수치해석 Project1
eigenface

2016025478 컴퓨터전공 서경욱

목표

- Cropped face image(real data)를 data set으로 가지고 SVD(singular value decomposition)을 통해서 eigenvector인 basis vector를 구한다.

- 32*32 image를 다루는데 1024개의 basis가 아닌 eigenvalue가 가장 큰 45개의 basis의 linear combination을 통해 image를 구성하는 식으로 바꾼다.

- 사람 10명의 5장씩의 image를 가지고 45개의 basis로 eigenface, coefficient를 이용해 image를 generate, recognize 해본다.

개발 환경

- Python 3.8

- 사용한 Library

- Numpy : 행렬 연산, 벡터 연산, SVD, covariance matrix 구하기, 내적
- Opencv : image file 처리(0~256 값으로 읽어 오기, image 출력)

Data Set


- 좌측과 같이 **face**만 **crop** 된 **image**들을 사용했습니다

- 다만, **image**크기가 **64*64**이므로 **resize**해서 **data**를 과제 명세에 맞게 **32*32**로 바꿔주는 과정이 필요했습니다.





- 같은 사람으로 **recognize**하기 위해 올바른 **coefficient**값을 구했는지 비교하기 위해 한사람당 **5**장의 **image**를 따로 구성해주었습니다.
- 제가 구한 **data set**은 무표정에 가깝지만 좀 다른 사진들도 있어 오차의 요인으로 고려해주었습니다

1. Collect face images

- Crop the same image size for face area

```
for i in range(len(files)): # 3000
    img = cv2.imread(join(img_path,files[i]),0) # 0 해줘야 흑백
    resize_img = cv2.resize(img,(32,32))
    tmp = []
    for j in range(32): # tmp_len 1024
        for k in range(32):
            #astype cast 안해주면 RuntimeWarning: overflow encountered in ubyte_scalars
            tmp.append(resize_img[j][k].astype(float))
    print("A[%d] is done" %i)
    A[i] = tmp
```

- 저는 eigenface를 생성하기 위해 3000장의 image를 사용했습니다
- 32*32라 1024 dimensional의 data를 다루나 1024개의 independent vector 가 일반적으로 존재하지 않기 때문에 충분한 개수의 image를 사용했습니다.

2. Construct data matrix, A

- 우선 처음 받아온 image를 읽어오면 32*32 꼴 행렬로 읽혀지므로 1024 dimensional vector로 형태를 바꿔줍니다.
- 그리고 A라는 행렬(3000*1024)의 row vector에 3000개의 image를 읽은 vector들을 넣어줍니다.(이유는 뒤에서 설명하겠습니다)

```
_M = np.mean(A)
M = [_M] * 1024
M = np.array(M)
```

- 그 후 A의 mean vector을 생성해줍니다.

2. Construct data matrix, A

- 앞에서 **A**의 **row vector**에 관측치를 넣어준 이유는 **covariance matrix**와 관련이 있습니다.
- **Covariance matrix**는 A^tA 의 형태로 **symmetric**한 형태를 지닙니다. 또한 **eigenvalue**가 **non-zero**라는 특징을 가집니다.
- 후에 **SVD**에서 관측치를 **row vector**로 둔지, **col vector**로 둔지에 따라 **basis vector**를 구하는 출처가 달라집니다.
- A^tA 형태는 **row**가 큰 경우 **더 작은 크기를 가지므로** 연산 속도에서의 이득을 볼 수 있습니다.

3. Apply SVD

```
# covariance matrix
# True -> row가 변수, col이 관측치
# False -> row가 관측치, col이 변수
# C.shape :  (1024, 1024)
C = np.cov(A.astype(float), rowvar=False)
```

- **Library**의 함수를 이용해 **covariance** 함수를 만들어주었습니다. 생성된 **C**의 크기는 **1024*1024**의 크기를 가집니다
- 함수 내에서 **mean vector**를 빼주는 과정을 진행하므로 따로 작성하지 않았습니다.

```
# #s = singular values 1차원 1024개 나옴
# V- row space basis U - col space basis
# orthonormal 값 return

U,s,Vh =  np.linalg.svd(C, full_matrices = True)
# V is unitary, Vt 안에는 row vector로 row space basis 들어있음
```

- **Library**의 함수를 이용해 **SVD**를 진행해줍니다.
- 이 함수에서는 **singular value**들을 내림차순으로 저장하고 있는 **s**, 그 크기 순서의 **column space basis**를 저장하는 **U**, **row space basis**를 저장하는 **V**의 **hermite**된 V^H가 **return**됩니다.

3. Apply SVD

- Eigenface의 개수는 45개로 설정했습니다.
- Eigenface의 해당하는 basis(row space basis)들은 V^H의 row vector(V 의 column vector)들로 구성되어 있을 것입니다. (A의 row에 data vector들을 넣었으므로)
- 또한 라이브러리 함수의 policy에 따라 eigenvector들의 순서도 singular value가 큰 순으로 나열되어 있으므로 1~45행의 eigenvector를 eigenface로 사용할 것입니다. 그리고 eigenvector들은 normalize된 상태로 return됩니다.(**orthonormal basis**)

4. Test face recognition

- Coefficient를 비교하기 위해 c_1 , …. , c_{45} 까지를 구하는 과정이 필요합니다.

```
test_path = 'test'
files = [ f for f in listdir(test_path) if isfile(join(test_path,f)) ]
res = []
for i in range(10):
    c = [[0 for _ in range(45)] for _ in range(5)]
    for j in range(5):
        img = cv2.imread(join(test_path,files[i*5 + j]) ,0)
        resize_img = cv2.resize(img,(32,32))
```

- Test file(50장)을 가진 폴더를 읽어와 처리합니다. Size의 변환까지 이뤄집니다. C에는 45개의 coefficient를 한사람당 5개 저장하는 배열을 만듭니다.

```
test = []
for k in range(32):
    for l in range(32):
        test.append(resize_img[k][l].astype(float))
test = np.array(test)
test -= M
for k in range(45):
    c[j][k] = np.dot(test,Vh[k])
res.append(c[j])
```

- Test file을 1024 dimensional vector로 변환 해준 뒤 Mean값을 빼줍니다.
- Original face에 mean값을 빼준 뒤 basis vector와 내적을 해주면 k번째 coefficient를 가진 항 외에는 **orthonormal**하므로 날아가고 c_k만 남게 되고 이를 c에 저장해주었습니다.

Generate face image using eigenfaces

```
show = np.zeros(1024)
res = np.array(res)
for k in range(45):
    data = res[1][k] * Vh[k]
    show += data
show = show.reshape(32,32)
cv2.imshow('image',show)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- 좌측과 같은 연산을 통해서 구해준 coefficient와 basis를 가지고 다시 원래의 image를 만들어 볼 수 있습니다.

동일 인물

orthonormal basis vector using SVD

- Real data는 32*32 인 1024 dimensional vector이고 이들의 vector space를 다루려면 1024개의 orthogonal한 basis가 필요합니다.

- SVD를 통해 구한 basis vector들을 eigenvalue가 큰 N(< 1024)개 만큼 뽑는다면 N개만으로 1024 dimensional을 근사하여 다룰 수 있게 됩니다.

- 또한 eigenface를 저장할 때 1024개가 아닌 N개의 저장으로 처리할 수 있으므로 **Data compression** 효과도 있습니다.

How to compare the identity of faces

- 구해진 coefficient들은 각 face의 identity입니다.
- 같은 사람의 얼굴에서는 다른 사람과 구분되는 **coefficient들의 특징**이 있을 것이고 그것을 구해보겠습니다.
- 우선, coefficient들의 차이를 기준으로 분석해보았습니다.
  (각 coefficient element들의 차의 제곱의 합)

How to compare the identity of faces

	case1	case2	case3	case4	case5	case6	case7	case8	case9	case10
Min	**1188728**.38 2131625	**968157**.89 666638	**959818**.267 0403817	**862563**.599 6891961	**574340**.319 3817368	**428710**.481 3321352	**972001**.053 9061085	**822445**.536 9504367	**871119**.703 4460879	**715245**.329 60419
Max	**3240681**.06 83010714	**4766375**.1 52521488	**2449717**.19 1647623	**8761882**.35 8880356	**3144126**.02 5213473	**5464182**.54 5276318	**4538396**.01 8659971	**6343168**.42 85601545	**4855311**.54 28323755	**3326781**.59 7087886

	3-1	3-2	3-4	3-5	3-6	3-7	3-8	3-9	3-10
Min	**1475510**.196 5318355	1466165.077 5925685	686915.86164 20296	1511653.0391 142073	939476.3348 603544	906753.14002 81844	2450311.0372 979953	1162403.5455 89016	840448.66093 31636
max	**4400787**.985 939894	**3231960**.477 152646	**7910298**.655 865726	**3880156**.918 701375	**3471225**.423 2052383	**4530364**.098 228127	**6112850**.957 720483	**4552336**.371 027894	**5321478**.700 995225

coefficient끼리의 차이가 가장 적은 case3의 경우를 예로 들겠습니다. 이 경우 case3과 나머지 coefficient와의 차이를 각 **element들의 차의 제곱으로** 보면 아래 표와 같습니다. 이 경우 case 3-4, cas3-8, case3-10 끼리는 차이가 심해 기준치를 case3의 max값으로 두면 distance만으로도 recognize를 확률 높게 할 수 있습니다.

How to compare the identity of faces

```
# 자기 자신 중 가장 variance 작은 coefficient
for i in range(10):
    tmp = [0 for _ in range(45)]
    for j in range(len(comb)):
        #tmp = [0 for _ in range(45)]
        a,b = comb[j]
        for k in range(45):
            tmp[k] += (res[5*i+a][k] - res[5*i+b][k]) ** 2
    print("min distance at %d" %(i+1),tmp.index(min(tmp))+1)
```

- 동일 인물의 coefficient 정보에서 어떤 coefficient의 variance가 작은 지 구하는 과정입니다.

```
min distance at 1 12
min distance at 2 32
min distance at 3 17
min distance at 4 39
min distance at 5 40
min distance at 6 45
min distance at 7 27
min distance at 8 36
min distance at 9 35
min distance at 10 44
```

- 구한 **coefficient의 차이가 가장 작은 index**를 구했습니다.
- 예를 들어 case1인지 recognize하고 싶을 때는 c_{12}에 **가중치를 높게 둬서** coefficient의 element의 차로 분류할 때 case1이 아닌 경우로 쉽게 분류할 수 있겠습니다
  (dominant / main coefficient)

오차 요인

- Element가 1024개인 vector를 크기가 1인 vector로 normalize 해주다보니(계산의 편의상) basis vector의 element가 굉장히 작은 값으로 이루어지게 됩니다.

- 이로 인해 floating point 문제로 후의 연산에서 truncate된 부분이 있을 것이고 오차가 생길 수 있다고 생각합니다.

- 마지막으로 data image에서 완전히 비슷한 각도, 표정이 아니기에 생기는 오차도 있다고 생각합니다