Sentiment Analysis on COVID-related Tweets Using Data Mining Techniques

08/2022

Group Name: Neem
Group member: Yuanzhi Xu, Chia-wei Lin, Wen Xie

Abstract

The proliferation of user-generated content (UGC) on social media platforms has made user opinion tracking a strenuous job(Raj et al., 2019). Twitter, being a huge microblogging social network, provides a platform for people to share and express their views towards topics, happenings, products and other services. And it could be used to accumulate views about the above mentioned aspects. Sentiment analysis is a mining technique employed to peruse opinions, emotions, and attitude of people toward any subject. The results could be wielded to provide an edge for businesses and governments in rolling out new entities (policies, products, topic, event).

This project contains implementation of data mining, machine learning, and natural language processing techniques, trains models on a labeled Tweets dataset, then applies the best-performing model to a public twitter dataset which captured COVID-19 related tweets posted on four days in Feb 2022, and thus proposes a sentiment analysis approach to classify the Tweets into positive and negative categories.

The project first chooses the best performing word vector among unigram and bigram vectorizer, unigram and bigram tf-idf vectorizer, and pretrained gloVe embedding. BernoulliNB model is applied to each of the 5 vectorizers, and it turns out that tf-idf bigram vectorizer has the best performance. With this best vectorizer, the project builds BernoulliNB, logistic regression, random forest, xgboost, and SVM models on the labeled Tweets dataset. It turns out that BernoulliNB Model yields the best performance. The project also compares performance of the derived model with two existing sentiment analyzing tools (nltk sentiment analyzer and textblob) and confirms the derived model yields best performance.

With this selected model, the project investigates the sentiment of Tweets of all Fridays in February 2022: Feb 4th, Feb 11th, Feb 18th, and Feb 25th. Result shows among 174,660 Tweets, 51.02% (89,113) are negative, and 48.98% (85,547) are positive. On each particular day investigated, no observable difference is detected on the ratio of negative sentiment to positive sentiment.

1. Introduction

Commencing on January, 2020, the Covid-19 pandemic has been placing great impact upon people's life all over the world. Economy, politics, infrastructure, medical service, and culture are all under great influence. Specifically, supply chain, housing, and even people's lifestyles are continuously changing. This project focuses on one particular aspect: people's emotional attitude (positive and negative) towards the Covid-19.
In the age of social media, sentiment analysis can efficiently analyze copious amounts of textual data, much of which is personal and can be freely accessed and analyzed. In fact, Twitter has been used extensively for public health surveillance, from monitoring and prediction to gauging public response (Jordan et al., 2019). This project performed sentiment analysis upon Tweets content, using data mining and machine learning techniques, aiming at generating insights on the public's attitude on particular days.

This study is of great importance. Covid-19 has been prevailing for such a long time and it has brought in such a lot of influence. Countries around the world are facing extraordinary challenges in implementing various measures to slow down the spread of COVID-19. Government and organizations have been taking actions such as lock down, isolation, heath monitoring, work from home, etc. When making a decision on whether to lay or lift a requirement, the public's attitude is a factor of great weight. Sentiment analysis can provide critical information about communities' risk perceptions. This project would help government and organizations to identify public sentiment towards certain words or topics, and in turn aids them to roll out new entities.

Twitter has become one of the most popular social media for years; data mining and machine learning science and techniques are also quite mature and have been always developing. However, Covid-19 is a comparably new thing: it started in 2020 and it has been no more than 3 years till now. Thus, research that combines Twitter, data mining, and Covid is a young and promising field. There has been recent studies upon Covid related topics, with data mining and machine learning techniques applied. The topics cover a wide range: predict the Covid positive case trend, study the impact of Covid vaccine, analyze the online platform that has been used to replace on-ground face to face communication, etc. Specifically focusing on Tweets, there have also been many studies and research, most focusing on sentiment analysis, since Tweets, to some extent, is a representation of public voice.

2. Methodology

There are many ways that people are expressing their attitudes, and this project chooses Twitter text as a perspective.

First, read and preprocess data. The project uses *Sentiment140 dataset with 1.6 million tweets* (https://www.kaggle.com/datasets/kazanova/sentiment140) from Kaggle. This dataset contains 1,600,000 tweets extracted using the twitter api. The tweets have been annotated (0 =

negative, 4 = positive) and they can be used to detect sentiment. This dataset is used as the train dataset since it is clean and already has target value labeled. The project also analyzes the data in this dataset. It is composed of 80,000 negative Tweets and 80,000 positive Tweets. A word cloud is done for better visualization of the data.

Then, the project preprocesses the text data. The preprocessing work includes: decode emoji, decode html, remove mention, remove url, remove punctuation, switch to lowercase, lemmatize stemming, and remove stop words. Also, if a Tweet is empty after the pre-process, then this particular data is removed from the dataset. After the removal, the data set contains 1,593,540 Tweets, among which 796895 are negative, taking up 50.00%, and 796645 are positive, taking up 49.99%. This is a balanced dataset.

Second, train-test split. The project splits train and test dataset to be 80:20.

Third, word vectorization. Unigram count vectorizer, bigram count vectorizer, tf-idf unigram vectorizer, tf-idf bigram vectorizer, and pretrained gloVe embedding are used with Bernoulli model in order to generate the best word vectorizer that can mathematically represent the dataset. The gloVe embedding turns out to be extremely low efficiency: fit time of 14.618990182876587 seconds, around 30 times of the time other vectorizers takes, while the accuracy rates are not observerbally different: gloVe embedding renders 66.8169% accuracy rate, while unigram count vectorizer results in 76.49%, bigram count vectorizer results in 77.39%, and unigram tf-idf vectorizer results in 76.80%, bigram tf-idf vectorizer results in 77.82. So the project chooses bigram tf-idf vectorizer as the word vectorizer.

Fourth, create and compare models. The project develops 5 models (bernoulliNB, logistic regression, random forest, xgboost, SVM) to interpret and classify sentiment (positive or negative) on the dataset with bigram tf-idf vectorizer. It turns out that BernoulliNB model has the best performance. Also, 2 existing Tweets sentiment analyzing tools (nltk sentiment analyzer and textblob) are introduced to test the performance of the model trained by the project. Results show that our model has better performance than existing tools.

Fifth, apply the best model to real world data. BernoulliNB model with bigram tf-idf vectorizer is employed to perform sentiment analysis and prediction on real world data collected from Tweets text: *Collect real-world tweets* (https://github.com/echen102/COVID-19-TweetIDs). The project first attempts to gather historical Tweets that mention specific keywords associated with Covid-19 within the time range from 01/2022 to 06/2022. Unfortunately, acquiring those data consumes tons of time and effort due to downloading api restrictions and limits. So the project focuses on 4 particular days in Feb 2022(02/04, 02/11, 02/18/, 02/25), and applies the derived model to generate sentiment analysis.

Result shows among 174,660 Tweets, 51.02% (89,113) are negative, and 48.98% (85,547) are positive. On each particular day investigated, no observable difference is detected on the ratio of negative sentiment to positive sentiment.

3. Code

3.1 Read and preprocess data

READ DATA

In [3]:
```
DATASET_CITATION = """
@ONLINE {Sentiment140,
    author = "Go, Alec and Bhayani, Richa and Huang, Lei",
    title  = "Twitter Sentiment Classification using Distant Supervision",
    year   = "2009",
    url    = "http://help.sentiment140.com/home"
}
"""
DATASET_DESCRIPTION = """
Sentiment140 allows you to discover the sentiment of a brand, product, or topic on Twitter.
The data is a CSV with emoticons removed. Data file format has 6 fields:
0. the polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)
1. the id of the tweet (2087)
2. the date of the tweet (Sat May 16 23:58:44 UTC 2009)
3. the query (lyx). If there is no query, then this value is NO_QUERY.
4. the user that tweeted (robotickilldozr)
5. the text of the tweet (Lyx is cool)
For more information, refer to the paper
Twitter Sentiment Classification with Distant Supervision at
https://cs.stanford.edu/people/alecmgo/papers/TwitterDistantSupervision09.pdf
"""
DATASET_DOWNLOAD_URL = "http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip"
DATASET_HOMEPAGE_URL = "http://help.sentiment140.com/home"
DATASET_FILE = "training.1600000.processed.noemoticon.csv"
DATASET_COLUMNS = ["target", "ids", "date", "flag", "user", "text"]
DATASET_ENCODING = "ISO-8859-1"
DATASET_LABEL = {0:"negative", 2:"neutral", 4:"positive"}
```

In [25]:
```
# # download dataset
# ! wget http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip
# ! unzip trainingandtestdata.zip
```

In [4]:
```
# read dataset
df = pd.read_csv(DATASET_FILE, encoding=DATASET_ENCODING, names=DATASET_COLUMNS)
df.head(5)
```

Out[4]:

	target	ids	date	flag	user	text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....

PREPROCESS TEXT

In [5]:
```
def decode_emoji(text):
    output = []
    for t in emoji.demojize(text).split():
        output.extend(t.split("_"))
    return ' '.join(output)

def decode_HTML(text):
    return BeautifulSoup(text, 'lxml').get_text()

def remove_mention(text):
    return re.sub(r'@[A-Za-z0-9]+', '', text)

def remove_URL(text):
    return re.sub(r"http\S+", "", text)

def remove_punctuation(text):
    return re.sub(r'[^\w\s]', ' ', text)

def correct_repeatted_letters(text):
    return re.sub(r'(.)\1+', r'\1\1', text)

def to_lowercase(text):
    return ' '.join([w.lower() for w in text.split()])

def lemmatize_stemming(text):
    return ' '.join([stemmer.stem(lemmatizer.lemmatize(w, pos='v'))
                     for w in text.split()])

def remove_stop_words(text):
    return ' '.join([w for w in text.split() if w not in stop_words])
```

In [6]:
```
def preprocess(text):
    text = decode_emoji(text)
    text = decode_HTML(text)
    text = remove_mention(text)
    text = remove_URL(text)
    text = remove_punctuation(text)
    text = correct_repeatted_letters(text)
    text = to_lowercase(text)
    text = lemmatize_stemming(text)
    text = remove_stop_words(text)
    return text
```

In [9]:
```
# preprocess text
tqdm_notebook.pandas()
df['text_cleaned'] = df['text'].progress_apply(lambda x: preprocess(x))
df['sentiment'] = df['target'].apply(lambda x: DATASET_LABEL[x])
df.head(5)
```

```
  0%|          | 0/1600000 [00:00<?, ?it/s]
```

Out[9]:

	target	ids	date	flag	user	text	text_cleaned	sentiment
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...	aww bummer shoulda get david carr third day	negative
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...	upset updat facebook text might cri result sch...	negative
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...	dive mani time ball manag save 50 rest go bound	negative
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire	whole bodi feel itchi like fire	negative
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....	behav mad whi becaus see	negative

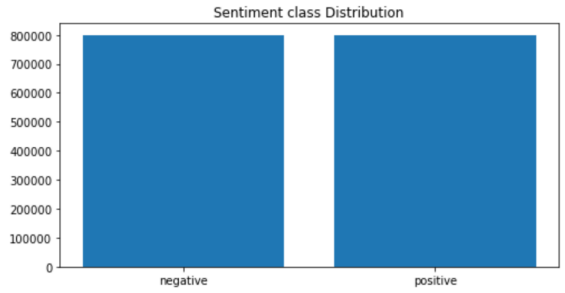
ANALYZE DATA

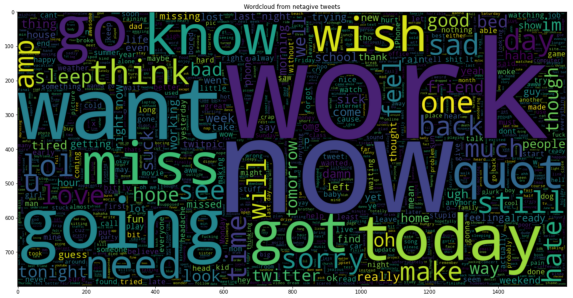
In [13]:
```
# analyse class distribution
val_count = df.sentiment.value_counts()
val_count
```

Out[13]:
```
negative    800000
positive    800000
Name: sentiment, dtype: int64
```

In [11]:
```
# visuallize class distribution
plt.figure(figsize=(8,4))
plt.bar(val_count.index, val_count.values)
plt.title("Sentiment class Distribution")
```

Out[11]:
```
Text(0.5, 1.0, 'Sentiment class Distribution')
```





3.2 Train-test split

TRAIN-TEST SPLIT

```
# drop rows with empty text_cleaned
print("Before remove empty text")
print(df.label.value_counts())

print()
df = df.dropna(subset=['text_cleaned'])

print("Removed empty text")
print(df.label.value_counts())
```

```
Before remove empty text
0    800000
1    800000
Name: label, dtype: int64

Removed empty text
0    796895
1    796645
Name: label, dtype: int64
```

```
# train test split
df_train, df_test = train_test_split(df[["text_cleaned", "label"]],
                                     test_size=1-TRAIN_SIZE, random_state=42, shuffle=True)
print("TRAIN size:", len(df_train))
print("TEST size:", len(df_test))
```

```
TRAIN size: 1274832
TEST size: 318708
```

```
y_train = np.array(df_train.label)
y_test = np.array(df_test.label)
```

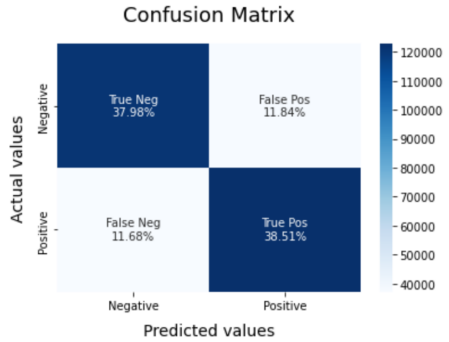
3.3 Word vectorization

test Bernoulli model with unigram count vectorizer

```
# apply unigram vectorizer to train data and transform tran and test data
unigram_vectorizer = CountVectorizer(stop_words='english', analyzer='word', ngram_range=(1, 1))
unigram_vectorizer.fit(df_train.text_cleaned)
save_model('unigramCountVectorizer', unigram_vectorizer)
```

```
# train on BernoulliNB Model
BNBmodel = BernoulliNB(alpha=2)
train_model("BernoulliNB | unigramCountVectorizer", BNBmodel, unigram_vectorizer)
```

```
==> BernoulliNB | unigramCountVectorizer: acc = 76.4891%, fit time = 0.33333587646484375 sec
              precision    recall  f1-score   support

           0       0.76      0.76      0.76    158773
           1       0.76      0.77      0.77    159935

    accuracy                           0.76    318708
   macro avg       0.76      0.76      0.76    318708
weighted avg       0.76      0.76      0.76    318708
```

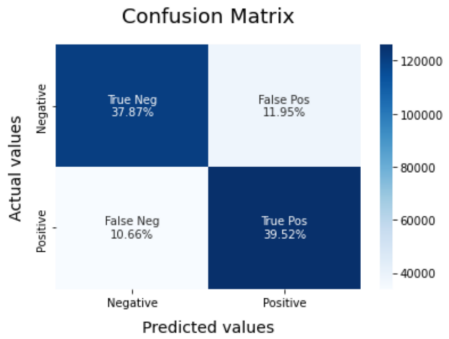


test Bernoulli model with bigram count vectorizer

```
# apply bigram vectorizer to train data and transform tran and test data
bigram_vectorizer = CountVectorizer(stop_words='english', analyzer='word', ngram_range=(1, 2))
bigram_vectorizer.fit(df_train.text_cleaned)
save_model('bigramCountVectorizer', bigram_vectorizer)
```

```
# train on BernoulliNB Model
BNBmodel = BernoulliNB(alpha=2)
train_model("BernoulliNB | bigramCountVectorizer", BNBmodel, bigram_vectorizer)
```

```
==> BernoulliNB | bigramCountVectorizer: acc = 77.3896%, fit time = 0.6653060913085938 sec
              precision    recall  f1-score   support

           0       0.78      0.76      0.77    158773
           1       0.77      0.79      0.78    159935

    accuracy                           0.77    318708
   macro avg       0.77      0.77      0.77    318708
weighted avg       0.77      0.77      0.77    318708
```

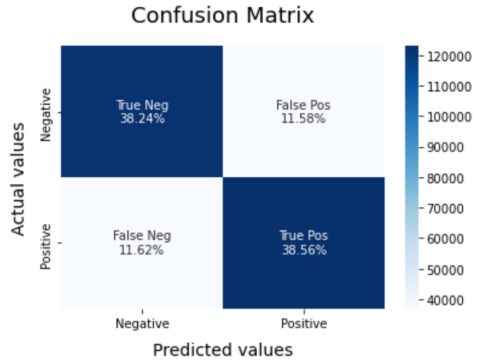


test Bernoulli model with tf-idf unigram vectorizer

```
# apply tf-idf vectorizer to train data
uni_tfidf_vectorizer = TfidfVectorizer(ngram_range=(1,1), max_features=500000)
uni_tfidf_vectorizer.fit(df_train.text_cleaned)
save_model('unigramTfidfVectorizer', uni_tfidf_vectorizer)
```

```
BNBModel = BernoulliNB(alpha = 2)
train_model("BernoulliNB | unigramTfidfVectorizer", BNBModel, uni_tfidf_vectorizer)
```

```
==> BernoulliNB | unigramTfidfVectorizer: acc = 76.8035%, fit time = 0.3609731197357178 sec
              precision    recall  f1-score   support

           0       0.77      0.77      0.77    158773
           1       0.77      0.77      0.77    159935

    accuracy                           0.77    318708
   macro avg       0.77      0.77      0.77    318708
weighted avg       0.77      0.77      0.77    318708
```

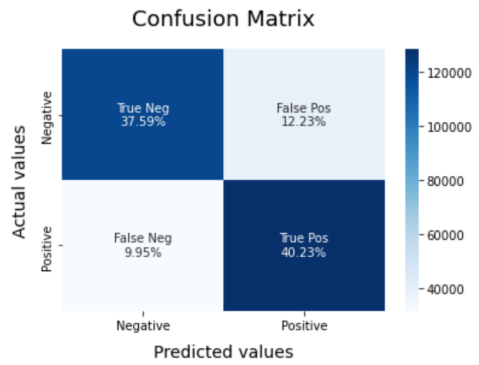


test Bernoulli model with tf-idf bigram vectorizer

```
# apply tf-idf vectorizer to train data
tfidf_vectorizer = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
tfidf_vectorizer.fit(df_train.text_cleaned)
save_model('bigramTfidfVectorizer', tfidf_vectorizer)
```

```
BNBModel = BernoulliNB(alpha = 2)
train_model("BernoulliNB | bigramTfidfVectorizer", BNBModel, tfidf_vectorizer)
```

```
==> BernoulliNB | bigramTfidfVectorizer: acc = 77.8189%, fit time = 0.49230194091796875 sec
              precision    recall  f1-score   support

           0       0.79      0.75      0.77    158773
           1       0.77      0.80      0.78    159935

    accuracy                           0.78    318708
   macro avg       0.78      0.78      0.78    318708
weighted avg       0.78      0.78      0.78    318708
```



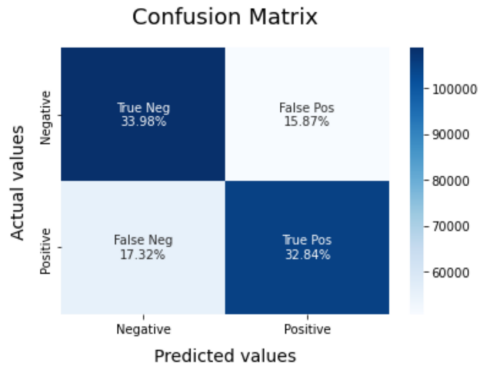
test Bernoulli model with pretrained gloVe embedding

```
# Download pretrained GloVe embedding
# ! wget http://nlp.stanford.edu/data/glove.twitter.27B.zip
# ! unzip glove.twitter.27B.zip
```

```
# print('loading word embeddings from GloVe...')

# embeddings_index = {}
# f = codecs.open('glove.twitter.27B.200d.txt', encoding='utf-8')

# for line in tqdm(f, total=1193515):
#     values = line.rstrip().rsplit(' ')
#     word = values[0]
#     coefs = np.asarray(values[1:], dtype='float32')
#     embeddings_index[word] = coefs
# f.close()

# print('found %s word vectors' % len(embeddings_index))
```

```
loading word embeddings from GloVe...
100%|██████████| 1193515/1193515 [02:07<00:00, 9381.47it/s]
found 1193514 word vectors
```

```
==> BernoulliNB with Glove Embedding: acc = 66.8169%, fit time = 14.618990182876587 sec
              precision    recall  f1-score   support

    negative       0.66      0.68      0.67    159494
    positive       0.67      0.65      0.66    160506

    accuracy                           0.67    320000
   macro avg       0.67      0.67      0.67    320000
weighted avg       0.67      0.67      0.67    320000
```



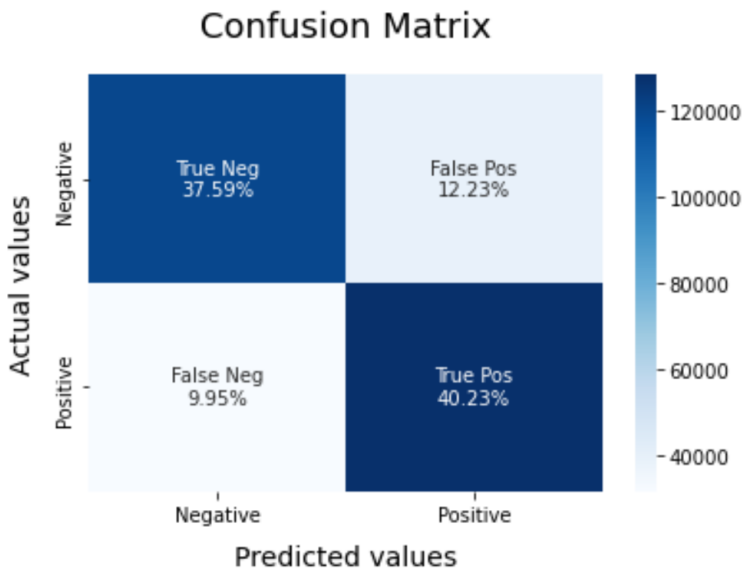
3.4 Create and compare models

3.4.1 BernoulliNB

bernoulliNB

```
# BernoulliNB Model
model = BernoulliNB(alpha = 2)
train_model("BernoulliNB | " + best_vec_name, model, best_vectorizer)
```

```
==> BernoulliNB | bigramTfidfVectorizer: acc = 77.8189%, fit time =
1.5670058727264404 sec
              precision    recall  f1-score   support

           0       0.79      0.75      0.77    158773
           1       0.77      0.80      0.78    159935

    accuracy                           0.78    318708
   macro avg       0.78      0.78      0.78    318708
weighted avg       0.78      0.78      0.78    318708
```

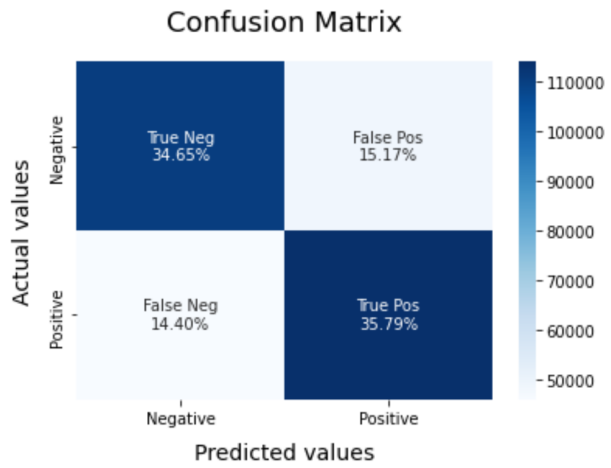


3.4.2 Logistic Regression

logistic regression

```
# Logistic Regression Model
LRmodel = LogisticRegression(n_jobs=-1, random_state=42)
train_model("Logistic Regression | " +  best_vec_name, LRmodel, best_vectorizer)
```

```
==> Logistic Regression | bigramTfidfVectorizer: acc = 70.4375%, fit time = 14.675590038
29956 sec
              precision    recall  f1-score   support

           0       0.71      0.70      0.70    158773
           1       0.70      0.71      0.71    159935

    accuracy                           0.70    318708
   macro avg       0.70      0.70      0.70    318708
weighted avg       0.70      0.70      0.70    318708
```

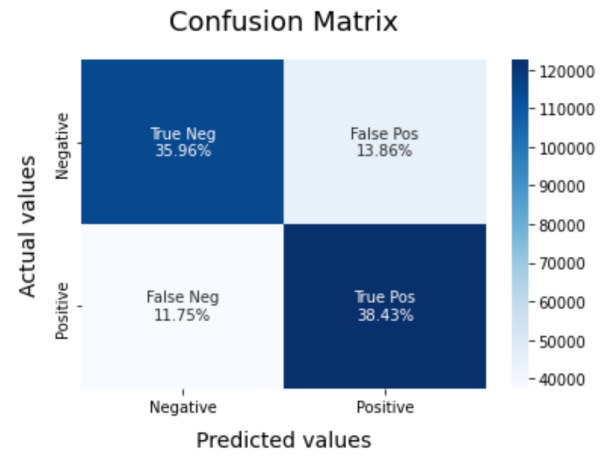


3.4.3 Random Forest

random forest

```
# Random Forest Model
RFModel = RandomForestClassifier(max_depth=40, random_state=42)
train_model("RandomForest | " + best_vec_name, RFModel, best_vectorizer)
```

```
==> RandomForest | bigramTfidfVectorizer: acc = 74.3897%, fit time = 170.8633029460907 s
ec
              precision    recall  f1-score   support

           0       0.75      0.72      0.74    158773
           1       0.73      0.77      0.75    159935

    accuracy                           0.74    318708
   macro avg       0.74      0.74      0.74    318708
weighted avg       0.74      0.74      0.74    318708
```

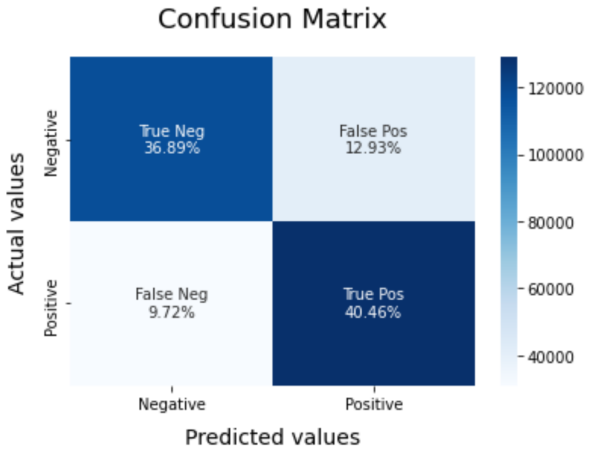


3.4.4 xgboost

xgboost

```
# Xgboost Model
XGBmodel = xgb.XGBClassifier(objective="binary:logistic", max_depth=40, random_state=42)
train_model("Xgboost | " +  best_vec_name, XGBmodel, best_vectorizer)
```

```
==> Xgboost | bigramTfidfVectorizer: acc = 77.3539%, fit time = 713.8064482212067 sec
              precision    recall  f1-score   support

           0       0.79      0.74      0.77    158773
           1       0.76      0.81      0.78    159935

    accuracy                           0.77    318708
   macro avg       0.77      0.77      0.77    318708
weighted avg       0.77      0.77      0.77    318708
```

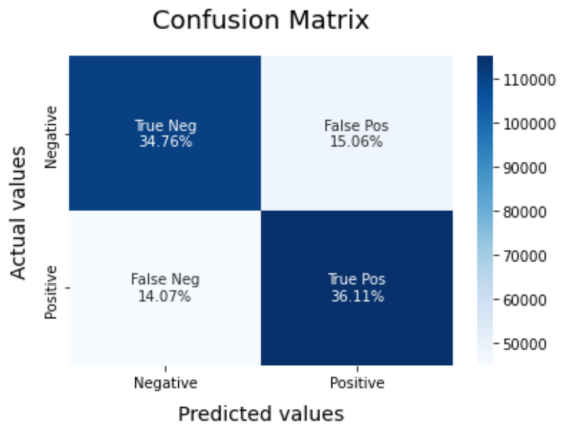


3.4.5 SVM

SVM

```
# LinearSVC Model
SVCmodel = LinearSVC(random_state=42, dual=False, max_iter=1000)
train_model("LinearSVC | " + best_vec_name, SVCmodel, best_vectorizer)
```

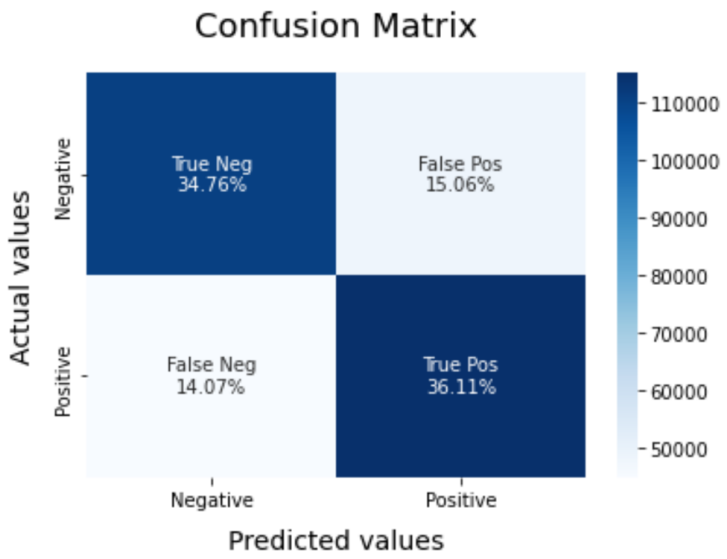
```
==> LinearSVC | bigramTfidfVectorizer: acc = 70.8667%, fit time = 222.1342430114746 sec
              precision    recall  f1-score   support

           0       0.71      0.70      0.70    158773
           1       0.71      0.72      0.71    159935

    accuracy                           0.71    318708
   macro avg       0.71      0.71      0.71    318708
weighted avg       0.71      0.71      0.71    318708
```



3.4.6 nltk sentiment intensity analyzer

```
from nltk.sentiment import SentimentIntensityAnalyzer
import operator
nltk.download('vader_lexicon')

name = "NLTK-SIA"
sia = SentimentIntensityAnalyzer()

test = df_test.copy()
start = time.time()
test["sentiment_score"] = test["text_cleaned"]
.apply(lambda x: sia.polarity_scores(x)["compound"])
y_pred = np.select([test["sentiment_score"] < 0, test["sentiment_score"] >= 0],
                   [NEGATIVE, POSITIVE])
end = time.time()

model_Evaluate(name, sia, y_test, y_pred, end-start)
```

```
==> LinearSVC | bigramTfidfVectorizer: acc = 70.8667%, fit time = 22
2.1342430114746 sec
              precision    recall  f1-score   support

           0       0.71      0.70      0.70    158773
           1       0.71      0.72      0.71    159935

    accuracy                           0.71    318708
   macro avg       0.71      0.71      0.71    318708
weighted avg       0.71      0.71      0.71    318708
```

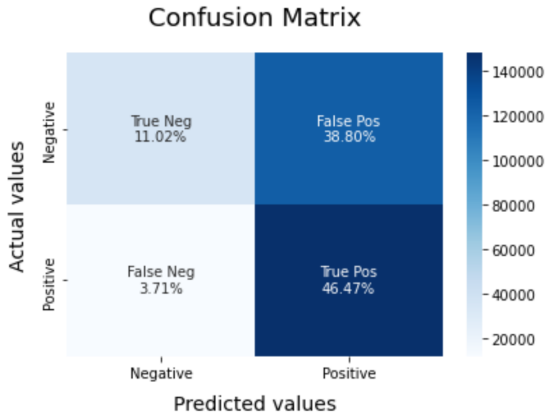


3.4.7 textblob

textblob

```
from textblob import TextBlob

name = "TextBlob"

start = time.time()
y_pred_pol = df_test["text_cleaned"].apply(lambda x: TextBlob(str(x)).sentiment.polarity)
y_pred = np.select([y_pred_pol < 0, y_pred_pol >= 0], [NEGATIVE, POSITIVE])
end = time.time()

model_Evaluate(name, None, y_test, y_pred, end-start)
```

```
==> TextBlob: acc = 57.4887%, fit time = 25.368539333343506 sec
              precision    recall  f1-score   support

           0       0.75      0.22      0.34    158773
           1       0.54      0.93      0.69    159935

    accuracy                           0.57    318708
   macro avg       0.65      0.57      0.51    318708
weighted avg       0.65      0.57      0.51    318708
```



3.5 Apply the best model to real world data

Read and preprocess data is similar to what has been done during training the best model. After data is prepared, the project applies the best model to the real world data and derives results.

3.5.1 Read data

```
## zip files
OUT_DIR = "../COVID-19-TweetIDs/"
JSON_DIR = OUT_DIR + "2022-02/"

# Fridays in Ferburary 2022
files= ["coronavirus-tweet-id-2022-02-04-07.jsonl",
        "coronavirus-tweet-id-2022-02-18-02.jsonl",
        "coronavirus-tweet-id-2022-02-11-02.jsonl",
        "coronavirus-tweet-id-2022-02-25-08.jsonl"]
```

```
from datetime import datetime
def get_date(string):
    return datetime.strptime(string, "%Y-%m-%dT%H:%M:%S.%fZ")
```

```
def process(line):
    # from json to single line df
    df_inter = pd.DataFrame([line])
    df_inter.columns = ['json_element']
    df_inter['json_element'].apply(json.loads)
    df_inter = pd.json_normalize(df_inter['json_element'].apply(json.loads))
    # extract needed columns
    df_needed = df_inter[['created_at', 'id', 'text', 'lang']]
    # format date
    df_needed["date"] = df_needed["created_at"].apply(get_date)
    df_needed['date'] = df.apply(lambda row: get_date(row['created_at']), axis =1)

    df_needed.drop(["created_at"], axis=1)

    return df_needed
```

```
import tqdm
df = pd.DataFrame()
```

```
# jsonl -> csv -> dataframe
for file in files:
    with tqdm.tqdm(total=os.path.getsize(JSON_DIR + file)) as pbar:
        with open(JSON_DIR + file, 'r') as reader:
            for line in reader:
                pbar.update(len(line))
                df_line = process(line.strip("\n"))
                if df_line["lang"][0] == "en":
                    frames = [df, df_line]
                    df = pd.concat(frames)
```

```
100%|██████████| 647278565/647278565 [02:35<00:00, 4175772.36it/s]
100%|██████████| 623715971/623715971 [04:47<00:00, 2167417.20it/s]
100%|██████████| 895503024/895503024 [09:53<00:00, 1509551.01it/s]
100%|██████████| 434427423/434427423 [05:41<00:00, 1273840.36it/s]
```

```
df.head(5)
```

	created_at	id	text	lang
0	2022-02-04T06:59:55.000Z	1489493905406296064	RT @DINESHK98891550: In the whole world, 100% ...	en
0	2022-02-04T06:59:55.000Z	1489493905426956288	RT @nicksey: Lol we're in trouble when he star...	en
0	2022-02-04T06:59:55.000Z	1489493905410121733	Vaccine manufacturers cannot be sued … what an...	en
0	2022-02-04T06:59:55.000Z	1489493905619832834	We just gave China 1.2 Billion, yes Billion, f...	en
0	2022-02-04T06:59:55.000Z	1489493905951186947	RT @nicksey: Lol we're in trouble when he star...	en

```
# Preprocess and clean text
df['text_cleaned'] = df['text'].progress_apply(lambda x: preprocess(x))
```

3.5.2 Pre-process data

```
# Preprocess and clean text
df['text_cleaned'] = df['text'].progress_apply(lambda x: preprocess(x))
```

```
  0%|          | 0/174660 [00:00<?, ?it/s]
```

```
df = df.reset_index()
static_df = pd.DataFrame()
static_df = df.copy()
df.head(5)
```

	index	created_at	id	text	lang	text_cleaned
0	0	2022-02-04T06:59:55.000Z	1489493905406296064	RT @DINESHK98891550: In the whole world, 100% ...	en	rt whole world 100 cure diseas like cancer aid...
1	0	2022-02-04T06:59:55.000Z	1489493905426956288	RT @nicksey: Lol we're in trouble when he star...	en	rt lol troubl start tweet indonesian loud cri ...
2	0	2022-02-04T06:59:55.000Z	1489493905410121733	Vaccine manufacturers cannot be sued … what an...	en	vaccin manufactur cannot sue idiot
3	0	2022-02-04T06:59:55.000Z	1489493905619832834	We just gave China 1.2 Billion, yes Billion, f...	en	give china 1 2 billion yes billion covid test ...
4	0	2022-02-04T06:59:55.000Z	1489493905951186947	RT @nicksey: Lol we're in trouble when he star...	en	rt lol troubl start tweet indonesian loud cri ...

3.5.3 Apply model

```
# predict the sentiments

# test SVM model with tf-idf vectorizer
X_predict_transform = vectorizer.transform(df['text_cleaned'])
X_predict = preprocessing.scale(X_predict_transform, with_mean=False)

df['predicted_sentiment'] = BNBmodel.predict(X_predict)

df.head(5)
```

	index	created_at	id	text	lang	text_cleaned	predicted_sentiment
0	0	2022-02-04T06:59:55.000Z	1489493905406296064	RT @DINESHK98891550: In the whole world, 100% ...	en	rt whole world 100 cure diseas like cancer aid...	positive
1	0	2022-02-04T06:59:55.000Z	1489493905426956288	RT @nicksey: Lol we're in trouble when he star...	en	rt lol troubl start tweet indonesian loud cri ...	negative
2	0	2022-02-04T06:59:55.000Z	1489493905410121733	Vaccine manufacturers cannot be sued … what an...	en	vaccin manufactur cannot sue idiot	negative
3	0	2022-02-04T06:59:55.000Z	1489493905619832834	We just gave China 1.2 Billion, yes Billion, f...	en	give china 1 2 billion yes billion covid test ...	negative
4	0	2022-02-04T06:59:55.000Z	1489493905951186947	RT @nicksey: Lol we're in trouble when he star...	en	rt lol troubl start tweet indonesian loud cri ...	negative

4. Results

4.1 Best vectorizer

Result of vectorization:

vectorizer	accuracy	fit time (sec)
CountVetorizer - unigram	0.7649	0.33333587646484375
CountVectorizer - bigram	0.7739	0.6653060913085938
TfidfVectorizer - unigram	0.7680	0.3609731197357178
TfidfVectorizer - bigram	0.7782	0.49230194091796875
gloVe embedding	0.6682	14.618990182876587

BernoulliNB model with bigram tf-idf vectorizer has the best performance.

4.2 Best model

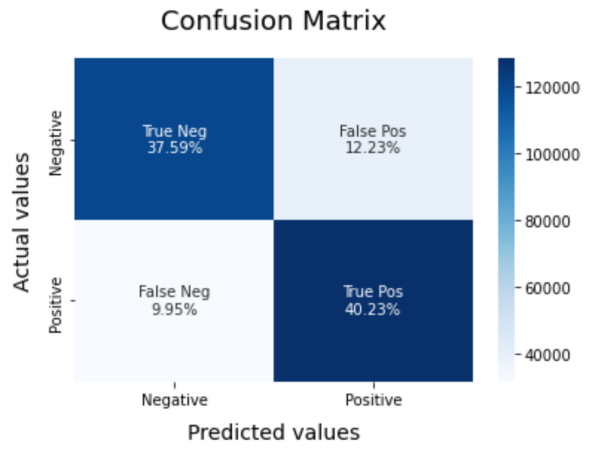
Applying the best vector, after testing bernoulliNB, logistic regression, random forest, xgboost, and SVM models on the given dataset, also comparing performance with two

existing sentiment analyzing tools (nltk sentiment analyzer and textblob) , it turns out that BernoulliNB with bigram tf-idf vectorizer outperforms all the other models.

```
models_in_order = sorted(models.items(), key=lambda x: x[1]['acc_test'], reverse=True)
for m in models_in_order:
    print(m)
```

```
('BernoulliNB | bigramTfidfVectorizer', {'acc_test': 0.7781888123297815, 'fit_time': 1.5670058727264404, 'model
': BernoulliNB(alpha=2)})
('BernoulliNB | bigramCountVectorizer', {'acc_test': 0.7738964820462618, 'fit_time': 0.6653060913085938, 'model
': BernoulliNB(alpha=2)})
('Xgboost | bigramTfidfVectorizer', {'acc_test': 0.7735387878559685, 'fit_time': 713.8064482212067, 'model': XG
BClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=40, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=42,
              reg_alpha=0, reg_lambda=1, ...)})
('BernoulliNB | unigramTfidfVectorizer', {'acc_test': 0.7680353175947889, 'fit_time': 0.3609731197357178, 'mode
l': BernoulliNB(alpha=2)})
('BernoulliNB | unigramCountVectorizer', {'acc_test': 0.764891373922211, 'fit_time': 0.33333587646484375, 'mode
l': BernoulliNB(alpha=2)})
('RandomForest | bigramTfidfVectorizer', {'acc_test': 0.7438972350866624, 'fit_time': 170.8633029460907, 'model
': RandomForestClassifier(max_depth=40, random_state=42)})
('LinearSVC | bigramTfidfVectorizer', {'acc_test': 0.7086674950111074, 'fit_time': 222.1342430114746, 'model':
LinearSVC(dual=False, random_state=42)})
('Logistic Regression | bigramTfidfVectorizer', {'acc_test': 0.7043751647275877, 'fit_time': 14.67559003829956,
'model': LogisticRegression(n_jobs=-1, random_state=42)})
('NLTK-SIA', {'acc_test': 0.6264323455953412, 'fit_time': 22.067135095596313, 'model': <nltk.sentiment.vader.Se
ntimentIntensityAnalyzer object at 0x7fe3d6dee190>})
('TextBlob', {'acc_test': 0.5748867301730738, 'fit_time': 25.368539333343506, 'model': None})
```

```
==> BernoulliNB | bigramTfidfVectorizer: acc = 77.8189%, fit time = 1.5670058727264404 sec
              precision    recall  f1-score   support

           0       0.79      0.75      0.77    158773
           1       0.77      0.80      0.78    159935

    accuracy                           0.78    318708
   macro avg       0.78      0.78      0.78    318708
weighted avg       0.78      0.78      0.78    318708
```



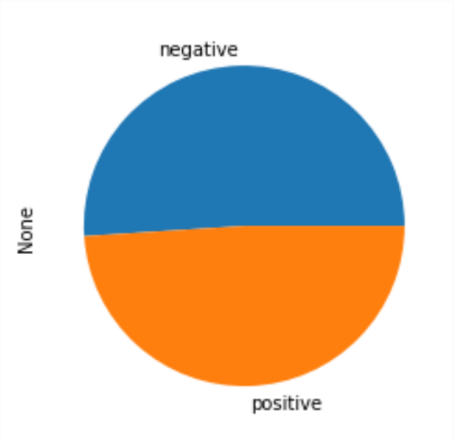
Also, this best model has better performance on predicting positive sentiment.

4.3 Real world data analysis

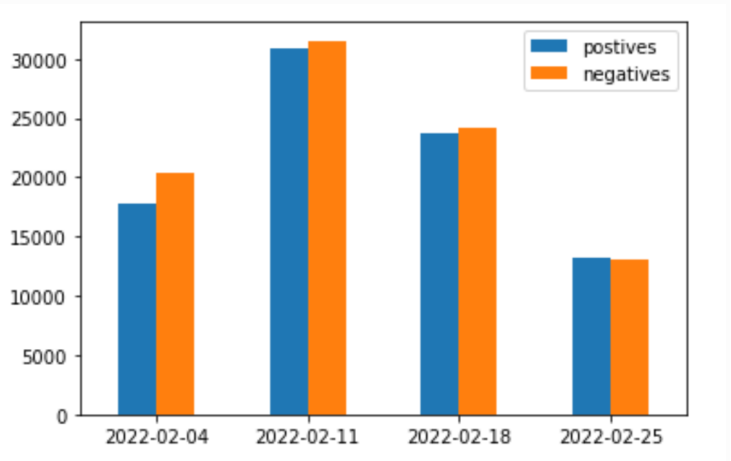
Result shows that under BernoulliNB model with bigram tf-idf vectorizer, among 174,660 Tweets, 51.02% (89,113) are negative, and 48.98% (85,547) are positive. On each particular day investigated, no observable difference is detected on the ratio of negative sentiment to

positive sentiment. On 02/04/2022, 20380 Tweets are negative, 17813 are positive; On 02/11/2022, 31538 Tweets are negative, 30853 are positive; On 02/18/2022, 24112 Tweets are negative, 23671 are positive; On 02/25/2022, 13083 Tweets are negative, 13210 are positive.

overall distribution of Tweets



distribution of Tweets in each selected day



5. Discussion

5.1 Sentiment analysis result

Data analysis shows that the positive and negative sentiments are not observably unequal in the 4 specific days in 02/2022. Negative takes a minor more distribution. This indicates that the public are not extremely anxious, but also not absolutely optimistic towards Covid-19.

5.2 Model performance

There has not been a huge difference among various vectors and models, however, the time consuming is of great difference. A balance between prediction accuracy and time consumed should be maintained in data analysis works. This project does not choose gloVe embedding due to its low efficiency.

From the perspective of natural language processing, the accuracy rate this project achieved is not high. The comparatively low accuracy is caused by the nature of Tweets: short expressions not containing much meaningful information. Even a human being sometimes will find it hard to detect the sentiment of a certain Tweet.

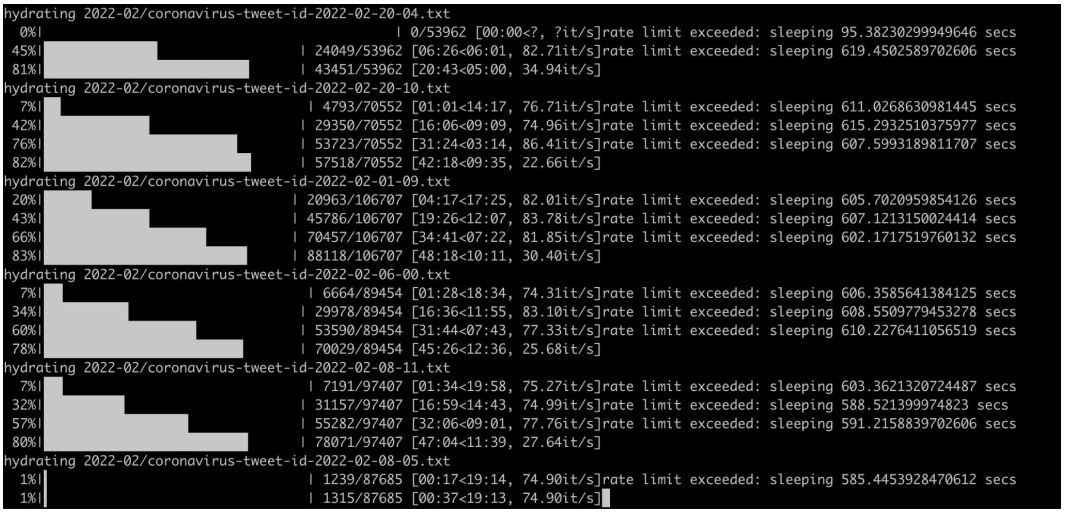
The model trained by the project outperforms the existing tools. One possible explanation would be the project merely chooses Tweets as the content rather than a comprehensive corpus source.

6. Future Work

First, due to time and resource limits, the best model developed in this project is based on tf-idf vectorizer. This vector is chosen upon BernoulliNB model. To make the model more accurate and predictive, more models should be applied when deciding the word vector. The best scenario would be picking the best vector together with the best model among all combinations of the 3 vectors and 5 models, that is, the best out of 3*15= 15 choices.

Second, obtaining real world Tweets and the pre-processing of these Tweets is time consuming and effort demanding. Given limited time for this study, with the purpose of controlling dataset size, this study chose to limit the source data, the real world tweets, to specific days in Feb 2022. And the insights generated can reflect the public's attitude within this restricted time, but it can not reflect the trend of attitude during the entire, long period since the pandemic began in 2020. Further research that covers tweets from 2020 to present will help a lot to generate the attitude trend, and in turn offer more thorough and persuasive conclusions.

Downloading real world data is time consuming and traffic is limited during downloading



Third, the model could be fine-tuned by parameter adjustment, or even, a neural network model could be employed to understand the meaning of a full sentence rather than simply analyze the word in the sentence and then classify the sentiment of the Tweet. Better model and more comprehensive approach would most probably render more convincing results.

Last but not least, this project selected Twitter to be the user-generated content (UGC) source. There are many other content sources which are also popular and can stand for people's sentiment from some perspective, for example: Reddit, FaceBook, Blind, etc. Simply using content from one social media platform is not strong enough to speak for the entire public. More content sources should be involved and a more comprehensive result would be given.

7. Conclusion

The project sanitizes and analyzes the text source, tests among 5 word vectorizors and decides the bigram tf-idf vectorizer to be employed. With this vector, the project compares accuracy rate as well as fit time among 5 models, and it turns out that BernoulliNB with bigram tf-idf vectorizer is the best. The project also confirms this model outperforms 2 existing sentiment analysis tools. Then, the BernoulliNB with bigram tf-idf vectorizer is applied to investigate the sentiment of Tweets of Feb 4th, Feb 11th, Feb 18th, and Feb 25th, 2022. Result shows that under BernoulliNB model with bigram tf-idf vectorizer, among 174,660 Tweets, 51.02% (89,113) are negative, and 48.98% (85,547) are positive. On each particular day investigated, no observable difference is detected on the ratio of negative sentiment to positive sentiment.

To make the prediction more accurate, a more comprehensive approach could be used to select the best word vector. A better model would be gained by tuning the parameter of the model during the training process. A neural network model could be employed to understand the meaning of a full sentence.

To offer a better public attitude report to the government or organizations, a larger real world dataset would aid in showing the trend of the public sentiment rather than currently the 4 specific days' result. More public voices from diverse social media platforms should be included instead of solely using Tweets.

8. References

1. Raj P. Mehta, Meet A. Sanghiv, Darshin K. Shah, Artika Singh (2019). Sentiment Analysis of Tweets Using Supervised Learning Algorithms: https://link.springer.com/chapter/10.1007/978-981-15-0029-9_26
2. Ankur Goel, Jyoti Gautam, Sitesh Kumar (2016). Real time sentiment analysis of tweets using Naive Bayes: https://ieeexplore.ieee.org/abstract/document/7877424
3. Mohammed Alhajji (2020). Sentiment analysis of tweets in Saudi Arabia regarding governmental preventive measures to contain COVID-19: https://www.researchgate.net/profile/Mohammed-Alkhalifah/publication/340482763_Sentiment_Analysis_of_Tweets_in_Saudi_Arabia_Regarding_Governmental_Preventive_Measures_to_Contain_COVID-19/links/5ffec2b192851c13fe0a3345/Sentiment-Analysis-of-Tweets-in-Saudi-Arabia-Regarding-Governmental-Preventive-Measures-to-Contain-COVID-19.pdf
4. Jordan, S. E., Hovet, S. E., Fung, I. C. H., Liang, H., Fu, K. W., & Tse, Z. T. H. (2019). Using Twitter for public health surveillance from monitoring and prediction to public response. Data, 4(1), 6.

9. Annex-A

Chia-Wei: Drives the project; trains model using Kaggle dataset; also helps writing the report.

Yuanzhi: Deals with real world data; applies the trained model to Tweets data and does analysis; also helps writing the report.

Wen: Report proposal, draft, and final report.