텍스트 마이닝

Term Project

경제금융학부 2018025205 김예은

융합전자공학부 2018006171 김경호

목차

(5) Word Cloud

(6) 세 필드 간 상관 관계

(7) EDA 해석

5. **분석 방안**

6. **데이터 분석 결과**

7. **결론**

(1) 분석 결과의 요약

(2) 주요 시사점

(3) 분석 결과의 개선 방안

Term Project

1. 서론

(1) 분석 목적

게임 이름을 입력하면 그 게임에서 얻을 수 있는 텍스트 데이터를 기반으로 해서 게임을 추천해 주는 모델을 만드는 것을 목표로 한다.

게임 리뷰를 텍스트 마이닝을 통해서 분석하고 얻을 수 있는 인사이트와 추천 모델 구현을 동시에 하는 것을 프로젝트의 목적으로 삼는다.

(2) 분석 배경

조선 일보 '스팀, AI로 '숨은 보석' 게임 찾아 권하는 새 추천 시스템 도입'에 따르면 기존의 게임 추천 방식은 태그나 리뷰 점수 등을 이용한 것들이 대부분이며 고객의 의사 파악보다는 게임 흥행을 기반으로 한 추천 알고리즘이 대부분이었다고 한다. 하지만 이 방식은 재빠르게 변화하는 게임 시장에는 뒤쳐진 알고리즘이기에, 스팀에서 기계 학습(머신러닝:Machine Learning) 데이터를 바탕으로 고객에 게임을 권하는 '상호작용 추천(Interactive Recommender)' 시스템을 공개했다.

게임 시장에 확장됨에 따라 새로운 추천 알고리즘 기술을 도입하는 게 플랫폼을 운영하는 입장에서 큰 관건이 되었다. 본 추천 알고리즘을 MSA 아키텍처로 모듈화하여 API서버로 Deploy한다면, 효율적으로 확장가능한 새로운 게임 플랫폼을 운영할 수 있게 된다.

분석 배경 출처: https://it.chosun.com/site/data/html_dir/2019/07/12/2019071201130.html

(3) 분석의 필요성

한국콘텐진흥원의 2020년 기준 콘텐츠 사업 보고 결과에 따르면 게임 시장은 타 분야와 비교했을 때 최근 5년 간 가장 엄청난 성장세를 보이는 사업이다. 사회적 거리두기의 영향과 COVID-19의 펜데믹으로 온라인 기반의 디지털 게임이 새로운 커뮤니케이션의 일종으로 자리잡았고 덕분에 게임에 대한 부정적인 시선도 긍정적으로 바뀌는 거에 한몫을 차지했다.

그만큼 게임 시장은 지금까지와 비교되지 않도록 빠른 속도로 성장했다. 트렌드에 맞춰 게임 회사도 다양해지고, 게임 수도 증가했다. 소비자들이 선택할 수 있는 게임의 폭이 넓어진 것이다. 이 말을 반대로 해석하자면 소비자들의 다양한 게임을 하기 원하는 욕구는 증가했지만 선택 범위가 넓어짐에 따라 선택에 어려움을 겪을 수 있게 되는 것이다. 이렇기에 게임 플랫폼에서 제공하는 게임 추천 서비스는 중요한 요소로 다가올 수밖에 없다.

이번 Term Project에서는 세계 최대 규모의 전자 소프트웨어 유통망인 Steam이라는 플랫폼을 선정하여 그들이 제공하는 게임 추천 알고리즘에서 발전해서 더 좋은 알고리즘으로 이용자들의 편의를 충족했으면 하는 마음에 분석을 시작하게 되었다. 본 프로젝트의 사이트가 외국 사이트라는 점과 한국어 리뷰보다는 외국어 리뷰 수가 많다는 점을 기반으로 하여 영어로 된 리뷰를 분석하고자 했다.

(4) 분석을 통하여 예상되는 결과 또는 이점

스팀에서 현재 제공하고 있는 추천 서비스는 이용자가 구매한 게임을 바탕으로 단순히 비슷한 분야 추천에 그친다. 혹은 이용자가 구매한 게임이 엄청나게 많은 경우 최근에 플레이했던 게임을 기반으로 다음에 플레이할 게임을 추천해 주는 시스템과 게임을 리뷰 수 기반, 인기 기반 등 몇 가지의 기준으로 나열해 주는 기능을 제공하고 있다.

지금까지 Steam에서 제공하고 있는 서비스와는 다르게 이용자가 남긴 텍스트를 기반으로 게임을 추천해 주는 서비스기에 이용자들의 목소리를 기반으로 하여 좀 더 다양한 종류의 게임을 추천해 줄 수 있을 것으로 보이며, 그것이 강점으로 작용할 것이다.

2. 분석 대상 데이터의 수집 또는 획득 방안

(1) Steam 사이트에서 직접 크롤링

API에서 가져온 Game ID를 기반으로 All Reviews(전체 리뷰 개수)를 Get 방식을 사용해서 직접 크롤링한다.

Game ID 크롤링

ㅁ 리뷰 수가 많은 순서대로 Game을 정렬해 주는 스팀 내 화면에서 상위 2만개의 Game ID를 크롤링해 오는 함수 구축

```
import requests
from bs4 import BeautifulSoup
from tqdm import tqdm

def get_n_appids(n=20000, filter_by='Reviews_DESC'):
    appids = []
    url = f'https://store.steampowered.com/search/?category1=998&filter={filter_by}&page='
    page = 0

    while page*25 < n:
        page += 1
        response = requests.get(url=url+str(page), headers={'User-Agent': 'Mozilla/5.0'})
        soup = BeautifulSoup(response.text, 'html.parser')
        for row in tqdm(soup.find_all(class_='search_result_row')):
            try:
                appids.append(row['data-ds-appid'])
            except:
                pass

    return appids[:n]
```

```
get_n_appids()
```

Game Name, Game Tag, Game Description 크롤링

▫ Steam 사이트에서 가져온 Game ID를 바탕으로 get 방식으로 페이지 url 접속 함수 구축

```
def get_gameinfo(appid):
    url = 'https://store.steampowered.com/app/'
    response = requests.get(url=url+appid, headers={'User-Agent': 'Mozilla/5.0'})
    a = response.content
    soup = BeautifulSoup(a, 'html.parser')
    return soup
```

▫ Game Tag, Game Description, Game Name을 CSS를 사용해서 가져오기

- 중간에 오류가 날 가능성을 대비하여 try, except 문으로 작성해 준다.

```
import pandas as pd
from tqdm import tqdm

df_gameinfo = pd.DataFrame(columns=("app_id", "game_name", "game_desp", "game_tag"))

for k, j in tqdm(enumerate(s)):
    try:
        pp = get_gameinfo(j)
        test_name = pp.select_one("#appHubAppName_responsive").text
        test_description = pp.select_one("div.game_description_snippet").text.replace("\r", "").replace("\n", "").replace("\t", "")
        test_tag = pp.select("a.app_tag")
        tag_list = []

        for i in range(len(test_tag)):

            tag = test_tag[i].text.replace("\r", "").replace("\n", "").replace("\t", "")
            tag_list.append(tag)

        tag_str = " ".join(tag_list)
        tag_str

        df_gameinfo.loc[k] = [j, test_name, test_description, tag_str]

    except:
        pass
```

All reviews 크롤링

▫ Game ID별 get 방식으로 페이지 url 접속 함수 구축

```
import requests
from bs4 import BeautifulSoup

def get_gameinfo(appid):
    url = 'https://store.steampowered.com/app/'
    response = requests.get(url=url+appid, headers={'User-Agent': 'Mozilla/5.0'})
    a = response.content
    soup = BeautifulSoup(a, 'html.parser')
    return soup
```

▫ Game ID별 get 방식으로 페이지 url 접속 후 각 페이지에서 All Reviews를 CSS를 사용해서 가져오기

- 중간에 오류가 날 가능성을 대비하여 try, except 문으로 작성해 준다.

```
import pandas as pd
from tqdm import tqdm

df_game_review_num = pd.DataFrame(columns=("appid", "review_all_num"))

for k, j in tqdm(enumerate([gameList["appid"].values[i] for i in range(len(gameList["appid"].values))])):
    try:
        pp = get_gameinfo(j)
        review_all_num = pp.select("span.responsive_hidden")[1].text.replace("\r", "").replace("\n", "").replace("\t", "").replace("("



        df_game_review_num.loc[k] = [j, review_all_num]

    except:
        pass
```

(2) Steam API 사이트를 크롤링

```
import requests

def get_reviews(appid, params={'json':1}):
        url = 'https://store.steampowered.com/appreviews/'
        response = requests.get(url=url+appid, params=params, headers={'User-Agent': 'Mozilla/5.0'})
        return response.json()
```

```
def get_n_reviews(appid, n=20):
    reviews = []
    cursor = '*'
    params = {
            'json' : 1,
            'filter' : 'all',
            'language' : 'english',
            'day_range' : 9223372036854775807,
            'review_type' : 'all',
            'purchase_type' : 'all'
            }

    while n > 0:
        params['cursor'] = cursor.encode()
        params['num_per_page'] = min(20, n)
        n -= 20

        response = get_reviews(appid, params)
        cursor = response['cursor']
        reviews += response['reviews']

        if len(response['reviews']) < 20: break

    return reviews
```

▫ Steam 사이트에서 가져온 Game ID를 바탕으로 review를 가져오기

- 사이트를 확인해 본 결과 (데이터 몇 개 뽑아서 계속 실험) 가장 상위에 나오는 데이터가 공감 수, 좋아요 수 등을 합친 weighted_vote_score - helpfulness score이 가장 높은 순서대로 불러오는 것으로 확인되었기에 가장 위의 1개의 데이터씩만 가져온다.

```
import pandas as pd
from tqdm import tqdm

df_review = pd.DataFrame(columns=("app_id", "review"))

for k, j in tqdm(enumerate(s)):
    try:
        df_review.loc[k] = [j, get_n_reviews(j)[0]['review']]

    except:
        pass
```

💡 수집한 데이터들은 하나의 데이터 프레임으로 관리한다. try, except문을 사용해서 오류가 나더라도 넘어가면서 크롤링할 수 있게 설정했기 때문에 각 부분들을 Game ID 기준으로 하나의 데이터프레임으로 합친 후 한 부분이라도 빈 곳이 있는 행을 제거하는 절차를 거쳐 처음에는 2만개의 데이터를 가져오는 것을 목표로 했지만 **총 `18973` 개의 데이터**를 가져올 수 있게 되었다.

3. 데이터 전처리 및 정제

(1) 결측 데이터 처리

1. **모델을 만들기 위해 All Reviews 데이터의 결측 데이터를 0으로 바꿔 준다. 나머지 이상한 값 (텍스트 값)들을 확인하여 모두 0으로 변환해 준다.**

```
df_game_review_num.loc[df_game_review_num['review_all_num'] == ":", 'review_all_num'] = 0
df_game_review_num.loc[df_game_review_num['review_all_num'] == "NaN", 'review_all_num'] = 0
df_game_review_num.loc[df_game_review_num['review_all_num'] == 'Explore and Customize', 'review_all_num'] = 0
```

해당 코드를 실행하면 모든 All Reviews 데이터가 float 형태로 변환되게 된다.

(2) 유형 변환

1. **모델을 만들기 위해 데이터 프레임 안에 들어 있는 All Reviews 데이터를 정렬하기 쉽게 int 형태로 바꿔 준다.**

```
gameList_model_df_new.astype({'review_all_num':'int'})
```

관련해서는 뒤에 분석 방안 부분에서 더 자세하게 설명한다.

- 빈도 변경
- 데이터 그룹화

(3) 데이터 결합

1. 모델을 만들기 위해 필요한 데이터 프레임으로 Game ID, Game Name, All Reviews 데이터를 합쳐 준다.

생성된 벡터와 숫자를 맞춰 주기 위해서 left join으로 결합을 실시한다.

데이터 결합 예시

```
import pandas as pd

df_game_review_num = pd.read_csv("./game_review_num.csv")
gameList_model = gameList[["appid", "name"]]
gameList_model_df = pd.merge(gameList_model, df_game_review_num, on="appid", how="left")
```

gameList_model_df

	appid	name	review_all_num
0	1174180	Red Dead Redemption 2	292346.0
1	578080	PUBG: BATTLEGROUNDS	2110680.0
2	1687950	Persona 5 Royal	15001.0
3	1172470	Apex Legends™	540204.0
4	1811260	EA SPORTS™ FIFA 23	29129.0
...	...	...	...
18968	906600	Gunkid 99	0.0
18969	1737200	Ardarium	0.0
18970	1823570	Age of Valakas: Vietnam	35.0
18971	1516320	Skeleton King	0.0
18972	857950	Incredible Mandy	0.0

18973 rows × 3 columns

(4) 텍스트 전처리

위와 같은 방식으로 Game Review, Game Description, Game Tag에 대한 Dataframe이 완성되었고 아래와 같은 규칙에 의해 텍스트를 변환, 삭제하였다.

1. HTML 태그는 삭제: steam페이지에서 사용하는 HTML 태그와 일반적인 HTML태그모두 삭제
2. 괄호는 삭제, 내용물은 그대로 둠: 괄호에 묶여있는 내용은 그대로 두고 괄호를 띄어쓰기로 바꾼다
3. 숫자, 특수문자는 제거. 알파벳은 소문자화
4. Lemmatization 진행
5. 불용어가 아니면서 nltk 사전에 포함되는 단어만 남긴다

```
# 토큰화 함수
def tokenization(sentences):
    return [word_tokenize(sentence) for sentence in sentences]

# 전처리 함수
def preprocess_sentence(sentence):
  # HTML 태그 삭제
  sentence = [re.sub(r"<(\"[^\"]*\"|'[^']*'|[^'\">])*>", '', word) for word in sentence]
  # stema용 Html태그 삭제
  sentence = [re.sub(r"\[[^\]]*]?", '', word) for word in sentence]
  # 괄호 삭제
  sentence = [re.sub(r"[\[\]\(\)]", '', word) for word in sentence]
  # url 삭제
  sentence = [re.sub(r'https?:\/\/(www\.)?[-a-zA-Z0-9@:%._\+~#=]{1,256}\.[a-zA-Z0-9()]{1,6}\b([-a-zA-Z0-9()@:%_\+.~#?&//=]*)', '', wor
  # 영어를 제외한 숫자, 특수 문자 등은 전부 제거. 모든 알파벳은 소문자화
  sentence = [re.sub(r'[^a-zA-z\s]', '', word).lower() for word in sentence]
  # lemmatization 진행
  lmtzr = WordNetLemmatizer()
  sentence = [lmtzr.lemmatize(word) for word in sentence]
  # 불용어가 아니면서 단어가 실제로 존재해야 한다.
  return [word for word in sentence if word not in stop_words and word in words.words() and len(word) > 1]

# 위 전처리 함수를 모든 문장에 대해서 수행. 이 함수를 호출하면 모든 행에 대해서 수행.
def preprocess_sentences(sentences):
    return [preprocess_sentence(sentence) for sentence in sentences]

def flatten(l):
    return [item for sublist in l for item in sublist]
```

(5) Corpus Embedding

word embedding을 하여 그 평균값으로 sentence vector를 구하고 그 평균으로 corpus vector를 구하는 작업.word embedding시,

- word2vec(CBOW, skip-gram)은 맥락을 고려하므로 단어 간 유추작업에는 좋은 성능을 보이지만 임베딩 벡터가 윈도우 크기 내에서만 주변 단어를 고려하기 때문에 코퍼스의 전체적인 통계 정보를 반영하지 못한다.
- GloVe는 통계적 정보와 맥락 모두를 고려하는 임베딩 방법이다.

CBOW와 GloVe 방식을 비교하기 위해 해당 두 가지 워드 임베딩을 진행하였다.

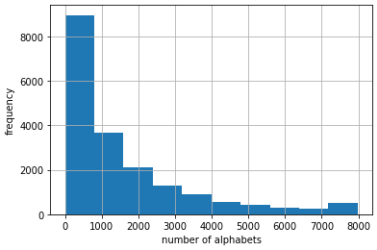
Word to Sentence, Sentence to Corpus

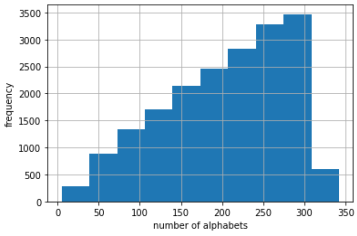
```
# 단어 벡터의 평균으로부터 문장 벡터를 얻는다.
def calculate_sentence_vector(sentence):
    if len(sentence) != 0:
        return sum([glove_dict.get(word, zero_vector) for word in sentence])/len(sentence)
    else:
        return zero_vector


# 각 문장에 대해서 문장 벡터를 반환
def sentences_to_vectors(sentences):
    return [calculate_sentence_vector(sentence) for sentence in sentences]


def corpus_to_vectors(corpus):
    return sum(corpus)/len(corpus)
```

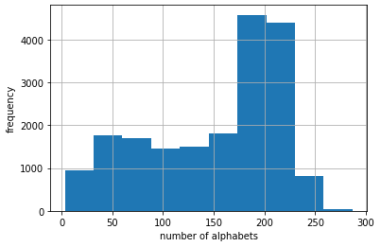
`calculate_sentence_vector` 함수의 2번째 라인에 glove_dict를 사용하는 대신 CBOW model을 사용하면 CBOW 워드 임베딩을 기반으로 한 Corpus Embedding이 나온다.

4. 탐색적 데이터 분석

(1) 전체 길이 분포



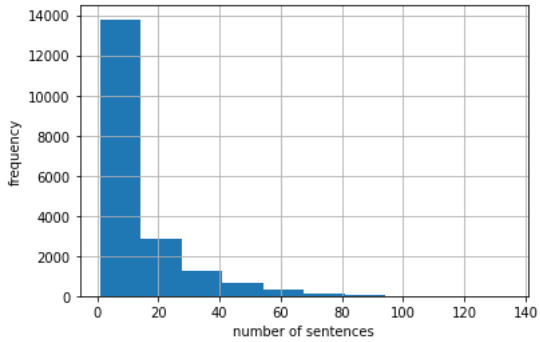


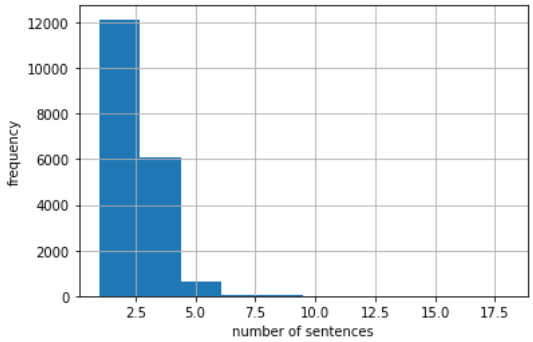


차례대로 Game Review, Game Description, Game Tag에 대한 전체 길이 분포이다.

- Game Review의 경우 Right-skewed이며 X축 Scale이 1000단위, 가장 많은 빈도는 1000길이 이하이다.
- Game Description의 경우 Left-skewed이며 X축 Scale이 50단위, 가장 많은 빈도는 300길이이다.
- Game Tag의 경우 Left-skewed에 가깝고 X축 Scale이 50단위, 가장 많은 빈도는 200길이이다.

(2) 문장 수 분포

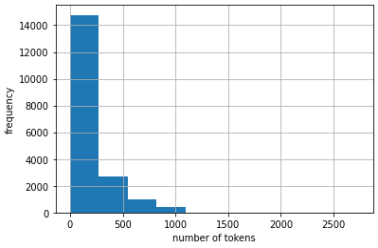


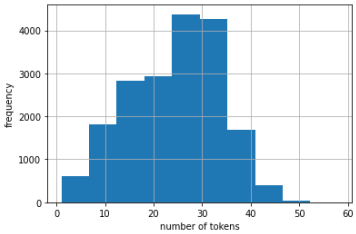


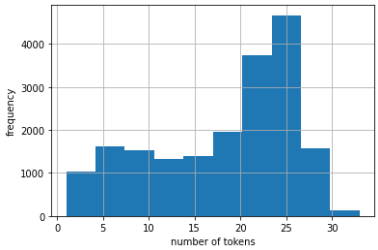
차례대로 Game Review, Game Description에 대한 문장 수 분포이다.

- Game Review의 경우 Right-skewed이며 X축 Scale이 20단위, 가장 많은 빈도는 20개 이하이다.
- Game Description의 경우 Right-skewed이며 X축 Scale이 2.5단위, 가장 많은 빈도는 2.5개 이하이다.

(3) Token 분포





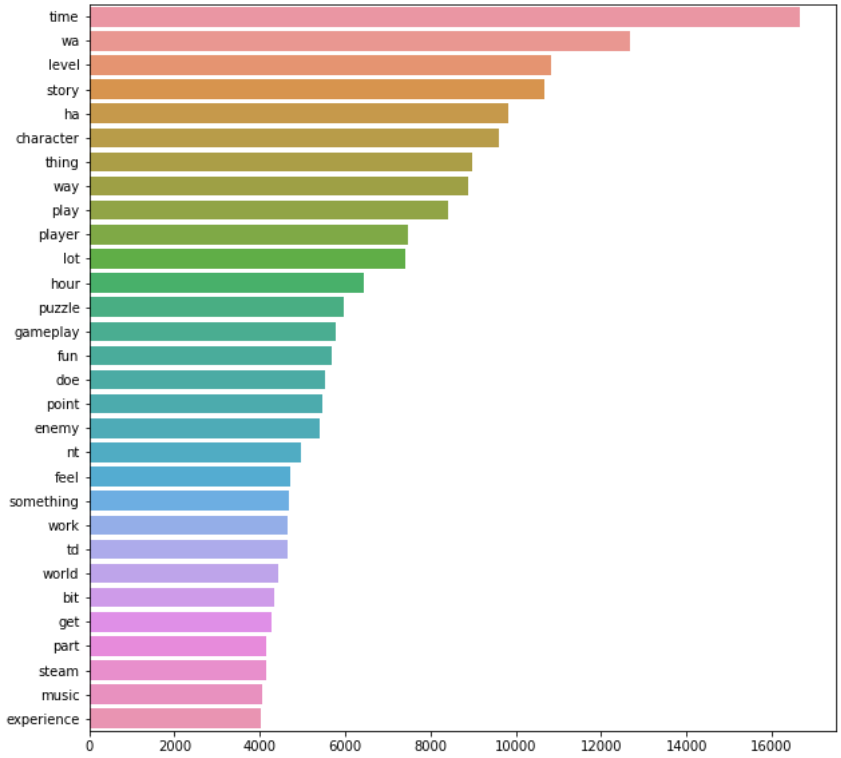


차례대로 Game Review, Game Description, Game Tag에 대한 Token 길이 분포이다.

- Game Review의 경우 Right-skewed이며 X축 Scale이 250단위, 가장 많은 빈도는 250개 이하이다.
- Game Description의 경우 Symmetric이며 X축 Scale이 10단위, 가장 많은 빈도는 25~30개이다.
- Game Tag의 경우 Left-skewed이며 X축 Scale이 5단위, 가장 많은 빈도는 25개이다.

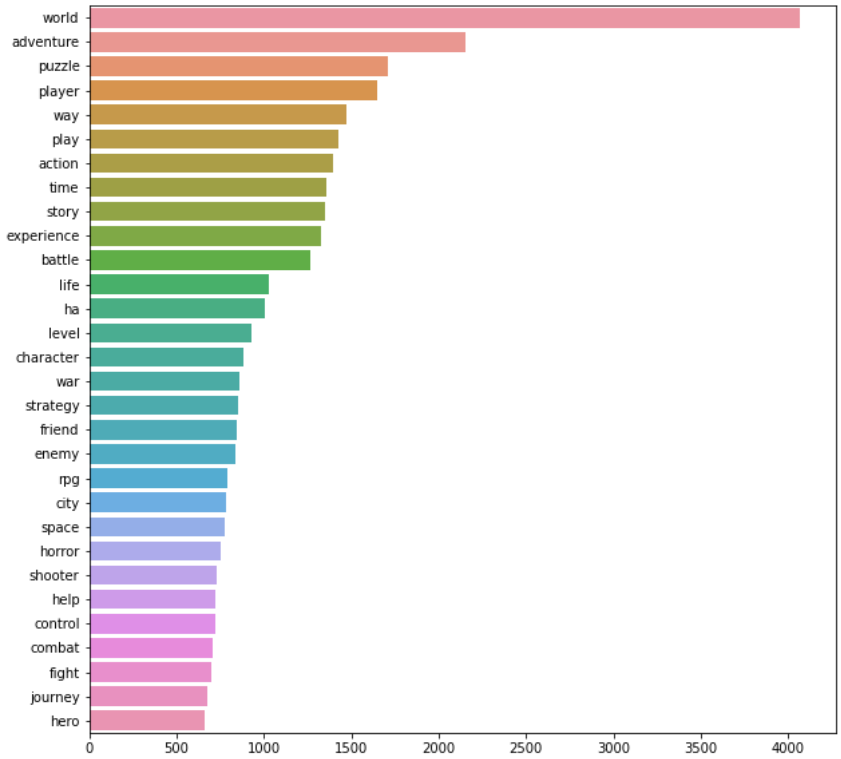
(4) 명사통계

- Game Review



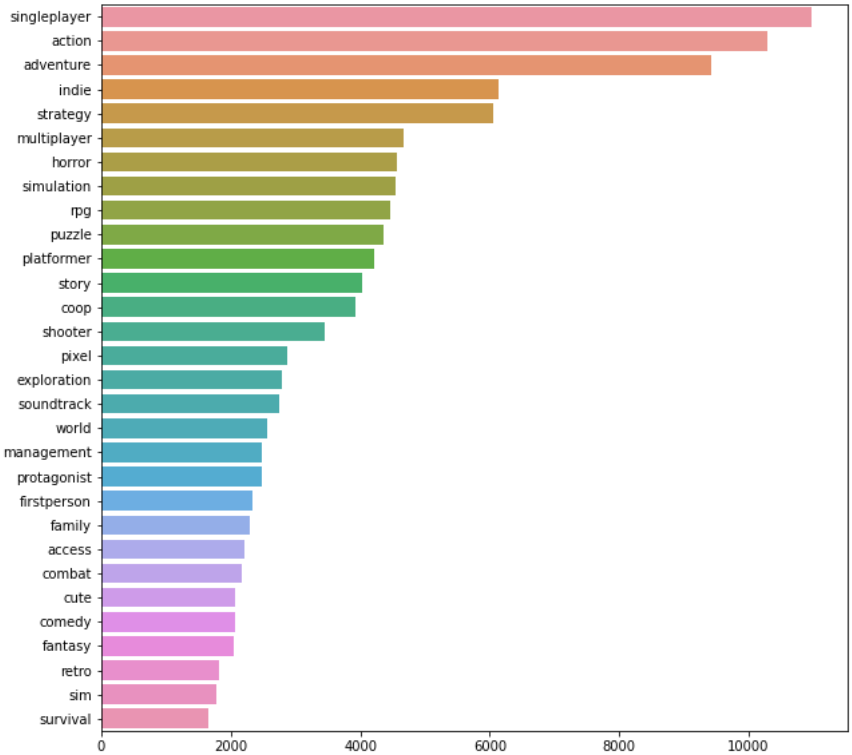
명사 통계시 플레이타임 혹은 스토리 관련 단어가 상위에 등장한다.

- Game Description



명사 통계시 스토리 혹은 게임 플레이와 캐릭터에 관련한 단어가 상위에 등장한다.

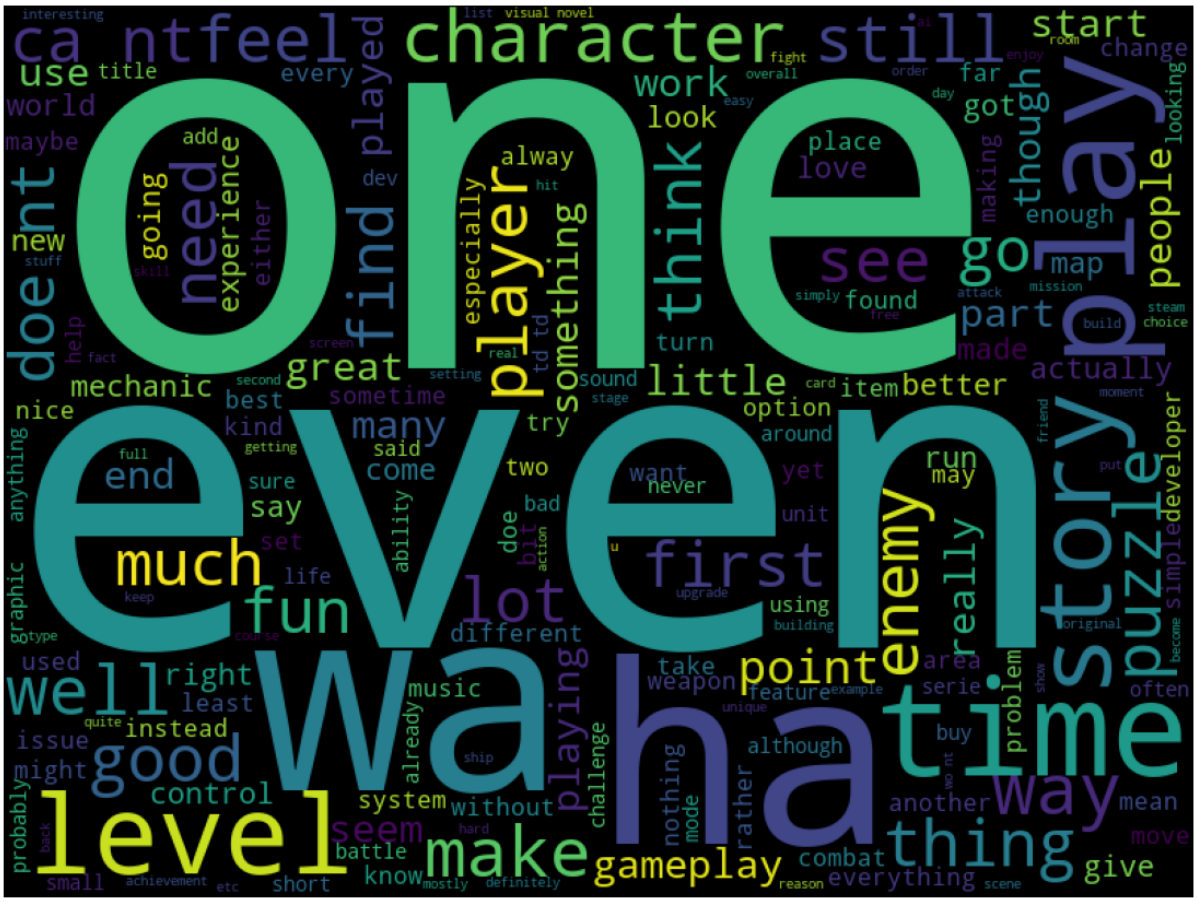
- Game Tag



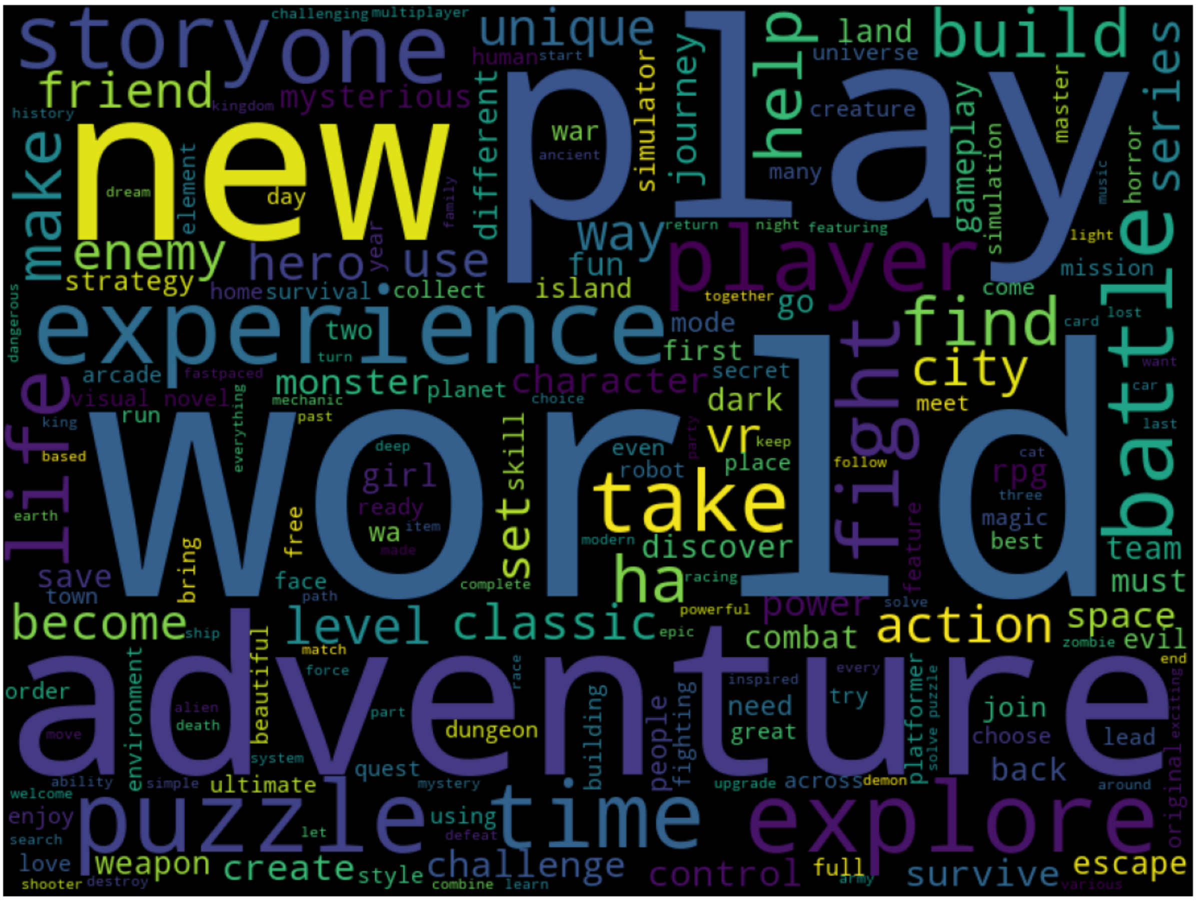
명사 통계시 게임 장르에 대한 단어가 상위에 등장한다.

(5) Word Cloud

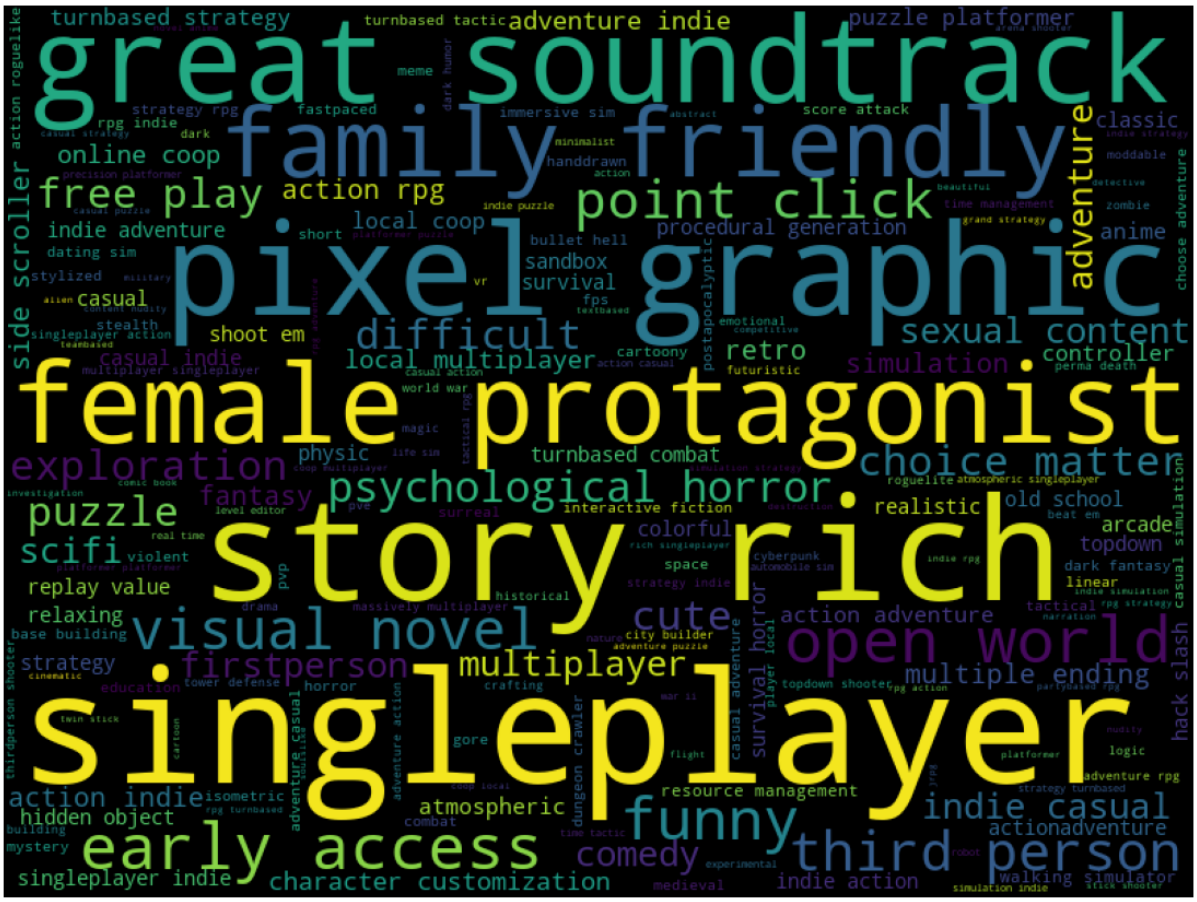
- Game Review



- Game Description

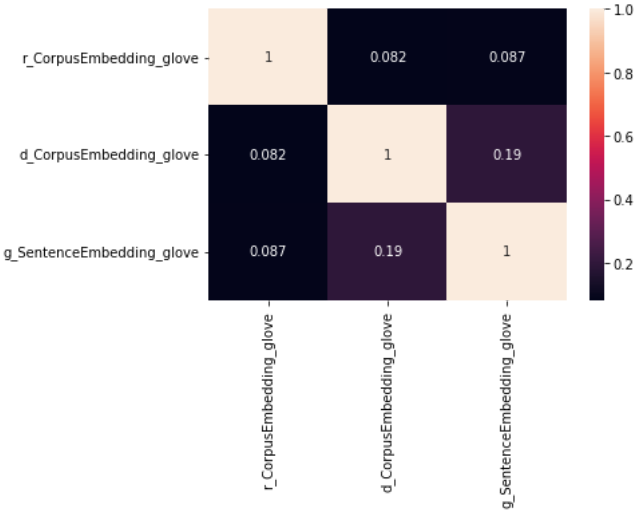

world
adventure
play
new
experience
story
one
unique
build
help
player
puzzle
time
explore
battle
find
fight
take
friend
enemy
hero
use
life
make
level
classic
action
become
city
series
journey
simulator
gameplay
different
way
fun
island
strategy
collect
survival
home
monster
planet
character
dark
vr
set
skill
girl
discover
ha
combat
space
evil
team
must
save
town
bring
face
free
order
environment
beautiful
dungeon
need
great
join
choose
back
lead
quest
ultimate
weapon
create
style
challenge
control
full
survive
escape
love
enjoy
using
mysterious
land
universe
creature
master
simulation
horror
mission
come
go
first
mode
secret
meet
city
arcade
visual novel
run
two
war
many
day
platformer
building
people
fighting
across
try
power
magic
best
rpg
robot
place
even
original

- Game Tag



(6) 세 필드간 상관관계

Corpus Vectorization을 한 후 그 벡터들의 flatten하여 각 필드에 대해 correlation coef를 구한 결과이다.



세 필드 간에 선형적인 상관관계를 찾을 수 없다.

(7) EDA 해석

세 필드의 분포가 확연히 다른 것을 알 수 있다. 이런 경우, 고려할 수 있는 방법은 크게 두 가지이다.

1. 세 필드 각각에 대한 결과를 도출하여 Majority Vote를 진행
2. 세 필드의 평균을 구하되 각 필드의 평균,분산을 기반으로 large variation을 제거

두 가지 방법에서 우리가 입력으로 주는 것이 Corpus Vector라는 현실적인 특징(각 Element가 특징화하고 있는 것을 해석하기 어려움)을 감안하여 아래 분석방안을 고안했다.

5. 분석 방안

- **분석 주제:** 게임 이름을 입력하면 그 게임에서 얻을 수 있는 텍스트 데이터를 기반으로 해서 게임을 추천해 주는 모델을 만들어 추천 시스템을 구축한다.
  - **방안 1:** Game Tag, Game Review, Game Description를 cbow를 사용해서 벡터화한 데이터를 각각 K-means를 활용하여 그룹화한 후 추천 시스템 이용자가 게임의 이름을 넣으면 각각의 그룹 내에서 리뷰 수가 많은 상위 n개의 게임을 추천해 주는 모델을 만든다.
  - **방안 2:** Game Tag, Game Review, Game Description를 cbow를 사용해서 벡터화한 데이터를 하나로 합치고 K-means를 활용하여 그룹화한 후 추천 시스템 이용자가 게임의 이름을 넣으면 그룹 내에서 리뷰 수가 많은 상위 n개의 게임을 추천해 주는 모델을 만든다.

- **분석 순서:**

(1) Game ID, Game Name, All Reviews가 있는 데이터 프레임 생성하기(데이터 결합 사용)

(2) All Reviews 데이터를 전처리한다. (결측값 채우기, 텍스트 데이터 처리, 데이터 유형 변환)

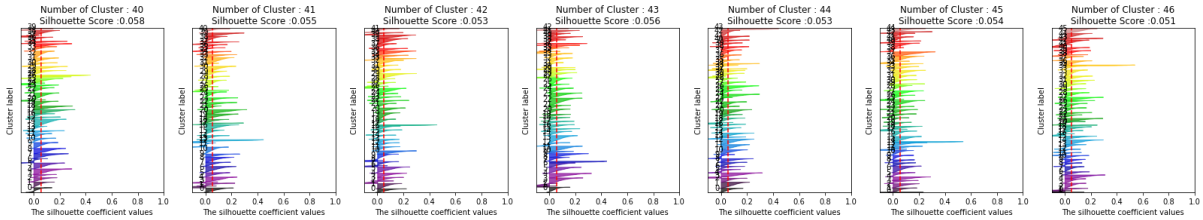
(3) 각 방안에 맞는 데이터를 준비한다.

(4) K-means를 활용하여 벡터를 클러스터링한다. (알맞는 그룹 개수를 알아내기 위해 실루엣 계수를 이용한다.)

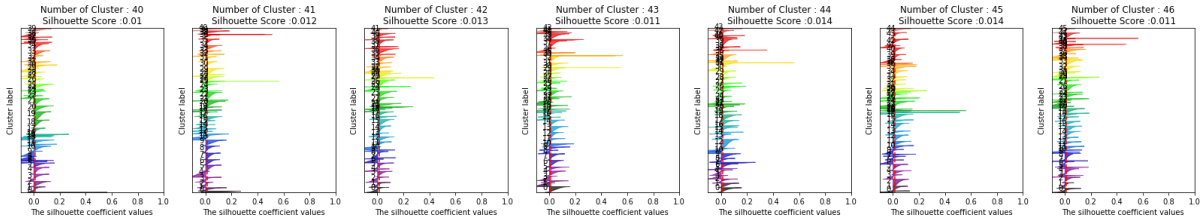
(5) [게임 이름, 추천받고 싶은 게임 수]를 입력하면 그 게임과 같은 그룹 내에 있으면서 전체 리뷰 수가 많은 상위 n개의 게임을 추천해 주는 모델을 만든다.

- **사용한 package:** numpy, pandas, sklearn.cluster.Kmeans, sklearn.datasets.make_blobs, sklearn.metrics.silhouette_samples, sklearn.metrics.silhouette_score, matplotlib.pyplot, matplotlib.cm, math
- **사용한 데이터**: cbow를 사용해서 벡터화한 데이터 (가장 성능이 좋았다.)

cbow를 사용해서 벡터화한 데이터 실루엣 계수 그림



glove를 사용해서 벡터화한 데이터 실루엣 계수 그림



위의 두 개 사진을 비교했을 때 cbow를 사용해서 벡터화한 데이터에 관한 실루엣 계수가 좀 더 안정적이고 나눠진 그룹 간의 부피와 길이가 비슷한 양상을 보이는 것을 확인할 수 있다. 따라서 cbow를 사용해서 벡터화한 데이터를 사용했다.

또한 실루엣 계수를 통해서 K-means를 사용해서 그룹화할 때 그 그룹의 개수를 43개로 지정한다.

공통 코드

1. 모델 생성에 필요한 데이터 프레임

```
import pandas as pd

gameList = pd.read_csv("./gameList4EDA.csv")
df_game_review_num = pd.read_csv("./game_review_num.csv")
gameList_model = gameList[["appid", "name"]]
gameList_model_df = pd.merge(gameList_model, df_game_review_num, on="appid", how="left")
```

2. Game Tag, Game Review, Game Description를 cbow를 사용해서 벡터화한 데이터 불러오기

```
import numpy as np

cbow_tag = np.load("./cbow_tag.npy")
cbow_review = np.load("./cbow_review.npy")
cbow_description = np.load("./cbow_description.npy")
```

3. 실루엣 계수 함수 코드

```
def visualize_silhouette(cluster_lists, X_features):

    from sklearn.datasets import make_blobs
    from sklearn.cluster import KMeans
    from sklearn.metrics import silhouette_samples, silhouette_score

    import matplotlib.pyplot as plt
    import matplotlib.cm as cm
    import math

    # 입력값으로 클러스터링 갯수들을 리스트로 받아서, 각 갯수별로 클러스터링을 적용하고 실루엣 개수를 구함
    n_cols = len(cluster_lists)

    # plt.subplots()으로 리스트에 기재된 클러스터링 수만큼의 sub figures를 가지는 axs 생성
    fig, axs = plt.subplots(figsize=(4*n_cols, 4), nrows=1, ncols=n_cols)

    # 리스트에 기재된 클러스터링 갯수들을 차례로 iteration 수행하면서 실루엣 개수 시각화
    for ind, n_cluster in enumerate(cluster_lists):

        # KMeans 클러스터링 수행하고, 실루엣 스코어와 개별 데이터의 실루엣 값 계산. max_iter: 클러스터 최대 반복 횟수
        clusterer = KMeans(n_clusters = n_cluster, max_iter=500, random_state=0)
        cluster_labels = clusterer.fit_predict(X_features)

        sil_avg = silhouette_score(X_features, cluster_labels)
        sil_values = silhouette_samples(X_features, cluster_labels)

        y_lower = 10
        axs[ind].set_title('Number of Cluster : '+ str(n_cluster)+'\n' \
                           'Silhouette Score :' + str(round(sil_avg,3)) )
        axs[ind].set_xlabel("The silhouette coefficient values")
        axs[ind].set_ylabel("Cluster label")
        axs[ind].set_xlim([-0.1, 1])
        axs[ind].set_ylim([0, len(X_features) + (n_cluster + 1) * 10])
        axs[ind].set_yticks([])  # Clear the yaxis labels / ticks
        axs[ind].set_xticks([0, 0.2, 0.4, 0.6, 0.8, 1])

        # 클러스터링 갯수별로 fill_betweenx( )형태의 막대 그래프 표현. cm.nipy_spectral 컬러 지정
        for i in range(n_cluster):
            ith_cluster_sil_values = sil_values[cluster_labels==i]
            ith_cluster_sil_values.sort()

            size_cluster_i = ith_cluster_sil_values.shape[0]
            y_upper = y_lower + size_cluster_i

            color = cm.nipy_spectral(float(i) / n_cluster)
            axs[ind].fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_sil_values, \
                                   facecolor=color, edgecolor=color, alpha=0.7)
            axs[ind].text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
            y_lower = y_upper + 10
```

```
        axs[ind].axvline(x=sil_avg, color="red", linestyle="--")
```

방안 1

1. Game Tag, Game Review, Game Description를 cbow를 사용해서 벡터화한 데이터를 각각 K-means 그룹화시킨다.

```
import numpy as np
from sklearn.cluster import KMeans

k_means = KMeans(n_clusters = 43, n_init=12)
k_means.fit(cbow_tag)

k_means_labels1 = k_means.labels_

k_means = KMeans(n_clusters = 43, n_init=12)
k_means.fit(cbow_review)

k_means_labels2 = k_means.labels_

k_means = KMeans(n_clusters = 43, n_init=12)
k_means.fit(cbow_description)

k_means_labels3 = k_means.labels_
```

2. 그룹화한 후 추천 시스템 이용자가 게임의 이름을 넣으면 각각의 그룹 내에서 리뷰 수가 많은 상위 n개의 게임을 추천해 주는 함수를 구현한다.

```
gameList_model_df = pd.read_csv("./gameList_model.csv")
```

```
gameList_model_df["segment1"] = k_means_labels1
gameList_model_df["segment2"] = k_means_labels2
gameList_model_df["segment3"] = k_means_labels3
```

```
def recommend_new(steam_game, int_num):
    global gameList_model_df
    game_index=gameList_model_df[gameList_model_df['name']==steam_game].index[0]
    segment1=k_means_labels1[game_index]
    segment2=k_means_labels2[game_index]
    segment3=k_means_labels3[game_index]


    gameList_model_df1_new = gameList_model_df.fillna(0)
    gameList_model_df1_new = gameList_model_df1_new.astype({'review_all_num':'int'})
    gameList_model_df1_new = gameList_model_df1_new.sort_values(['segment1', 'review_all_num'], ascending=False)

    game_list_df_final1 = gameList_model_df1_new[gameList_model_df1_new["segment1"]==segment1].reset_index(drop=True)
    game_list_df_final1 = game_list_df_final1[game_list_df_final1["name"]!=steam_game]
    game_list_df_final1 = game_list_df_final1.drop_duplicates(subset='name',keep='first')

    print("★ 태그 관련도를 통해서 게임을 추천합니다. ★")
    print("-----------------------------------------------")
    for i in range(int_num):
        print(game_list_df_final1[0:5].values[i][1])

    print()
    print()

    gameList_model_df2_new = gameList_model_df.fillna(0)
    gameList_model_df2_new = gameList_model_df2_new.astype({'review_all_num':'int'})
    gameList_model_df2_new = gameList_model_df2_new.sort_values(['segment2', 'review_all_num'], ascending=False)

    game_list_df_final2 = gameList_model_df2_new[gameList_model_df2_new["segment2"]==segment2].reset_index(drop=True)
    game_list_df_final2 = game_list_df_final2[game_list_df_final2["name"]!=steam_game]
    game_list_df_final2 = game_list_df_final2.drop_duplicates(subset='name',keep='first')

    print("★ 사용자 리뷰 관련도를 통해서 게임을 추천합니다. ★")
    print("-----------------------------------------------------")
    for i in range(int_num):
        print(game_list_df_final2[0:5].values[i][1])
```

```
    print()
    print()

    gameList_model_df3_new = gameList_model_df.fillna(0)
    gameList_model_df3_new = gameList_model_df3_new.astype({'review_all_num':'int'})
    gameList_model_df3_new = gameList_model_df3_new.sort_values(['segment3', 'review_all_num'], ascending=False)

    game_list_df_final3 = gameList_model_df3_new[gameList_model_df3_new["segment3"]==segment3].reset_index(drop=True)
    game_list_df_final3 = game_list_df_final3[game_list_df_final3["name"]!=steam_game]
    game_list_df_final3 = game_list_df_final3.drop_duplicates(subset='name',keep='first')

    print("★ 사용자 리뷰 관련도를 통해서 게임을 추천합니다. ★")
    print("-----------------------------------------------------")
    for i in range(int_num):
        print(game_list_df_final3[0:5].values[i][1])
```

```
recommend_new("Persona 5 Royal", 5)
```

```
recommend_new("Persona 5 Royal", 5)

★ 태그 관련도를 통해서 게임을 추천합니다. ★
-----------------------------------------------
Persona 4 Golden
OMORI
South Park™: The Stick of Truth™
Pathfinder: Kingmaker - Enhanced Plus Edition
FINAL FANTASY VII


★ 사용자 리뷰 관련도를 통해서 게임을 추천합니다. ★
-----------------------------------------------------
ELDEN RING
Hollow Knight
No Man's Sky
Sekiro™: Shadows Die Twice - GOTY Edition
Half-Life 2


★ 사용자 리뷰 관련도를 통해서 게임을 추천합니다. ★
-----------------------------------------------------
Rust
Unturned
Dying Light 2 Stay Human
Muse Dash
BattleBlock Theater®
```

방안 2

1. Game Tag, Game Review, Game Description를 cbow를 사용해서 벡터화한 데이터를 하나로 합쳐서 K-means 그룹화시킨다.

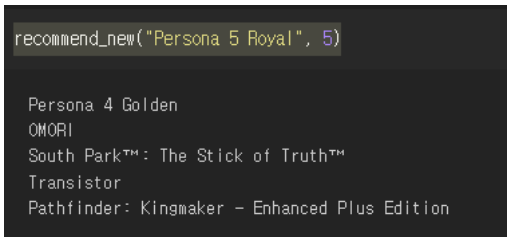
```
cbow_whole = np.concatenate([cbow_tag, cbow_review, cbow_description], 1)

import numpy as np
from sklearn.cluster import KMeans

k_means = KMeans(n_clusters = 43, n_init=12)
k_means.fit(cbow_whole)

k_means_labels = k_means.labels_
k_means_labels
```

2. 그룹화한 후 추천 시스템 이용자가 게임의 이름을 넣으면 각각의 그룹 내에서 리뷰 수가 많은 상위 n개의 게임을 추천해 주는 함수를 구현한다.

```
gameList_model_df = pd.read_csv("./gameList_model.csv")
gameList_model_df["segment"] = k_means_labels
```

```
def recommend_new(steam_game, int_num):
    global gameList_model_df
    game_index=gameList_model_df[gameList_model_df['name']==steam_game].index[0]
    segment=k_means_labels[game_index]

    gameList_model_df_new = gameList_model_df.fillna(0)
    gameList_model_df_new = gameList_model_df_new.astype({'review_all_num':'int'})
    gameList_model_df_new = gameList_model_df_new.sort_values(['segment', 'review_all_num'], ascending=False)

    game_list_df_final = gameList_model_df_new[gameList_model_df_new["segment"]==segment].reset_index(drop=True)
    game_list_df_final = game_list_df_final[game_list_df_final["name"]!=steam_game]

    for i in range(int_num):
        name_print = game_list_df_final[0:int_num].values[i][1]
        print(name_print)
```

```
recommend_new("Persona 5 Royal", 5)
```

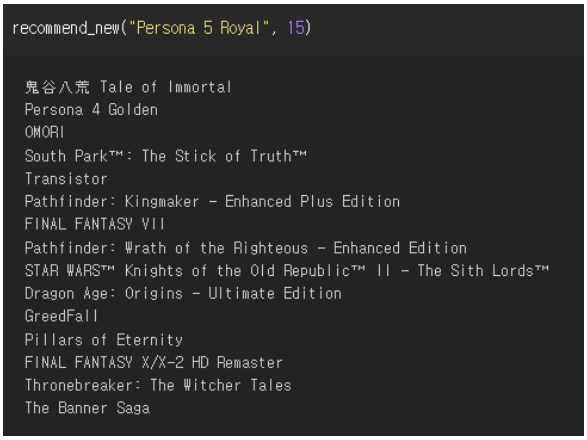


6. 데이터 분석 결과

- **추천 시스템 비교**

방안 1

방안 2



```
recommend_new("PUBG: BATTLEGROUNDS", 5)

 ★ 태그 관련도를 통해서 게임을 추천합니다. ★
 ------------------------------------------------
 Counter-Strike: Global Offensive
 Tom Clancy's Rainbow Six® Siege
 Apex Legends™
 Destiny 2
 Unturned


 ★ 사용자 리뷰 관련도를 통해서 게임을 추천합니다. ★
 ------------------------------------------------------
 DARK SOULS™ III
 Robocraft
 Assassin's Creed® Odyssey
 Sid Meier's Civilization® V
 Squad


 ★ 사용자 리뷰 관련도를 통해서 게임을 추천합니다. ★
 ------------------------------------------------------
 Destiny 2
 Brawlhalla
 Human: Fall Flat
 Counter-Strike
 World of Warships
```

```
recommend_new("PUBG: BATTLEGROUNDS", 15)

 Counter-Strike: Global Offensive
 Dota 2
 Tom Clancy's Rainbow Six® Siege
 Team Fortress 2
 Apex Legends™
 Destiny 2
 PAYDAY 2
 War Thunder
 Paladins®
 Sea of Thieves
 Halo Infinite
 Heroes & Generals
 Deep Rock Galactic
 World of Tanks Blitz
 Battlefield™ 2042
```

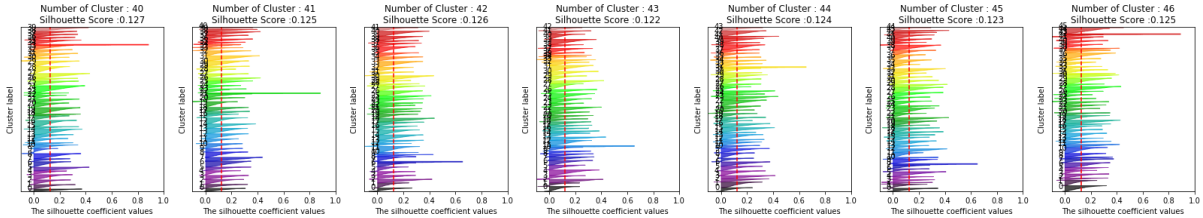
```
recommend_new("The Sims™ 4", 5)

 ★ 태그 관련도를 통해서 게임을 추천합니다. ★
 ------------------------------------------
 Stardew Valley
 Totally Accurate Battle Simulator
 Slime Rancher
 SPORE™
 PC Building Simulator


 ★ 사용자 리뷰 관련도를 통해서 게임을 추천합니다. ★
 ------------------------------------------------
 Phasmophobia
 New World
 Halo: The Master Chief Collection
 Battlefield™ V
 Splitgate


 ★ 사용자 리뷰 관련도를 통해서 게임을 추천합니다. ★
 ------------------------------------------------
 Don't Starve
 Starbound
 Wolcen: Lords of Mayhem
 Saints Row IV
 Planet Zoo
```

```
recommend_new("The Sims™ 4", 15)

 Stardew Valley
 Totally Accurate Battle Simulator
 House Flipper
 Clicker Heroes
 AdVenture Capitalist
 SPORE™
 Cookie Clicker
 The Sims™ 3
 Spiritfarer®: Farewell Edition
 Two Point Hospital
 Dorfromantik
 Leaf Blower Revolution - Idle Game
 Townscaper
 Cell to Singularity - Evolution Never Ends
 Mini Motorways
```

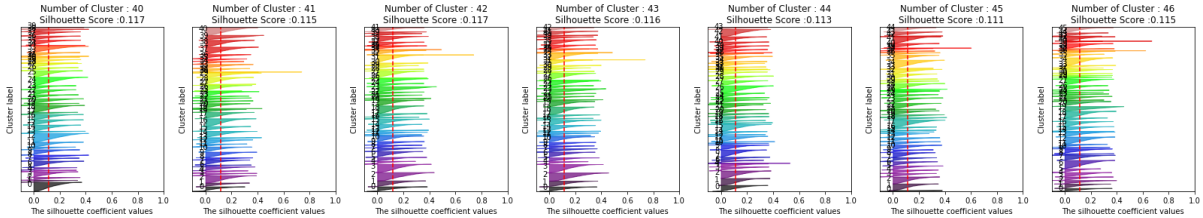
방안 1과 방안 2에서 만들어진 추천 서비스에 세 개의 스팀 게임 이름을 넣어 보았다.

- Game Tag, Game Description, Game Review를 합친 추천 서비스의 결과와 Game Tag만으로만 만든 추천 서비스의 결과가 비슷하다는 점을 확인할 수 있다.

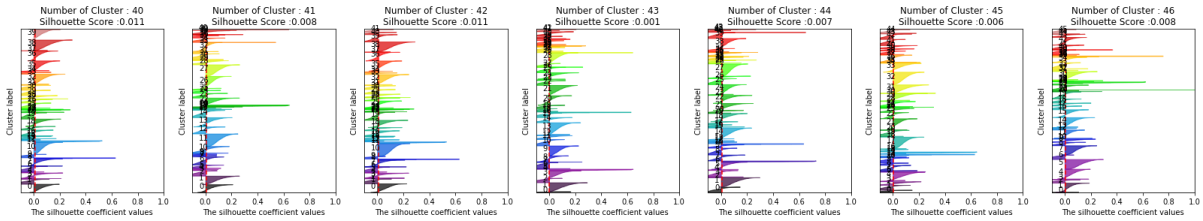
Game Tag 실루엣 계수



Game Description 실루엣 계수



Game Review 실루엣 계수



세 개의 실루엣 계수를 봤을 때 Game Tag의 실루엣 계수 그림이 가장 잘 나뉘어져 있는 것을 확인할 수 있는데, 이 점 때문에 Game Tag, Game Description, Game Review를 한 번에 합쳐서 만든 방안 2의 추천 서비스에서도 클러스터링을 할 때 Game Tag의 영향을 많이 받았지 않을까 예측할 수 있다.

- 방안 2에서 추천받은 게임을 방안 1에서 나온 각 주제별 추천과 비교하면 아예 주제별에서 추천되지 않은 게임이 나오는 경우도 있다는 것을 확인할 수 있다. Game Tag가 그룹화에 많은 영향을 준 건 확실하나 방안 2에서 K-means를 실행할 때 각 데이터가 합쳐져 있으면 다양한 결과물이 나올 수 있다.
- 두 방식 다 K-means를 사용하여 비슷한 장르의 게임을 추천해 주는 것을 확인할 수 있다.
- 방안 1의 결과를 보면 Game Tag, Game Description, Game Review를 각각 클러스터링해서 결과를 도출했는데, 셋 중에 겹치는 게임이 그렇게 많지 않다는 것을 확인할 수 있다. 게임의 장르, 게임 회사에서 설명하는 게임 정보, 이용자가 직접 플레이하고 사용하는 리뷰가 제작자의 의도와 다르게 사람들에게 전달이 될 수 있다. 의도가 같고 이용자들도 비슷하게 느낀다면 게임 정보와 리뷰의 클러스터링이 비슷하게 되었어야 하기에 셋을 잘 결합한 추천 서비스를 만드는 것이 중요한 과제로 보인다.

7. 결론

(1) 분석 결과의 요약

모델링 이전에는, CBOW보다 문서 전체의 통계적 정보를 포함하고 있는 GloVe기반의 Corpus Vector 입력이 더 좋을 것이라고 예측했다. 하지만 실제 클러스터링을 해보면 CBOW방식을 통한 Corpus Embedding이 의외로 GloVe방식을 통한 Corpus Embedding보다 하나의 Corpus를 특징화하는 것에 효과가 있음을 알 수 있다.

EDA과정에서 세 분포의 길이, 문장개수, 토큰, 상위 명사통계의 분포가 세 필드에 대해 다르고 선형적인 관계 또한 찾아볼 수 없었다. 방법1에서 해당 해석을 고려한 추천모델을 만들었고, 방법2는 세 필드를 합쳐 클러스터링을 하였다. 방법2의 경우 유사장르를 추천하는 기존 steam 추천목록 시스템과 유사한 모습을 보이고 방법1은 유사장르를 5개, 유사리뷰를 5개, 유사설명을 5개 추천한다.

(2) 주요 시사점

장르는 Formula, Convention, Icon 이 세가지에 의해 결정된다. 특정 장르의 게임을 좋아하는 게이머라는 뜻은 해당 장르를 특징화하는 세 가지 필드의 유사도를 기반으로 비슷한 게임을 플레이하려고하는 게이머라고 볼 수 있다.

그러나 실제로는 도트 그림체의 게임만 골라서 하는 게이머, 플레이가 어렵고 시간이 오래 걸리는 게임만 골라서 하는 게이머, 게임패드를 이용할 수 있는 게임만 골라서 하는 게이머 등의 사례가 존재한다. 이건 리뷰나 게임 설명을 직접 살펴보지 않는다면 얻기 힘든 정보이다. 만약 사용자가 같은 장르나 같은 게임사의 게임만 플레이하려고 한다면 위와 같은 상황은 발생하지 않아야 한다. 이게 기존 Steam등의 플랫폼에서 제공하는 추천목록 시스템의 한계이다.

우리가 만든 추천시스템은 장르 외적인, 특히 텍스트 형태의 데이터를 2개밖에 사용하지 않았음에도 사용자의 흥미를 끌 수 있는 결과를 도출해내었다. 더 다양한 필드와 효과적인 전처리 과정을 통해 기존 추천시스템보다 더 효과적인 추천 시스템을 만들 수 있을 것이다.

(3) 분석 결과의 개선 방안

- 데이터 수집 시, review를 가져올 때 공감을 가장 많이 받은 최고 리뷰 하나만 가져왔는데 여러 리뷰를 가져와서 corpus 단위로 분석할 수 있었으면 보다 정확한 성능의 모델을 개발할 수 있을 것이다.
- 데이터 수집 시, 리뷰, 게임 설명, 게임 카테고리만 수집했으나 제작사 등 더 많은 필드가 있다면 더 효과적인 모델을 구축할 수 있을 것이다.
- Item Based recommendation이 아닌 User Based recommendation을 적용하여 게임 뿐 아니라 유저 자체를 Vectorizing하여 모델을 설계할 수 있을 것이다.