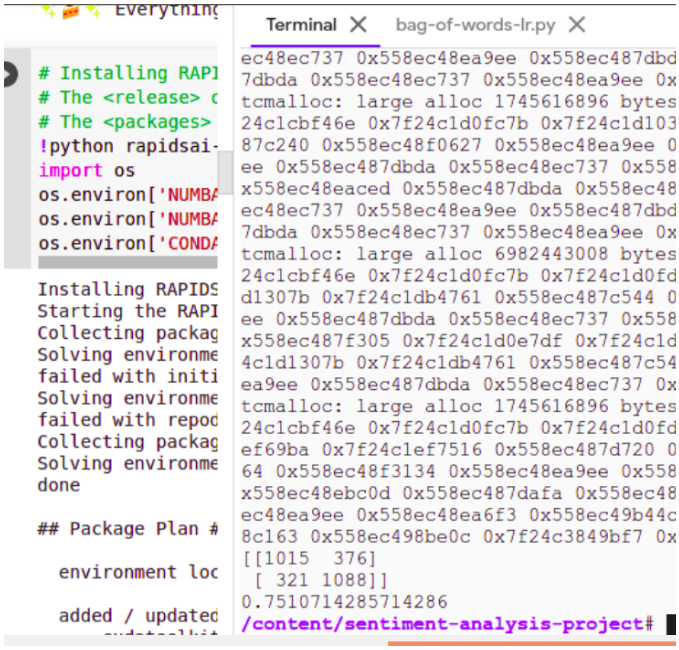
4. bag of words, 1-gram, random forest.



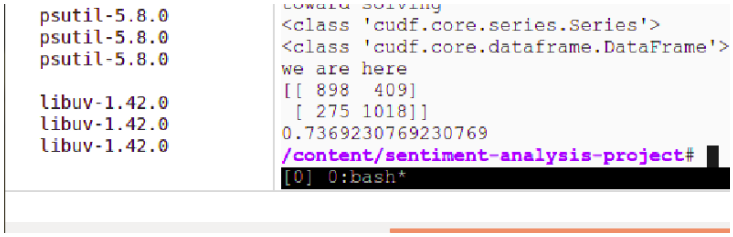
5. bag of words, 1-gram, KNN.



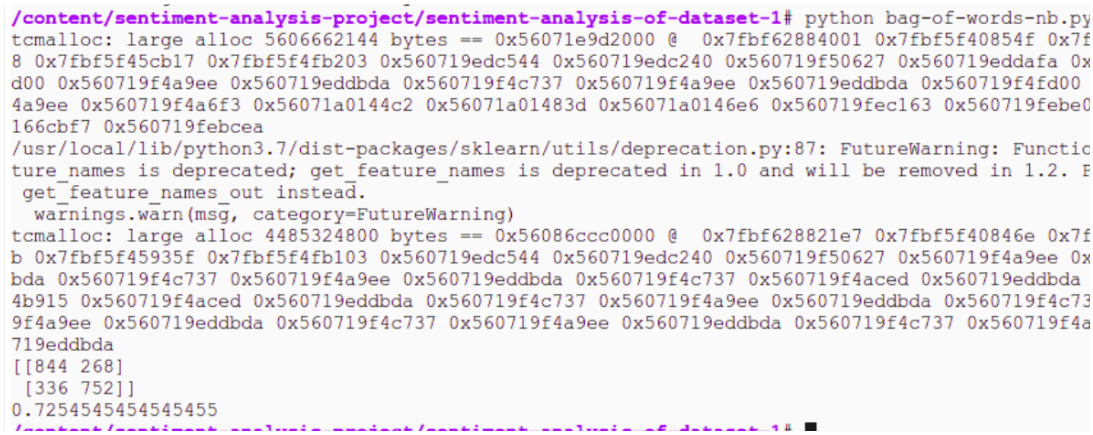
6. bag of words, 2-grams, logistic regression.



7. bag of words, 2-grams, SVM.

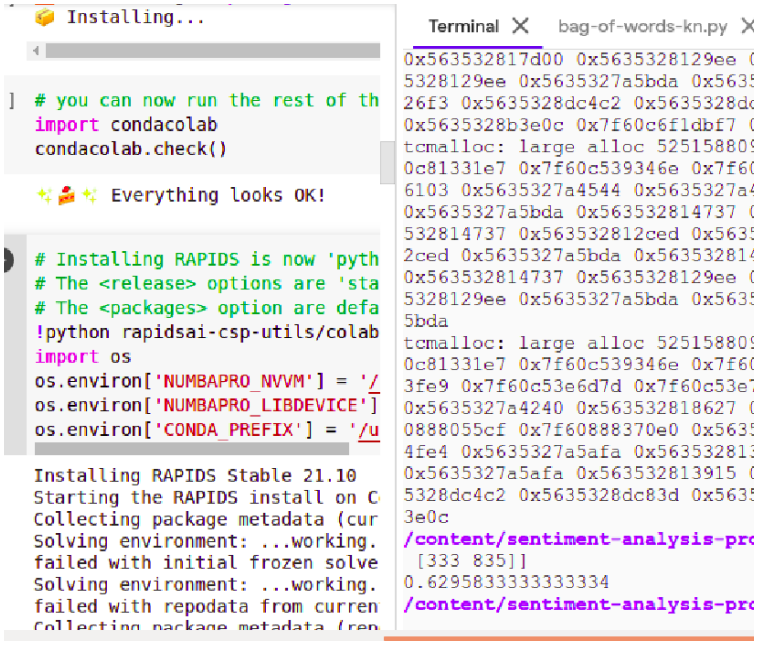


8. bag of words, 2-grams, naive bayes.

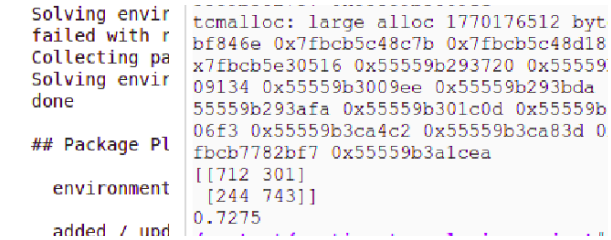


9. bag of words, 2-grams, random forest.

10. bag of words, 2-grams, KNN.



11. bag of words, 3-grams, logistic regression.

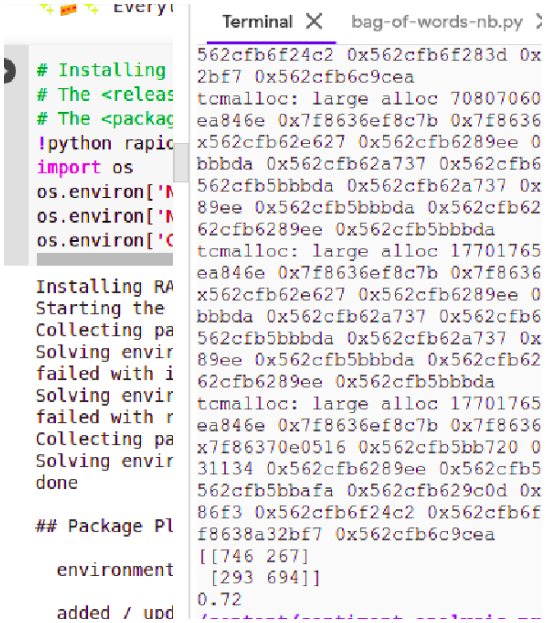


12. bag of words, 3-grams, SVM.

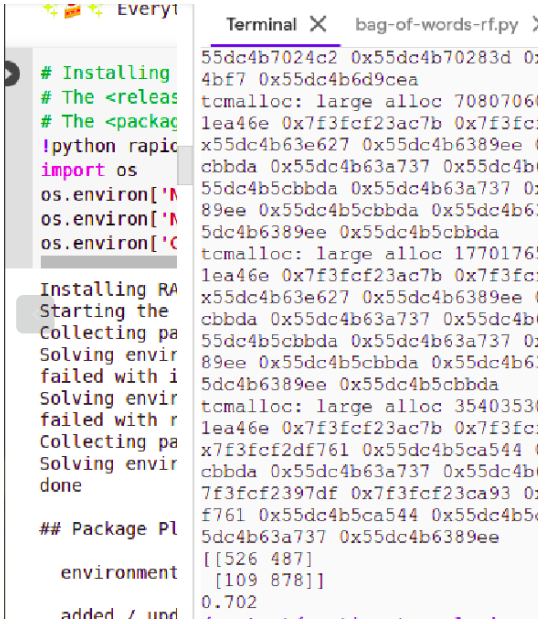
```
backports-1.0          | 4 KB
backports-1.0          | 4 KB

jsonschema-4.2.1       | 118 K
jsonschema-4.2.1       | 118 K
jsonschema-4.2.1       | 118 K

poppler-21.03.0        | 15.9
```

```
<class 'cudf.core.series.Series'>
<class 'cudf.core.dataframe.DataFrame'
we are here
[[595 322]
 [178 705]]
0.7222222222222222
/content/sentiment-analysis-project#
```

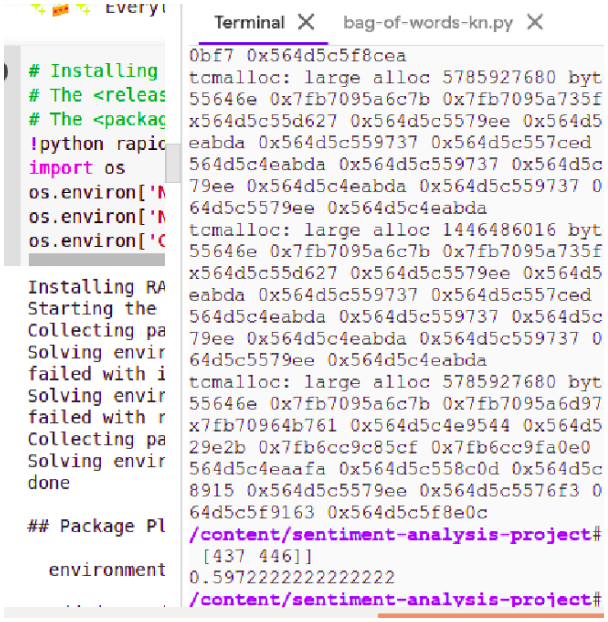
13. bag of words, 3-grams, naive bayes.



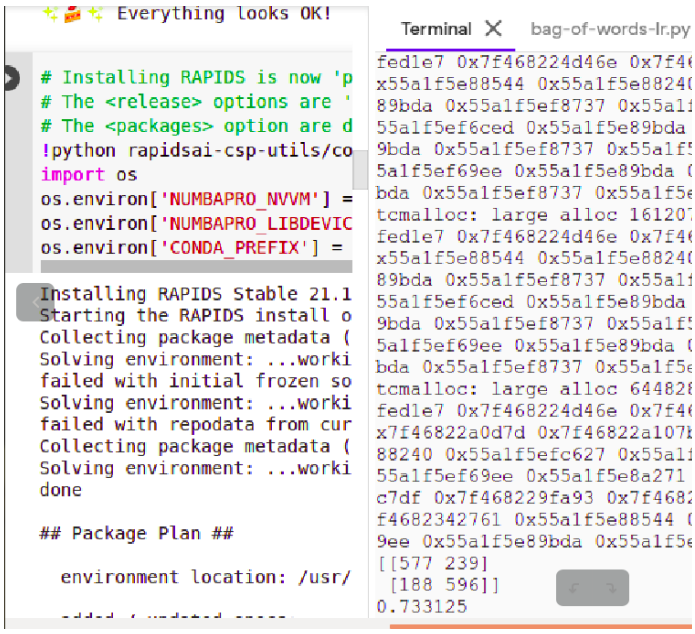
14. bag of words, 3-grams, random forest.



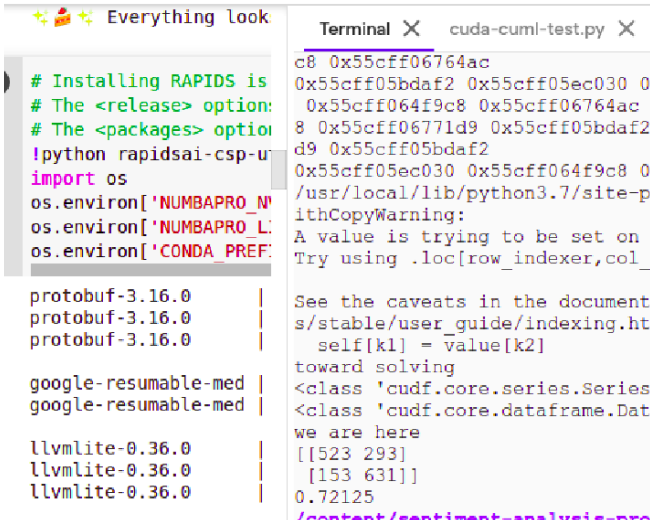
15. bag of words, 3-grams, KNN.



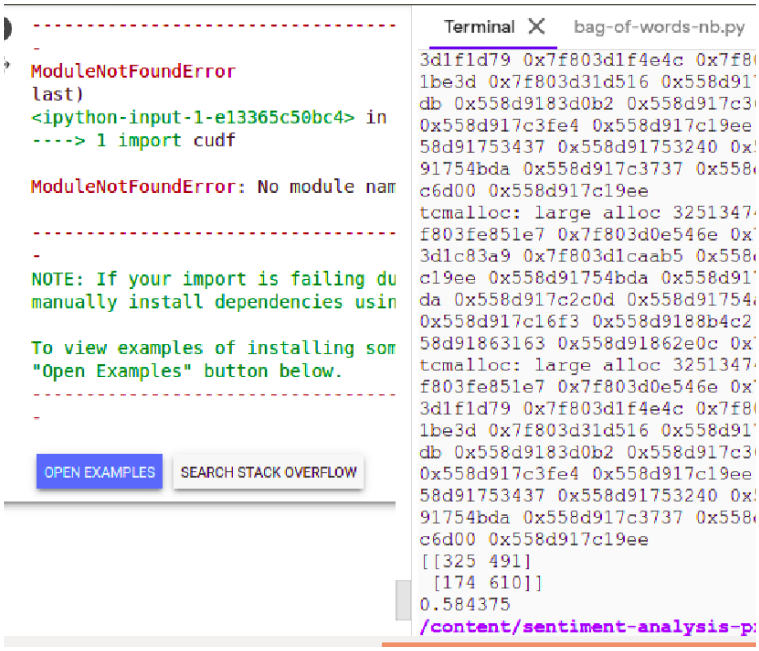
16. bag of words, 4-grams, logistic regression.



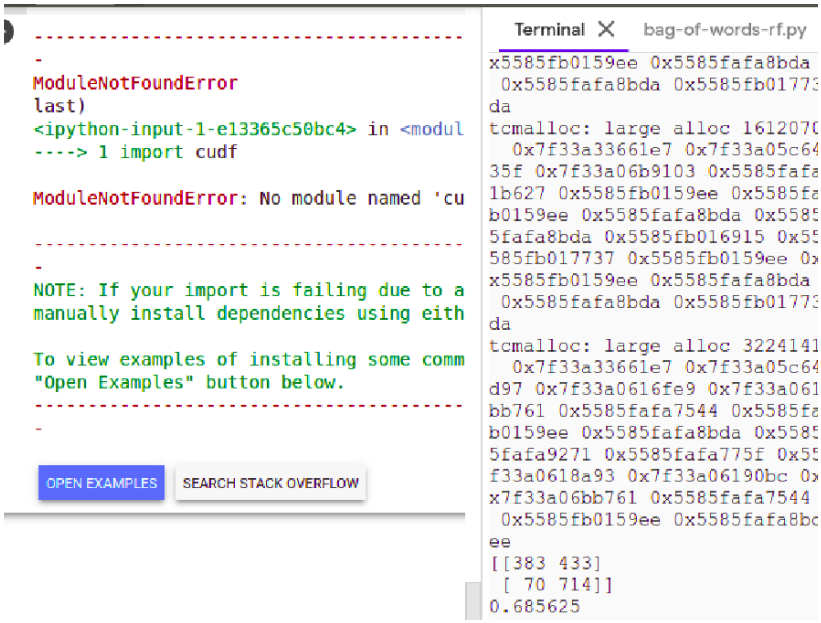
17. bag of words, 4-grams, SVM.



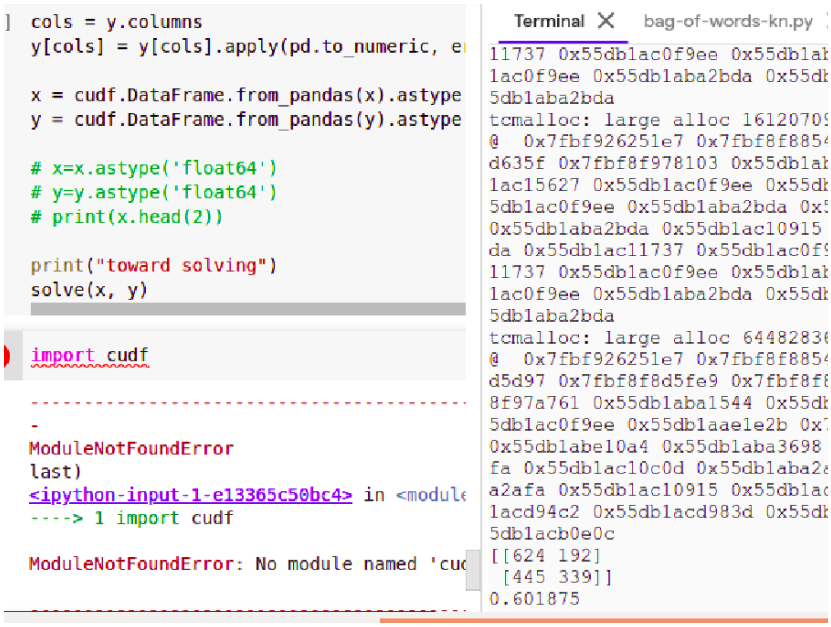
18. bag of words, 4-grams, naive bayes.



19. bag of words, 4-grams, random forest.

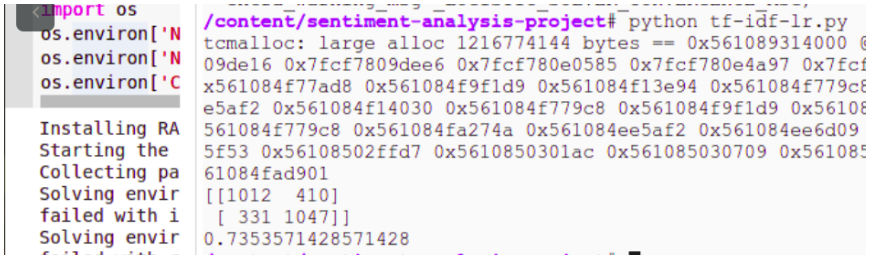


20. bag of words, 4-grams, KNN.

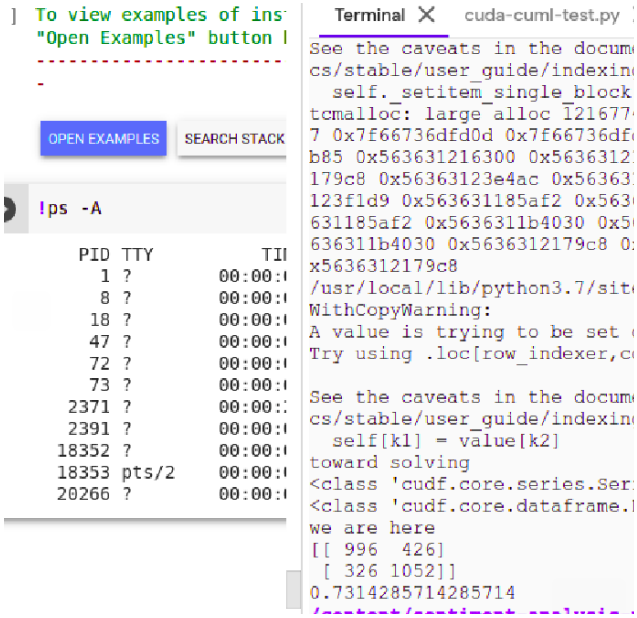


Following are the screenshots of confusion matrix and accuracy obtained for dataset-1 and TF-IDF :

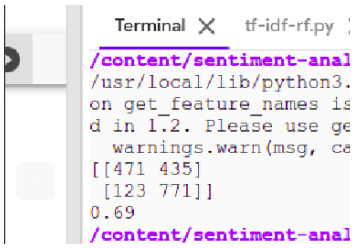
1. TF-IDF, 1-gram, logistic regression.



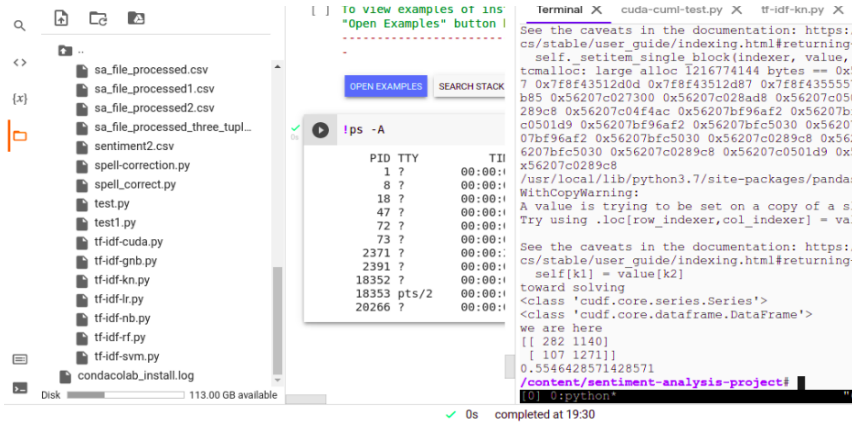
2. TF-IDF, 1-gram, SVM.



3. TF-IDF, 1-gram, random forest.



4. TF-IDF, 1-gram, KNN.



5. TF-IDF, 2-grams, logistic regression.

```
# you can now
import condac
condacolab.ch

Everyt

# Installing
# The <releas
# The <packag
!python rapid
import os
os.environ['N
os.environ['N
os.environ['C

Installing RA
Starting the
Collecting pa
Solving envir
failed with i
Solving envir
```

```
/content/sentiment-analysis-project# python tf-idf-lr.py
tcmalloc: large alloc 8728051712 bytes == 0x55da88078000
2d6e16 0x7ff04c2d6ee6 0x7ff04c319585 0x7ff04c31da97 0x7ff
x55da84492ad8 0x55da844ba1d9 0x55da8442ee94 0x55da844929c
00af2 0x55da8442f030 0x55da844929c8 0x55da844ba1d9 0x55da
55da844929c8 0x55da844bd74a 0x55da84400af2 0x55da84401d09
0f53 0x55da8454afd7 0x55da8454b1ac 0x55da8454b709 0x55da8
5da844c8901
tcmalloc: large alloc 6982443008 bytes == 0x55dc90c4a000
2d6d0d 0x7ff04c2d6d87 0x7ff04c319574 0x7ff04c31a0ff 0x7ff
x55da84492ad8 0x55da844ba1d9 0x55da8442ee94 0x55da844929c
00af2 0x55da8442f030 0x55da844929c8 0x55da844ba1d9 0x55da
55da844929c8 0x55da844ba1d9 0x55da84400af2 0x55da8442f030
94ac 0x55da84400af2 0x55da8442f030 0x55da844929c8 0x55da8
5da8442f030 0x55da844929c8
tcmalloc: large alloc 1745616896 bytes == 0x55de30f44000
2d6d0d 0x7ff04c2d6d87 0x7ff04c319574 0x7ff04c31a0ff 0x7ff
x55da84492ad8 0x55da844ba1d9 0x55da8442ee94 0x55da844929c
00af2 0x55da8442f030 0x55da844929c8 0x55da844ba1d9 0x55da
55da844929c8 0x55da844ba1d9 0x55da84400af2 0x55da8442f030
94ac 0x55da84400af2 0x55da8442f030 0x55da844929c8 0x55da8
5da8442f030 0x55da844929c8
[[1002  389]
 [ 309 1100]]
0.7507142857142857
```

6. TF-IDF, 2-grams, SVM.

```
Restarting

# you can now
import condac
condacolab.ch

Everyt

# Installing
# The <releas
# The <packag
!python rapid
import os
os.environ['N
os.environ['N
os.environ['C

Installing RA
Starting the
Collecting pa
Solving envir
failed with i
Solving envir
```

```
Terminal  cuda-cuml-test.py  tf-idf-svm.py
aaaf2 0x55c6970d9030 0x55c69713c9c8 0x55c69716
55c69713c9c8 0x55c6971641d9 0x55c6970aaaf2 0x5
41d9 0x55c6970aaaf2 0x55c6970d9030 0x55c69713c
5c6970d9030 0x55c69713c9c8
tcmalloc: large alloc 7610204160 bytes == 0x55
b73d0d 0x7f388db73d87 0x7f388dbb6574 0x7f388db
x55c69713cad8 0x55c6971641d9 0x55c6970d8e94 0x
aaaf2 0x55c6970d9030 0x55c69713c9c8 0x55c69716
55c6971634ac 0x55c6970aaaf2 0x55c6970d9030 0x5
aaf2 0x55c6970d9030 0x55c69713c9c8 0x55c697164
5c69713c9c8 0x55c6971641d9
/usr/local/lib/python3.7/site-packages/pandas/
rning:
A value is trying to be set on a copy of a sli
Try using .loc[row_indexer,col_indexer] = valu

See the caveats in the documentation: https://
user_guide/indexing.html#returning-a-view-vers
  self[k1] = value[k2]
toward solving
<class 'cudf.core.series.Series'>
<class 'cudf.core.dataframe.DataFrame'>
we are here
[[ 917  390]
 [ 287 1006]]
0.7396153846153846
```

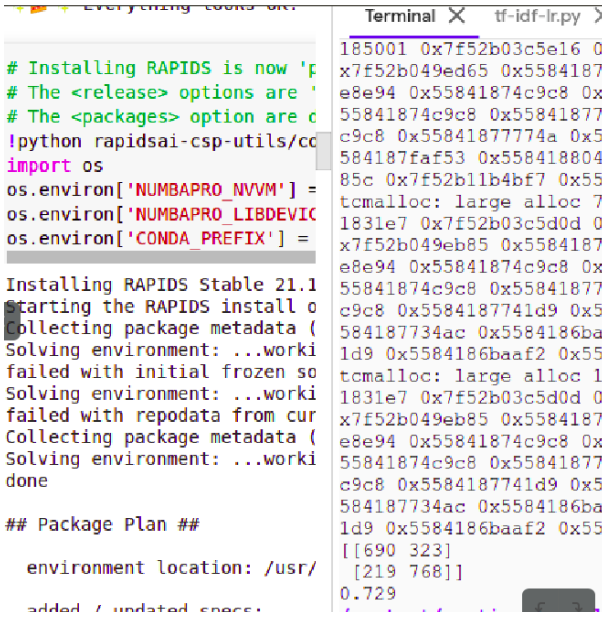
7. TF-IDF, 2-grams, random forest.

```
/content/sentiment-analysis-project/sentiment-analysis-of-dataset-1# python tf-idf-rf.py
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Func
et_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed
. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
[[476 430]
 [135 759]]
0.6861111111111111
```

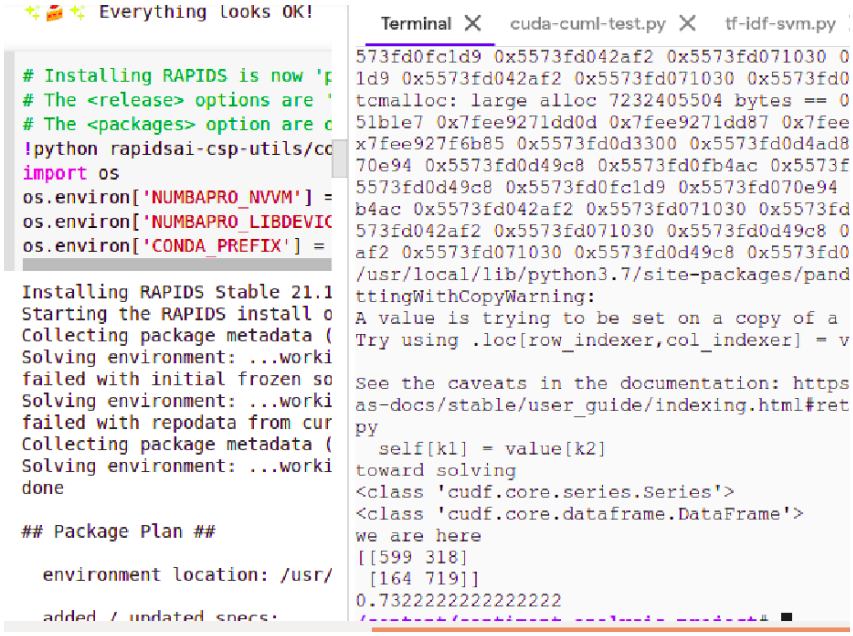
8. TF-IDF, 2-grams, KNN.

```
/content/sentiment-analysis-project/sentiment-analysis-of-dataset-1# python tf-idf-kn.py
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Fu
eature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in
 use get_feature_names_out instead./sentiment-analysis-of-dataset-1# python tf-idf-kn.py
  warnings.warn(msg, category=FutureWarning)rn/utils/deprecation.py:87: FutureWarning: Fu
[[160 746]entiment-analysis-project/sentiment-analysis-of-dataset-1# d will be removed in
 [ 21 873]]
0.5738888888888889
```

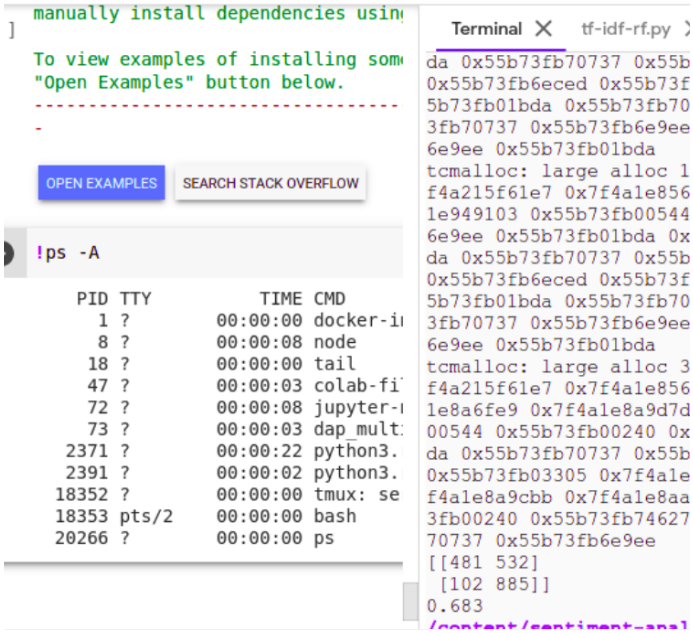
9. TF-IDF, 3-grams, logistic regression.



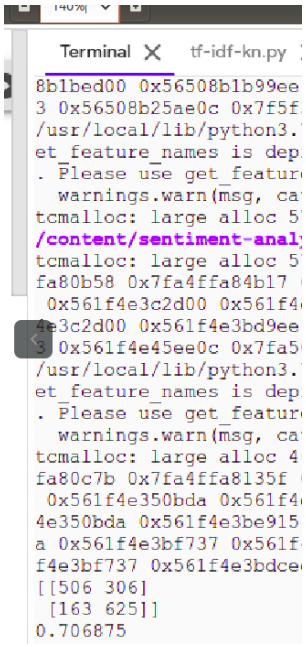
10. TF-IDF, 3-grams, SVM.



11. TF-IDF, 3-grams, random forest.



12. TF-IDF, 3-grams, KNN.



13. TF-IDF, 4-grams, logistic regression.

```
/content/sentiment-analysis-project/sentiment-analysis-of-dataset-1# python tf-idf-lr.py
tcmalloc: large alloc 6217400320 bytes == 0x5557974e0000 @  0x7f04c3740001 0x7f04c02c454f
0314b58 0x7f04c0318b17 0x7f04c03b7203 0x5557938cd544 0x5557938cd240 0x555793941627 0x5557
 0x555793940d00 0x55579393b9ee 0x5557938cebda 0x55579393d737 0x55579393b9ee 0x5557938cebd
93940d00 0x55579393b9ee 0x55579393b6f3 0x555793a054c2 0x555793a0583d 0x555793a056e6 0x555
 0x5557939dce0c 0x7f04c2528bf7 0x5557939dccea
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Fu
et_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be remove
. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
tcmalloc: large alloc 4973920256 bytes == 0x55590a040000 @  0x7f04c373e1e7 0x7f04c02c446e
0314c7b 0x7f04c031535f 0x7f04c03b7103 0x5557938cd544 0x5557938cd240 0x555793941627 0x5557
 0x5557938cebda 0x55579393d737 0x55579393b9ee 0x5557938cebda 0x55579393d737 0x55579393bce
938cebda 0x55579393c915 0x55579393bced 0x5557938cebda 0x55579393d737 0x55579393b9ee 0x555
a 0x55579393d737 0x55579393b9ee 0x5557938cebda 0x55579393d737 0x55579393b9ee 0x5557938ceb
79393d737 0x55579393bced 0x5557938cebda
[[479 219]
 [161 541]]
0.7285714285714285
```

14. TF-IDF, 4-grams, SVM.

```
] Done in 0:00:23
  Restarting kernel...

] # you can now run the rest of
  import condacolab
  condacolab.check()

  ✨🍰✨ Everything looks OK!

  # Installing RAPIDS is now 'p
  # The <release> options are '
  # The <packages> option are d
  !python rapidsai-csp-utils/co
  import os
  os.environ['NUMBAPRO_NVVM'] =
  os.environ['NUMBAPRO_LIBDEVIC
  os.environ['CONDA_PREFIX'] =

  Installing RAPIDS Stable 21.1
  Starting the RAPIDS install o
  Collecting package metadata (
  Solving environment: ...worki
  failed with initial frozen so
  Solving environment: ...worki
```

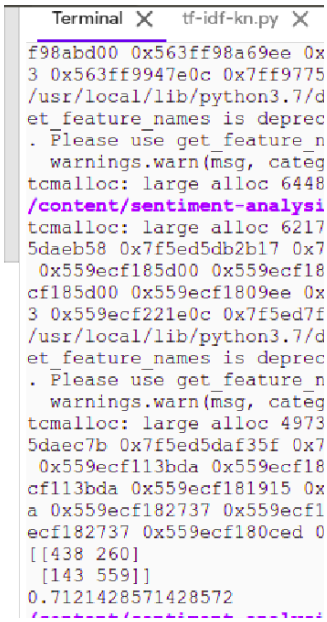
```
Terminal ✕  cuda-cuml-test.py
65296c761d9 0x565296bbcaf2 0x
1d9 0x565296bbcaf2 0x565296be
tcmalloc: large alloc 806035
ddf1e7 0x7f3d3f021d0d 0x7f3d
x7f3d3f0fab85 0x565296c4d300
eae94 0x565296c4e9c8 0x56529
565296c4e9c8 0x565296c761d9
54ac 0x565296bbcaf2 0x56529
65296bbcaf2 0x565296beb030 0x
af2 0x565296beb030 0x565296c
/usr/local/lib/python3.7/site
ttingWithCopyWarning:
A value is trying to be set
Try using .loc[row_indexer,co

See the caveats in the docume
as-docs/stable/user_guide/ind
py
  self[k1] = value[k2]
toward solving
<class 'cudf.core.series.Seri
<class 'cudf.core.dataframe.D
we are here
[[544 272]
 [169 615]]
0.724375
```

15. TF-IDF, 4-grams, random forest.

```
] Done in 0:
  Restarting

2] # you can now
  import condac
  condacolab.ch

  ✨🍰✨ Everyt

  # Installing
  # The <releas
  # The <packag
  !python rapid
  import os
  os.environ['N
  os.environ['N
  os.environ['C

  Installing RA
  Starting the
  Collecting pa
  Solving envir
```

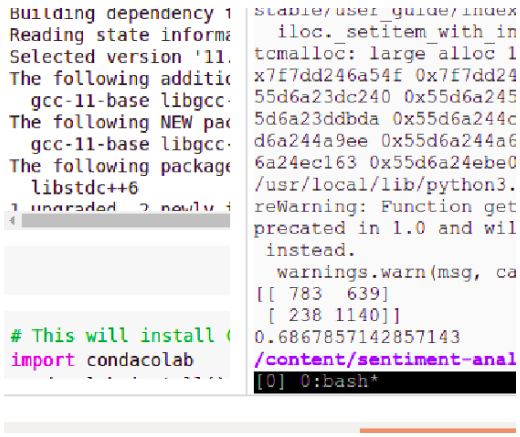
```
Terminal ✕  cuda-cuml-test.py ✕
/usr/local/lib/python3.7/site-pa
rning: To use pickling or GPU-ba
t the estimator
  ret_val = func(*args, **kwargs
/usr/local/lib/python3.7/site-pa
rning: GPU based predict only ac
float64 data hence cannot use GP
Defaulting to CPU-based Predicti
To predict on float-64 data, set
  ret_val = func(*args, **kwargs
tcmalloc: large alloc 1612070912
62f136 0x7f2d23610bea 0x7f2d6ab9
x55ddcd8b63a8 0x55ddcd91bf93 0x7
6eaa7 0x55ddcd8b4af2 0x55ddcd8e3
55ddcd9469c8 0x55ddcd96d4ac 0x55
4f53 0x55ddcd9fefd7 0x55ddcd9ff1
5ddcd97c901
<class 'cudf.core.series.Series'
<class 'cudf.core.dataframe.Data
we are here
/content/sentiment-analysis-proj
 [ 73 711]]
0.6775
/content/sentiment-analysis-proj
```

16. TF-IDF, 4-grams, KNN.

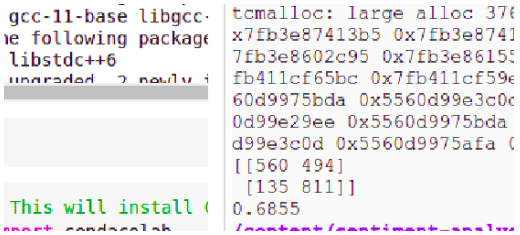


Following are the screenshots of confusion matrix and accuracy obtained for dataset-1 and XGBoost :

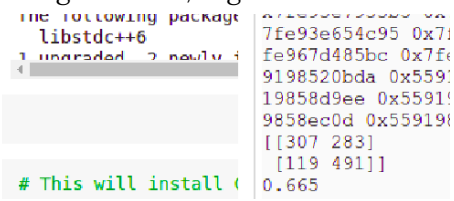
1. bag of words, 1-gram.



2. bag of words, 2-grams.



3. bag of words, 4-grams.

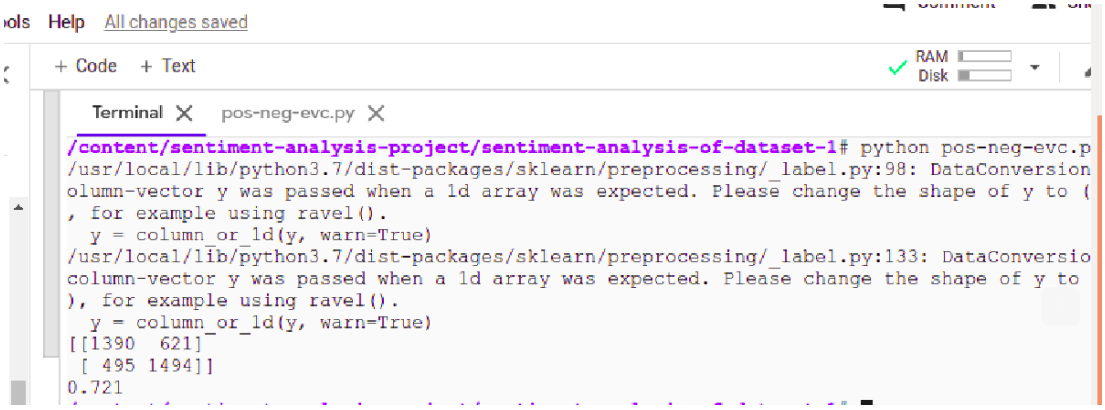


Hard voting was used for ensemble learning. I chose algorithms giving the highest accuracy and then final sentiment was based on the highest votes. Following are the screenshots of confusion matrix and accuracy obtained for dataset-1 and ensemble learning :

1. bag of words, 1-gram, classifiers used for voting were : logistic regression, naive bayes, random forest and svm.

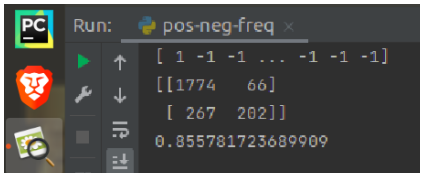
```
    https://scikit-learn.org/stable/modules/preprocessing.html
/content/sentiment-analysis-project/sentiment-analysis-of-dataset-1# python bag-of-words-evc.py
tcmalloc: large alloc 1216774144 bytes == 0x5627a05dc000 @  0x7fe10cb76001 0x7fe1096fa54f 0x7fe1
8 0x7fe10974eb17 0x7fe1097ed203 0x56279cc51544 0x56279cc51240 0x56279ccc5627 0x56279cc52afa 0x56
d00 0x56279ccbf9ee 0x56279cc52bda 0x56279ccc1737 0x56279ccbf9ee 0x56279cc52bda 0x56279ccc4d00 0x
bf9ee 0x56279ccbf6f3 0x56279cd894c2 0x56279cd8983d 0x56279cd896e6 0x56279cd61163 0x56279cd60e0c
b95ebf7 0x56279cd60cea
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function
ture_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Ple
 get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning
 failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
[[1067  355]
 [ 398  980]]
0.7310714285714286
/content/sentiment-analysis-project/sentiment-analysis-of-dataset-1#
```

2. positive-negative frequency, 4-grams, classifiers used for voting were : logistic regression, naive bayes and random forest.

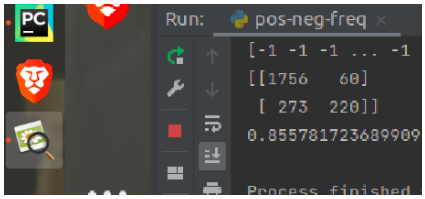


Following are the screenshots of confusion matrix and accuracy obtained for dataset-2 and positive-negative frequency :

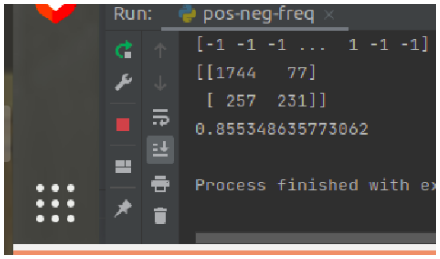
1. positive-negative frequency, 1-gram, logistic regression.



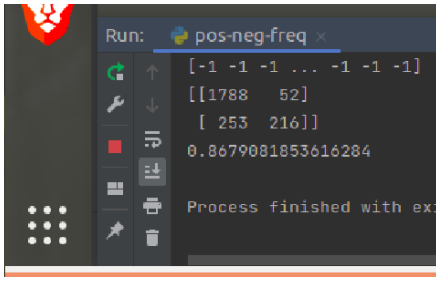
2. positive-negative frequency, 2-grams, logistic regression.



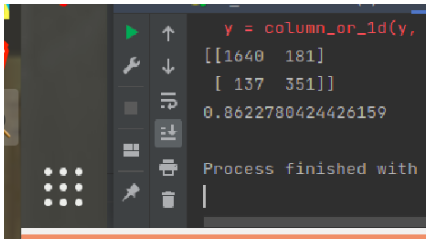
3. positive-negative frequency, 3-grams, logistic regression.



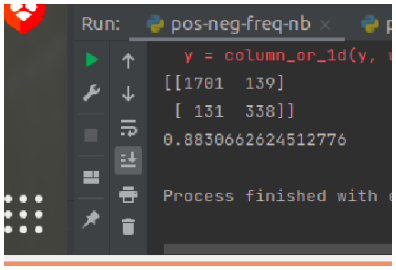
4. positive-negative frequency, 4-grams, logistic regression.



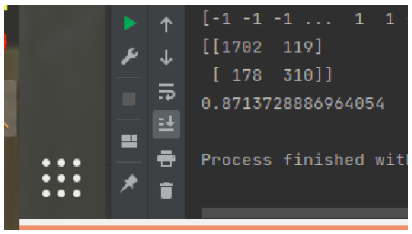
5. positive-negative frequency, 3-grams, naive bayes.



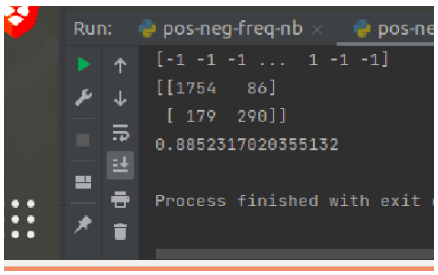
6. positive-negative frequency, 4-grams, naive bayes.



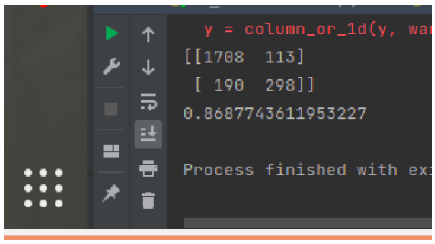
7. positive-negative frequency, 3-grams, random forest.



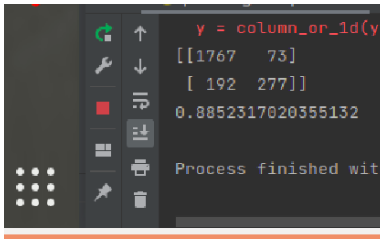
8. positive-negative frequency, 4-grams, random forest.



9. positive-negative frequency, 3-grams, SVM.

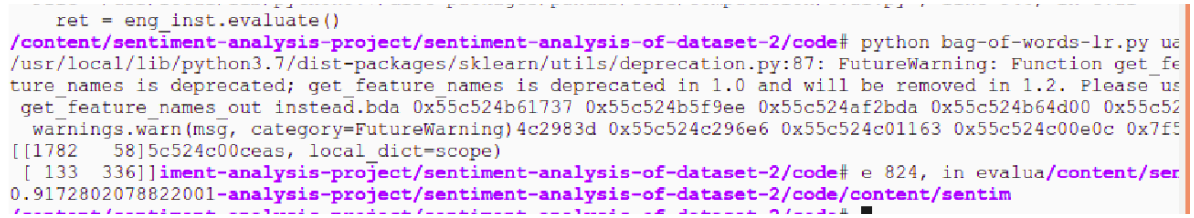


10. positive-negative frequency, 4-grams, SVM.

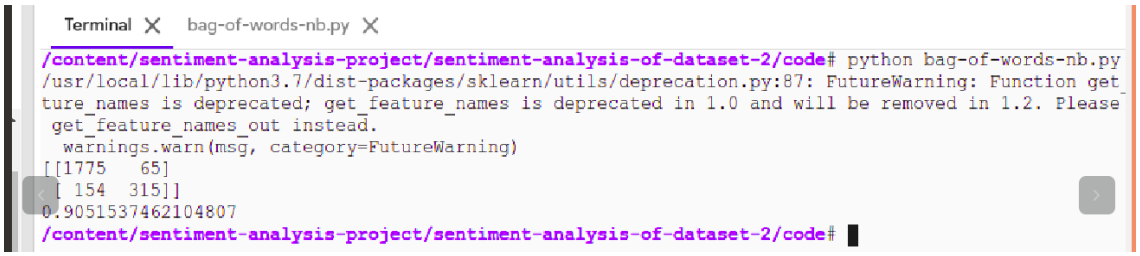


Following are the screenshots of confusion matrix and accuracy obtained for dataset-2 and bag of words :

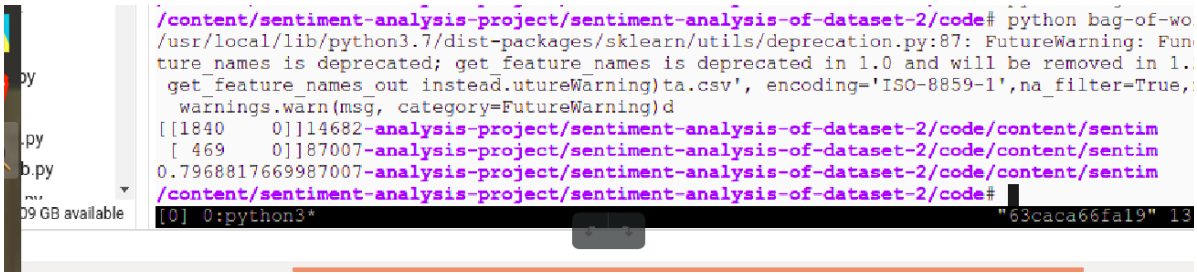
1. bag of words, 1-gram, logistic regression.



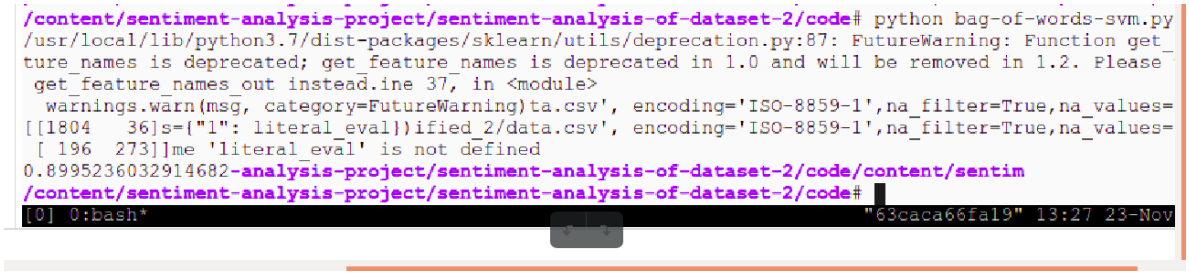
2. bag of words, 1-gram, multinomial naive bayes.



3. bag of words, 1-gram, random forest.

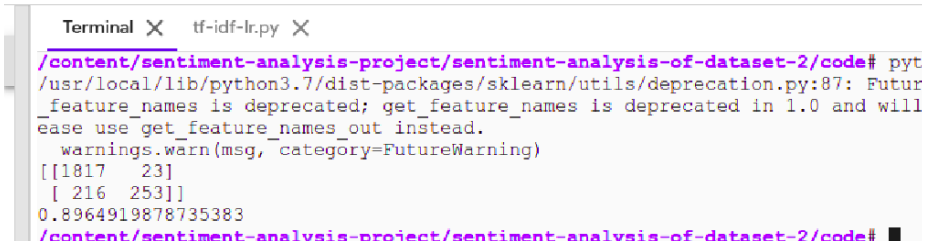


4. bag of words, 1-gram, SVM.



Following are the screenshots of confusion matrix and accuracy obtained for dataset-2 and TF-IDF :

1. TF-IDF, 1-gram, logistic regression.



2. TF-IDF, 1-gram, random forest.

```
/content/sentiment-analysis-project/sentiment-analysis-of-dataset-2/code# python tf-idf-rf.py
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function
_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2
ease use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
[[1840    0]timent-analysis-project/sentiment-analysis-of-dataset-2/code# python tf-idf-nb.py
 [ 469    0]]iment-analysis-project/sentiment-analysis-of-dataset-2/code# tureWarning: Fun/cont
0.7968817669987007
/content/sentiment-analysis-project/sentiment-analysis-of-dataset-2/code#
```

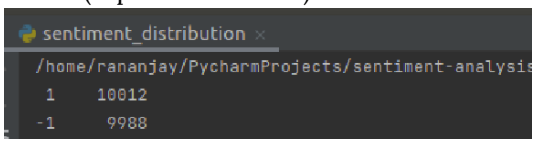
3. TF-IDF, 1-gram, SVM.

```
  warnings.warn(msg, category=FutureWarning)t-analysis-of-dataset-2/code# python tf-idf-nb.py
[[1812   28]]iment-analysis-project/sentiment-analysis-of-dataset-2/code# tureWarning: Fun/cont
 [ 204  265]]87007
0.8995236032914682-analysis-project/sentiment-analysis-of-dataset-2/code/content/sentim
/content/sentiment-analysis-project/sentiment-analysis-of-dataset-2/code#
[0] 0:bash*                                                    "63caca66fa19" 14:58 23-No
```

Data Collection

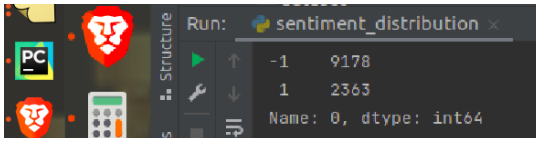
- Source of dataset-1 : link.
- Source of dataset-2 : link.

Following is the screenshot of distribution of positive(represented as 1) and negative sentiment tweets(represented as -1) in the dataset-1 :



We can see that both positive and negative sentiment tweets are around 50% of the whole dataset. It's expected since tweets are not from any specific domain.

Following is the screenshot of distribution of positive(represented as 1) and negative sentiment tweets(represented as -1) in the dataset-2:



Percentage of negative and positive sentiment tweets is 79.525 and 20.475, respectively. Since tweets are related to airlines it is expected that most of the tweets are from customers facing some issues and thus there are more negative sentiment tweets.

Following is the screenshot of distribution of positive(represented as 1), negative(represented as -1) and neutral sentiment(represented as 0) tweets in the airline dataset (the complete dataset-2 which doesn't exclude neutral sentiments):

Percentage of negative, neutral and positive tweets are 62.69, 21.16 and 16.14, respectively.

Evaluation of Solution

Formula for some of the metrics which will be used to evaluate the results :

$$Accuracy = \frac{correct\ predictions\ count}{total\ number\ of\ data}$$

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

$$F_1\ score = \frac{2*precision*recall}{precision+recall}$$

Note that F_1 and F score refers to the same formula and can be used interchangeably.

For the naive solution (runned on dataset-1) maximum accuracy of 63.6 percent was obtained using a support vector machine. For a very naive solution it's a good result since one of the papers I went through got accuracy around 75 percent after doing all pre-processing, feature extraction.
It's interesting to observe however that this naive solution works well in terms of false negatives. In the worst case it's 14.55 percent and in the best case it's 6 percent. So there are very few cases where it says that a tweet has negative sentiment but it's positive in reality.

After I implemented all the techniques that I thought on paper, for dataset-1 maximum accuracy of 0.751 was obtained with bag of words (2-grams, logistic regression) and for dataset-2 maximum accuracy of 0.917 was obtained with bag of words (1-gram, logistic regression). It's expected that accuracy for dataset-2 would be higher because all of it's tweets are related to airlines and thus training would produce better classifiers compared to dataset-1 where tweets don't belong to any specific domain.

For the preprocessing part, following example was runned to illustrate here :
tweet="RT @dynamo #amazing wowwww Thanksss!!! for the trip, It was so gooood 👍"
After preprocessing, the list of word obtained was :
['amaze', 'wow', 'thanks', 'thanks', 'trip', 'good', 'thumb', '_amaze_wow', '_wow_thanks', '_thanks_thanks', '_thanks_trip', '_trip_good', '_good_thumb', '_amaze_wow_thanks', '_wow_thanks_thanks', '_thanks_thanks_trip', '_thanks_trip_good', '_trip_good_thumb', '_amaze_wow_thanks_thanks', '_wow_thanks_thanks_trip', '_thanks_thanks_trip_good', '_thanks_trip_good_thumb']

In above example we can see that "#amazing" has been converted to "amaze" after removing hashtag and doing lemmatization, "wowwww" has been spell corrected to "wow", "thanks" is repeated twice since in original tweet it had exclamation mark at the end, n-grams till size 4 has been created (example : '_wow_thanks_thanks_trip'), emoji has been replaced by its corresponding word.

Note that the sizes of dataset I choose while running any code was maximum possible for resources available on google colab. Dataset size for different code varied from 7000 (tf-idf, 4-gram, logistic regression) to 20000 (positive-negative frequency, 1-gram, logistic regression).

It was observed that accuracy increased with more tuple sizes (n-grams). Let's see some of the results obtained from the same dataset sizes and different n-grams for positive-negative frequency for dataset-1.

Table-1

	Model	Accuracy
1.1	training size=16000, test size=4000, 1-gram, SVM	0.6925
1.2	training size=16000, test size=4000, 2-grams, SVM	0.6942
1.3	training size=16000, test size=4000, 3-grams, SVM	0.6942
1.4	training size=16000, test size=4000, 4-grams, SVM	0.7072

Table-2

	Model	Accuracy
1.1	training size=16000, test size=4000, 1-gram, logistic regression	0.7055
1.2	training size=16000, test size=4000, 4-grams, logistic regression	0.7182

Some of the results obtained from the bag of words model for dataset-1.

Table-3

	Model	Accuracy
1.1	training size=11200, test size=2800, 1-gram, logistic regression	0.7246
1.2	training size=11200, test size=2800, 2-grams, logistic regression	0.7510

Table-4

	Model	Accuracy
1.1	training size=11200, test size=2800, 1-gram, logistic regression	0.7271
1.2	training size=11200, test size=2800, 2-grams, logistic regression	0.7510

It's not correct to compare different n-grams with very different dataset sizes. While running the algorithms, memory limit was exceeded in google colab for 4-grams but it ran for 1-gram for the same large input dataset size.

Some of the results obtained from the positive-negative frequency for dataset-2.

Table-5

	Model	Accuracy
1.1	training size=9232, test size=2300, 3-grams, naive bayes	0.8622
1.2	training size=9232, test size=2309, 4-grams, naive bayes	0.8830

From the above tables we can say that accuracy increased around 2% in some cases when the number of tuples (n-grams) were increased.

Following table represents F_1 scores for the cases where best accuracy was obtained for each dataset :

Table-6

	Model	F_1
dataset-1	training size=11200, test size = 2800, bag of words, 2-grams, logistic regression	0.7573
dataset-2	training size=9232, test size=2309, bag of words, 1-gram, logistic regression	0.7786

Following table has results of ensemble learning for bag of words and 1-gram for dataset-1:

Table-7

	Model	Accuracy
1.1	training size=11200, test size = 2800, hard voting (logistic regression, naive bayes, random forest, SVM)	0.7310
1.2	XGBoost	0.6867

Let's compare the result of hard voting with individual algorithms:

Table-8

	Model	Accuracy
1.1	training size=11200, test size = 2800, logistic regression	0.7246
1.2	training size=11200, test size = 2800, naive bayes	0.7121
1.3	training size=11200, test size = 2800, random forest	0.7142
1.4	training size=11200, test size = 2800, SVM	0.7271

We can see that the maximum accuracy in the above table is 0.7271 and by using the hard voting ensemble learning technique it increased to 0.7310.

Also I choose only those algorithms for hard voting whose accuracy are comparable. This is because if we choose lets say 25 models which can vote and suppose 10 of them have accuracy 0.72 and 15 of them have accuracy 0.63, chances are that there won't be any improvement from 0.72.
There is another ensemble learning technique called soft voting which uses probability but since SVM doesn't output probability thus it is not possible to use soft voting with it.

Comparison with other works

Let's compare the results in this report with those of Samuel, Ali, Rahman, Esawi, Samuel[2]. There is a study of the effect of accuracy on the number of characters in COVID-19 related tweets.

Following are the confusion matrices (in the paper) for Naive Bayes for various tweet lengths :

Number of characters less than 77, accuracy=0.9143

	Negative	Positive
Negative	34	1
Positive	5	30

Number of characters less than 120, accuracy=0.5714

	Negative	Positive
Negative	34	1

Positive	29	6

Following are the confusion matrices (in the paper) for Logistic Regression for various tweet lengths :

Number of characters less than 77, accuracy=0.7429

	Negative	Positive
Negative	30	5
Positive	13	22

Number of characters less than 120, accuracy=0.52

	Negative	Positive
Negative	21	14
Positive	19	16

Following is the confusion matrix obtained for positive-negative frequency, unigram, multinomial naive bayes for dataset-1 in my work :

Accuracy : 0.7022

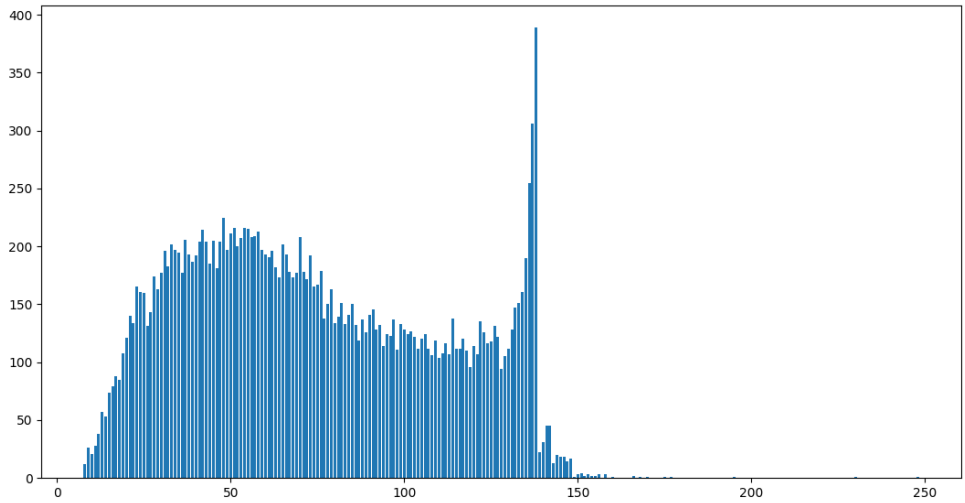
	Negative	Positive
Negative	1357	630
Positive	561	1452

Following is the confusion matrix obtained for bag of words, 2-grams, logistic regression for dataset-1 in my work :

Accuracy : 0.751

	Negative	Positive
Negative	1015	376
Positive	321	1088

Following is the distribution of tweet length in the dataset :



Out of 20000 tweets, 8564 have length greater than 77 and 11434 have length less than 77.

Let's compare the F scores (2nd column has F score for the research paper and 3rd column has F score for my work) :

Naive Bayes	0.9090, 0.2857	0.7091
Logistic Regression	0.7096, 0.4923	0.7573

Note that the 2nd column represents the F score of both : number of characters less than 77 and less than 120 (for example 0.9090, 0.2857 for naive bayes) in the research paper. We can say that Logistic Regression performed much better in my case. Also note that dataset-1 is generic whereas the dataset used by authors is related to COVID-19. A test with a dataset related to some single domain is expected to give better F score since words which play an important role in determining the sentiment would be less and occur a lot more time than in dataset which contains tweets from every domain.

Following table represents accuracy obtained by Kharde, Sonawane[3] and by me for naive bayes :

Algorithm	Kharde, Sonawane	dataset-1	dataset-2

Naive Bayes (unigram)	74.56	71.21	86.53
Naive Bayes (bigram)	76.44	72.54	86.4
Naive Bayes (trigram)	75.41	72	86.22

The results in the column of dataset-1 is for bag of words and results in the column of dataset-2 is for positive-negative frequency.

We can see that there is a slight decrease in accuracy for dataset-2 and dataset-1 with increase in n-grams. Similar patterns can be observed for Kharde, Sonawane when we compare bigram and trigram. There is an increase in the number of tokens with n-grams and when a sufficiently large dataset is not available the tokens formed with higher n-grams would be present in less number of tweets. It can then lead to poor training and those tokens would just skew the true results while testing.

Following table represents accuracy obtained by Kharde, Sonawane and by me for SVM :

Algorithm	Kharde, Sonawane	dataset-1	dataset-2
SVM with unigram	76.68	72.71	85.57
SVM with bigram	77.73	73.69	85.57

The results in the column of dataset-1 is for bag of words and results in the column of dataset-2 is for positive-negative frequency.

Similar to Kharde, Sonawane we can see that for dataset-1 we obtained more accuracy for bigram than unigram.

From the results it seems that type of algorithm plays an important role in determining whether accuracy will increase with increase in n-grams. It is observed from the screenshot section (“Code and Output”) that logistic regression and support vector machines usually give more increase in accuracy with increase in n-grams than naive bayes. For example, for bag of words, 72.46 accuracy was obtained for unigram and 75.10 for bigram with logistic regression whereas for naive bayes 71.21 accuracy was obtained for unigram and 72.54 for bigram.

Following table compares results obtained from 3 different sentiment analysis tools[6] :

	Proposed System	Alchemy API	SVM (Pre-processed tweets)	SVM, dataset-1
Accuracy	59.85%	58.87%	64.95%	73.69%
Precision	53.65%	40.82%	66.54%	71.33%
F-measure	0.48	0.43	0.57	70.21%

Column ‘SVM, dataset-1’ is for preprocessed tweets with bag of words and bigram. We can clearly see that I obtained better results compared to those in the first three columns. Note that dataset-1 contains almost 50% percent positive tweets and 50% negative tweets (to be precise : 10012 positive tweets and 9988 negative tweets from randomly chosen 20000 tweets from dataset-1).

For the airline dataset, I also ran algorithms for tweets with three categories : positive,neutral and negative sentiment. Results can be compared with those obtained by Taufic, Touhid[7] who also experimented with the same dataset.

Following is a comparison for the bag of words.

Algorithm	Accuracy Taufic, Touhid	Accuracy obtained by me (unigram, bigram)
SVM	0.77	0.78, 0.735
Multinomial Naive Bayes	0.74	0.762, 0.765
Logistic Regression	0.77	0.783, 0.78

Following is a comparison for the TF-IDF.

Algorithm	Accuracy Taufic, Touhid	Accuracy obtained by me (unigram, bigram)
SVM	0.77	0.743, 0.716
Multinomial Naive Bayes	0.70	0.686, 0.676
Logistic Regression	0.77	0.755, 0.766

We can see that for TF-IDF, the results of Taufic, Touhid are slightly better than my results. I have obtained slightly better results for SVM, Multinomial Naive Bayes, Logistic Regression in bag of words. Note that for TF-IDF with bigram I am getting less accuracy because I used less dataset size due to the memory limit of google colab (for example I used 9500 tweets for training and testing in TF-IDF with bigram and SVM whereas total dataset size is 14640).

Since I obtained better result for bag of words than the authors, let’s compare weighted F scores (author has also used weighted F scores since dataset is imbalanced) for the same :

	F score Taufic, Touhid	F score obtained by me (unigram)
SVM	0.75	0.769
Multinomial Naive Bayes	0.72	0.745

Logistic Regression	0.77	0.780

We can clearly see that I obtained better F scores than the authors for bag of words.

Comparison with python sentiment analysis tools

	Accuracy from Vader	Accuracy from Textblob	Accuracy obtained by me
dataset-1	0.52405	0.6192	0.751 (bag of words, logistic regression, bigram)
dataset-2 (with positive,negative and neutral tweets)	0.490	0.464	0.78 (bag of words, logistic regression, unigram)

Following are confusion matrices obtained for vader (-1 refers to negative sentiment, 0 refers to neutral, 1 refers to positive sentiment):

dataset-1

	-1	0	1
-1	4243	2548	3197
0	0	0	0
1	1016	2758	6238

dataset-2

	-1	0	1
-1	4018	1445	3715
0	350	1027	1722
1	65	159	2139

Following are confusion matrices obtained for textblob :

dataset-1

	-1	0	1

-1	6781	0	3207
0	0	0	0
1	4409	0	5603

dataset-2

	-1	0	1
-1	3231	3287	2660
0	331	1759	1009
1	104	453	1806

Following are confusion matrices obtained from my solution where I divided data into 80-20 for training and testing :

dataset-1

	-1	0	1
-1	1015	0	376
0	0	0	0
1	321	0	1088

dataset-2

	-1	0	1
-1	1629	156	53
0	192	332	69
1	94	70	333

From above we can see that I have obtained the highest accuracy in all cases. If we compare F score for highest accuracy case : F score for Textblob is 0.559 for dataset-1 and we have obtained F score of 0.757.

F score, dataset-1

Textblob	My code
0.59	0.757

other techniques used for improving performance

I have used the effect of exclamation marks and utilized information of words with underscore and hashtags which can increase the accuracy. A large number of tweets related to politics and sports have hashtags, words with underscores and usually they contain words other than stop words and keeping them can be useful compared to the cases where we just remove any token having character other than alphabet. Also, every technique is not used in every research paper. For example some researchers did not use n-grams, however I have tried to use each and every preprocessing technique which can improve the results.

Conclusion

- Based on comparison using different parameters (accuracy, precision, F_1 score, tweets with different numbers of characters) with 4 different research papers, I obtained better results than 2 of them and quite close to the other two. Note that dataset-1 has tweets from every domain and almost equal number of positive and negative tweets are present. If any dataset has an unequal number of positive-negative tweets or is from some specific domain (say COVID-19, movie review) then it's likely to give better results for some parameters as we observed with airline dataset (dataset-2). I obtained 0.917 accuracy for dataset-2 and 0.751 accuracy for dataset-1. However the F_1 scores, 0.75 for dataset-1 and 0.77 for dataset-2 are very close.
- I obtained better results in terms of F score and accuracy if we compare numbers obtained with vader and textblob.
- F_1 score of 0.7614 was obtained for dataset-1 and F_1 score of 0.8045 was obtained for dataset-2.
- Maximum accuracy of 0.751 was obtained for dataset-1 and maximum accuracy of 0.917 for dataset-2.
- Accuracy for dataset-1 is less because it's tweets are not from any specific domain. Dataset-2 contains airline dataset and so some words would occur more frequently and thus we got more accuracy for it.
- Bag of Words and TF-IDF gave comparable results and performed significantly better than Positive-Negative frequency. This is because in Bag of Words and TF-IDF we are considering all words whereas in Positive-Negative frequency all information is put into 2-dimension. Bag of words and TF-IDF have more time and memory complexity compared to positive-negative frequency.
- Joining multiple words (n-grams) gave better results compared to unigrams (1-gram).

Future Work

Currently I have thought about following ways to make further improvements :

- We can replace all synonym words with a common word in each tweet.
- We used n-grams for taking account of negation (for example : “not sad”). However we can try using antonyms (i.e replace “not sad” with “happy”) along with n-grams (for cases like “not very happy”, “not” and “happy” have one word between them and thus we can either use antonym for words few distance apart or we can use higher n-grams like 3-grams, 4-grams).
- Trying to improve preprocessing and better ways of representing the data mathematically.

References

[1] Alonso, Vilares, Gómez-Rodríguez, Vilares, Sentiment Analysis for Fake News Detection, 2021
[2] Samuel, Ali, Rahman, Esawi, Samuel, COVID-19 Public Sentiment Insights and Machine Learning for Tweets Classification, 2020
[3] Kharde, Sonawane, Sentiment Analysis of Twitter Data: A Survey of Techniques, 2016
[4] Alsaeedi, Khan, A Study on Sentiment Analysis Techniques of Twitter Data, 2019
[5] Twitter character limit
[6] Chong, Selvaretnam, Soon, Natural Language Processing for Sentiment Analysis, 2014
[7] Taufic, Touhid, A Comparative Study of Sentiment Analysis Using NLP and Different Machine Learning Techniques on US Airline Twitter Data, 2021