

Python Programming and Practice

Put on your song!
: Music Recommendation
System

Proposal

Date : 2023.11.03

Name : Seokyoung Kim

ID : 182207

1. Introduction

1) Background

According to the International Federation of the Phonographic Industry (IFPI)'s Global Music Report 2023, the global music industry generated \$25.9 billion in revenue in 2022, up 18.5% from 2021. These revenues have been reaching new highs every year, with this year's growth being the largest since the 1990s. In addition, the 2021 Music User Survey report released by the Korea Creative Content Agency found that 71.5% of respondents said they have spent more time listening to music since the coronavirus (COVID-19) pandemic. 88.3% of all respondents said they listen to music "at least once a week," with 51.7% saying they listen to music "almost every day. The number of people streaming music has also been on the rise, with 60.5% in 2019, 63.2% in 2020, and 65% in 2021. To compete in this rapidly growing music market, differentiated service technologies are being developed by streaming companies such as Melon, Genie, and YouTube Music.



Figure 1. Global music industry market size trends

2) Project goal

In this project, we aim to build a system that recommends music to users by analyzing existing music data. It also aims to automatically create new playlists based on the user's playlists and generate titles for them.

3) Differences from existing programs

Traditional recommendation systems use other users with similar tastes to the user to recommend songs that the user has listened to. However, these systems simply recommend songs based on what you've listened to recently, rather than what you've listened to in the past, even on the same day, depending on your mood and the time of day. In addition, this requires a large enough number of users to collect data, and this limitation makes it difficult to recommend music from new users or independent artists. Especially with newer music, it can be difficult to recommend it to users before we have enough information to recommend it. In addition, the music recommendation systems currently used by many streaming apps are mainly limited to paid subscribers or song purchasers, and free users have limited access to them.

To solve these problems, this project proposes a system that recommends songs without existing user data based on a systematic analysis of music information. This is expected to reduce the gap in the music industry caused by the imbalance of user data.

2. Functional Requirement

*** The following functions can be changed during implementation.**

1) Load Dataset Function

- Download the dataset for analysis
 - (1) Load dataset – music genre and file

2) Data Preprocessing Function

- Preprocessing data to extract clean features
 - (1) Analyzing singer-sepecific data
 - (2) Analyzing data by gender
 - (3) Analyzing data by genre
 - (4) Analyzing of the frequency of lyrics
 - (5) Top-ranked frequency analysis
 - (6) Analysis of the frequency of songwriting

3) Remove specific words Function

- Remove strange words/english combinations
 - (1) Analyzing singer-sepecific data
 - (2) Delete outlier or incorrect word

4) Tokenize Function

- Tokenize the words in sentences
 - (1) Using tokenize library in tensorflow
 - (2) Tokenize words and divide to 'singer', 'genre', etc

5) Clustering Data Function

- Clustering the data with traditional algorithm
 - (1) K-means Clustering
 - (2) Agglomerative Clustering
 - (3) Analysis the morpheme

6) Main Function

- Set the epoch, save path, load path, samples, data file paths, batch size
 - (1) Download the model (ex. GPT2LMHeadModel, kogpt2)
 - (2) Download vocabulary
 - (3) Call the tokenizer
 - (4) Train the model using dataset
 - (5) Evaluate the trained model
 - (5) Save the results

7) Get a song/singer/playlist input and recommend system Function

- Recommended by receiving the song title and checking if the song is in the database you own.
 - (1) Recall 'doc2vec' model and run similarity
 - (2) Find the cluster to which each song belongs
 - (3) Call the tokenizer
 - (4) Choose randomly selected songs from songs in the cluster
 - (5) If the input is a singer, check the cluster that contains the most songs by the singer and select the randomly selected song
 - (6) Run similarly if the input is a playlist.

3. Progress

* The part where i applied what we learned is marked in **red**

1) Load dataset function

- Download the dataset for analysis

- (1) Load font

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

fe = fm.FontEntry(
    fname=r'./NanumSquareNeo-Variable.ttf', # ttf 파일이 저장되어 있는 경로
    name='NanumGothic') # 이 폰트의 원하는 이름 설정
fm.fontManager.ttflist.insert(0, fe) # Matplotlib에 폰트 추가
plt.rcParams.update({'font.size': 10, 'font.family': 'NanumGothic'}) # 폰트 설정
```

- (2) Load dataset

- read dataset file(csv) using pandas library

```
data = pd.read_csv("1992_2020_kpop.csv", engine='python', encoding='CP949')
train_data = pd.DataFrame(data)
```

(3) Verifying the data

```
train_data.head()
```

	년도	가수	제목	성별	장르	최고순위	작사	작곡	소속사	가사
0	19920000	쥬	난 멈추지 않는다	mixed	댄스	1	조진호	조진호	DSP 미디어	이제 모든걸 다시 시작해 내겐 아직도 시간이 있어 때론 상처가 좌절로 남아 돌아갈수...
1	19920000	쥬	우리 모두 사랑하자	mixed	댄스	7	조진수	조진수	DSP 미디어	워 우리 모두 사랑하자 우리의 젊은날을 위하여 우리 모두 춤을 추자 가벼운 인스텝 ...
2	19920000	쥬	이유	mixed	발라드	0	조진호	조진호	DSP 미디어	어들은 드리워지고 이제는 우리들만의 시간이 지천듯이 내 가슴속에 밀려와 나를 재우고...
3	19920000	쥬	18번가의 비밀	mixed	댄스	0	조진호	조진호	DSP 미디어	끝없는 여름 속을 천천히 걷고 있어 어디서 본 듯한 아련한 느낌이야 거리는 비에 젖...
4	19920000	쥬	요즘 친구들	mixed	댄스	0	조진수	조진호	DSP 미디어	요즘 친구들은 흥 정말 진정한 친구가 뭔지 잘 몰라 말로만 떠돌고 자기들이 다만 그...

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 7235 entries, 0 to 7234  
Data columns (total 10 columns):  
#   Column  Non-Null Count  Dtype    
---  --  
0   년도    7235 non-null    int64    
1   가수    7235 non-null    object   
2   제목    7235 non-null    object   
3   성별    7235 non-null    object   
4   장르    7235 non-null    object   
5   최고순위 7235 non-null    object   
6   작사    7194 non-null    object   
7   작곡    7183 non-null    object   
8   소속사  7026 non-null    object   
9   가사    7233 non-null    object   
dtypes: int64(1), object(9)  
memory usage: 565.4+ KB
```

- Check the data shape and information

```
train_data.shape # 7235개 곡, 10개 속성
```

```
(7235, 10)
```

```
train_data.columns
```

```
Index(['년도', '가수', '제목', '성별', '장르', '최고순위', '작사', '작곡', '소속사', '가사'], dtype='object')
```

```
train_data['소속사'].describe()
```

```
count      7026  
unique      251  
top      (주)SM엔터테인먼트  
freq       716  
Name: 소속사, dtype: object
```

(4) Confirm the not available values in data using 'for' statements

```
# 각 attribute에 따른 N/A값의 수  
for attribute in train_data.columns:  
|     print(train_data[attribute].isnull().sum())  
  
0  
0  
0  
0  
0  
0  
41  
52  
209  
2
```

2) Data Preprocessing Function

- Preprocessing data to extract clean features

(1) Analyzing singer-specific data based on **index**

```
train_data['가수'].describe()

count      7235
unique      455
top      방탄소년단
freq       317
Name: 가수, dtype: object

train_data['소속사'].describe()

count      7026
unique      251
top      (주)SM엔터테인먼트
freq       716
Name: 소속사, dtype: object
```

```
#groupby 가수: 빈도수 계산
singer_count=train_data.groupby('가수',sort = False).count().rename(columns = {'년도': 'count'})
singer_count=singer_count.reset_index().rename(columns={"index": "singer"})
singer_count=singer_count[['가수','count']]

topsinger=singer_count.sort_values('count',ascending = False).head(50)
topsinger.head(10)
```

	가수	count
267	방탄소년단	317
63	동방신기	280
334	TWICE (트와이스)	196
319	몬스타엑스	176
72	슈퍼주니어	155
47	오렌지	143
296	Red Velvet (레드벨벳)	141
291	마마무 (Mamamoo)	139
34	쥬얼리	134
74	빅뱅	134

```
#groupby 소속사: 빈도수 계산
E_count=train_data.groupby('소속사',sort = False).count().rename(columns = {'년도': 'count'})
E_count=E_count.reset_index().rename(columns={"index": "소속사"})
E_count=E_count[['소속사','count']]

topE=E_count.sort_values('count',ascending = False).head(50)

topE.head(10)
```

	소속사	count
43	(주)SM엔터테인먼트	716
179	JYP 엔터테인먼트	401
121	빅히트 엔터테인먼트	290
0	DSP 미디어	212
5	SM 엔터테인먼트	211
77	스타쉽 엔터테인먼트	201
50	(주)YG엔터테인먼트	171
39	YG 엔터테인먼트	148
25	스타제국	147
36	오렌지엔터테인먼트	143

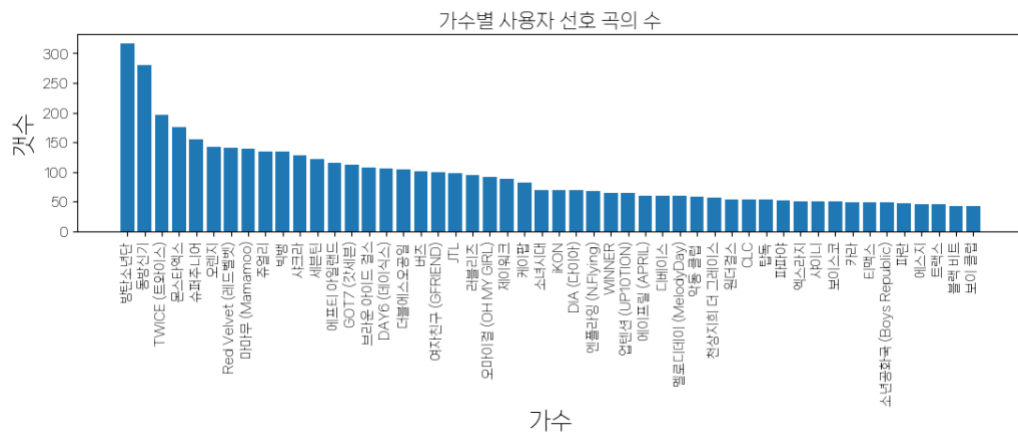
- Visualization using matplotlib library and **index**

```
#시각화
index = np.arange(topsinger.shape[0])
x_name = topsinger['가수'].tolist()
plt.figure(figsize=(13,6))
plt.subplot(211)
plt.bar(index, topsinger['count'])
plt.title('가수별 사용자 선호 곡의 수', fontsize=15)
plt.xlabel('가수', fontsize=18)
plt.ylabel('갯수', fontsize=18)
plt.xticks(index, x_name, rotation= 90)
plt.rc('font', family='NanumGothic')

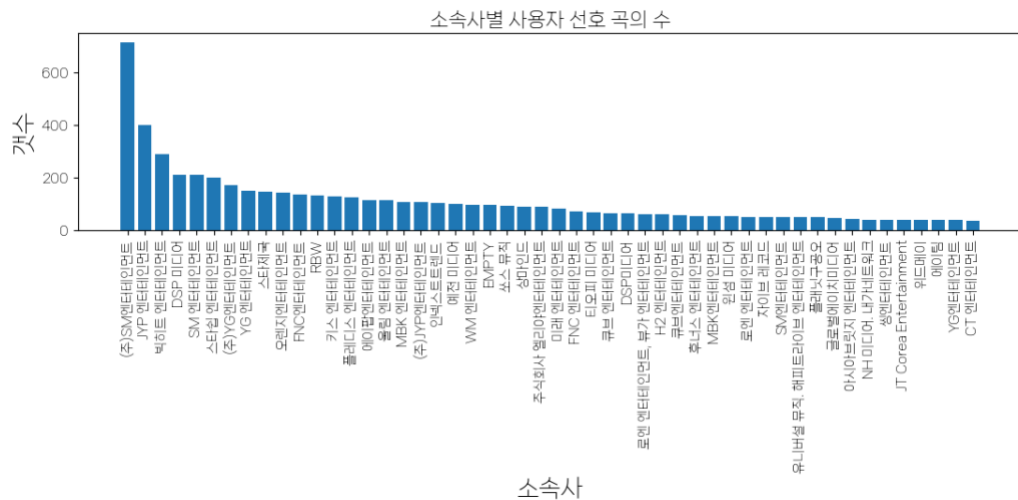
index = np.arange(topE.shape[0])
x_name = topE['소속사'].tolist()
plt.figure(figsize=(13,6))
plt.subplot(212)
plt.bar(index, topE['count'])
plt.title('소속사별 사용자 선호 곡의 수', fontsize=15)
plt.xlabel('소속사', fontsize=18)
plt.ylabel('갯수', fontsize=18)
plt.xticks(index, x_name, rotation= 90)
plt.rc('font', family='NanumGothic')

plt.show()
```

- Number of user favorite songs by singer



- Number of user favorite songs by agency



(2) Analyzing data by gender

- check the data of gender based on **index**

```
train_data['성별'].describe()

count    7235
unique     15
top      male
freq     2543
Name: 성별, dtype: object
```

- group by gender using **count** function

```
#groupby 성별
gender_count=train_data.groupby('성별',sort = True).count()

train_data.loc[train_data["성별"] == "MALE", "성별"] = "male"
train_data.loc[train_data["성별"] == "Male", "성별"] = "male"
train_data.loc[train_data["성별"] == "male", "성별"] = "male"
train_data.loc[train_data["성별"] == "남", "성별"] = "male"

train_data.loc[train_data["성별"] == "Famale", "성별"] = "female"
train_data.loc[train_data["성별"] == "Female", "성별"] = "female"
train_data.loc[train_data["성별"] == "female", "성별"] = "female"
train_data.loc[train_data["성별"] == "여", "성별"] = "female"

train_data.loc[train_data["성별"] == "Mixed", "성별"] = "mixed"
train_data.loc[train_data["성별"] == "Mixed voices", "성별"] = "mixed"
train_data.loc[train_data["성별"] == "all", "성별"] = "mixed"
train_data.loc[train_data["성별"] == "mixed voices", "성별"] = "mixed"
train_data.loc[train_data["성별"] == "mix", "성별"] = "mixed"
train_data.loc[train_data["성별"] == "mixed", "성별"] = "mixed"

gender_count=train_data.groupby('성별',sort = True).count()
gender_count #남자 가수 4366, 여자가수 2786, 혼성 83
```

	년도	가수	제목	장르	최고순위	작사	작곡	소속사	가사
성별									
female	1	1	1	1	1	1	1	1	1
female	2785	2785	2785	2785	2785	2782	2782	2674	2785
male	4366	4366	4366	4366	4366	4331	4320	4268	4364
mixed	83	83	83	83	83	80	80	83	83

(3) Analyzing data by genre

- check the data of genre

```
train_data['장르'].describe()

count    7235
unique    54
top      댄스
freq     3419
Name: 장르, dtype: object
```

- ```
#장르
gnr_count=train_data.groupby('장르').count().rename(columns = {'년도': 'count'})
gnr_count.loc[:, '비율(%)'] = round(gnr_count['count']/sum(gnr_count['count'])*100,2)
gnr_count=gnr_count[['count', '비율(%)']]
gnr_count=gnr_count.reset_index().rename(columns={"index": "gnr"})
gnr_list=gnr_count.sort_values('count',ascending = False)

gnr_count.sort_values('count',ascending=False).head(10)
```
- |    | 장르         | count | 비율(%) |
|----|------------|-------|-------|
| 12 | 댄스         | 3419  | 47.26 |
| 27 | 발라드        | 1458  | 20.15 |
| 21 | 랩/힙합       | 692   | 9.56  |
| 24 | 록/메탈       | 517   | 7.15  |
| 2  | J-POP      | 280   | 3.87  |
| 4  | R&B/Soul   | 277   | 3.83  |
| 32 | 발라드, 국내드라마 | 111   | 1.53  |
| 3  | POP        | 69    | 0.95  |
| 16 | 댄스, 국내드라마  | 49    | 0.68  |
| 18 | 댄스, 랩/힙합   | 41    | 0.57  |

- ```
index = np.arange(gnr_list.shape[0])
x_name = gnr_list['장르'].tolist()

plt.figure(figsize=(13,6))
plt.bar(index, gnr_list['count'])
plt.title('장르별 사용자 선호 곡의 수', fontsize=20)
plt.xlabel('장르', fontsize=18)
plt.ylabel('갯수', fontsize=18)
plt.xticks(index, x_name, rotation=90)
plt.rc('font', family='NanumGothic')

plt.show()
```

- [illegible]

(4) Analyzing of the frequency of lyrics

- Check out the song without lyrics based on `isnull` and `sum` function : 2

```
print(train_data['가사'].isnull().sum()) #2 가사 NULL
```

2

- Check the overlapping lyrics

```
# 중복된 가사 확인
train_data.shape[0] - train_data['가사'].nunique()
if (train_data.duplicated('가사')).any == True:
    print("중복된 가사 확인")
```

- Calling morpheme analyzer (Okt) and save the lyrics in variable 'okja' list based on 'for' statements and 'append' function

```
#객체 호출
okt = Okt()

# 변수 okja에 전체가사 다시저장
words=train_data['가사']
okja = []
for line in words:
    okja.append(line)
```

- Separating morpheme by sentence using 'for' statements, 'if' statements, 'append' function, 'len' function, and try-except

```
# 각 문장별로 형태소 구분하기
sentences_tag = []
i=0
for sentence in okja:
    try:
        morph = okt.pos(sentence)
        sentences_tag.append(morph)
        if i % 1000 == 0:
            print("[{} / {}] done".format(i, len(okja)))
            i += 1
    except:
        print("NULL")
print(len(sentences_tag))
```

[0/7235]done
[1000/7235]done
[2000/7235]done
[3000/7235]done
[4000/7235]done
[5000/7235]done
[6000/7235]done
NULL
NULL
[7000/7235]done
7233

- Select and list only parts of speech that are nouns or adjectives and remove one word length based on `list`, `'for'` statements, `'pop'`, `'append'` function

```
# 명사 혹은 형용사인 품사만 선별해 리스트에 담기 ### (이후 다른 품사에도 적용 가능)
noun_adj_list = []
for sentence1 in sentences_tag:
    for word, tag in sentence1:
        if tag in ['Noun', 'Adjective']:
            noun_adj_list.append(word)

# 단어의 길이가 하나인거 제거
for i, v in enumerate(noun_adj_list):
    if len(v) < 2:
        noun_adj_list.pop(i)
```

- Calculate the frequency of each selected part of the speech & output top 10 frequencies

```
# 선별된 품사별 빈도수 계산 - 6·상위·빈도·10위·출력
counts = Counter(noun_adj_list)
print(counts.most_common(10))

[('사람', 16281), ('너', 8689), ('그대', 7764), ('내', 7072), ('우리', 5778), ('나', 5713), ('이제', 4504), ('없어', 4310), ('다시', 4159), ('맘', 3974)]
```

- Visualize with WordCloud

```
#WordCloud 사용하여 시각화
wc = WordCloud(font_path='./NanumSquareNeo-Variable.ttf', background_color='white', width=800, height=600)
print(dict(counts))
cloud = wc.generate_from_frequencies(dict(counts))
plt.figure(figsize=(10, 8))
plt.axis('off')
plt.imshow(cloud)
plt.show()
```



(5) Top-ranked frequency analysis

- Check the Top-ranked frequency and confirm null values using **sum** function

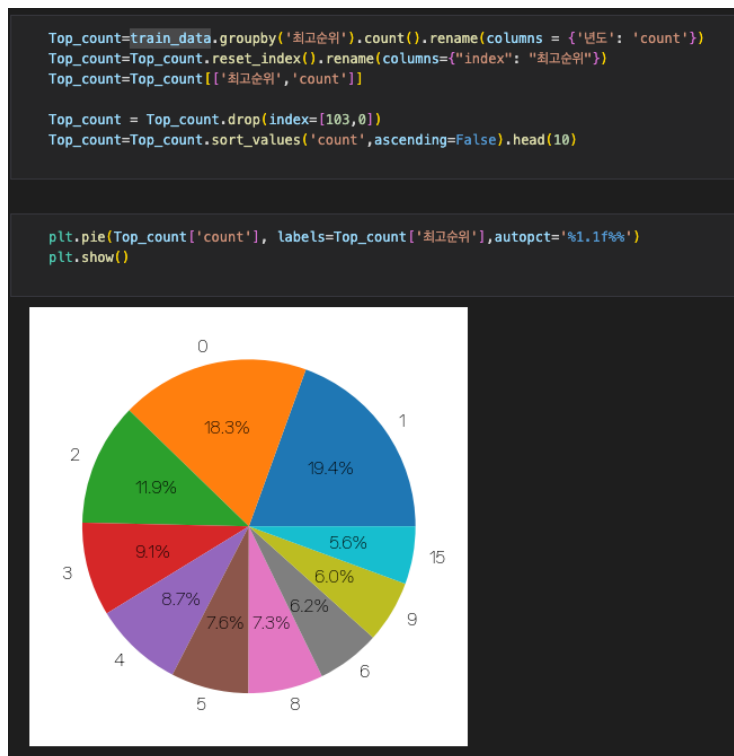
```
train_data['최고순위'].describe()

count      7235
unique      104
top         EMPTY
freq       3128
Name: 최고순위, dtype: object

train_data['최고순위'].isnull().sum()

0
```

- Top-ranked frequency visualization



(6) Analysis of the frequency of songwriting

- Check the songwriting, songcomposing

```
print(train_data['작사'].describe(), "\n=====")
print(train_data['작곡'].describe())

count      7194
unique     2186
top         EMPTY
freq      1246
Name: 작사, dtype: object
=====
count      7183
unique     2476
top         EMPTY
freq      1148
Name: 작곡, dtype: object
```

- Group by songwriter and count the values using **count** function

```
ma_count=train_data.groupby('작사').count().rename(columns = {'년도': 'count'})
ma_count=ma_count.reset_index().rename(columns={"index": "작사"})
ma_count=ma_count[['작사', 'count']]
ma_count.sort_values('count',ascending=False).head(10)

mu_count=train_data.groupby('작곡').count().rename(columns = {'년도': 'count'})
mu_count=mu_count.reset_index().rename(columns={"index": "작곡"})
mu_count=mu_count[['작곡', 'count']]

ma_count.sort_values('count',ascending=False).head(10)
```

	작사	count
173	EMPTY	1246
1243	박진영	79
197	G-DRAGON	72
2121	한성호	67
708	Young K	61
918	김이나	58
1631	유영진	53
1368	서지음	51
905	김영아	50
322	KENZIE	49

```
mu_count.sort_values('count',ascending=False).head(10)
```

	작곡	count
292	EMPTY	1148
1599	박진영	61
1882	윤갑환 황제	57
1937	유영진	49
1295	김승수	45
2120	이철도	38
1541	박근태	37
2006	이민수	36
2401	한상원	34
2458	황성제	34

3) Remove specific words(Preprocessing) Function

- In this function refines the data and preprocessing process

(1) Analyzing singer-sepecific data

- After storing data in train_data, check the data information with the head function, '**len**' function, '**for**' statements, **index**

```
train_data = pd.DataFrame(data)

train_data.head(3) # 데이터 구성요소
```

	년도	가수	제목	성별	장르	최고순위	작사	작곡	소속사	가사
0	19920000	켄	난 멈추지 않는다	mixed	댄스	1	조진호	조진호	DSP 미디어	이제 모둔걸 다시 시작해 내겐 아직도 시간이 있어 때론 상처가 좌절로 남아 돌아킬수...
1	19920000	켄	우리 모두 사랑하자	mixed	댄스	7	조진수	조진수	DSP 미디어	워 우리 모두 사랑하자 우리의 젊은날을 위하여 우리 모두 손을 추자 가벼운 인스텔...
2	19920000	켄	이유	mixed	발라드	0	조진호	조진호	DSP 미디어	어들은 드리워지고 이제는 우리들만의 시간이 지친듯이 내 가슴속에 밀려와 나를 재우고...

```

train_data.shape[0] # 곡 수

5619

len(train_data['가수'].unique()) # 유니크한 가수의 수

454

train_data.columns # 특성 목록

Index(['년도', '가수', '제목', '성별', '장르', '최고순위', '작사', '작곡', '소속사', '가사'], dtype='object')

```

- Check the number of N/A values for each attribute and delete them using 'for' statements, 'sum' function

```

# 각 attribute에 따른 N/A값의 수
for attribute in train_data.columns:
    print(train_data[attribute].isnull().sum())

# 아래 결과에서 소속사가 없는 가수에 대한 정보가 다수 포함되어 있음을 알 수 있음.
# 클러스터링 단계에서는 가사만 사용되므로, 가사가 N/A값인 데이터만 삭제.

0
0
0
0
0
0
0
41
52
99
1

train_data = train_data[train_data['가사'].notnull()]

train_data.head(3)

```

	년도	가수	제목	성별	장르	최고순위	작사	작곡	소속사	가사
0	19920000	젬	난 멈추지 않는다	mixed	댄스	1	조진호	조진호	DSP 미디어	이제 모든걸 다시 시작해 내겐 아직도 시간이 있어 때론 상처가 최절로 남아 돌아갈수...
1	19920000	젬	우리 모두 사랑하자	mixed	댄스	7	조진수	조진수	DSP 미디어	워 우리 모두 사랑하자 우리의 결혼날을 위하여 우리 모두 춤을 추자 가벼운 인스텝 ...
2	19920000	젬	이유	mixed	발라드	0	조진호	조진호	DSP 미디어	어들은 드리워지고 이제는 우리들만의 시간이 지천듯이 내 가슴속에 밀려와 나를 채우고...

```

print(train_data['가사'].isnull().sum())

0

```

(2) Delete outlier or incorrect word

- Remove duplicate lyrics

```

# 중복된 가사 제거
train_data.shape[0] - train_data['가사'].nunique()
train_data.drop_duplicates(subset=['가사'], inplace=True)

```


- Check the total number of samples

```
print('총 샘플의 수 :', len(train_data))
```

총 샘플의 수 : 5618

4) Tokenize Function

- Tokenize the words in sentences

- (1) Specify an unspecified term and calling the Okt object. and then check the train data using **index**

```
# 불용어 지정
stopwords = ['의', '가', '이', '은', '들', '는', '좀', '잘', '강', '과', '도', '를', '으로', '자', '에', '와', '한', '하다']

okt = Okt()

train_data['가사'][0]
```

'이제 모든걸 다시 시작해 내겐 아직도 시간이 있어 때론 상처가 좌절로 남아 돌이킬수 없는 후회도 하고 그러나 우리 잊어선 안돼 지금의 나는 내가

- (2) Use Okt, classify words by morphemes, and perform some level of normalization with the **'for'** statements, **'append'** function

```
# Okt를 사용, 형태소 단위로 단어 분류, 일정 수준의 정규화 실행
X_train = []
for sentence in train_data['가사']:
    temp_X = []
    temp_X = okt.morphs(sentence, stem=True) # 토큰화
    temp_X = [word for word in temp_X if not word in stopwords] # 불용어 제거
    X_train.append(temp_X)
```

- (3) Save the tokenized X_test as a .npz file

```
# 토큰화한 X_test를 저장

X_train_narray = np.array(X_train, dtype='object')
np.save('./X_train', X_train_narray)
```

```
X_train = np.load('./X_train.npz', allow_pickle=True).tolist()
```

- (4) Doc2vec based weight training code is implemented, but I will write down in function 7

5) Clustering Data Function

- Clustering the data with traditional algorithm

(1) K-means Clustering

- Two different tokenizers are used on '가수' and '장르' columns of the train_data_all DataFrame. The fit_on_texts method is used to update the internal vocabulary based on the given texts, and texts_to_sequences converts each text to a sequence of integers
- A new list X2 is created to build input sequences. Using 'for' statements entry in the tokenized '장르' (X_genre) and '가수' (X_singer) lists, the code appends the first element of each to a temporary list (temp). This forms pairs of integers representing the genre and singer for each entry
- The list X2 is converted into a NumPy array named X3. This array could be used as input data for clustering algorithms.

```
# 클러스터링 전처리
import copy
from gensim.models import Doc2Vec
from keras.preprocessing.text import Tokenizer
import numpy as np

# Assuming you have already trained a Doc2Vec model and stored it in the variable 'model'
# If not, you need to train a Doc2Vec model before using it.

tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_data_all['가수'])
X_singer = tokenizer.texts_to_sequences(train_data_all['가수'])

tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_data_all['장르'])
X_genre = tokenizer.texts_to_sequences(train_data_all['장르'])

# Assuming 'model' is your trained Doc2Vec model
X2 = []
for n, x_genre in enumerate(X_genre):
    temp = []
    temp.append(x_genre[0])
    temp.append(X_singer[n][0])
    X2.append(temp)

X3 = np.array(X2)
```

- Perform K-means clustering based on 8 clusters (using **dictionary** and **append function, for statements**)

```
print("K-Means Clustering")

M_KMeans = KMeans(n_clusters=8, random_state=0)
X = X3 # document vector 전체를 가져옴.
M_KMeans.fit(X)# fitting

'''
M_KMeans = KMeans(n_clusters=8, random_state=0)
X = model.docvecs.vectors_docs # document vector 전체를 가져옴.
M_KMeans.fit(X)# fitting
'''

cluster_dict = {i:[] for i in range(0,8)}
for text_tags, label in zip(common_texts_and_tags, M_KMeans.labels_):
    text, tags = text_tags
    cluster_dict[label].append([tags, text])

'''
for label, lst in cluster_dict.items():
    print(f"Cluster {label}")
    for x in lst:
        print(x)
'''
```

- Create an empty lyrics list, turn cluster_dict into a **for statement**, and for each title, singer, **append** the lyrics that correspond to the cluster

```
gasas = []
cluster_n = []
for clster in range(len(cluster_dict)):
    #print('cluster ' + str(cluster))
    cluster_n_song = []
    gasa = []
    for song in cluster_dict[clster]:
        temp1 = list(train_data_all['제목'][(train_data_all['제목']==song[0])[0]].index)
        temp2 = list(train_data_all['가수'][(train_data_all['가수']==song[0][1]).index])

        for title in temp1:
            x = -1
            if title in temp2:
                x = title
                break

        #print(train_data_all['제목'][x])
        #print(train_data_all['장르'][x])
        #print(train_data_all['가수'][x])
        gasa.append(train_data_all['가사'][x])
        cluster_n_song.append([train_data_all['장르'][x],train_data_all['가수'][x],train_data_all['소속사'][x],int(train_data_all['년도'][x]/10000),train_data_all['작곡'][x]])
    cluster_n.append(cluster_n_song)
    gasas.append(gasa)
```

(2) Agglomerative Clustering

- Perform Agglomerative clustering based on 10 clusters

```
print("Agglomerative Clustering")

n_clusters = 10

M_Agglo = AgglomerativeClustering(linkage='ward',
                                   connectivity=None, n_clusters=n_clusters)

# Assuming 'model' is my trained Doc2Vec model
X = model.docvecs.vectors_docs

result = M_Agglo.fit_predict(X)
```

- Populating a list of lists (cluster_dict1), where each sublist corresponds to a cluster identified by a clustering algorithm. After the 'for' loop completes, cluster_dict1 is a list where each element (sublist) corresponds to a cluster, and each sublist contains a list of tags and texts belonging to that cluster. The structure is essentially a way to organize the data by clusters

```
cluster_dict1 = []
for i in range(n_clusters):
    cluster_dict1.append([])

for n, i in enumerate(result):
    text, tags = common_texts_and_tags[n]
    cluster_dict1[i].append([tags, text])
```

- Create an empty lyrics list, turn cluster_dict into a for statement, and for each title, singer, append the lyrics that correspond to the cluster (same as K-means clustering)

```
gasas = []
cluster_n = []
for cluster in range(len(cluster_dict1)):
    #print('cluster ' + str(cluster))
    cluster_n_song = []
    gasa = []
    for song in cluster_dict1[cluster]:
        temp1 = list(train_data_all['제목'] == song[0][0]).index()
        temp2 = list(train_data_all['가수'] == song[0][1]).index()

        for title in temp1:
            x = -1
            if title in temp2:
                x = title
                break

        #print(train_data_all['제목'][x])
        #print(train_data_all['장르'][x])
        #print(train_data_all['가수'][x])
        gasa.append(train_data_all['가사'][x])
        cluster_n_song.append([train_data_all['장르'][x], train_data_all['가수'][x], train_data_all['소속사'][x], int(train_data_all['년도'][x]/10000), train_data_all['작곡'][x]])
    cluster_n.append(cluster_n_song)
    gasas.append(gasa)
```

(3) Analysis the morpheme and clustering data will be covered in function 7

4. Changes in Comparison to the Plan

- None

5. Schedule

Task	11/3	11/10	11/17	11/24
Proposal	----- >			
Function 1		----- >		
Function 2,3			----- >	
Function 4,5				----- >
Task	12/1	12/8	12/15	-
Function 6	----->			
Function 7	----->			
Fix bugs		----- >		
Retrospect			- ----- >	