코로나 트위터1 과제#4

R E P O R T

조원 : 정주은(팀장), 양진욱(팀원), 배수현(팀원), 송태현(팀원)

1)분석목표

딥러닝 기반 코로나19 관련 트위터 감정분석 성능 비교

본 연구에서는 전세계적인 소셜 미디어 '트위터'에서 수집한 데이터를 가지고 코로나19 관련 감정분석을 수행하고자 한다. 트위터가 가지는 구조적 특징을 유지하면서도 불필요한 문자, 기호 등을 제거하기 위한 전처리 방식을 제시하였다. 텍스트를 컴퓨터가 이해할 수 있는 언어로 변환하기 위해서 통계적 기반의 임베딩 기법과 문장 수준의 임베딩 기법을 적용하였다. 자연어 처리 분야에서 대표적으로 사용되는 머신러닝과 딥러닝기법을 사용하여 감정분석을 수행하고 각 성능을 비교분석하였다.

2)자료설명

트위터는 소셜미디어 중 마이크로 블로그에 해당한다. 140자의 단문 소통, 다자간의 대화, 실시간성이라는 구조적 특징을 지니고 있다(Hur and Choi, 2012). 트위터 메세지인 '트윗'은 통신언어에서 나타나는 언어적 특징을 가지며, 이는 이모티콘의 사용, 축약어, 불필요한 자음 첨가 등이 있다. 또한 실시간적인 다자 소통을 위해 '멘션'이나 '해시태그'가 다수 포함되어 있다.
본 연구에서는 코로나19를 키워드로 하여 2020년 3월부터 4월 까지 검색된 44,955개의 트위터 메세지 기반으로 단어들을 추출한 데이터(이하 트위터 데이터)를 사용하였다. 이는 예측모델 및 분석대회 플랫폼의 일종인 ‘캐글(Kaggle)’에 공개된 데이터셋으로부터 수집하였다.
본 연구에서 사용하는 데이터는 사용자 이름, 해당 트윗이 작성된 지역, 시간, 메세지 원문과 감정구분 컬럼을 포함한 csv 파일이다. 각 이름은 개인정보 보호를 위해 코드화되었고, 감정구분 컬럼은 해당 데이터셋의 공유자가 수동으로 태깅하여 작성되었다.

# UserName Name	# ScreenName Screen Name	Location Tweeted from	TweetAt Tweeted At	OriginalTweet Twitter Text	Sentiment Label
 1 3798	 45.0k 48.8k	[null] 22% United States 2% Other (2889) 76%	13-03-2020 32% 12-03-2020 18% Other (1880) 49%	3798 unique values	Negative 27% Positive 25% Other (1810) 48%
1	44953	NYC	02-03-2020	TRENDING: New Yorkers encounter empty supermarket shelves (pictured, Wegmans in Brooklyn), sold-out ...	Extremely Negative
2	44954	Seattle, WA	02-03-2020	When I couldn't find hand sanitizer at Fred Meyer, I turned to #Amazon. But $114.97 for a 2 pack of ...	Positive

csv 파일을 읽어온 후, 데이터 프레임 형식으로 변환하였다.

```
## 데이터 읽기
train=pd.read_csv("Corona_NLP_train.csv",encoding='latin1')
test=pd.read_csv("Corona_NLP_test.csv",encoding='latin1')


## 데이터프레임으로 바꾸기
## df = train + test
df=pd.concat([train,test])
df['OriginalTweet']=df['OriginalTweet'].astype(str)
df['Sentiment']=df['Sentiment'].astype(str)

## DataFrame 열을 문자열로 변환
train['OriginalTweet']=train['OriginalTweet'].astype(str)
train['Sentiment']=train['Sentiment'].astype(str)

test['OriginalTweet']=test['OriginalTweet'].astype(str)
test['Sentiment']=test['Sentiment'].astype(str)
df.head()
```

	UserName	ScreenName	Location	TweetAt	OriginalTweet	Sentiment
0	3799	48751	London	16-03-2020	@MeNyrbie @Phil_Gahan @Chrisitv https://t.co/i...	Neutral
1	3800	48752	UK	16-03-2020	advice Talk to your neighbours family to excha...	Positive
2	3801	48753	Vagabonds	16-03-2020	Coronavirus Australia: Woolworths to give elde...	Positive
3	3802	48754	NaN	16-03-2020	My food stock is not the only one which is emp...	Positive
4	3803	48755	NaN	16-03-2020	Me, ready to go at supermarket during the #COV...	Extremely Negative

3)자료처리

쓰레기를 넣으면 쓰레기가 나온다(garbage in, garbage out)는 것은 데이터 분석의 기본 전제다. 바른 분석 결과를 얻기 위해서는 올바른 데이터를 입력해야 한다. 딥러닝 학습에 있어서 데이터는 가장 중요한 핵심 요소이다. 수집한 원시 데이터는 딥러닝 학습에 적절하게 활용될 수 있도록 전처리 과정이 필요하다. 본 연구에서는 아래 과정을 거쳐 데이터 가공 및 전처리를 진행하였다.

1. 데이터 중복값 처리
데이터를 수집하고 병합하는 과정에서 발생할 수 있는 오류로 인해 데이터가 중복되는 경우가 발생할 수 있다. 따라서 데이터 분석에 앞서 중복 데이터를 확인하고 처리하는 과정이 필요하다. 본 연구에서는 데이터의 중복값을 찾고, 그 중 하나의 값만 남길 수 있도록 처리하였다.

```
# Drop duplicates
## 데이터 중복값 중 하나만 남기고 처리
train.drop_duplicates()
print(" Shape of dataframe after dropping duplicates: ", df.shape)
```

```
Shape of dataframe after dropping duplicates:  (44955, 6)
```

2. 결측치 제거
트위터 데이터에서 발생하는 결측치의 특성을 분석하고 결측치 제거를 진행하였다. 총 6개의 컬럼으로 구성되어있는 해당 데이터에서 결측치가 존재하는 컬럼은 'Location' 컬럼이 유일하였다. 트위터는 이용자의 위치 정보를 연결하기 위해 '정확한 위치' 기능을 제공하고 있지만, 이용자가 선택적으로 이 기능을 해제할 수 있다. 다른 컬럼의 정보들과 달리 위치와 같은 개인정보 제공과 수집에 동의하지 않는 이용자에 의해 특정 컬럼에 대해서만 결측치가 발생한 것으로 분석할 수 있다. 'Location' 컬럼에 대해 0.2% 정도의 결측치가 발생하여 해당 행을 모두 제거하였다.

```
## isnull() : 관측치가 결측이면 True, 결측이 아니면 False 반환
## isnull().sum() : 컬럼별 결측값 개수 구하기
## sort_values() : DF 정렬 / ascending=False : 내림차순
null= df.isnull().sum().sort_values(ascending=False)
## df.shape[0] : 행 개수 세기
total =df.shape[0]
```

```
## 전체 중에 결측치 비율
percent_missing=
(df.isnull().sum()/total).sort_values(ascending=False)

## 컬럼(데이터 구분)별로 몇 개의 행(자료)에서 결측치가 존재하는지와 그 비율
missing_data= pd.concat([null, percent_missing], axis=1, keys=['Total
missing', 'Percent missing'])

## 행 인덱스 초기화 (인덱스를 다시 처음부터 재배열) / inplace=True : 새 df
생성하지 않고 기존df를 바꿈
missing_data.reset_index(inplace=True)
## columns= { 기존 컬럼명 : 새 컬럼명 }
missing_data= missing_data.rename(columns= { "index": " column
name"})

print ("Null Values in each column:\n", missing_data)
```

```
Null Values in each column:
      column name  Total missing  Percent missing
0        Location           9424         0.209632
1       Sentiment              0         0.000000
2   OriginalTweet              0         0.000000
3         TweetAt              0         0.000000
4      ScreenName              0         0.000000
5        UserName              0         0.000000
```

3. 감정구분 단순화

수동으로 태깅된 'Sentiment' 컬럼의 값은 총 5가지로 이루어져 있다. 이는 'Extremely Negative, Negative, Neutral, Positive, Extremely Positive'의 단계로 구분된다. 그러나 해당 범위 내에 존재하는 데이터의 양적인 증가와 학습 및 추론의 효율성을 위해 세분화된 감정구분을 'Negative, Neutral, Positive'의 3단계로 재분류하였다.

감정구분을 단순화하는 과정에서 데이터 불균형 문제가 발생할 수 있다. Negative와 Positive 클래스의 경우, 기존 2개였던 클래스를 하나로 합쳐 데이터의 크기가 증가하였다. 그러나 Neutral 클래스의 경우, 기존 하나였던 클래스만큼의 데이터 크기를 유지하여, 다른 두 클래스와의 데이터 비율 차이가 크게 나타남을 확인하였다.

```
# Data has 5 classes, let's convert them to 3
## 감정구분을 3가지로 줄이고자 함
def classes_def(x):
    if x ==  "Extremely Positive":
```

```
        return "positive"
    elif x == "Extremely Negative":
        return "negative"
    elif x == "Negative":
        return "negative"
    elif x ==  "Positive":
        return "positive"
    else:
        return "neutral"

## Sentiment 컬럼 데이터를 class_def에 적용하여 바꿈
df['sentiment']=df['Sentiment'].apply(lambda x:classes_def(x))
train['sentiment']=train['Sentiment'].apply(lambda x:classes_def(x))
test['sentiment']=test['Sentiment'].apply(lambda x:classes_def(x))
target=df['sentiment']

## 어떤 컬럼/Series의 unique value들을 count해주는 함수
## normalize=True : 비율 확인
df.sentiment.value_counts(normalize= True)
```

```
positive    0.435814
negative    0.378846
neutral     0.185341
Name: sentiment, dtype: float64
```

4. 정규표현식

정규표현식을 이용하여 전처리를 진행하였다. 인터넷 언어의 특징을 가지고 있는 트위터 메세지 '트윗'은 일반적인 텍스트와 달리, 이모티콘을 사용하는 경우가 많고, 다른 사용자를 언급하는 '멘션'이나 관련된 글들을 모아 분류할 수 있는 '해시태그' 등을 포함하고 있다. 이에 따라 1) url이나 html 링크 제거, 2) '@'로 시작하는 멘션 제거, 3) 반복되는 공백, 숫자, 구두점 제거, 4) 텍스트가 아닌 이모티콘 제거 과정을 진행하였다. 해시태그의 경우, 트위터 사용자가 관심있는 주제 혹은 관련된 내용을 분류하기 위해 지정하므로 이는 감정 분석에 활용하기 위해 남겨두었다.

```
# 소문자로 변경
def lower(text):
    low_text= text.lower()
    return low_text

train['text_new']=train['text'].apply(lambda x:lower(x))
```

```
test['text_new']=test['text'].apply(lambda x:lower(x))

# Urls and HTML links 제거
def remove_url(text):
    url_remove = re.compile(r'https?://\S+|www\.\S+') # \S : 공백 아님,
+ : 최소 한 번 이상 반복, | : 여러 개의 표현식 중 하나(or)
    return url_remove.sub(r'', text)
train['text_new']=train['text'].apply(lambda x:remove_url(x))
test['text_new']=test['text'].apply(lambda x:remove_url(x))

def remove_html(text):
    html=re.compile(r'<.*?>')
    return html.sub(r'',text)    # 빈 칸으로 변경
train['text']=train['text_new'].apply(lambda x:remove_html(x))
test['text']=test['text_new'].apply(lambda x:remove_html(x))

# @로 시작하는 문자나 숫자 제거(멘션 제거)
def remove_mention(x):
    text=re.sub(r'@\w+','',x) # \w: 숫자나 문자, + : 최소 한 번 이상 반복
    return text
train['text_new']=train['text'].apply(lambda x:remove_mention(x))
test['text_new']=test['text'].apply(lambda x:remove_mention(x))

# 반복되는 공백 제거
def remove_space(text):
    space_remove = re.sub(r"\s+"," ",text).strip() # \s : 공백, + :
최소 한 번 이상 반복
    return space_remove
train['text_new']=train['text'].apply(lambda x:remove_space(x))
test['text_new']=test['text'].apply(lambda x:remove_space(x))


# 반복되는 숫자 제거
def remove_num(text):
    remove= re.sub(r'\d+', '', text)  # \d : 숫자, + : 최소 한 번 이상
반복
    return remove
train['text']=train['text_new'].apply(lambda x:remove_num(x))
test['text']=test['text_new'].apply(lambda x:remove_num(x))

# 구두점 제거
def punct_remove(text):
    punct = re.sub(r"[^\w\s\d]","", text)  # # \w: 숫자나 문자, \s :
```

```
공백, \d : 숫자, [^ ] : 반대
    return punct
train['text_new']=train['text'].apply(lambda x:punct_remove(x))
test['text_new']=test['text'].apply(lambda x:punct_remove(x))

# 이모지(emoji) 제거
def remove_emoji(text):
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols &
pictographs
                               u"\U0001F680-\U0001F6FF"  # transport &
map symbols
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)
train['text']=train['text_new'].apply(lambda x:remove_emoji(x))
test['text']=test['text_new'].apply(lambda x:remove_emoji(x))
```

5. 불용어 제거

파이썬에서 제공하는 NLTK(Natural Language ToolKit) 내 영어 불용어(stopwords) 리스트를 다운받아 불용어 제거를 진행하였다.

```
#Remove stopwords & Punctuations
import nltk
nltk.download('stopwords')

#Remove stopwords & Punctuations
from nltk.corpus import stopwords
", ".join(stopwords.words('english'))
STOPWORDS = set(stopwords.words('english'))

def punct_remove(text):
    punct = re.sub(r"[^\w\s\d]","", text)
    return punct

train['text_new']=train['text'].apply(lambda x:punct_remove(x))
test['text_new']=test['text'].apply(lambda x:punct_remove(x))
```

```
def remove_stopwords(text):
    """custom function to remove the stopwords"""
    return " ".join([word for word in str(text).split() if word not
in STOPWORDS])
train['text']=train['text_new'].apply(lambda x:remove_stopwords(x))
test['text']=test['text_new'].apply(lambda x:remove_stopwords(x))
```

6.TF-IDF

TF-IDF(Term Frequency - Inverse Document Frequency)는 정보 검색과 텍스트 마이닝에서 이용하는 가중치로, 여러 문서로 이루어진 문서군이 있을 때 어떤 단어가 특정 문서 내에서 얼마나 중요한 것인지를 나타내는 통계적 수치이다.

```
## sublinear_tf : 높은 tf값을 완만하게 처리하여 아웃라이어 문제를 해결하고자
함 (tf = 1+ln(tf))
## min_df : DF의 최소값을 설정하여 특정 단어가 나타난 문서 수가 5개 미만인
경우, 해당 단어를 제외하고 인덱스를 부여하지 않음
tfidf = TfidfVectorizer(sublinear_tf=True, min_df=5,
                        stop_words='english')

# We transform each text into a vector
features = tfidf.fit_transform(train.text).toarray()
labels = train.label

print("Each of the %d tweets is represented by %d features (TF-IDF
score of unigrams and bigrams)" %(features.shape))
```

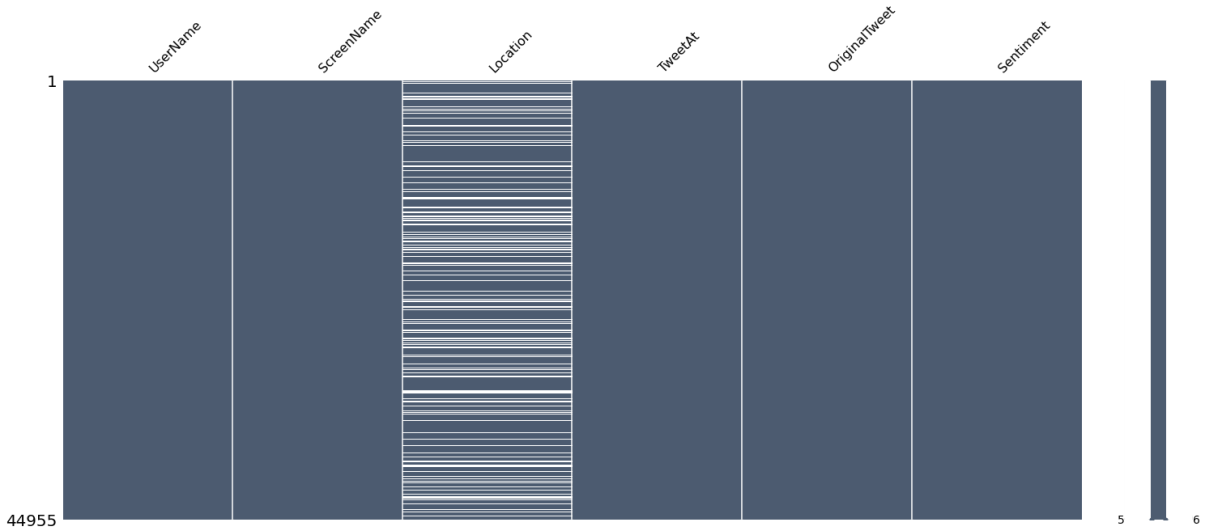
```
Each of the 41157 tweets is represented by 10615 features (TF-IDF score of unigrams and bigrams)
```

4)자료탐색

캐글에 공유되는 아래 코드를 참고하여 탐색적 자료 분석을 진행하였다.

1. 결측치 시각화

```
#Missing data as white lines
## 매트릭스 형태로 결측치 시각화
import missingno as msno
msno.matrix(df,color=(0.3,0.36,0.44))
```



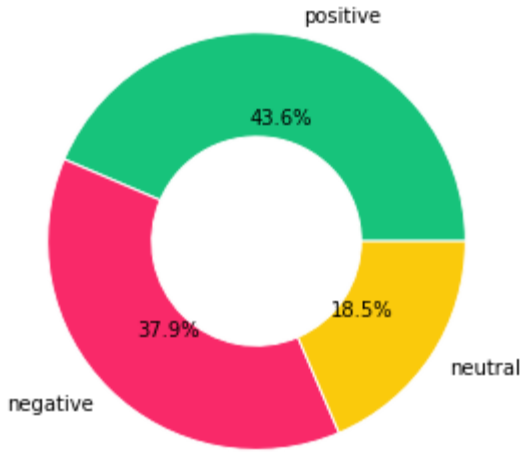
2. 클래스별 분류

```
## groupby() : 그룹별로 요약
## count() : 해당 그룹에 대한 데이터 세기
## df.groupby('sentiment').count()['text'] : text 컬럼의 자료 개수만 세기
## reset_index() : 인덱스 재배열
## sort_values(by='text') : 어떤 컬럼의 값을 기준으로 정렬해야 하는지 명시적으로 설정
## ascending=False : 내림차순으로 정렬
class_df = df.groupby('sentiment').count()['text'].reset_index().sort_values(by='text',ascending=False)
## df 숫자에 따라서 색을 다르게 입힘
class_df.style.background_gradient(cmap='winter')
```

	sentiment	text
2	positive	19592
0	negative	17031
1	neutral	8332

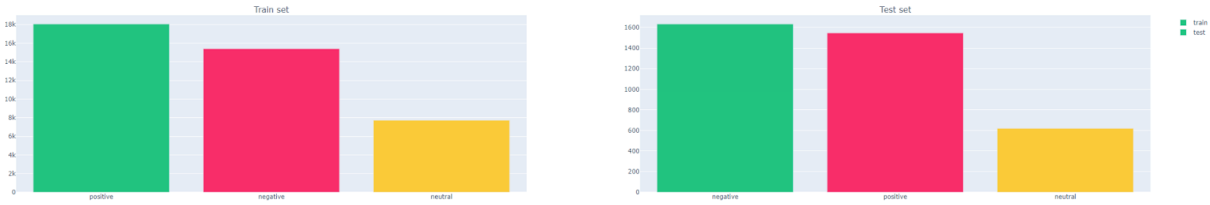
```
import matplotlib.pyplot as plt
import os
```

```
import numpy as np
import pandas as pd

import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

## text 값
percent_class=class_df.text
## 감정구분
labels= class_df.sentiment

colors = ['#17C37B','#F92969','#FACA0C']

## plt.plot()을 만들면 3가지 데이터(그래프 모양, 레이블, 값)가 생성되므로 사용하지 않더라도 좌변 변수가 3개여야 함
## autopct="%.1f%%" : 소수점 한자리까지 표시하도록 설정
my_pie,_,_ = plt.pie(percent_class,radius = 1.2,labels=labels,colors=colors,autopct="%.1f%%")


## setp() : 다양한 조정
plt.setp(my_pie, width=0.6, edgecolor='white')

plt.show()
```



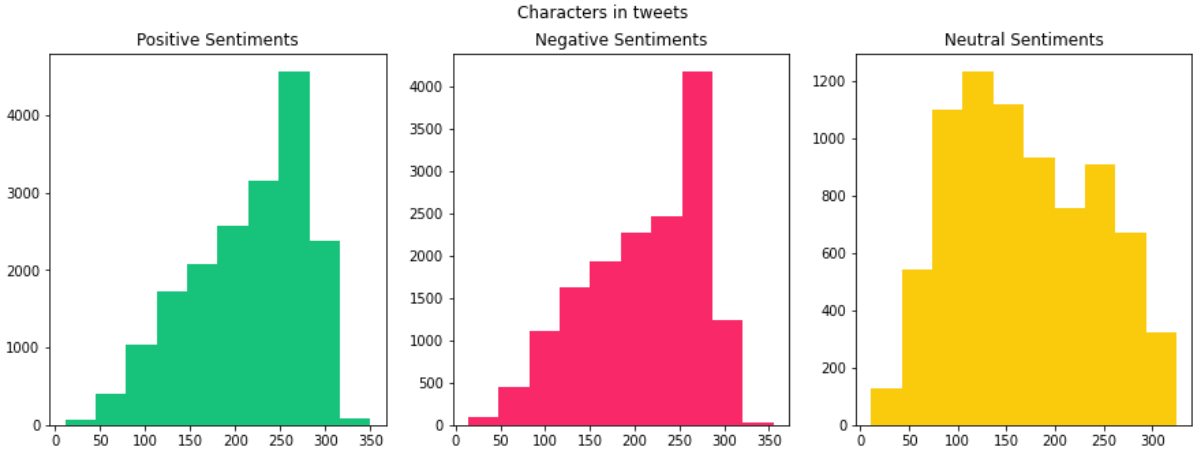

```
## train/test별 그래프 따로 생성
fig=make_subplots(1,2,subplot_titles=('Train set','Test set'))
```

```
## 감정구분별로 자료 개수 세기
x=train.sentiment.value_counts()

fig.add_trace(go.Bar(x=x.index,y=x.values,marker_color=['#17C37B','#F92969','#FACA0C'],name='train'),row=1,col=1)
x=test.sentiment.value_counts()
fig.add_trace(go.Bar(x=x.index,y=x.values,marker_color=['#17C37B','#F92969','#FACA0C'],name='test'),row=1,col=2)
```

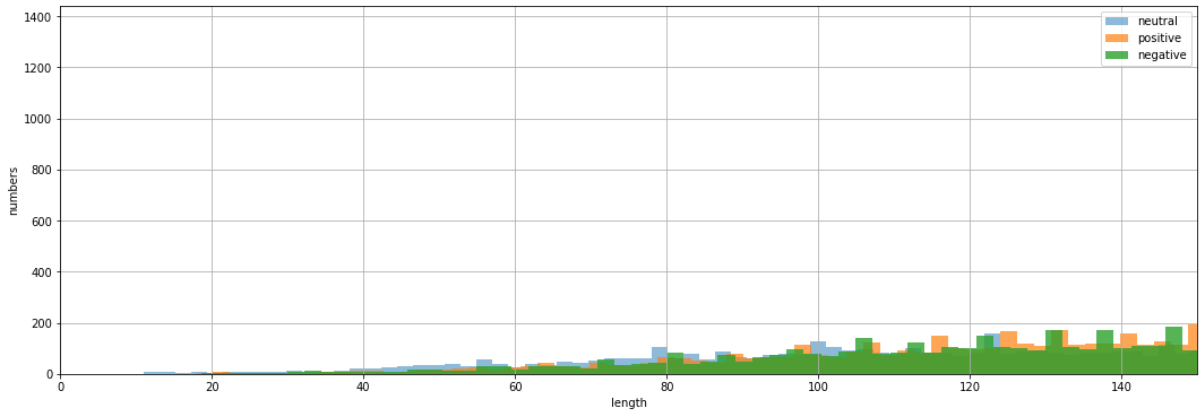


3. 글자 수

```
fig,(ax1,ax2,ax3)=plt.subplots(1,3,figsize=(15,5))

tweet_len=train[train['sentiment']=="positive"]['text'].str.len()
ax1.hist(tweet_len,color='#17C37B')
ax1.set_title('Positive Sentiments')

tweet_len=train[train['sentiment']=="negative"]['text'].str.len()
ax2.hist(tweet_len,color='#F92969')
ax2.set_title('Negative Sentiments')

tweet_len=train[train['sentiment']=="neutral"]['text'].str.len()
ax3.hist(tweet_len,color='#FACA0C')
ax3.set_title('Neutral Sentiments')

fig.suptitle('Characters in tweets')
plt.show()
```

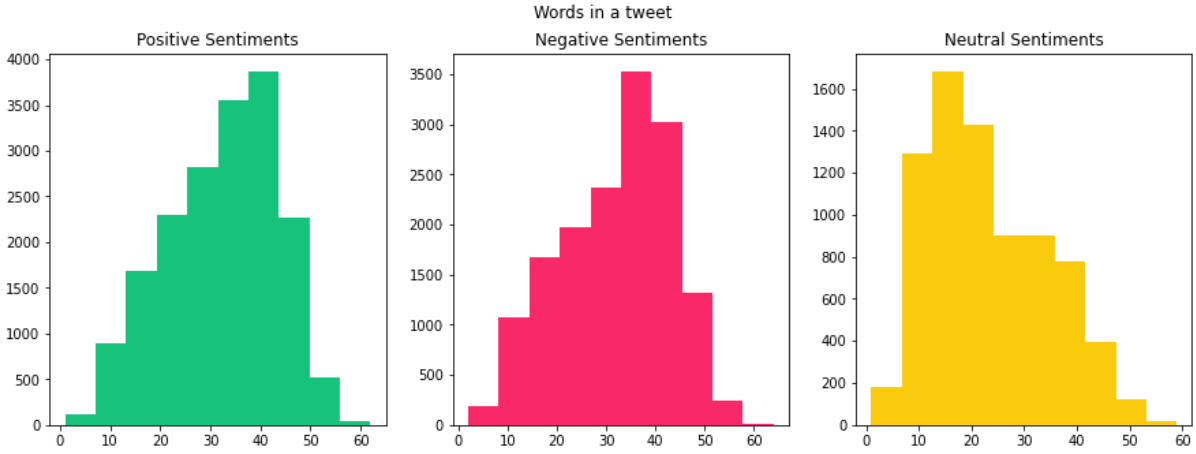



```
def length(text):
    '''a function which returns the length of text'''
    return len(text)
df['length'] = df['text'].apply(length)

plt.rcParams['figure.figsize'] = (18.0, 6.0)
bins = 150
plt.hist(df[df['sentiment'] == "neutral"]['length'], alpha = 0.5,
bins=bins, label='neutral')
plt.hist(df[df['sentiment'] == "positive"]['length'], alpha = 0.7,
bins=bins, label='positive')
plt.hist(df[df['sentiment'] == "negative"]['length'], alpha = 0.8,
bins=bins, label='negative')
plt.xlabel('length')
plt.ylabel('numbers')
plt.legend(loc='upper right')
plt.xlim(0,150)
plt.grid()
plt.show()
```



4. 단어 수

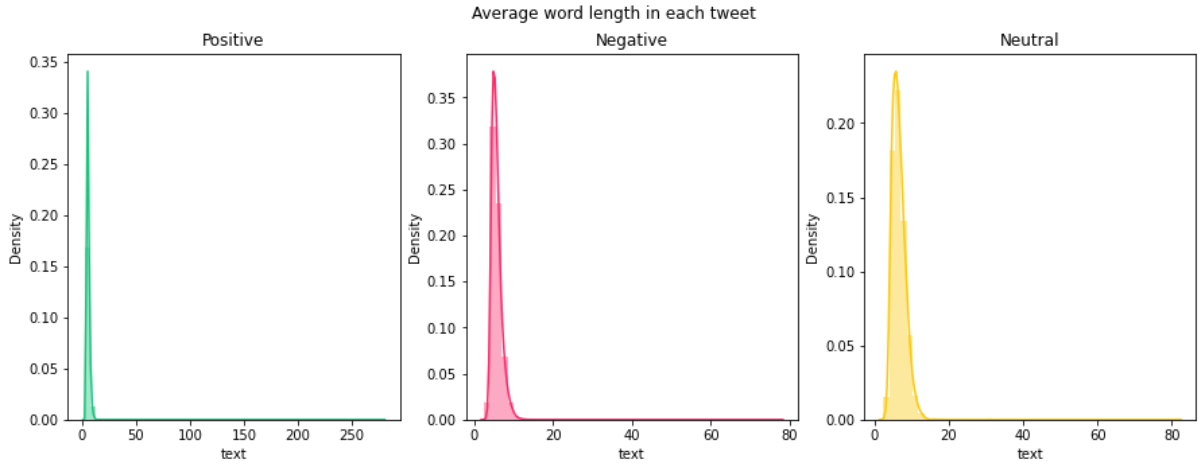
```
fig,(ax1,ax2,ax3)=plt.subplots(1,3,figsize=(15,5))

tweet_len=train[train['sentiment']=="positive"]['text'].str.split().m
ap(lambda x: len(x))
ax1.hist(tweet_len,color='#17C37B')
ax1.set_title('Positive Sentiments')


tweet_len=train[train['sentiment']=="negative"]['text'].str.split().m
ap(lambda x: len(x))
ax2.hist(tweet_len,color='#F92969')
ax2.set_title('Negative Sentiments')

tweet_len=train[train['sentiment']=="neutral"]['text'].str.split().ma
p(lambda x: len(x))
ax3.hist(tweet_len,color='#FACA0C')
ax3.set_title('Neutral Sentiments')

fig.suptitle('Words in a tweet')
plt.show()
```



5. 평균 단어 길이

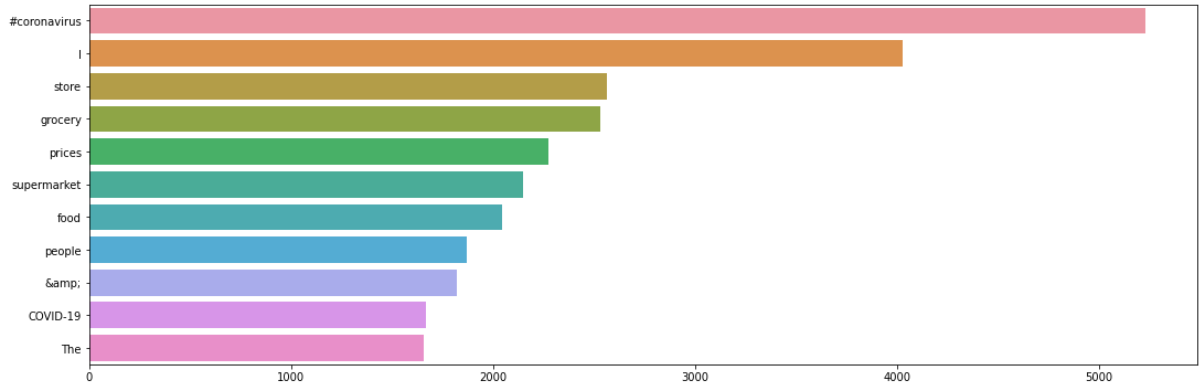
```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt
##  Matplotlib을 기반으로 다양한 색상 테마와 통계용 차트 등의 기능을 추가한 시각화 패키지
import seaborn as sns

fig,(ax1,ax2, ax3)=plt.subplots(1,3,figsize=(15,5))

word=train[train['sentiment']=="positive"]['text'].str.split().apply(lambda x : [len(i) for i in x])
sns.distplot(word.map(lambda x: np.mean(x)),ax=ax1,color='#17C37B')
ax1.set_title('Positive')


word=train[train['sentiment']=="negative"]['text'].str.split().apply(lambda x : [len(i) for i in x])
sns.distplot(word.map(lambda x: np.mean(x)),ax=ax2,color='#F92969')
ax2.set_title('Negative')

word=train[train['sentiment']=="neutral"]['text'].str.split().apply(lambda x : [len(i) for i in x])
sns.distplot(word.map(lambda x: np.mean(x)),ax=ax3,color='#FACA0C')
ax3.set_title('Neutral')
```

```
fig.suptitle('Average word length in each tweet')
```



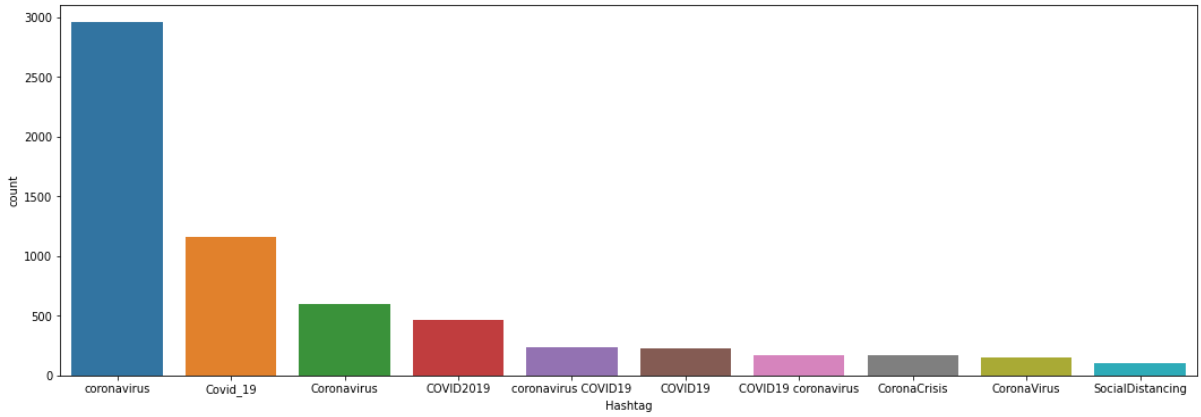
6. 가장 많이 쓰인 단어

```
 from collections import Counter

counter=Counter(corpus)
## 코퍼스 내 최빈값 구하기 (딕셔너리 형태로 반환)
most=counter.most_common()
x=[]
y=[]
for word,count in most[:40]:
    if (word not in stop) :
        x.append(word)
        y.append(count)
sns.barplot(x=y,y=x)
```



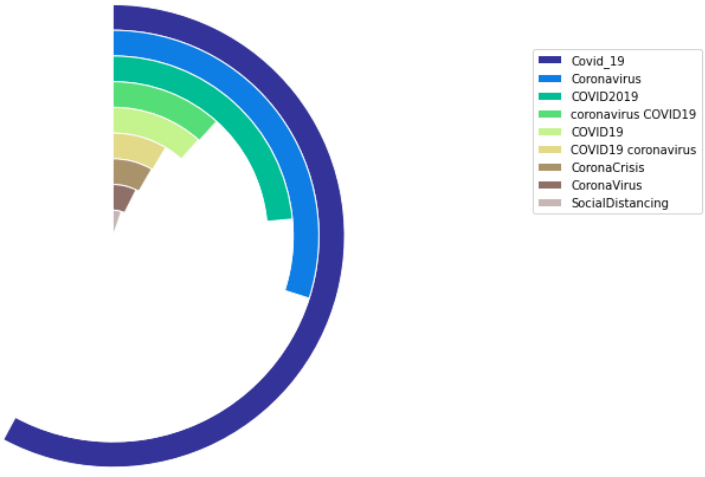
7. 해시태그 분석

```
import re

def find_hash(text):
    line=re.findall(r'(?<=#)\w+',text)
    return " ".join(line)
df['hash']=df['text'].apply(lambda x:find_hash(x))
## 어떤 컬럼/Series의 unique value들을 count
temp=df['hash'].value_counts()[:][1:11]
## 단일 Pandas Series를 Dataframe 으로 변환;
temp=
temp.to_frame().reset_index().rename(columns={'index':'Hashtag','hash
':'count'})
sns.barplot(x="Hashtag",y="count", data = temp)
```




```
from matplotlib import cm
from math import log10

labels = df['hash'].value_counts()[:][2:11].index.tolist()
data = df['hash'].value_counts()[:][2:11]

df['hash'].value_counts()[:][1:11].index.tolist()
#number of data points
n = len(data)
#find max value for full ring
k = 10 ** int(log10(max(data)))
m = k * (1 + max(data) // k)
```

```
#radius of donut chart
r = 1.5
#calculate width of each ring
w = r / n

#create colors along a chosen colormap
colors = [cm.terrain(i / n) for i in range(n)]

#create figure, axis
fig, ax = plt.subplots()
ax.axis("equal")

#create rings of donut chart
for i in range(n):
    #hide labels in segments with textprops: alpha = 0 - transparent,
alpha = 1 - visible
    innerring, _ = ax.pie([m - data[i], data[i]], radius = r - i * w,
startangle = 90, labels = ["", labels[i]], labeldistance = 1 - 1 /
(1.5 * (n - i)), textprops = {"alpha": 0}, colors = ["white",
colors[i]])
    plt.setp(innerring, width = w, edgecolor = "white")

plt.legend()
plt.show()
```

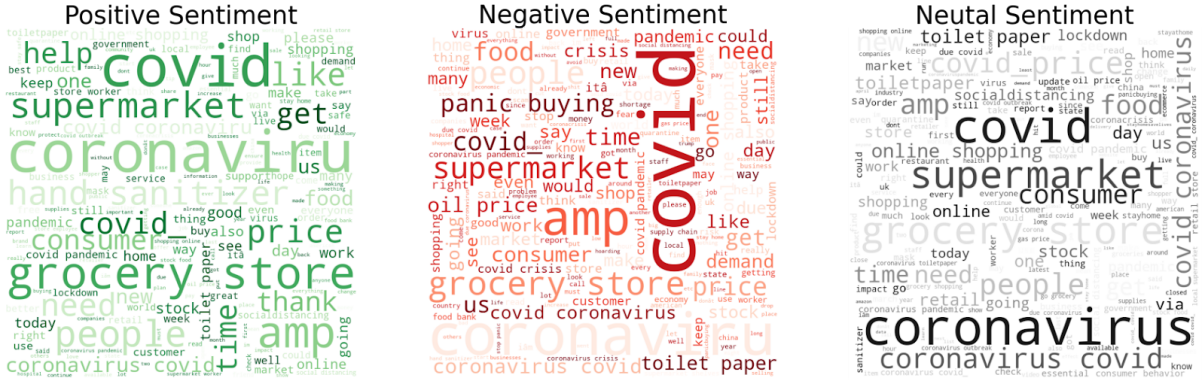



```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=[30, 15])
```

```
df_pos = df[df["sentiment"]=="positive"]
df_neg = df[df["sentiment"]=="negative"]
df_neu = df[df["sentiment"]=="neutral"]

comment_words = ''
stopwords = set(STOPWORDS)

for val in df_pos.text:

    # typecaste each val to string
    val = str(val)

    # split the value
    ## 문자열 구분해서 리스트로 반환
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        ## 대문자를 소문자로
        tokens[i] = tokens[i].lower()
    ## '구분자'.join(리스트) : 리스트에 있는 요소 하나하나를 합쳐서 하나의
문자열로 바꾸어 반환
    comment_words += " ".join(tokens)+" "

## comment_words에 대한 워드클라우드 생성
wordcloud1 = WordCloud(width = 800, height = 800,
                background_color ='white',
                colormap="Greens",
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

ax1.imshow(wordcloud1)
ax1.axis('off')
ax1.set_title('Positive Sentiment',fontsize=35);

comment_words = ''

for val in df_neg.text:

    # typecaste each val to string
    val = str(val)
```

```
    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    comment_words += " ".join(tokens)+" "




wordcloud2 = WordCloud(width = 800, height = 800,
                background_color ='white',
                colormap="Reds",
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)
ax2.imshow(wordcloud2)
ax2.axis('off')
ax2.set_title('Negative Sentiment',fontsize=35);



comment_words = ''
for val in df_neu.text:

    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    comment_words += " ".join(tokens)+" "

wordcloud3 = WordCloud(width = 800, height = 800,
                background_color ='white',
                colormap="Greys",
                stopwords = stopwords,
```

```
                  min_font_size = 10).generate(comment_words)
ax3.imshow(wordcloud3)
ax3.axis('off')
ax3.set_title('Neutal Sentiment',fontsize=35);
```



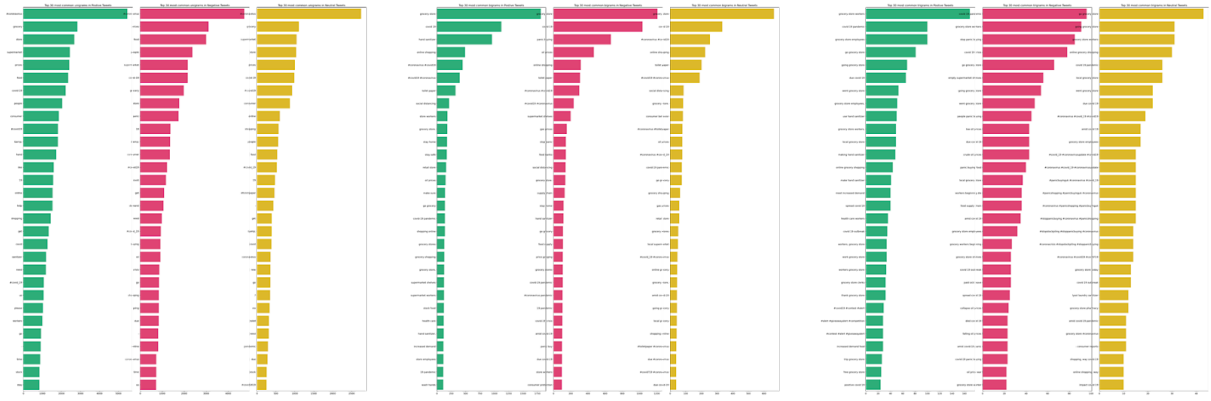
8. N-grams

generate_ngram 함수를 만들어 Uni-gram, bi-gram, tri-gram을 수행하였다.

```
# Define functions
## n_gram의 파라미터를 조정하여 bigram, trigram 등을 수행한다.
def generate_ngrams(text, n_gram=1):
    token = [token for token in text.lower().split(' ') if token !=
'' if token not in STOPWORDS]
    ngrams = zip(*[token[i:] for i in range(n_gram)])
    return [' '.join(ngram) for ngram in ngrams]

N = 30

positive= train["sentiment"]== "positive"
negative= train["sentiment"]== "negative"
neutral= train["sentiment"]== "neutral"

positive_unigrams = defaultdict(int)
neutral_unigrams = defaultdict(int)
negative_unigrams = defaultdict(int)

# Unigrams
for tweet in train[positive]['text']:
    for word in generate_ngrams(tweet):
        positive_unigrams[word] += 1
```

```
for tweet in train[negative]['text']:
    for word in generate_ngrams(tweet):
        negative_unigrams[word] += 1

for tweet in train[neutral]['text']:
    for word in generate_ngrams(tweet):
        neutral_unigrams[word] += 1

df_positive_unigrams = pd.DataFrame(sorted(positive_unigrams.items(),
key=lambda x: x[1])[::-1])
df_negative_unigrams = pd.DataFrame(sorted(negative_unigrams.items(),
key=lambda x: x[1])[::-1])
df_neutral_unigrams = pd.DataFrame(sorted(neutral_unigrams.items(),
key=lambda x: x[1])[::-1])


fig, axes = plt.subplots(ncols=3, figsize=(27, 30), dpi=150)
plt.tight_layout()

sns.barplot(y=df_positive_unigrams[0].values[:N],
x=df_positive_unigrams[1].values[:N], ax=axes[0], color='#17C37B')
sns.barplot(y=df_negative_unigrams[0].values[:N],
x=df_negative_unigrams[1].values[:N], ax=axes[1], color='#F92969')
sns.barplot(y=df_neutral_unigrams[0].values[:N],
x=df_neutral_unigrams[1].values[:N], ax=axes[2], color='#FACA0C')


for i in range(3):
    axes[i].spines['right'].set_visible(False)
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')
    axes[i].tick_params(axis='x', labelsize=13)
    axes[i].tick_params(axis='y', labelsize=13)

axes[0].set_title(f'Top {N} most common unigrams in Postive Tweets',
fontsize=15)
axes[1].set_title(f'Top {N} most common unigrams in Negative Tweets',
fontsize=15)
axes[2].set_title(f'Top {N} most common unigrams in Neutral Tweets',
fontsize=15)
```

```
plt.show()
```



5)추론

트위터 데이터를 사용하여 텍스트 기반 감정 분석을 진행하고, 모델별 성능을 분석하기 위해 머신러닝, 딥러닝 기법을 적용하였다.

1. 머신러닝

총 3가지의 머신러닝 기법을 적용하였다.

랜덤 포레스트는 앙상블 머신러닝 모델로, 다수의 의사결정 트리를 만들고 각 트리의 분류를 집계해서 최종적인 분류를 수행한다. 모든 의사결정 트리는 학습 데이터셋에서 임의의 하위 데이터셋을 추출하여 생성된다. 랜덤 포레스트의 장점으로는 다수의 트리로부터 분류를 집계하기 때문에 오버피팅을 해소하는데 효과적이라는 점이 있다. 랜덤 포레스트의 결정 트리가 많을수록 깔끔한 Decision Boundary가 나오지만 훈련 시간과 메모리의 소비가 증가하기 때문에 적절한 값이 주어져야 하며, 본 연구에서는 n_estimators = 100으로 설정하였다 또한, max_depth = 5로 설정하였다. 이는 트리의 깊이가 깊어지면 오버피팅이 일어날 수 있으므로 적절한 제어가 필요하다. 동일한 예측 결과를 위해 난수를 발생시키지 않고자 random_state = 0으로 설정하였다.

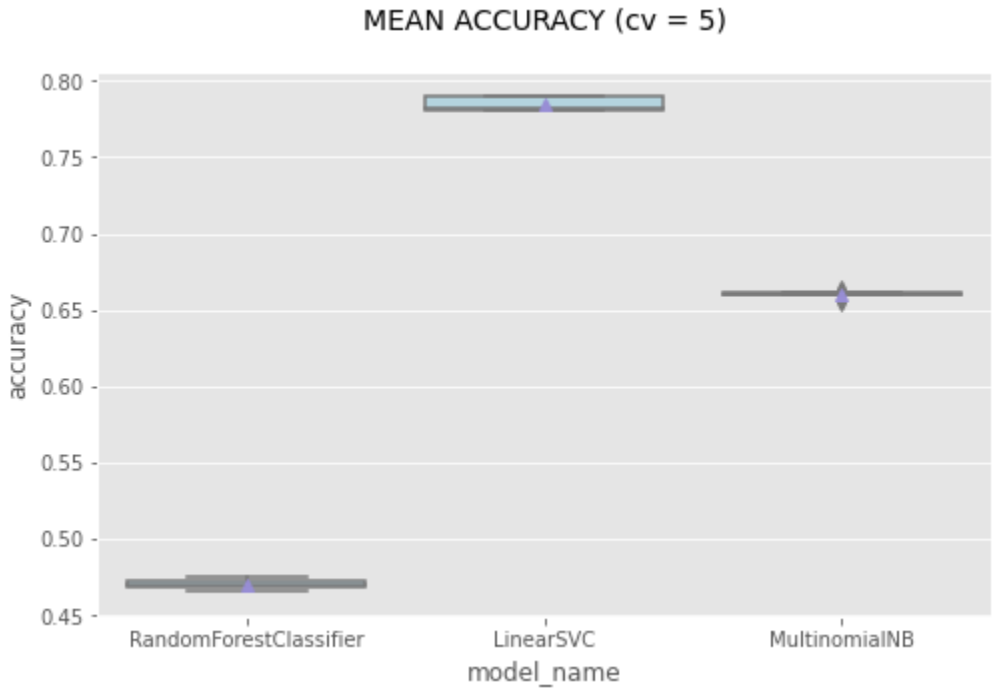
서포트 벡터 머신은 분류, 회귀 및 특이치 검출을 위해 사용되는 감독된 학습 방법으로 마진을 최대로 하는 서포트벡터와 직선을 찾는 것이 목표이다.두 클래스 사이 가장 가까이 있는 점들(support vector)과의 거리가 가장 큰 직선(margin)을 찾는다. LinearSVC는 보통의 SVM 구현과 달리 규제에 편향을 포함시키고 있어서, 스케일링 작업을 수행하지 않는다.

다항분포 나이브 베이즈는 데이터의 특징이 출현횟수로 표현되었을 경우, 데이터의 출현횟수에 따라 값을 달리한 데이터에 사용한다.

```
models = [
    RandomForestClassifier(n_estimators=100, max_depth=5,
random_state=0),
    LinearSVC(),
    MultinomialNB(),
]

# 5 Cross-validation
CV = 5
cv_df = pd.DataFrame(index=range(CV * len(models)))

entries = []
for model in models:
  model_name = model.__class__.__name__
  accuracies = cross_val_score(model, features, labels,
scoring='accuracy', cv=CV)
  for fold_idx, accuracy in enumerate(accuracies):
    entries.append((model_name, fold_idx, accuracy))

cv_df = pd.DataFrame(entries, columns=['model_name', 'fold_idx',
'accuracy'])

mean_accuracy = cv_df.groupby('model_name').accuracy.mean()
std_accuracy = cv_df.groupby('model_name').accuracy.std()

acc = pd.concat([mean_accuracy, std_accuracy], axis= 1,
          ignore_index=True)
acc.columns = ['Mean Accuracy', 'Standard deviation']
acc
```

model_name	Mean Accuracy	Standard deviation
LinearSVC	0.784411	0.004532
MultinomialNB	0.659791	0.003452
RandomForestClassifier	0.470029	0.003683

```
plt.figure(figsize=(8,5))
sns.boxplot(x='model_name', y='accuracy',
            data=cv_df,
            color='lightblue',
            showmeans=True)
plt.title("MEAN ACCURACY (cv = 5)\n", size=14);
```




```
X_train, X_test, y_train, y_test,indices_train,indices_test =
train_test_split(features,

labels,

train.index, test_size=0.10,

random_state=1)
model =   LinearSVC()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Classification report

from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
```

```
from sklearn import metrics

print('\t\t\t\tCLASSIFICATIION METRICS\n')
print(metrics.classification_report(y_test, y_pred,
                                    target_names=
train['label'].unique()))
```

CLASSIFICATIION METRICS

	precision	recall	f1-score	support
1	0.80	0.79	0.80	1554
2	0.69	0.67	0.68	760
0	0.82	0.84	0.83	1802
accuracy			0.79	4116
macro avg	0.77	0.77	0.77	4116
weighted avg	0.79	0.79	0.79	4116

2. 딥러닝(LSTM)

LSTM은 기존의 RNN이 출력과 먼 위치에 있는 정보를 기억할 수 없다는 단점을 보완하여 장/단기 기억을 가능하게 설계한 신경망의 구조를 말한다. 주로 시계열 처리나, 자연어 처리에 사용된다. RNN과 같은 체인 구조로 되어 있지만, 반복 모듈은 단순한 한 개의 tanh layer가 아닌 4개의 layer가 서로 정보를 주고받는 구조로 되어 있다.

```
# Tokenize Texts
max_features = 20000
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(list(X_train))
list_tokenized_train = tokenizer.texts_to_sequences(X_train)
list_tokenized_test = tokenizer.texts_to_sequences(X_test)
```

```
# convert tokenized texts into same padding size
maxlen = 200
embed_size = 128
X_train_final = pad_sequences(list_tokenized_train,
maxlen=maxlen)
X_test_final = pad_sequences(list_tokenized_test, maxlen=maxlen)

# Create Model
inp = Input(shape=(maxlen, ))
x   =  Embedding(max_features, embed_size)(inp)
x   =  LSTM(60, return_sequences=True,name='lstm_layer')
x   =  GlobalMaxPool1D()
x   =  Dense(64, activation="relu")
x   =  Dropout(0.2)
x   =  Dense(3, activation="softmax")

model = Model(inputs=inp, outputs=x)
model.compile(loss='categorical_crossentropy',
             optimizer='adam',
             metrics=['accuracy'])
```

	precision	recall	f1-score	support
0	0.90	0.85	0.87	654
1	0.84	0.84	0.84	263
2	0.88	0.89	0.88	603
micro avg	0.88	0.86	0.87	1520
macro avg	0.87	0.86	0.86	1520
weighted avg	0.88	0.86	0.87	1520
samples avg	0.86	0.86	0.86	1520

3. 딥러닝(GRU)

GRU는 게이트 메커니즘이 적용된 RNN 프레임워크 일종으로 LSTM의 구조를 간단하게 개선한 모델이다.
토큰화 작업 이후 모델 생성의 경우 케라스 Sequential 모델 적용했으며 성능 향상을 위한 단어 주변의 문맥분석을 위한 Bidirectional(양방향) 처리작업을 하였다. confusion matrix로 확인 결과 정확도(acc) 80%가 나왔지만 모델 최적화 및 정확도를 높이기 위한 10번의 에포크를 진행으로 시간이 오래 걸렸으며, 훈련데이터의 정확도와 비교했을때 과대적합(overfitting) 보여 새로운 샘플에 대한 예측에는 정확성이 떨어질 수도 있다고 예측된다.

```
tokenizer.fit_on_texts(x_train)
vocab_length = len(tokenizer.word_index) + 1
x_train = tokenizer.texts_to_sequences(x_train)
x_test = tokenizer.texts_to_sequences(x_test)
x_train = pad_sequences(x_train, maxlen=max_len, padding='post')
x_test = pad_sequences(x_test, maxlen=max_len, padding='post')

model = tf.keras.Sequential([
          tf.keras.layers.Embedding(vocab_length, embedding_dim,
          input_length=max_len),
          tf.keras.layers.Bidirectional(tf.keras.layers.GRU(256,
          return_sequences=True)),
          tf.keras.layers.GlobalAveragePooling1D(),
          tf.keras.layers.Dense(64, activation='relu'),
          tf.keras.layers.Dropout(0.4),
          tf.keras.layers.Dense(3, activation='softmax')

model.compile(loss='categorical_crossentropy',optimizer="adam",m
etrics=['accuracy'])
print(model.summary())
```

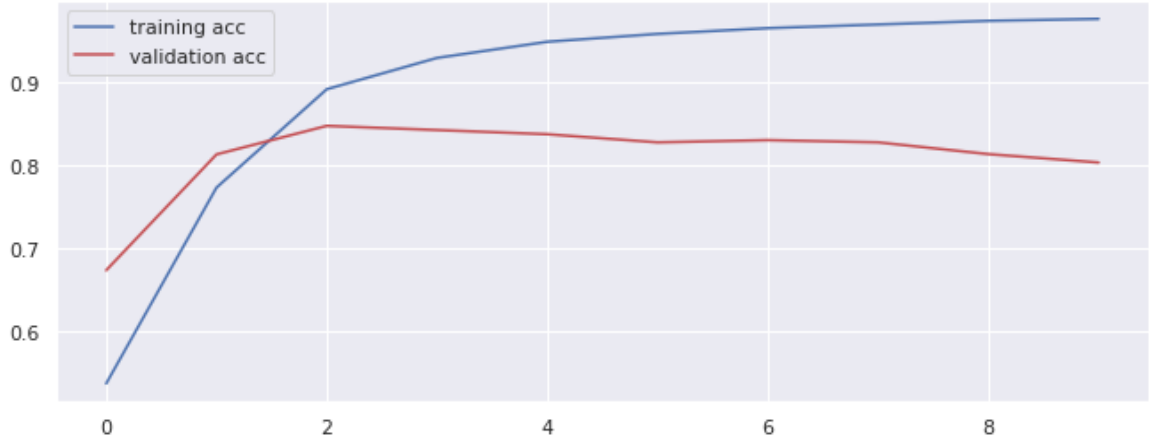
```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_4 (Embedding)      (None, 286, 16)           508144

 bidirectional_4 (Bidirectio  (None, 286, 512)          420864
```

```
 nal)

 global_average_pooling1d_4   (None, 512)              0
 (GlobalAveragePooling1D)

 dense_8 (Dense)              (None, 64)                32832

 dropout_4 (Dropout)          (None, 64)                0

 dense_9 (Dense)              (None, 3)                 195

=================================================================
Total params: 962,035
Trainable params: 962,035
Non-trainable params: 0
_________________________________________________________________
None
```

```
from tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train, 3)
y_test = to_categorical(y_test, 3)
num_epochs = 10
history = model.fit(x_train, y_train, epochs=num_epochs,
                    validation_data=(x_test, y_test))

temp = model.predict(x_test)
pred= np.argmax(temp, axis=1)
print(classification_report(np.argmax(y_test,1),pred))
```

```
              precision    recall  f1-score   support

           0       0.67      0.78      0.72       497
           1       0.85      0.79      0.82      1207
           2       0.83      0.83      0.83      1260

    accuracy                           0.80      2964
   macro avg       0.78      0.80      0.79      2964
weighted avg       0.81      0.80      0.81      2964
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

epochs = range(len(acc))

plt.plot(epochs, acc,'b',label='training acc')
plt.plot(epochs, val_acc, 'r', label='validation acc')
plt.legend()
plt.show()
```



4. BERT

BERT는 특정 과제를 하기 전 사전 훈련 Embedding을 통해 특정 과제의 성능을 더 좋게 할 수 있는 언어모델이다. 이전에는 word2vec, glove, fasttext 방식을 많이 사용하였지만, 요즘은 고성능을 내는 대부분의 모델에서 BERT를 많이 사용하고 있다. 대량 코퍼스로 BERT 언어모델을 적용하고, BERT 언어모델 출력에 추가적인 모델(RNN, CNN 등 머신러닝 모델)을 쌓아 원하는 Task를 수행한다.

(진행중)

6)실험결과

트위터 메세지 분석을 위해 다양한 모델을 사용하여 성능을 비교해보았다.

적용한 4가지 모델(ML, LSTM, GRU, BERT) 실험결과를 정리한다.

모델
[ML] precision recall f1-score support 1 0.80 0.79 0.80 1554 2 0.69 0.67 0.68 760 0 0.82 0.84 0.83 1802 accuracy 0.79 4116 macro avg 0.77 0.77 0.77 4116 weighted avg 0.79 0.79 0.79 4116
[LSTM] precision recall f1-score support 0 0.90 0.85 0.87 654 1 0.84 0.84 0.84 263 2 0.88 0.89 0.88 603 micro avg 0.88 0.86 0.87 1520 macro avg 0.87 0.86 0.86 1520 weighted avg 0.88 0.86 0.87 1520
[GRU] precision recall f1-score support 0 0.67 0.78 0.72 497 1 0.85 0.79 0.82 1207 2 0.83 0.83 0.83 1260 accuracy 0.80 2964 macro avg 0.78 0.80 0.79 2964 weighted avg 0.81 0.80 0.81 2964
[BERT] (진행중)

7)결론

model_name	Mean Accuracy	Standard deviation
LinearSVC	0.784411	0.004532
MultinomialNB	0.659791	0.003452
RandomForestClassifier	0.470029	0.003683

1. LinearSVC (서포트벡터머신) 결과가 0.78%
2. 랜덤포레스트 결과가 기대 이하로 나타남(0.470029)
3. LinearSVC로 Confusion Matrix로 확인 결과, 정확도(acc) 79% 임
4. f1-score(recall과 precision을 조화평균한 값)이 1('Neutral'),0('Negative') 는 0.80, 0.83으로 분류된 반면, 2('Positive')는 0.68로 성능이 상당히 떨어지는 것으로 나타남
5. GRU는 LSTM보다 학습 속도가 빠르다고 알려져있지만 반드시 LSTM 대신 사용하는 것이 좋다고 하지 않는다.(Denny Britz et al, 2017)

=> 실험결과 ML acc 79% LSTM acc 87% GRU acc 80% 이다. ML에서는 서포트벡터머신의 acc가 가장 높았으며 랜덤포레스트 결과가 기대이하로 나왔다. GRU와 LSTM의 둘 간의 성능 차이가 명확히 나타나고 있지 않기에 연구 및 프로젝트에 맞게 성능을 직접 테스트 해보고 알맞은 모델을 사용해야 한다.
LSTM과 GRU는 hidden state 벡터에 모든 단어의 의미를 담아야 하기 때문에 모든 정보를 담기 어렵다. 이 문제를 해결한 Attention 기반 BERT 모델을 실험해볼 예정이다.

8)github 주소

https://github.com/tongrabbit/covidSentiment

9) 참고문헌

- Hur, S. H., and Choi, K. S. (2012). A Study on characteristics and types of tweet in twitter. Hanminjok Emunhak, Vol. 61, pp. 455-494.

- 유승의 .(2018). 인공지능과 자연어 처리 기술 동향, 정보통신기획평가원, www.iitp.kr

- Denny Britz, Anna Goldie, Minh-Thang Luong, Quoc Le .(2017). Massive Exploration of Neural Machine Translation Architectures