2020-1 데이터분석캡스톤디자인

뮤지컬 장르 판별 및 장르 시각화

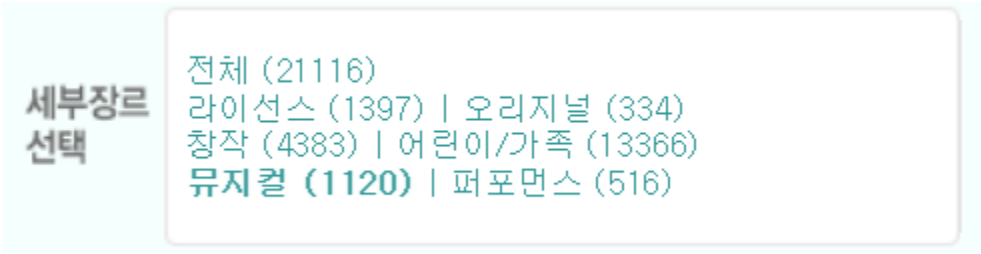
2018102098 소프트웨어융합학과 김서영

2016103208 응용물리학과 양윤지

목차

1.과제개요

현재 뮤지컬의 장르 : 창작, 라이선스, 오리지널, 어린이/가족
분류가 없는 것도 많음
⇒ 내용 및 취향 예측 불가



반면 영화의 경우 SF, 코미디, 공포, 판타지, 멜로
=> 한눈에 보고 예측 가능

뮤지컬도 작품의 특징적인 요소로 분류해보자!

장르로 구분하기 위한 명확한 정의가 있는 것은 아니며, 기준은 사람마다 다를 수 있음.

2. 과제 수행

뮤지컬 데이터 수집

https://broadwaymusicalhome.com/shows.htm 에서 307건 줄거리 크롤링

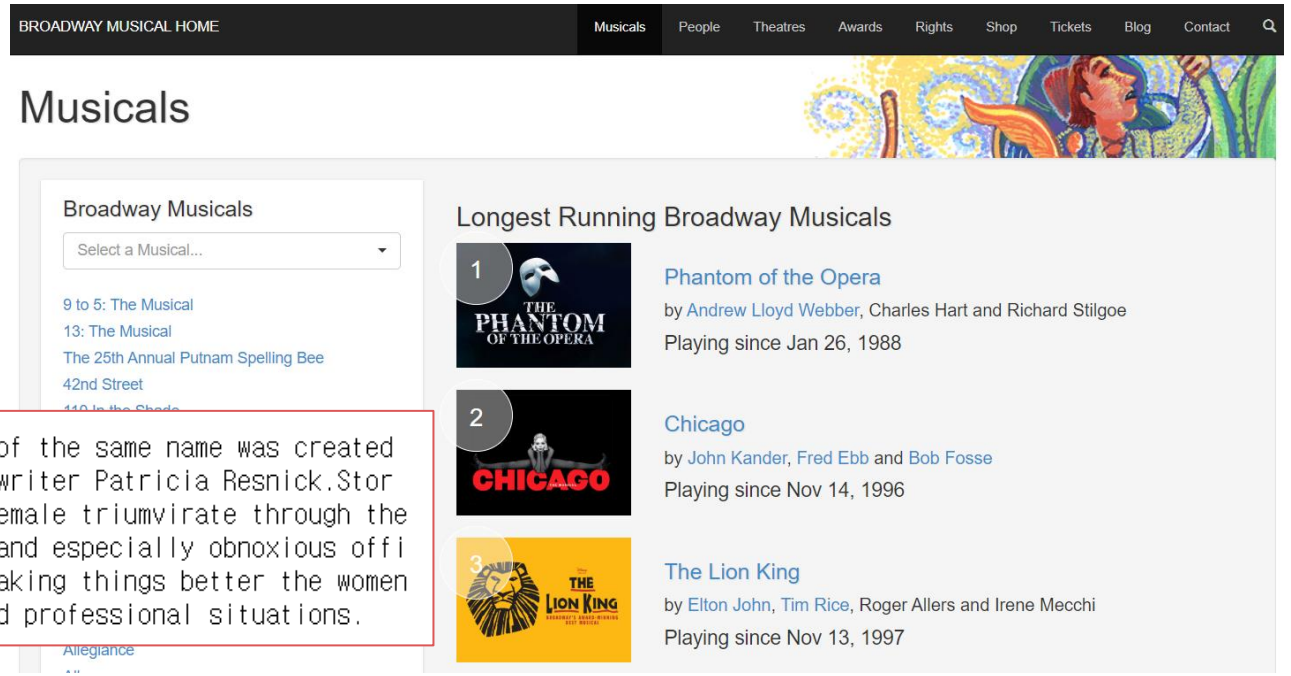
크롤링 결과 :

```
import urllib.request
from bs4 import BeautifulSoup
import requests

url = "https://broadwaymusicalhome.com/shows.htm"
req = urllib.request.urlopen(url)
res = req.read()
```

```
soup = BeautifulSoup(res,'html.parser')
for link in soup.find_all('a'):
```

```
9 to 5: The Musical,This musical based on the movie of the same name was created by star Dolly Parton and the film's original screenwriter Patricia Resnick.Story:Co-workers Violet Judy and Doralee form a strong female triumvirate through their mutual dissatisfaction with workplace conditions and especially obnoxious office boss Franklin Hart Jr.  After fantasizing about making things better the women work together to help improve both their personal and professional situations.
```



김서영

2. 과제수행

장르 분류 모델을 위한 데이터 수집

줄거리 데이터 http://www.cs.cmu.edu/~ark/personas/ 에서 수집(80000개)
부족한 뮤지컬 데이터를 영화로 보충

수집한 데이터 정리

줄거리 혹은 장르 태깅이 되어있지 않는 데이터는 제거 → 약 41000개

```
data = data.dropna(axis=0) #plot == NULL인 행 삭제
data = data[data.genre != "{}"] #genre == "{}"인 행 삭제
data = data.reset_index(drop = True)
data.head(5)
```

	id	genre	줄거리
0	330	{"/m/07s9rl0": "Drama", "/m/01t_vv": "Comedy-d...	In order to prepare the role of an important o...
1	3217	{"/m/01q03": "Cult", "/m/03npn": "Horror", "/m...	After being pulled through a time portal, Ash ...
2	3333	{"/m/06ppq": "Silent film", "/m/0219x_": "Indi...	The film follows two juxtaposed families: the ...
3	3746	{"/m/01jfsb": "Thriller", "/m/01qpc": "Cyberpu...	{{Hatnote}} In Los Angeles, November 2019, ret...
4	3837	{"/m/0hfjk": "Western", "/m/06nbt": "Satire", ...	In the American Old West of 1874, construction...

양윤지

2. 과제 수행

줄거리 데이터 장르 재태깅

총 364개의 장르 데이터 -> [romance, fanstasy, thriller, drama, history, social]
(애매한 장르들은 NULL값으로 처리)

최종장르

	0	
1	"/m/0hj3n0q": "Early Black Cinema"	#N/A
2	"/m/018jz": "Baseball"	drama
3	"/m/0hj3mtp": "Animals"	drama
4	"/m/02n4kr": "Mystery"	fantasy
5	"/m/0hj3n26": "Family-Oriented Adventure"	drama
6	"/m/026ny": "Dystopia"	fantasy
7	"/m/01chg": "Bollywood"	#N/A
8	"/m/0qdzd": "B-movie"	#N/A
9	"/m/075fzd": "Social issues"	social
10	"/m/0604r_": "Fictional film"	fantasy
11	"/m/0b5_s_": "Prison escape"	thriller
12	"/m/082gq": "War film"	history
13	"/m/03p5xs": "Comedy of manners"	drama
14	"/m/05jhg": "News"	drama

2. 과제수행

장르 데이터 전처리

약 35000여개,

- 단어 소문자화
- 불용어 제거
- 길이 2자 이하 단어 제거

모든 장르 줄거리

```
# 토큰화+전처리(3) 전체 불용어 처리
# 전체 플롯
from tqdm import tqdm
all_vocab = {}
all_sentences = []
stop_words = set(stopwords.words('english'))

for i in tqdm(allplot):
    all_sentences = word_tokenize(str(i)) # 단어 토큰화를 수행합니다.
    result = []
    for word in all_sentences:
        word = word.lower() # 모든 단어를 소문자화하여 단어의 개수를 줄입니다.
        if word not in stop_words: # 단어 토큰화 된 결과에 대해서 불용어를 제거합니다.
            if len(word) > 2: # 단어 길이가 2이하인 경우에 대하여 추가로 단어를 제거합니다.
                result.append(word)
                if word not in all_vocab:
                    all_vocab[word] = 0
                all_vocab[word] += 1
    all_sentences.append(result)
```

```
100%|██████████| 35230/35230
```

김서영

2. 과제수행

장르 데이터 전처리

단어 빈도수 순으로 인덱스 부여, 15000위 미만 단어는 정보 삭제

```
all_vocab_sorted = sorted(all_vocab.items(), key = lambda x:x[1], reverse = True)
```

```
#전처리(4) 인덱스 부여
all_word_to_index = {}
i=0
for (word, frequency) in all_vocab_sorted :
    if frequency > 1 : # 정제(Cleaning) 챕터에서 언급했듯이 빈도수가 적은 단어는 제외한다.
        i=i+1
        all_word_to_index[word] = i
#print(all_word_to_index)

vocab_size = 15000 #상위 15000개 단어만 사용
words_frequency = [w for w,c in all_word_to_index.items() if c >= vocab_size + 1] # 인덱스가 15000 초과인 단어 제거
for w in words_frequency:
    del all_word_to_index[w] # 해당 단어에 대한 인덱스 정보를 삭제


all_word_to_index['OOV'] = len(all_word_to_index) + 1
```

```
7426), ('tells', 16926), ('man', 16022), ('life', 1528
14707), ('also', 14507), ('find', 14411), ('family', 1
, ('mother', 12706), ('police', 12480), ('goes', 1220
', 10972), ('wife', 10799), ('first', 10792), ('help',
friend', 9461), ('death', 9433), ('killed', 9316), ('b
3), ('car', 8863), ('friends', 8848), ('woman', 8818),
6), ('daughter', 8495), ('decides', 8437), ('soon', 84
ve', 8031), ('old', 7799), ('meanwhile', 7643), ('gir
eets', 7281), ('eventually', 7249), ('finally', 7125),
, 6864), ('end', 6754), ('return', 6749), ('town', 668
ecome', 6437), ('sees', 6391), ('world', 6384), ('woul
d', 6380), ('wants', 6259), ('dead', 6240), ('falls', 6239), ('arrives', 6216), ('makes', 6206), ('city', 6184), ('fight', 6125), ('e
ven', 6069), ('come', 6046), ('ends', 6039), ('local', 5851), ('husband', 5779), ('discovers', 5775), ('gives', 5751), ('around', 574
6), ('children', 5743), ('says', 5704), ('like', 5642), ('head', 5621), ('well', 5591), ('turns', 5563), ('meet', 5550), ('reveals',
5487), ('job', 5469), ('left', 5464), ('body', 5429), ('along', 5360), ('last', 5358), ('place', 5356), ('attempts', 5298), ('relatio
nship', 5271), ('movie', 5262), ('starts', 5260), ('war', 5215), ('dr.', 5196), ('kills', 5114), ('sister', 5107), ('lives', 5101),
('party', 5052), ('parents', 5039), ('runs', 5035), ('set', 5020), ('boy', 4957), ('order', 4938), ('team', 4894), ('leaving', 4809),
```

김서영

2. 과제수행

장르 데이터 전처리

장르별 데이터에 적용

로맨스 스릴러 ¶ 역사 사회 판타지 뮤지컬

```
# 토큰화+전처리(3) 전체 불용어 처리
# 전체 플롯
from tqdm import tqdm
Mu_vocab = {}
Mu_sentences = []
Mu_stop_words = set(stopwords.words('english'))

for i in tqdm(Mu):
    Mu_sentence = word_tokenize(str(i)) # 단어 토큰화를 수행합니다.
    result = []

    for word in Mu_sentence:
        word = word.lower() # 모든 단어를 소문자화하여 단어의 개수를 줄입니다.
        if word not in Mu_stop_words: # 단어 토큰화 된 결과에 대해서 불용어를 제거합니다.
            if len(word) > 2: # 단어 길이가 2이하인 경우에 대하여 추가로 단어를 제거합니다.
                result.append(word)
                if word not in Mu_vocab:
                    Mu_vocab[word] = 0
                Mu_vocab[word] += 1

    Mu_sentences.append(result)
```

```
100%|██████████| 307/307
```

김서영

2. 과제 수행

Train 데이터 인코딩, Test 데이터 인코딩

줄거리의 길이가 짧은 경우 우측 사진처럼 0으로 채움

```
print(X_train[6000])
```

```
[692, 9385, 2345, 110, 65, 7, 853, 2276, 4248, 3049, 72, 15001, 252, 8415, 1097, 650, 15001, 102, 2501, 6068, 4184, 15001, 1371, 15001,
15001, 8535, 15001, 6187, 256, 3049, 11382, 2933, 2704, 1655, 638, 3049, 15001, 3443, 5395, 7992, 6877, 4524, 1810, 5157, 5857, 15001,
15001, 7287, 15001, 6816, 15001, 15001, 9320, 15001, 58, 14362, 3443, 15001, 11256, 15001, 30, 3332, 15001, 15001, 80, 15001, 1
001, 15001, 1346, 4880, 647, 3049, 1208, 4407, 15001, 15001, 6326, 15001, 2273, 5330, 507, 1614, 15001, 93, 3147, 2273, 324, 3
1, 8463, 4088, 3917, 15001, 15001, 6192, 132, 14085, 12141, 1985, 3899, 7663, 6634, 15001, 931, 15001, 46, 1710, 2276, 5699, 1
0, 3443, 1104, 2095, 1313, 3407, 15001, 833, 15001, 2456, 221, 4125, 3049, 1451, 436, 15001, 15001, 11221, 3443, 1218, 8415, 4
9, 15001, 196, 27, 15001, 252, 3443, 80, 15001, 1363, 13056, 2456, 5536, 3443, 94, 3588, 15001, 7648, 6068, 3999, 7097, 15001,
911, 1196, 3917, 15001, 15001, 358, 15001, 199, 2373, 2456, 15001, 108, 15001, 1885, 325, 8256, 9376, 3443, 1371, 2276, 110, 11
5, 2213, 15001, 722, 26, 613, 15001, 1655, 2636, 15001, 103, 1655, 3443, 419, 812, 2276, 535, 3443, 15001, 1394, 15001, 9033, 3
4, 110, 1099, 1399, 306, 15001, 9076, 1150, 4638, 206, 94, 6738, 535, 221, 25, 13189, 1194, 15001, 977, 307, 877, 935, 2345, 34
15001, 5118, 1994, 15001, 102, 15001, 1473, 5395, 1371, 3443, 4253, 1473, 192, 15001, 3917, 1093, 15001, 31, 977, 563, 4655, 23
3, 148, 259, 427, 11903, 4751, 573, 15001, 1428, 603, 2933, 1521, 2345, 13264, 110, 1897, 2526, 1874, 1473, 374, 31, 9521, 1056
395, 1028, 1473, 46, 260, 5395, 199, 15001, 4677, 563, 10481, 11758, 3166, 1291, 15001, 1473, 10803, 1144, 2847, 94, 188, 199,
5, 138, 1440, 786, 3772, 199, 148, 1388, 10246, 15001, 409, 9047, 1619, 29, 12550, 5395, 94, 188, 9773, 80, 1, 3166, 1144, 2847
31, 1144, 94, 9552, 15001, 700, 15001, 412, 3166, 5395, 175, 31, 1619, 29, 1720, 1028, 1473, 275, 122, 3166, 3443, 80, 3166, 16
94, 484, 2456, 11355, 4861, 1119, 15001, 239, 3166, 2605, 4464, 1473, 9436, 3166, 1282, 35, 2890, 1266, 1026, 43, 1619, 29, 61,
49, 126, 3443, 2194, 7967, 3443, 239, 2933, 13394, 43, 3443, 5501, 1606, 7496, 15001, 9033, 2933, 3272, 920, 477, 1212, 2516, 2
2, 3443, 304, 168, 70, 15001, 7017, 382, 693, 1664]
```

```
print(Y_test[1])
print(X_test[1])
```

```
[1 0 0 0]
[    0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0 15001 11704   354   691   447  2738  2441
 15001    11   220   163   447   658 15001   996   261   792   313   329
    40    45    69  2738  4051   432  2157  4968    70  2738   119   440
    11   447  2674   119     8  2108   257   386  1785   784  2735   102
   467   688   379 13040 15001   789   279   226   265  1640    91  9333
   113    18]
```

2. 과제수행

각 데이터 길이 패딩

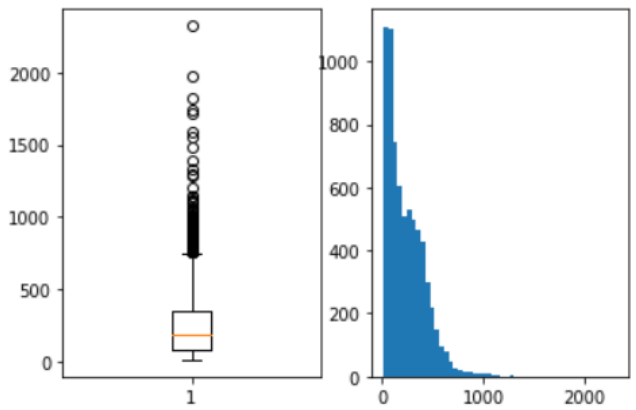
[학습 데이터]

```
import matplotlib.pyplot as plt
%matplotlib inline

len_result = [len(s) for s in X_train]
print("줄거리 최대 길이 : ",max(len_result))
print("줄거리 평균 길이 : ",sum(len_result)/len(len_result))

plt.subplot(1,2,1)
plt.boxplot(len_result)
plt.subplot(1,2,2)
plt.hist(len_result, bins=50)
plt.show()
```

```
줄거리 최대 길이 :  2324
줄거리 평균 길이 :  234.355
```



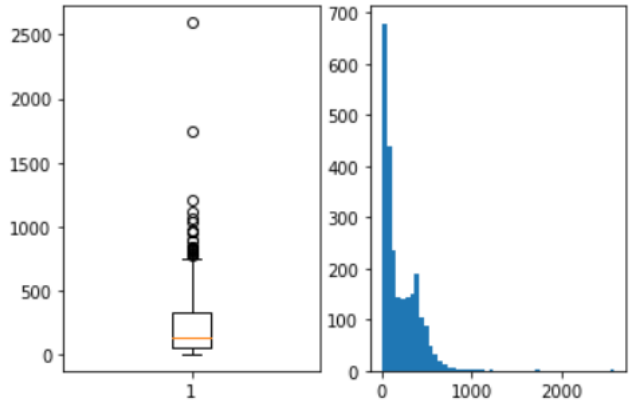
[테스트 데이터]

```
import matplotlib.pyplot as plt
%matplotlib inline

len_result = [len(s) for s in X_test]
print("줄거리 최대 길이 : ",max(len_result))
print("줄거리 평균 길이 : ",sum(len_result)/len(len_result))

plt.subplot(1,2,1)
plt.boxplot(len_result)
plt.subplot(1,2,2)
plt.hist(len_result, bins=50)
plt.show()
```

```
줄거리 최대 길이 :  2593
줄거리 평균 길이 :  203.25755102040816
```



양윤지

2. 과제수행

LSTM 모델 구현

4. LSTM 분류

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Embedding
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import numpy as np


M_test=Mu_encoded
M_test= np.array(M_test)
max_len = 230
X_train = pad_sequences(X_train, maxlen=max_len)
X_test = pad_sequences(X_test, maxlen=max_len)

model = Sequential()
model.add(Embedding(15002, 120))
model.add(LSTM(128))
model.add(Dense(4, activation='softmax'))

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('best_model.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)

X_train = np.array(X_train)
Y_train = np.array(Y_train)
X_test = np.array(X_test)
Y_test = np.array(Y_test)

model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['acc'])
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=7, batch_size=64, callbacks=[es, mc])
```

최대 줄거리 길이

Embedding 단어 수, 레이어

LSTM, softmax 사용, 4가지로 분류

2. 과제 수행

동작 영상

추가하기

2. 과제수행

Train 데이터 7000개, test 데이터 2450개
로맨스, 스릴러, 판타지, 기타 : 4가지 장르로 분류

```
Train on 7000 samples, validate on 2450 samples
Epoch 1/7
6976/7000 [============================>.] - ETA: 0s - loss: 1.2673 - acc: 0.4394
Epoch 00001: val_acc improved from -inf to 0.58816, saving model to best_model.h5
7000/7000 [==============================] - 55s 8ms/sample - loss: 1.2660 - acc: 0.4403 - val_loss: 1.0394 - val_acc: 0.5882
Epoch 2/7
6976/7000 [============================>.] - ETA: 0s - loss: 0.8254 - acc: 0.6832
Epoch 00002: val_acc improved from 0.58816 to 0.58898, saving model to best_model.h5
7000/7000 [==============================] - 54s 8ms/sample - loss: 0.8262 - acc: 0.6829 - val_loss: 1.0857 - val_acc: 0.5890
Epoch 3/7
6976/7000 [============================>.] - ETA: 0s - loss: 0.5897 - acc: 0.7931
Epoch 00003: val_acc improved from 0.58898 to 0.59388, saving model to best_model.h5
7000/7000 [==============================] - 55s 8ms/sample - loss: 0.5888 - acc: 0.7937 - val_loss: 1.0604 - val_acc: 0.5939
Epoch 4/7
6976/7000 [============================>.] - ETA: 0s - loss: 0.3968 - acc: 0.8635
Epoch 00004: val_acc did not improve from 0.59388
7000/7000 [==============================] - 52s 7ms/sample - loss: 0.3980 - acc: 0.8631 - val_loss: 1.2175 - val_acc: 0.5612
Epoch 5/7
6976/7000 [============================>.] - ETA: 0s - loss: 0.2513 - acc: 0.9160
Epoch 00005: val_acc did not improve from 0.59388
7000/7000 [==============================] - 52s 7ms/sample - loss: 0.2512 - acc: 0.9160 - val_loss: 1.4132 - val_acc: 0.5567
Epoch 00005: early stopping
```

```
Train on 7000 samples, validate on 2450 samples
Epoch 1/7
6976/7000 [============================>.] - ETA: 0s - loss: 1.2723 - acc: 0.4127
Epoch 00001: val_acc improved from -inf to 0.48571, saving model to best_model.h5
7000/7000 [==============================] - 66s 9ms/sample - loss: 1.2714 - acc: 0.4133 - val_loss: 1.2396 - val_acc: 0.4857
Epoch 2/7
6976/7000 [============================>.] - ETA: 0s - loss: 0.8693 - acc: 0.6637
Epoch 00002: val_acc improved from 0.48571 to 0.61755, saving model to best_model.h5
7000/7000 [==============================] - 70s 10ms/sample - loss: 0.8689 - acc: 0.6640 - val_loss: 0.9527 - val_acc: 0.6176
```

Train 데이터 정확도 91.6%

Test 데이터 정확도 61.7%

김서영

Thank You

감사합니다