Term Project

2016147538 임재훈

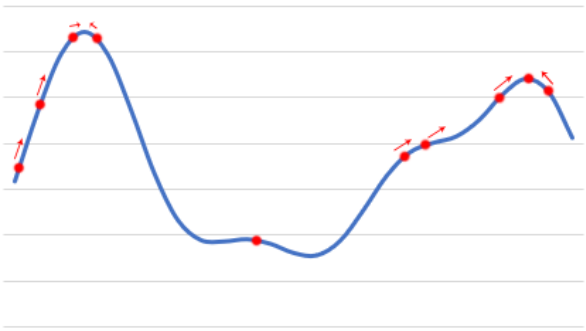
Contents

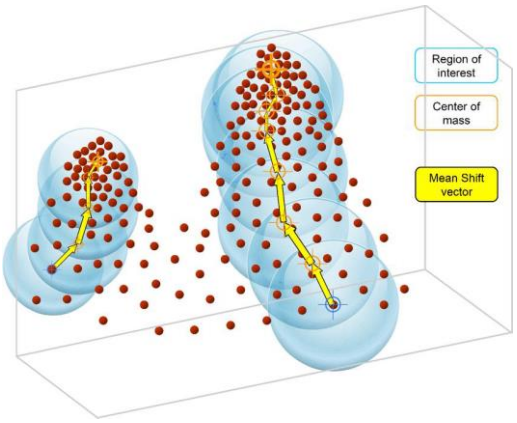
Phase Ⅰ. Geographical clustering

1. Clustering Method

Photo의 Geo-coordinate 정보를 이용한 Geographical clustering을 수행하기 위해 여러 가지 clustering 방법 중에서 MeanShift 방법을 사용했다.

MeanShift 방법의 경우, 현재 몇 개의 landmark(cluster)가 존재하는지 알 수 없는 상황에서, K-mean clustering과 같은 방법을 사용할 때와 같이 cluster의 개수를 정해줄 필요가 없다는 장점이 있어, 이 방법을 사용했다. 또한 MeanShift algorithm은 data에 대한 density function의 maxima를 shifting하면서 converge할 때까지 이를 반복하여 clustering을 수행하는 알고리즘이다. 이를 통해 data가 밀집된 center를 찾을 수 있다.





2. Implementation

Clustering의 경우, Scikit learn의 MeanShift 내장 함수를 활용하여 구현하였다.

```
[ ]   tag = pd.read_csv(loc1, names=['photo_id','label_num','label_name'])
      photo = pd.read_csv(loc2, names=['photo_id', 'user_id', 'geo_latitude','geo_longitude', 'taken_time','None'] )
      dataset = photo[['photo_id','geo_latitude','geo_longitude']]

[ ]   X = np.array(dataset[['geo_latitude','geo_longitude']]) # Make training set with geo coordinate
      bandwidth = estimate_bandwidth(X, random_state=2021, n_samples=10000) # calculate standard bandwidth value

[ ]   bandwidth

      0.06600568106466975
```

MeanShift clustering을 수행하기 위한 Dataset으로 geo-coordinate가 활용되므로, photo.csv 파일의 데이터에 있는 각 사진의 geo_coordinate를 활용하였다.

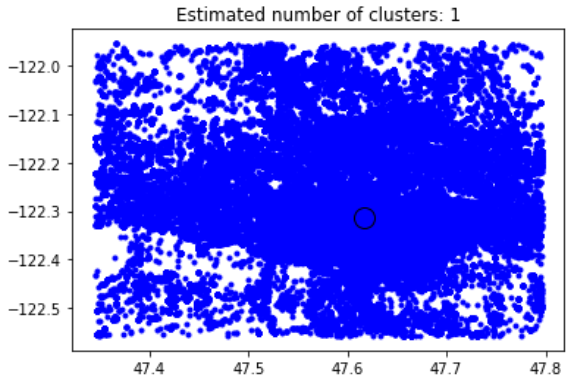
또한 MeanShift algorithm의 경우 clustering을 수행하는 bandwidth가 중요한데, 이를 위해 Scikit learn에서 제공하는 estimate_bandwidth 함수를 활용하여 dataset을 이용하여 dataset에 적합한 bandwidth를 계산하였다. Bandwidth를 계산하는 데에 $O(n^2)$가 소요되므로 전체 dataset을 이용하지 않고, dataset에서 sample을 추출하여 이를 통해 bandwidth를 계산하였다. 또한 이 bandwidth의 값을 기준으로 조금씩 조정하면서 clustering을 시도했다.

```
[ ]   clustering = MeanShift(bandwidth=bandwidth, bin_seeding=True) # Make model
      clustering.fit(X) # Train model
      labels = clustering.labels_ # label of input
      centers = clustering.cluster_centers_ # coordinate of cluster centers
      labels_unique = np.unique(labels) # set of labels
      n_clusters = len(labels_unique) # number of clusters
      len(labels) # total input size (# of coordinates)

      991766
```

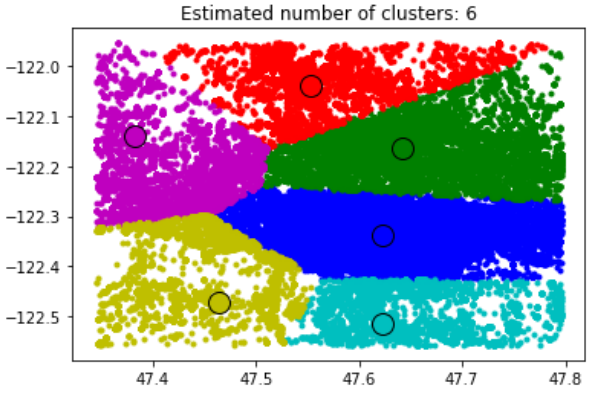
Clustering의 경우, MeanShift clustering model을 만들고, 이를 geo_coordinate dataset을 이용하여 training하면서, dataset에 대한 clustering을 수행하도록 하였다. 마찬가지로 MeanShift를 수행하는데 $O(n^2)$이 소모되므로, Dataset 전체를 initial point로 사용하여 이 작업을 수행하기에는 너무 많은 시간이 소모되므로, bin_seeding 옵션을 사용하여 speedup하였다. 또한 bandwidth를 조정하면서 다양한 결과를 관측하였다.

```
[ ]    # Draw figure
       import matplotlib.pyplot as plt
       from itertools import cycle

       plt.figure(1)
       plt.clf()

       colors = cycle('bgrcmykkbgrcmyk')
       for k, col in zip(range(n_clusters), colors):
           my_members = labels == k
           cluster_center = centers[k]
           plt.plot(X[my_members, 0], X[my_members, 1], col + '.')
           plt.plot(cluster_center[0], cluster_center[1], 'o', markerfacecolor=col,
                    markeredgecolor='k', markersize=14)
       plt.title('Estimated number of clusters: %d' % n_clusters)
       plt.show()
```

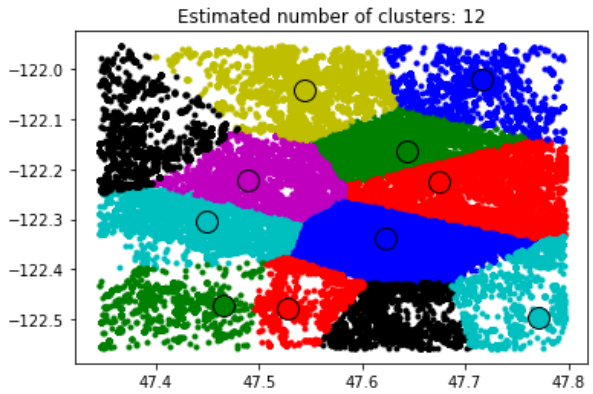
앞서 계산한 clustering을 시각적으로 확인할 수 있도록 cluster마다 다른 색상을 부여하여 2차원 좌표상으로 표시하도록 하였다.



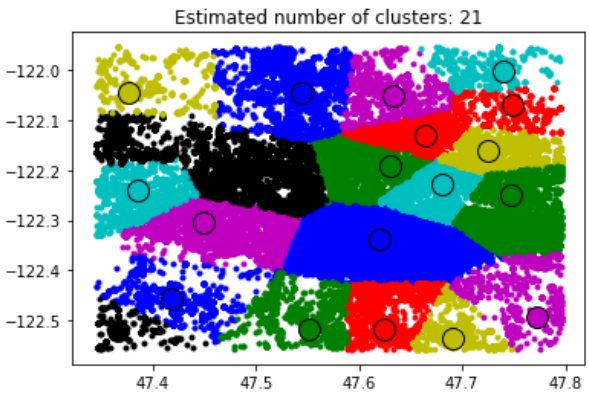
[Fig.1]



[Fig.2]



[Fig.3]



[Fig.4]

또한 bandwidth의 조정에 따른 clustering의 결과를 시각적으로 확인하면 위와 같다.

그림 상으로 확인할 수 있듯이, 너무 큰 bandwidth(=2)를 설정한 경우, **[Fig.1]**과 같이 clustering이 제대로 수행되지 않음을 확인할 수 있고, 너무 작은 bandwidth(=0.05)를 설정한 경우, **[Fig.4]**와 같이 clustering이 너무 자세하게 수행됨을 확인할 수 있다.

따라서, bandwidth(=bandwidth)를 사용한 **[Fig.2]**의 경우와 bandwidth(=0.06)을 사용한 **[Fig.3]**의 경우에 대해서만 clustering의 결과에 대한 분석을 수행하고자 계획하였다.

```
import shutil
import os
source = '/content/drive/MyDrive/Datas/in/' # source position
target_position = '/content/drive/MyDrive/Datas/out2/' # destination position
ids = list(dataset['photo_id']) # photo_id list of total data
filelist = os.listdir(source) # get total image file list
```

shutil을 활용하여 사진을 cluster별로 분류하여 복사하여 저장하도록 하였다.

```
total_file = []
for i in range(len(filelist)):
  total_file.append(filelist[i].split('.')[0]) # remove .jpg
total_file = [int(x) for x in total_file] # str to int

for i in range(len(ids)): # copy photos to each class directory
  if ids[i] in total_file:
    shutil.copy(source + str(ids[i]) + '.jpg', target_position + str(labels[i] + 1))
```

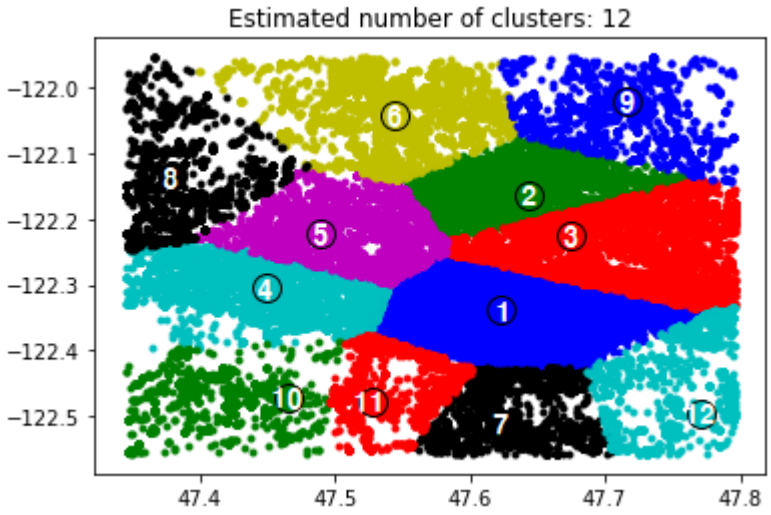
파일의 개수를 비교하여 사진의 복사가 제대로 됐는지 검증 작업도 수행하였다.

```
len(filelist)

sum = 0
for i in range(n_clusters):
  temp = os.listdir(target_position + str(i+1))
  sum += len(temp)
print(sum)
```

3. Result analysis & Conclusion

✓ Bandwidth=0.06, 12 clusters

이 경우, 아래와 같이 12개의 class로 분리되었는데, 각 그룹을 살펴보면 다음과 같다.



Class1의 경우, 뮤지컬, 럭비, 축구 경기장, 빌딩, 마을 축제, 스케이트 경기장, 타워, 선박, 바닷가, 시장, 야구장, 공연장, 수목원, 수족관, 박물관, 기차역, 미술관, 코스프레 사진과 같이 다양한 내용들로 구성되어 있었고, 시위 현장이나 개인의 사진과 같은 의미 없는 정보도 많이 존재했다.

Class2의 경우, 미술관의 사진도 일부 있었지만, 가족 사진, 시위 사진과 같은 의미 없는 정보들이 주를 이루었다.

Class3의 경우, 클럽의 사진, 럭비 경기 사진도 있었지만, 개인적인 사진도 많이 있었다.

Class4의 경우, 비행기 박물관, 동물탈 코스프레 사진 등이 주를 이루었다.

Class5의 경우, 공원, 바닷가의 사진이 주를 이루었다.

Class6의 경우, 럭비경기 사진이 주를 이루었다.

Class11의 경우, 화물 크레인의 사진이 주를 이루었다.

Class7, 9, 12의 경우, 의미 없는 개인적인 사진으로 이루어져 있었다.

Class8, 10의 경우 해당하는 사진이 존재하지 않았다.

처음에는 6개의 cluster로 분류한 경우도 분석할 계획이었으나, 12개의 cluster로 분류했을 때에도 충분히 만족할만한 clustering이 되지 않았기 때문에, 더 큰 규모로 clustering한 경우는 분석에서 배제하였다.

위와 같이 분류한 사진을 이용하여 각 class의 사진 구성을 분석한 결과, 대부분의 사진들이 class1으로 분류되었고, 그에 따라서 많은 종류의 사진들이 존재했고, landmark의 candidate가 될 가능성이 높은 빌딩, 타워, 축구 경기장 등과 같은 후보군의 사진 또한 많이 존재했다.

그에 비해 다른 class들의 경우, landmark를 어떻게 정의할 것이냐에 따라서 달라지긴 하겠지만, landmark의 candidate가 될 가능성이 높은 의미 있는 종류의 사진이 거의 존재하지 않았다.

또한, class3에서 유효한 럭비경기 사진이 class1에서 발견되는 것과 같은, 한 class에서는 유효한 사진이 다른 class에서 outlier로 발견되는 현상을 확인할 수 있었다. 이를 통해 강의에서 배운 geo-coordinate를 이용한 clustering의 한계점을 확인할 수 있었다. 물론, 이를 clustering이 완벽하지 않아서 발생한다고도 판단할 수도 있다.

추가적으로, geo-coordinate를 이용한 clustering을 통해서 분류한 outlier class들의 data point들을 remove함으로써 dataset을 더 refine하고 scaling할 수 있다는 장점도 있다고 생각한다. 또한 refine한 dataset을 다시 더 작은 규모(더 작은 bandwidth)로 clustering을 수행하는 방법도 생각해 볼 수 있을 것이다. 이 방법을 통해서 현재의 class1을 더 작은 규모의 여러 cluster로 나눌 수도 있을 것이다.

결론적으로 Geo-coordinate 만을 이용한 clustering으로는 landmark의 사진과 landmark에서 찍은 landmark와는 관계 없는 사진이 동일한 가치를 지닌다는 한계 때문에 완벽한 clustering이 불가능하고, 동일한 landmark라도 사진을 찍은 위치에 따라서 다른 cluster로 분류 될 수 있다는 문제점이 있고, landmark 간의 물리적 거리가 가까우면 이를 구분하기 어렵다는 문제점이 있다. 따라서, geo-coordinate 이외에 추가적인 다른 방법을 활용하여 outlier image의 filtering과, near class 간의 cluster reorganize를 할 필요가 있으며, clustering을 통해 얻은 landmark의 candidate가 다수 존재하는 class1을 다시 더 작게 clustering할 필요가 있다고 생각한다.

Phase Ⅱ. Visual clustering

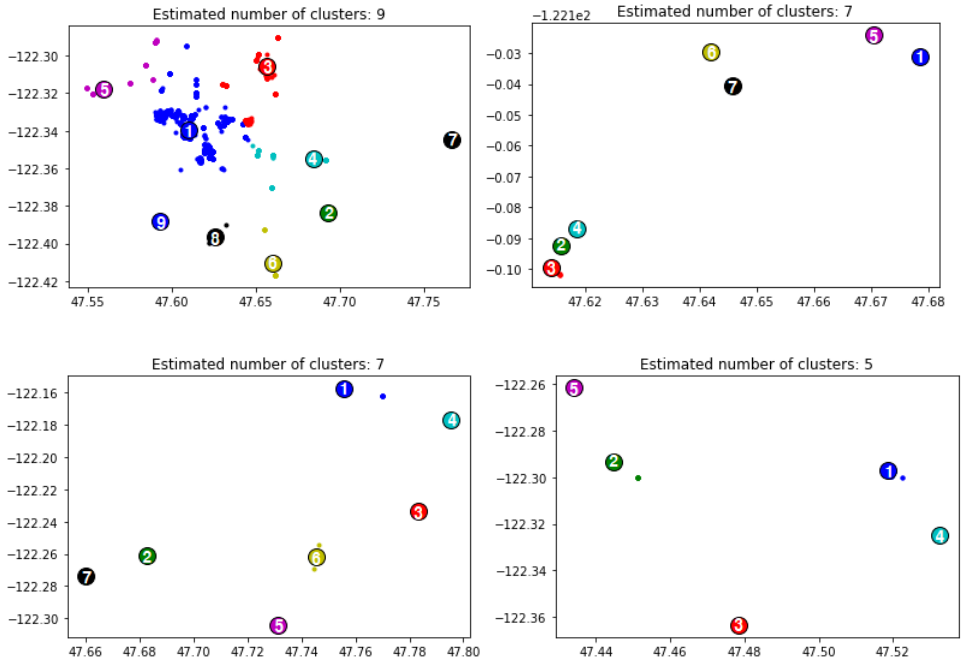
0. Preprocess

Phase Ⅰ에서 수행한 geo-cluster를 분석한 결과를 이용하여, 다음과 같은 outlier class들을 discard하였다.

- Class7, 9, 12 개인적인 사진
- Class11. 화물 크레인 사진
- Class8, 10. 해당하는 사진 없음

또한 O(n^2)의 수행시간을 소모하는 visual cluster를 수행하기에는 각 class의 image가 너무 많으므로, 6개의 각 class에 대해 cluster에 포함된 사진이 너무 많은 경우, 사진의 geo-coordinate를 이용한 clustering을 다시 실시하여 더 작은 subclass들로 나누었고, 각 class의 outlier class들을 discard하여 scaling 과정을 수행하였다. 또한 Csv data를 이용한 clustering의 generality를 잃지 않기 위해 cluster를 top-down 방식으로 나누었다.

Class1, 2, 3, 4에 대해 수행하였고, 그 결과는 아래와 같다.



새롭게 분류된 각 class의 cluster들을 분석하였고, cluster의 사진의 개수가 너무 적거나, landmark와 관련이 있다고 보기 너무 어려운 사진으로만 구성된 outlier class들을 discard하였다.

Class1의 경우 clustering을 진행한 이후에도 여전히 한 cluster가 많은 사진으로 구성되어 있었고, 이를 scaling하기 위해 geo-clustering을 bandwidth를 줄여가며 더 작은 규모에서 반복해서 진행하여 더 작은 cluster들로 나누며 outlier class를 찾아 discard하는 작업을 수행했다.

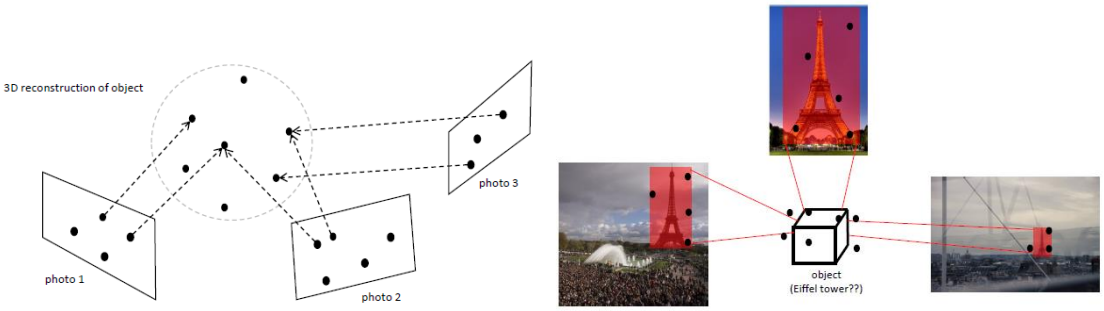
Geo-coordinate를 이용한 clustering이 충분하여 visual clustering이 필요 없는 경우나, Geo-coordinate clustering을 통해서 분류된 cluster가 너무 많은 사진을 포함한 경우(100장 이상), 따로 visual clustering을 수행하지 않았고 추후에 textual clustering을 진행하여 scaling을 진행한 후 필요하면 visual clustering을 진행할 예정이다. outlier의 경우 landmark일 가능성이 전혀 보이지 않는 경우와, 사진 상으로는 landmark일 가능성이 있지만, 그 사진의 개수가 현저히 적은 경우, 그 사진들의 모음이 outlier가 아닌 다른 cluster가 존재할 것으로 판단하고 현재 cluster를 outlier로 판단하고 discard를 진행하였다.

Preprocess 과정을 통해 최종적으로 다음과 같은 cluster를 얻었다.

Class				# photos	visual clustered?	kind	remarks
1	1	1	1	629	too large	mixed	# of photos does not include the outlier images
			2	245	too large	mixed	
			3	111	no need	library	
			4	32	no need	stairs	
		2	1	348	too large	mixed	
			2	213	too large	mixed	
			3	80	no need	artwork	
		3	1	281	no need	soccer stadium	
			2	189	no need	baseball stadium	
			3	76	o	mixed	
			4	55	no need	station	
		4	-	275	too large	mixed	
		5	-	43	no need	concert	
		6	-	10	no need	city view	
	3	1	-	97	o	mixed	
		2	-	97	no need	arboretum	
		3	-	64	o	mixed	
		4	-	36	no need	athletic stadium	
		6	-	13	no need	small concert	
	4	-	-	127	o	mixed	
	5	-	-	72	o	mixed	
	6	-	-	20	o	mixed	
2	3	-	-	28	o	art center	
3	2	-	-	71	o	concert	
4	1	-	-	254	o	airplane museum	
	2	-	-	170	o	cosplay	
	3	-	-	61	no need	beach	
5	-	-	-	73	no need	park	
6	-	-	-	61	o	football	
			Total	3831			

1. Clustering Method

Visual clustering을 수행하기 위하여 Microsoft Bundler라는 프로그램을 사용하였다. 이를 통해 Phase Ⅰ과 preprocess 과정에서 구분한 geo-cluster 내에서의 visual clustering을 진행하였다. SIFT를 통해 추출한 image의 key point를 이용해, image간의 shared object를 찾아 adjacency graph의 edge를 저장한 pairwise output을 얻었고, 이를 통해 adjacency graph를 구성하여 각 visual cluster를 구성하였다.



2. Implementation

Visual clustering을 수행하여 얻은 파일인

List_tmp.txt와 pairwise_scores.txt를 이용하여 다음과 같이 각 visual cluster를 계산하였다.

```
names = []
name = open('/content/list_tmp.txt', "r")
lines = name.readlines()
for line in lines:
  tmp = (line.split('/')[8]).split()[0]
  names.append(tmp)
name.close()
```

List_tmp.txt를 읽어 해당 cluster에 속한 사진 파일의 이름을 저장한다.

```
adjacents = []
adjacent= open('/content/pairwise_scores.txt', "r")
lines = adjacent.readlines()
lst=[]
for line in lines:
  i = int(line.split()[0])
  j = int(line.split()[1])
  if len(lst) == 0:
    lst.append(set({i,j}))
  else:
    temp = {}
    p = {i,j}
    merged = False
    for idx, s in enumerate(lst):
      if p.isdisjoint(s):
        continue
      else:
        if not merged: # merge and save the set
          merged = True
          s.update(p)
          temp = s
        else: # if new set connects two different set
          temp.update(s)
          lst.pop(idx)
          break
    if not merged:
      lst.append(p)
```

Pairwise_scores.txt 파일은 graph의 node를 연결하는 edge를 저장하고 있으므로, 해당 edge가 node를 connected component를 구성하는 유일한 edge일 경우 새로 추가하고, 아닐 경우 해당 edge가 connected component를 연장하는 경우 그래프를 연장하고, 만약 edge가 두 connected component를 연결할 경우 이를 합치도록 구현하였다.

```
for i, item in enumerate(lst):
  indices = list(item)
  for idx in indices:
    shutil.copy(source + names[idx], target_position + str(i+1) + '/')
    #print(source + names[idx])
    #print(target_position + str(i+1) + '/')
```

위에서 분류한 connected component가 저장된 list를 활용하여 파일을 복사하였다.

3. Result analysis & Conclusion

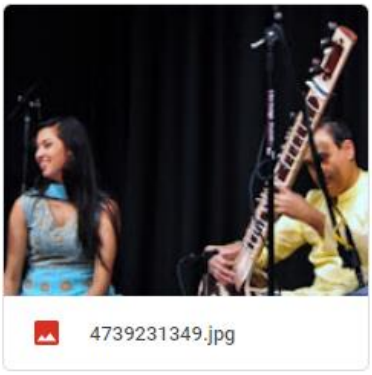
class	# photos	# clusters	avg # of photos
1-1-3-3	76	14	5.428571429
1-3-1	97	5	19.4
1-3-3	64	8	8
1-4	127	38	3.342105263
1-5	72	22	3.272727273
1-6	20	6	3.333333333
2-3	28	6	4.666666667
3-2	71	16	4.4375
4-1	254	22	11.54545455
4-2	170	31	5.483870968
6	61	16	3.8125

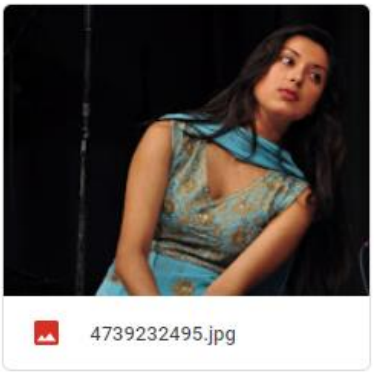
Visual clustering을 진행한 결과 위와 같은 결과를 얻었다.

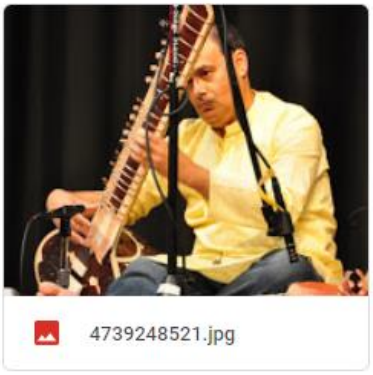
분류된 평균 사진의 개수를 보면 알 수 있듯이, 대부분의 visual clustering의 결과가 평균 3~4장 정도의 cluster로 분류되는 것을 확인할 수 있다. 하지만, 이 각각의 cluster를 하나의 landmark를 구성하는 cluster라고 보기는 어렵다. 하지만 어느 cluster에도 속하지 않은 사진들을 filtering 함으로써 outlier를 제거하는 효과를 낼 수 있다.

또한 위와 같은 결과가 나온 이유를 추측하기 위해, visual clustering을 수행한 cluster 중에서 비슷한 사진수에도 불구하고 현저한 cluster당 사진수의 차이를 가지는 **1-1-3-3**과 **1-3-1**의 결과를 대표로 비교해서 분석해본 결과는 다음과 같다.

1-1-3-3의 경우, 비록 landmark와 관련이 없는 outlier image여서 discard의 대상이 되었지만, 아래와 같이 두 번째와 세 번째 사진의 object를 공통적으로 가지고 있는 첫 번째 사진을 통해 하나의 cluster로 분류된 것을 확인할 수 있다.

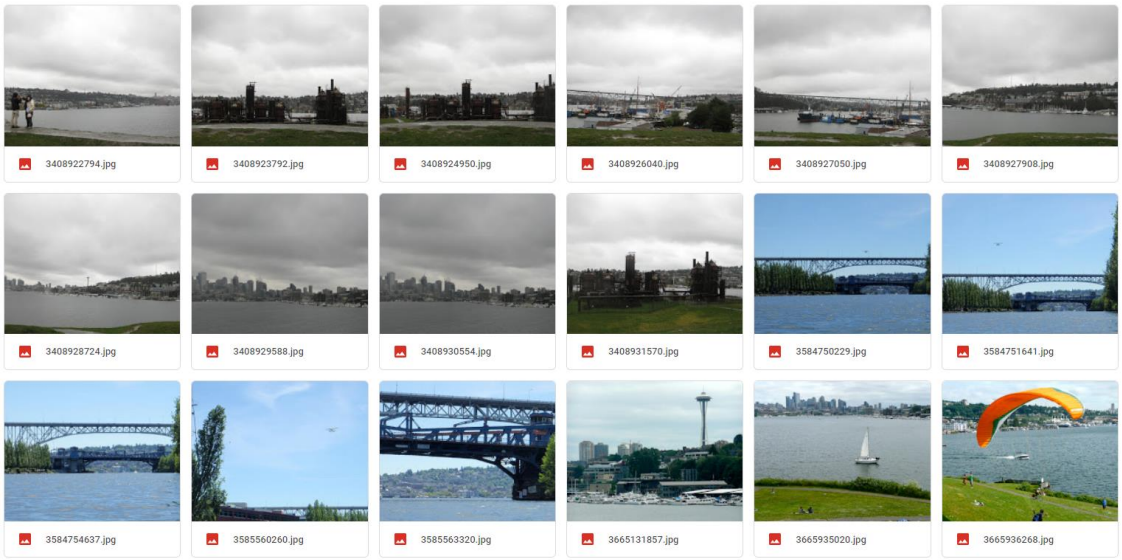


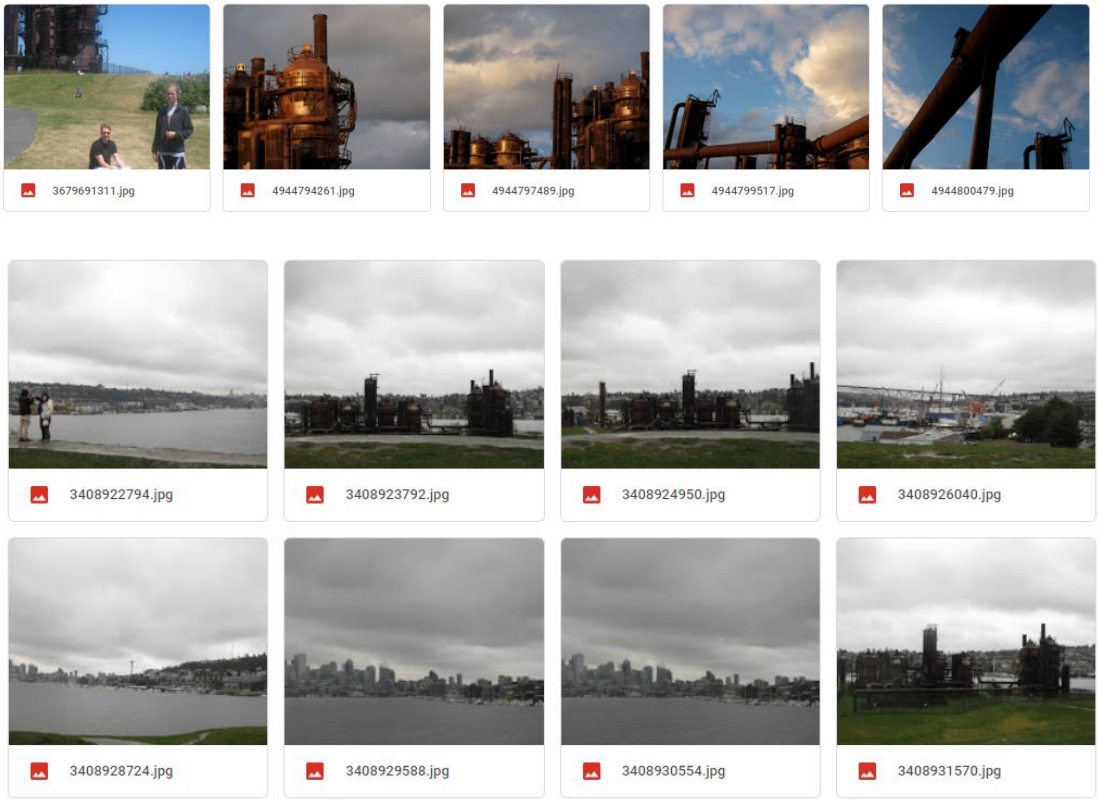




하지만, 왼쪽의 두 사진과 오른쪽의 사진이 다른 cluster로 분류된 것처럼, 분명히 같은 장소인 것을 알 수 있지만, 사진에서 보여지는 시각적인 특징에만 집중하기에 색상에 따라서 다른 cluster로 분류될 수 있는 문제점이 있을 것이라고 추측할 수 있다. 이러한 점은 같은 landmark의 사진이라도 날씨에 따라 또는 낮에 촬영한 사진과 밤에 촬영한 사진을 다르게 분류할 수 있다는 문제점을 야기할 수 있다.

1-3-1의 경우, 인물 사진이 주를 이루었던 위의 경우와는 다르게 풍경 사진이 주를 이루었고, visual clustering도 비슷한 풍경 사진을 우수하게 같은 cluster로 분류한 것을 확인할 수 있다. 이를 통해 visual clustering의 효과를 확인할 수 있었다.





하지만 위 그룹과 아래 모두 같은 물체를 촬영한 사진을 포함하고 있음에도 불구하고, 그 밝기에 따라서 다른 물체로 분류하여, 다른 그룹으로 분류된 것을 확인할 수 있었다. 이를 통해서 위에서 예측했던, 날씨나 시간에 따라 같은 물체를 다르게 분류할 수 있다는 문제점을 확인할 수 있었다.

위와 같은 두 case study를 통해서 visual clustering의 효과뿐만 아니라, 단점도 확인할 수 있었다. 확인한 Visual clustering의 단점인 dataset의 quality에 따라 cluster의 scale이 영향을 받을 수 있다는 점, 사진의 색상에 큰 영향을 받는다는 점, 또한 위 case study에서 직접적으로 확인하기는 어렵지만 사진만을 이용한 clustering을 진행하기 때문에, 사진 그 자체에만 집중하여, 맥락을 통해 파악을 하는 것이 불가능하다는 단점이 있다. 예를 들어, landmark 실내의 사진과 landmark 외부의 사진을 같은 cluster로 판단하지 못하게 되는 경우가 발생할 수 있다. 따라서 맥락을 통해 clustering을 수행할 수 있는 textual data를 이용한 cluster의 concatenation이나 division을 통해 보완작업이 필요할 것이다.

또한 bundler 프로그램이 완벽하게 작동하지 않는 경우도 있었기에 visual clustering의 결과를 무조건 맹신하기 보다는 그 결과에 보수적으로 접근하여 cluster를 판단할 필요도 있다.

Phase Ⅲ. Textual clustering and creative ideas

1. Clustering Method

Phase Ⅱ를 통해서 visual clustering 을 진행했지만, visual clustering 의 경우 너무 자세히 clustering 이 되는 경향이 있다는 단점이 있었다. 이는 visual clustering 이 색상에 영향을 받는다는 점, 사진에서 얻는 특징 이상의 맥락을 파악할 수 없다는 점 때문이라고 예상했었다. 또한 visual clustering 의 경우 많은 시간이 소요되기 때문에 큰 규모의 cluster 에서는 visual cluster 를 수행할 수 없었다. 이러한 visual clustering 의 문제점을 보완하기 위해 먼저, Tag.csv 파일에 담긴 'tag' 정보를 활용하여 너무 작게 쪼개진 visual cluster 의 경우 concatenate 하고, 규모가 너무 커서 visual clustering 을 하지 못한 cluster 의 경우 division 을 수행하였다. 너무 작은 규모의 cluster 가 생성될 경우 cluster 를 다른 cluster 와 합쳤고, 너무 큰 규모의 cluster 가 생성될 경우 textual clustering 을 다시 실행하여 더 작게 나누었다.

또한, 위 작업을 통해 분류한 최종 cluster 가 landmark 인지 아닌지를 판단하기 위해 Photo.csv 파일에 담긴 'user_id' 정보를 활용하였다. 한 인물이 한 장소에서 사진을 여러 장 촬영했다고 해서 그것이 landmark 라고 볼 수는 없기 때문에 이러한 경우를 걸러내기 위해, landmark 의 candidate 의 경우 다양한 인물이 촬영한 사진이 있어야 한다는 가정에서 출발한, cluster 내에 최소 3 개의 다른 user_id 가 존재해야 된다는 기준을 세워 landmark 후보군을 줄였다.

2. Implementation

```
[ ]   # Filter dataset
      Tag = tag_dataset[tag_dataset['photo_id'].isin(total_file)]
      Photo = photo_dataset[photo_dataset['photo_id'].isin(total_file)]

[ ]   tags = list(Tag['photo_id'])
      labels = list(Tag['label_name'])

[ ]   # Make {photo_id : [label_names]} dictionary
      tag_label_dict = {}
      iter = 0
      while iter < len(tags):
        id = tags[iter]
        temp = []
        while (iter < len(tags)-1) and (tags[iter+1] == id):
          temp.append(labels[iter])
          iter += 1
        tag_label_dict[id] = temp
        iter += 1
```

Tag.csv 파일 중 사진 파일에 대한 tag 정보만을 filtering 해서 photo_id 와 해당하는 tag 를 매칭시킨 dictionary 를 만든다.

```
      clusters = []
      for i in range(len(lst)):
        clusters.append([])
      for i, item in enumerate(lst):
        indices = list(item)
        for idx in indices:
          clusters[i].append(int(names[idx].split('.')[0]))

[ ]   # Only use in visual clustered images
      category = []
      for i in range(len(clusters)):
        category.append(set())
      for idx, items in enumerate(clusters):
        for item in items:
          try:
            for c in tag_label_dict[item]:
              category[idx].add(c)
          except KeyError: # if image has no tag
            pass
```

Visual clustering 을 진행한 cluster 의 경우, visual clustering 으로 생성된 visual cluster 에 포함된 image 들의 tag 로 set 을 만들고

```
[ ]  # Make sentence with categories
     train_data = []
     for item in category:
       temp = ''
       for word in item:
         temp += word + ' '
       train_data.append(temp[:-1])
```

```
[ ]  from sklearn.feature_extraction.text import CountVectorizer

[ ]  # visual cluster --> min_df: 2
     # textual cluster(hierachical) --> min_df: hyperparameter
     vectorizer = CountVectorizer(min_df=3)

[ ]  Mat = vectorizer.fit_transform(train_data)
```

각 cluster 의 tag 집합을 나열하여 textual clustering 을 위한 training data 를 만들고 CountVectorizer 를 통해 학습을 진행한다. 여기서 countvectorizer 의 min_df (등장 빈도)변수를 조절하여 적절한 clustering 이 이루어지도록 한다.

```
[ ]  Matarr = Mat.toarray()

[ ]  # convert
     for i in range(len(Matarr)):
       for j in range(len(Matarr[i])):
         Matarr[i,j] = min(Matarr[i,j], 1)
```

결과로 만들어진 document matrix 의 document row 를 최대 1 로 scaling 한다.

Calculate Similarity of documents

```
def cos_sim(A, B):
  return np.dot(A, B)/(np.linalg.norm(A)* np.linalg.norm(B))
threshold = 0.7 # similarity threshold
pairs = [] # document pair
for number in range(len(Mat.toarray())):
  temp = []
  temp.append(number)
  for i in range(len(Mat.toarray())):
    if i != number and (cos_sim(Matarr[number], Matarr[i])) > threshold:
      temp.append(i)
  temp.sort()
  pairs.append(temp)
```

document 간의 cosine similarity 를 계산하여 유사한 document 끼리 묶는다. Threshold 를 잘 조절하여 잘 묶이도록 설정한다.

```
pair = []
for i in range(len(pairs)):
  if list(Mat.toarray()[i]).count(1) > 1: # filter only one feature docs
    pair.append(pairs[i])
```

```
Set = []
for item in pair:
  if len(Set) == 0:
    Set.append(set(item))
  else:
    for idx, s in enumerate(Set):
      if len(s & set(item)) != 0:
        s.update(set(item))
        break
      if idx == len(Set) - 1:
        Set.append(set(item))
```

```
new_Set = [] # for outerjoin one more when size of 'pair' is so small
for item in Set:
  if len(new_Set) == 0:
    new_Set.append(set(item))
  else:
    for idx, s in enumerate(new_Set):
      if len(s & set(item)) != 0:
        s.update(set(item))
        break
      if idx == len(new_Set) - 1:
        new_Set.append(set(item))
Set = new_Set
```

유사한 document 를 set 으로 묶는다. 과정이 완전히 correct 하지 않기 때문에 올바른 결과가 나올 때까지 묶는 과정을 반복한다.

```
# Ex)
# clusters[0,1,2,3,4,5] --> 1
# clusters[7,8,9] --> 2
for idx, items in enumerate(clusters): # copy photos to each class directory
  label = None
  for num, item in enumerate(Set): # calculate target class
    if idx in item:
      label = num
      break
  for item in items:
    shutil.copy(source + str(item) + '.jpg', target_position + str(label))
    print(source + str(item) + '.jpg' + ' ' + target_position + str(label))
```

새롭게 재구성된 cluster 로 visual cluster 를 합친다.

Only Textual Clustering starts from here

```
[ ]   # When textual clustering visual clustered images, just skip this cell
      src = '/content/drive/MyDrive/Datas/textout/' # source position
      src += '1-1-1-1/None/'
      filelist = os.listdir(src) # get total image file list
      new_clusters = []
      for i in range(len(filelist)):
        new_clusters.append(filelist[i].split('.')[0]) # remove .jpg
      new_clusters = [int(x) for x in new_clusters] # str to int
      clusters = new_clusters

      category = []
      for i in range(len(clusters)):
        category.append(set())
      for idx, item in enumerate(clusters):
        try:
          for c in tag_label_dict[item]:
            category[idx].add(c)
        except KeyError: # if image has no tag
          pass
```

Text 만을 이용한 clustering 의 경우 다음과 같은 유사한 cluster tag set 을 만드는 과정을 수행한 후 동일하게 training data 를 만들어 학습을 진행한다.

그 다음 유사한 과정들을 거쳐 clustering 을 진행한 후

```
[ ]   # File copy for text only
      # Ex)
      # clusters[0,1,2,3,4,5] --> 1
      # clusters[7,8,9] --> 2
      for idx, items in enumerate(clusters): # copy photos to each class directory
        label = None
        for num, item in enumerate(Set): # calculate target class
          if idx in item:
            label = num
            break
        shutil.copy(source + str(items) + '.jpg', target_position + str(label))
        print(source + str(items) + '.jpg' + ' ' + target_position + str(label))
```

다음과 같이 해당하는 cluster 로 image 를 저장한다.

Filtering 과 Pruning 을 위해서 아래와 같은 방법을 사용했다.

Determine class name

```
[ ]   source = '/content/drive/MyDrive/Datas/final_output/' # source position
      source += '6(vt)/1/'
      filelist = os.listdir(source) # get total image file list
      total_file = []
      for i in range(len(filelist)):
        total_file.append(filelist[i].split('.')[0]) # remove .jpg
      total_file = [int(x) for x in total_file] # str to int
      Class = set()
      for id in total_file:
        try:
          Class.update(set(tag_label_dict[id]))
        except KeyError:
          pass
```

Cluster 의 이름을 결정할 tag 의 후보군 집합을 위와 같이 계산했다.

후보군 집합에서 적절한 이름을 선택하였다.

Filter by user_id

```
[83]  source = '/content/drive/MyDrive/Datas/final_output/' # source position
      source += '6(vt)/1'
      filelist = os.listdir(source) # get total image file list
      total_file = []
      for i in range(len(filelist)):
        total_file.append(filelist[i].split('.')[0]) # remove .jpg
      total_file = [int(x) for x in total_file] # str to int
      Photo = photo_dataset[photo_dataset['photo_id'].isin(total_file)]
```

Cluster image 의 user_id 의 종류를 위와 같이 계산했다.

User_id 집합의 원소 개수로 pruning 을 위한 user_id 종류의 수를 계산하였다.

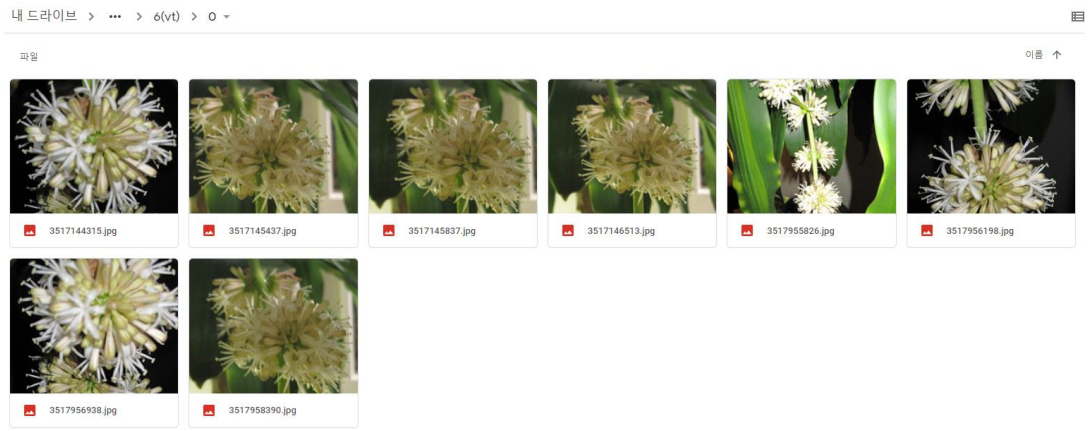
3. Result & Analysis

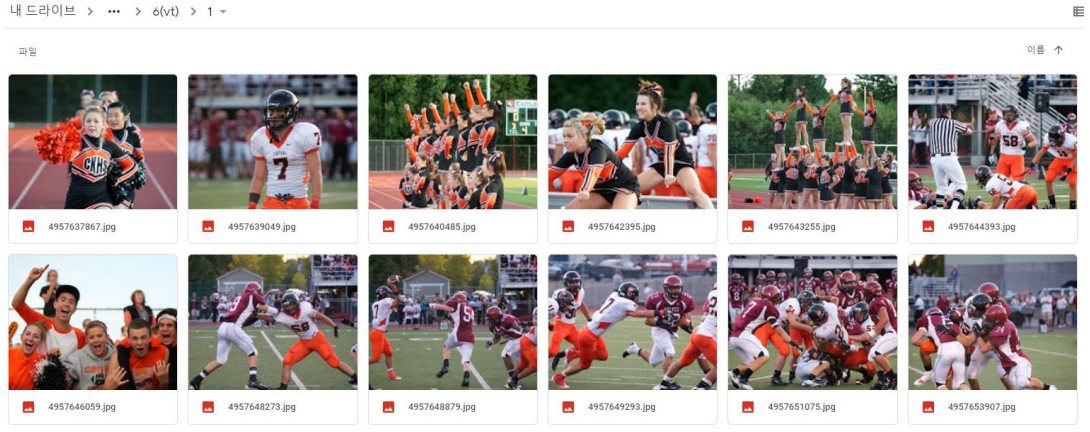
위와 같은 방법으로 clustering 을 수행하여 다음과 같은 결과를 얻었다.

visual clustered			
class	# photos	# clusters	# clusters(new)
1-1-3-3	76	14	**3**
1-3-1	97	5	**3**
1-3-3	64	8	**4**
1-4	127	38	**6**
1-5	72	22	**5**
1-6	20	6	**2**
2-3	28	6	**2**
3-2	71	16	**2**
4-1	254	22	**4**
4-2	170	31	**2**
6	61	16	**2**

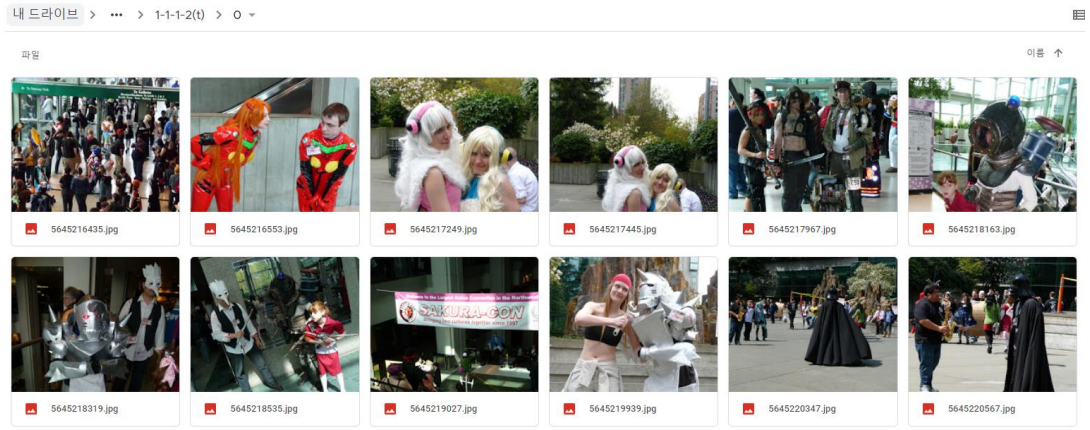
only text clustered		
class	# photos	# clusters
1-1-1-1	629	**10**
1-1-1-2	245	**7**
1-1-2-1	348	**7**
1-1-2-2	213	**6**
1-1-4	275	**3**

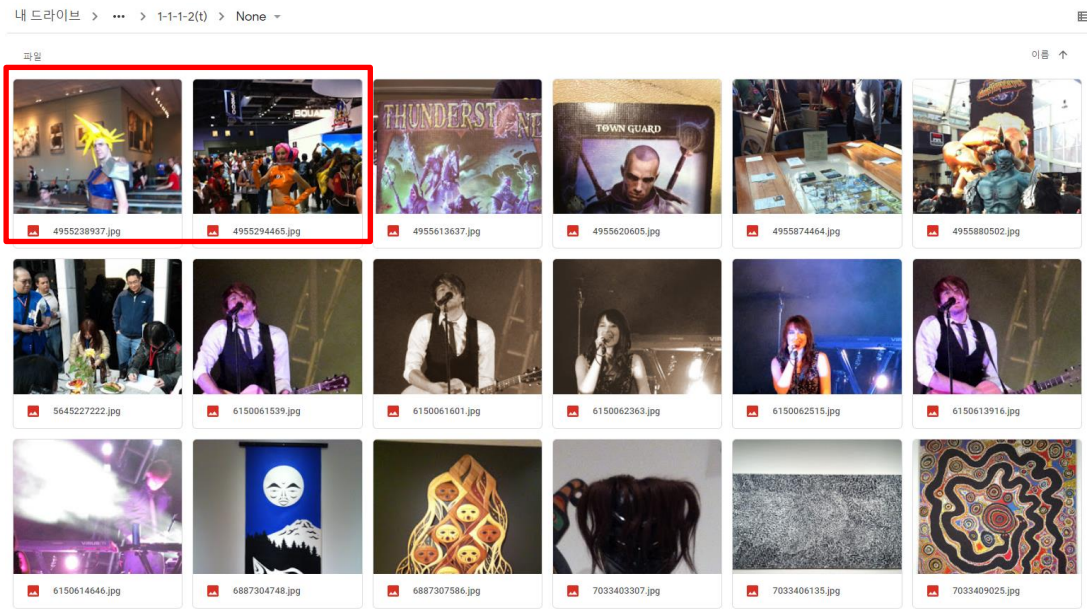
Textual clustering 을 수행하자, visual cluster 의 경우 확연히 cluster 의 개수가 효율적으로 줄어든 것을 확인할 수 있고, visual cluster 를 수행하지 않았던 cluster 도 작은 여러 개의 cluster 로 나누어진 것을 확인할 수 있었다.





위 사진의 class 6 을 예를 들자면, class 6 의 경우 visual clustering 을 통해 16 개의 작은 cluster 로 구분되었지만, 여기에 textual clustering 을 수행하자 2 개의 cluster 로 명확하게 구분된 것을 확인할 수 있다. 이를 통해 visual cluster 에 textual clustering 을 수행한 효과를 확인할 수 있었다.





반대로 textual clustering 만을 수행한 경우를 하나 살펴보면, 잘 분리되는 경우도 있지만, 위와 같이 유사한 주제의 코스프레 사진이라도 tag 가 없거나 1 개인 경우 clustering 이 잘 수행되지 않는 경우가 발생한 것을 확인할 수 있다.

위 결과를 통해서 visual cluster 를 수행한 결과에 대해서 textual clustering 을 수행하였을 때의 결과가 다음과 같이 더욱 효과적으로 나타났는데, 그 이유를 분석해보면 모든 image 가 tag 를 가지고 있는 것이 아니라, tag 가 없는 image 도 존재했다. 이러한 경우 textual clustering 만 수행할 경우 제대로 분리되지 않는 모습을 관찰할 수 있었다. 또한 'seattle'과 같이 tag 가 1 개만 있는 경우도 흔한 하나의 tag 만으로는 하나의 landmark cluster 라고 보기 어렵다고 가정하고 분리하였기 때문에 tag 가 1 개인 경우도 제대로 분리되지 않는 모습을 관찰할 수 있었다. 반면에, 먼저 visual clustering 을 수행하고 textual clustering 을 수행한 경우 tag 가 없거나 적은 사진이라도 같은 visual cluster 내에 있는 사진들과 같은 tag 를 가진 것으로 여겨지므로 상대적으로 잘 분리되었다고 할 수 있다.

Class					# photos	cluster method	kind	# user id
1	1	1	1	1	17	text only	market	1
				2	12		gumwall	2
				3	10		artmuseum or mirror	1
				5	25		aquarium	1
				6	67		cosplay	1
			2	0	62	text only	cosplay	1
				1	32		protest	1
				2	108		pax	6
				3	10		the triple door	1
			3	-	111	geo only	seattle central library	26
		2	1	1	35	text only	spaceneedle	8
				2	24		muralampi theatre	1
				3	18		totempole	1
			2	0	21	text only	intiman theatre	1
				1	28		muralampi theatre	1
				2	134		roller skating	2
			3	-	80	geo only	seattle art musueum	23
		3	1	-	281	geo only	qwest field	27
			2	-	189	geo only	safeco field	25
			3	1	16	visual + text	tower	2
			4	-	55	geo only	king street station	10
		4	0	-	117	text only	lake union	4
			2		150		lake union	1
		5	-	-	43	geo only	neumo concert hall	3
	3	1	-	-	47	visual + text	gas works park	9
		2	-	-	97	geo only	volunteer park	4
		3	0	-	6	visual + text	university of washington	2
		4	-	-	36	geo only	husky stadium	4
	4	0	-	-	33	visual + text	greenwood car show	2
		2	-	-	11		boat	1
		3	-	-	53		art	1
	5	0	-	-	32	visual + text	bellevue	2
3	2	-	-	-	71	visual + text	sea compression 2009	1
4	1	-	-	-	73	visual + text	the museum of flight	8
	2	0	-	-	150	visual + text	rainfurrest	1
	3	-	-	-	61	geo only	seahurst park	1
5	-	-	-	-	73	geo only	seward park	6
6	1	-	-	-	50	visual + text	highschool football	1
				Total	2438			

이를 통해 clustering 으로 구분된 너무 작은 cluster 들을 임의로 합치고 tag 를 통해 분류한 뒤 명확한 outlier class 를 pruning 한 결과는 위와 같았다. Textual clustering 을 수행하고도 명확히 분리되지 않은 class 의 경우는 임의로 수정하지 않고 clustering 이 실패한 것으로 처리하여 outlier 처리하였다. 또한 tag 가 없거나 부족한 None class 의 경우 candidate 로 선정하지 않고 pruning 하여 제거하였다.

이렇게 얻은 cluster 에 대해서 cluster 의 user id 개수(≥3)로 cluster pruning 을 수행하고자 계획했으나, textual clustering 만을 수행한 경우 tag 가 없거나 적어 버려진 사진과 textual clustering 이 잘 수행되지 않아 버려진 사진들 때문에 user id 의 개수로 이들을 판단하는 것은 적절하지 않다 생각하여, visual clustering 과 textual clustering 을 함께 한 경우나 geographical clustering 만으로도 clustering 이 수행된 cluster 에 대해서만 위의 조건으로 pruning 을 수행하고, geo cluster 상으로 인접한 cluster 이면서 tag 의 종류가 유사한 cluster 의 경우 같은 cluster 라고 판단하여 처리한 결과는 다음과 같다.

Class					# photos	cluster method	kind	# user id
1	1	1	1	1	17	text only	market	1
				2	12		gumwall	2
				3	10		artmuseum	1
				5	25		aquarium	1
				6	129		cosplay	2
			2	1	32	text only	protest	1
				2	108		pax	6
				3	10		the triple door	1
			3	-	111	geo only	seattle central library	26
		2	1	1	35	text only	spaceneedle	8
				2	52		muralampi theatre	2
				3	18		totempole	1
			2	0	21	text only	intiman theatre	1
				2	134		roller skating	2
			3	-	80	geo only	seattle art musueum	23
		3	1	-	281	geo only	qwest field	27
			2	-	189	geo only	safeco field	25
			4	-	55	geo only	king street station	10
		4	0	-	267	text only	lake union	5
		5	-	-	43	geo only	neumo concert hall	3
	3	1	-	-	47	visual + text	gas works park	9
		2	-	-	97	geo only	volunteer park	4
		4	-	-	36	geo only	husky stadium	4
4	1	-	-	-	73	visual + text	the museum of flight	8
5	-	-	-	-	73	geo only	seward park	6

최종적으로 위와 같은 25 개의 landmark candidate 를 얻었다.

Conclusion & Compare with the ground truth

이렇게 세 단계를 거쳐 clustering 을 한 결과를 ground truth 와 비교한 결과는 다음과 같았다.

	Predicted	Ground Truth
Candidate	Aquarium	Aqua
	Art museum	Art
	Cosplay	Convention
	Gas works park	Gas
	Gum wall	Gum
	Husky stadium	
	Intiman theatre	
	King street station	King
	Lake union	Lake
	Market	Market
	Muralampi theatre	
	Neumo concert hall	
	Pax	
	Protest	
	Qwest field	Century
	Roller Skating	Arena
	Safeco field	Safeco
	Seattle art museum	Olympic
	Seattle central library	Library
	Seward park	Seward
	Space needle	Space
	The museum of flight	Museum
	The tripledoor	
	Totempole	
	Volunteer park	Volunteer
		Water
		Univ
		Smith

너무 자세하게 clustering 되어 ground truth 에 없는 경우도 있었고, clustering 과정에서 제거되어 ground truth 에 없는 경우도 있었다.

Clustering 이 완벽하다고는 할 수 없겠지만, 어느 정도 비슷하게 예측했다는 것을 확인할 수 있었다.

또한 각 cluster 의 screen shot 을 제출파일에 첨부하였다.

Reference

https://bab2min.tistory.com/637 그림 참조

https://scikit-learn.org/stable/auto_examples/cluster/plot_mean_shift.html#sphx-glr-auto-examples-cluster-plot-mean-shift-py

Lecture Note: Week6_Term_project