

# 알고리즘 팀플

---

202135284 - Song Min Wi



# AlgorithmTermProject codeOrganizingDocument

## -Flow

1. 데이터 로드 및 전처리
2. 유사도 계산,
3. 그래프 생성
4. MST 생성
5. MST에서 엣지 제거하여 클러스터 형성
6. 클러스터 추출
7. 클러스터 활용하여 추천

# 데이터 로드 및 전처리

## - pandas 이용

: csv로 변경, 원-핫 인코딩, 결측치 제거, 유사한 태그를 하나로 정리

```
# 필요한 라이브러리 임포트
import pandas as pd
import numpy as np
from sklearn.neighbors import NearestNeighbors
from scipy.sparse import csr_matrix
from scipy.sparse.csgraph import minimum_spanning_tree, connected_components
import umap
import seaborn as sns
import matplotlib.pyplot as plt

# Step 1: CSV 파일 읽기
df = pd.read_csv('/content/영화목록_processed.csv')

# '장르_개수'와 '사회_통합' 컬럼 위치 변경
columns = list(df.columns)
idx1 = columns.index('장르_개수')
idx2 = columns.index('사회_통합')
columns[idx1], columns[idx2] = columns[idx2], columns[idx1]
df = df[columns]

# Step 2: 장르 벡터 추출
genre_columns = df.columns[3:58] # 'SF'부터 '로맨스통합'까지
genre_vectors = df[genre_columns].values
```

genre\_Column에 해당하는 value만 저장

```
print(genre_vectors)
```

✓ 0.0s

[[0 0 0 ... 0 0 1]  
[0 0 0 ... 0 0 0]  
[0 0 0 ... 0 0 1]  
...  
[0 0 0 ... 0 0 0]  
[0 0 0 ... 0 0 0]  
[0 0 0 ... 0 0 1]]

# 1. 차원 축소

```
# Step 3: UMAP을 사용하여 차원 축소
# n_components는 원하는 차원으로 설정 (예: 10)
reducer = umap.UMAP(n_components=5, random_state=42)
genre_umap = reducer.fit_transform(genre_vectors)
```

3~58에 해당하는 55차원 데이터를 -> 5차원으로 축소  
(데이터의 분석과 계산을 효율적이고 빠르게 해결하기 위함,  
이때 데이터 간의 관계를 최대한 보존)

# 2. 유사도 계산

```
# Step 4: KNN을 사용하여 유클리드 거리 기반의 유사도 계산
K = 1000 # 메모리 상황에 따라 조정 가능
nbrs = NearestNeighbors(n_neighbors=K, metric='euclidean', algorithm='auto').fit(genre_umap)
distances, indices = nbrs.kneighbors(genre_umap)
```

distances의

각 행은 각 영화마다 k개의 최근접 이웃에 대한 거리 값을 나타

indics는

각 영화마다 k개의 최근접 이웃에 대한 인덱스를 나타냄

distances = [  
[0.1, 0.15, 0.2, ...], # 첫 번째 포인트와 1000개의 이웃 거리  
[0.05, 0.12, 0.18, ...], # 두 번째 포인트와 1000개의 이웃 거리  
...]

• 유클리드 거리 공식:

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2}$$

k개의 최근접 이웃

거리 계산 방식: 유클리드 거리

최근접 이웃 검색 알고리즘: auto -> 자동

# 3. 그래프 생성을 위한 간선 리스트 생성

```
# Step 5: 간선 리스트 생성
num_movies = genre_umap.shape[0] #genre_umap의 행 개수를 가져옴
edges = []

for i in range(num_movies):
    for j in range(K):
        neighbor = indices[i][j]
        distance = distances[i][j]
        if i < neighbor: # 중복 간선 방지
            edges.append((i, neighbor, distance))

# 간선 리스트를 DataFrame으로 변환
edges_df = pd.DataFrame(edges, columns=['node1', 'node2', 'weight'])
```

edge의 저장 형태 (노드1, 노드2, 거리)

이중 루프를 통해 모든 영화에 대한 k개의 간선 생성

이 함수는 그래프에 가중치(weight)가 포함된 간선을 추가.

- 예: (0, 1, 0.3) → 노드 0과 1 사이에 가중치 0.3인 간선을 추가.

# 4. 그래프 생성 및 Kruskal 알고리즘을 사용해 MST제작

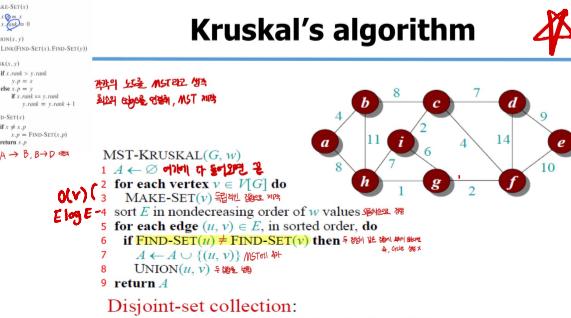
```
# Step 6: Kruskal 알고리즘을 사용하여 MST 생성
# 그래프 생성
G = nx.Graph()
G.add_weighted_edges_from(edges)

# Kruskal 알고리즘을 사용하여 MST 계산
mst = nx.minimum_spanning_tree(G, algorithm='kruskal')
```

edges를 참조하여 그래프 생성

NetworkX 라이브러리를 사용해 Kruskal 알고리즘을 사용

### Kruskal's algorithm



### Kruskal's algorithm (모든 가중치 정렬 사전에 피기)

- 그래프 내의 모든 간선을 가중치 오름차순으로 표기
- 간선 목록을 차례로 MST에 추가, But 사이클은 생김 X
- 사이클 방지를 위한 방법 → a forest of single node trees
- 각 정점이 각각의 분리집합이라 생각, 연결되면 합침 수행
- 이때, 연결된 두 정점이 같은 집합에 속하면 사이클 형성
- 예) A와 D를 연결, B가 A, C가 D에 속하는 경우 A, B, C, D는 같은 집합이므로 사이클 생김
- 시간 복잡도:  $O(E \log V) = O(E \log E)$
- 사이클 형성 피함

## 5. MST에서 특정 거리 이상의 엣지를 제거

```
# Step 7: 엣지 제거를 통한 클러스터 형성
# 엣지 가중치 추출
edge_weights = np.array([data['weight'] for u, v, data in mst.edges(data=True)])

# 거리 역치값 설정 (예: 상위 1% 거리의 엣지 제거)
threshold = np.percentile(edge_weights, 99)

# 역치값 이상의 엣지 제거
edges_to_remove = [(u, v) for u, v, data in mst.edges(data=True) if data['weight'] > threshold]
mst.remove_edges_from(edges_to_remove)
```

edge\_weights에 MST에 모든 edge의 가중치를 저장

그 중 상위 1%에 해당하는 edge를 threshold에 저장

threshold보다 큰 가중치의 edge list를 변수에 저장하고, 제거.

- 엣지 가중치가 [0.2, 0.5, 1.3, 2.1, 3.5]라면:
- 99번째 백분위값 → threshold = 3.5.

## 6. 제거된 edge를 바탕으로 클러스터 찾기

```
# Step 8: 연결된 구성 요소(클러스터) 찾기
clusters = list(nx.connected_components(mst))
labels = np.zeros(num_movies, dtype=int)

for cluster_id, cluster in enumerate(clusters):
    for node in cluster:
        # 하나의 클러스터를 끼내고, 인덱스 할당
        labels[node] = cluster_id + 1

# 클러스터 수 출력
n_components = len(clusters)
print(f"클러스터 수: {n_components}")

# 클러스터 레이블을 데이터프레임에 추가
df['cluster'] = labels
```

connected\_components = edge로 연결된 노드들의 집합

clusters = [{0, 1, 2}, {3, 4, 5}, {6, 7}]

labels는 각 영화 개수만큼 0으로 초기화된 배열  
-각 영화의 클러스터 ID를 저장

labels = [0, 0, 0, 1, 1, 1, 2, 2]

## 7. 클러스터링 결과 시각화 분석



현재 그래프는 총 443개의 클러스터를 포함

하지만 색상의 제한으로 인해, 서로 다른 클러스터임에도 일부가 같은 색상으로 표현되었다.

클러스터의 크기는 클러스터에 해당하는 영화의 개수를 의미

해석해볼점은 2가지

큰 클러스터:

점들이 조밀하게 모여있고, 경계가 흐릿한 것을 볼 수 있다.  
이는 여러 장르가 혼합된 영화들이 포함되었거나, 장르의 분포가 폭넓은 영화들일 가능성이 높다.

작은 클러스터:

점들이 고립되어 있거나, 소규모로 모여 있는 것이 특징이다.  
이는 특정한 속성이나 장르를 강하게 공유하는 영화들이 포함되었을 가능성이 높다.

## 8. 프론트 제작을 위한 csv파일 제작

```
# Step 10: 클러스터 요약 정보 CSV로 저장
# 장르 컬럼 이름 리스트 생성
genre_columns = df.columns[3:58]

# 클러스터 고유 값 추출
cluster_labels = sorted(df['cluster'].unique())
    # 예: [0, 1, 2, ..., 442]
    # 데이터프레임에서 유일한 클러스터 ID를 정렬된 리스트로 반환.

# 클러스터 정보를 저장할 리스트 초기화
cluster_info_list = []

# 상위 N개의 특징적인 태그 선택
top_N = 5 # 필요에 따라 변경 가능

# 각 클러스터에 대해 반복
for cluster_label in cluster_labels:
    # 해당 클러스터의 데이터 선택
    cluster_data = df[df['cluster'] == cluster_label]
    # 해당 클러스터에 해당하는 데이터 선택

    # 영화의 행 번호 리스트 추출
    movie_indices = cluster_data.index.tolist()

    # 장르 컬럼의 합계를 계산하여 특징적인 태그 파악
    tag_sums = cluster_data[genre_columns].sum()

    # 합계가 높은 상위 N개의 태그 선택
    top_tags = tag_sums.nlargest(top_N).index.tolist()
```

```
# 클러스터 정보를 딕셔너리에 저장
cluster_info = {
    'cluster': cluster_label,
    'movie_indices': ','.join(map(str, movie_indices)),
    'top_tags': ','.join(top_tags)
}

# 리스트에 추가
cluster_info_list.append(cluster_info)

# (variable) cluster_info: dict[str, Any]

# 클러스터 정보 리스트를 DataFrame으로 변환
cluster_info_df = pd.DataFrame(cluster_info_list)

# 열 순서 조정
cluster_info_df = cluster_info_df[['cluster', 'movie_indices', 'top_tags']]

# CSV 파일로 저장
cluster_info_df.to_csv('cluster_summary.csv', index=False)
```

각 클러스터에서 N개의 상위 태그