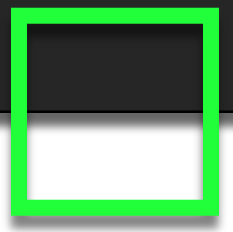
#수치 해석
#프로젝트 1

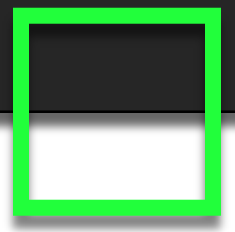
EigenFace

컴퓨터소프트웨어학부
2018008395 박정호



#0

Dataset



사용한 Dataset - Eigenface

MIT Faces Recognition Project database

3991개의 흑백 128*128 이미지가 바이너리 형식으로 저장된 데이터이다.
Eigenface를 구성하는데 사용했다. 시각화한 대략적인 형태는 아래와 같다.

이미지의 벡터화

```
def getByteArray(filename):
    byte_array = []
    file = open(filename, "rb")
    data = None
    while True:
        data = file.read(1)
        if data == b"":
            break
        try:
            byte_array.append(ord(data))
        except TypeError:
            pass

    return byte_array

def resizeByte(byte_array):
    image = Image.new('L', (128, 128))
    image.putdata(byte_array)
    image = image.resize((32, 32))

    return np.array(image.getdata(), dtype=int)
```

Eigenface를 구하는데 사용할 Dataset을 1024차원의 벡터로 만들 때 사용한 함수이다.

getByteArray는 주어진 파일을 가지고 바이트를 하나씩 읽어서 하나의 바이트 배열을 만드는 함수이고,

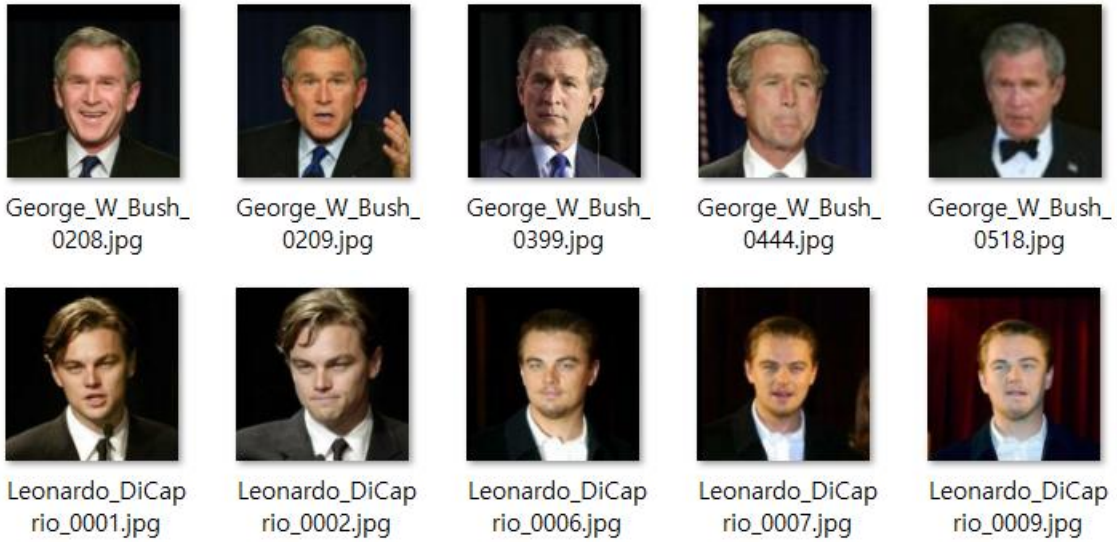
resizeByte는 주어진 바이트 배열을 128*128의 이미지로 두어서 이 이미지를 32*32로 압축한 뒤, 다시 바이트 배열로 돌려주는 함수이다.

이 두 함수를 이용해서 128*128짜리 이미지에서 1024차원의 벡터를 뽑아낼 수 있었다.

사용한 Dataset – Face Recognition

[Labeled Faces in the Wild]

5749명의 사진 13233장이 컬러 250*250 jpg 이미지로 저장된 데이터이다.
Face Recognition을 테스트하는데 사용했다. 대략적인 형태는 다음과 같다. 사용한 데이터는 조지 부시, 레오나르도 디카프리오, 실베스터 스탤론 등을 포함한 유명인 10명의 사진 5장씩이다.



이미지의 벡터화

```
def resizeImage(image_name, save_name):
    image = Image.open(image_name, 'r').convert('L')
    image = image.resize((32, 32))
    image.save(save_name)

def parseTestSet(src, dist):
    i = 0
    for filename in os.listdir('test_sample/'+src):
        resizeImage('test_sample/'+src+filename,
                    'test_sample/'+dist+str(i)+'.jpg')
        i+=1
```

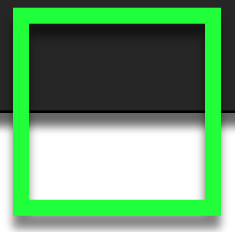
Face Recognition에 사용할 Dataset을 1024차원의 벡터로 만들 때 사용한 함수이다.

테스트는 다른 파일에서 진행할 것이기 때문에, 이 소스코드는 단순히 이미지의 흑백 변환과 크기 압축만 진행한다.

또한 테스트의 용이성을 위해 각 사람 별로 0~9까지 번호를 매기고, 이미지도 사람 별로 0~4까지 번호를 매겨서 저장했다.

#1

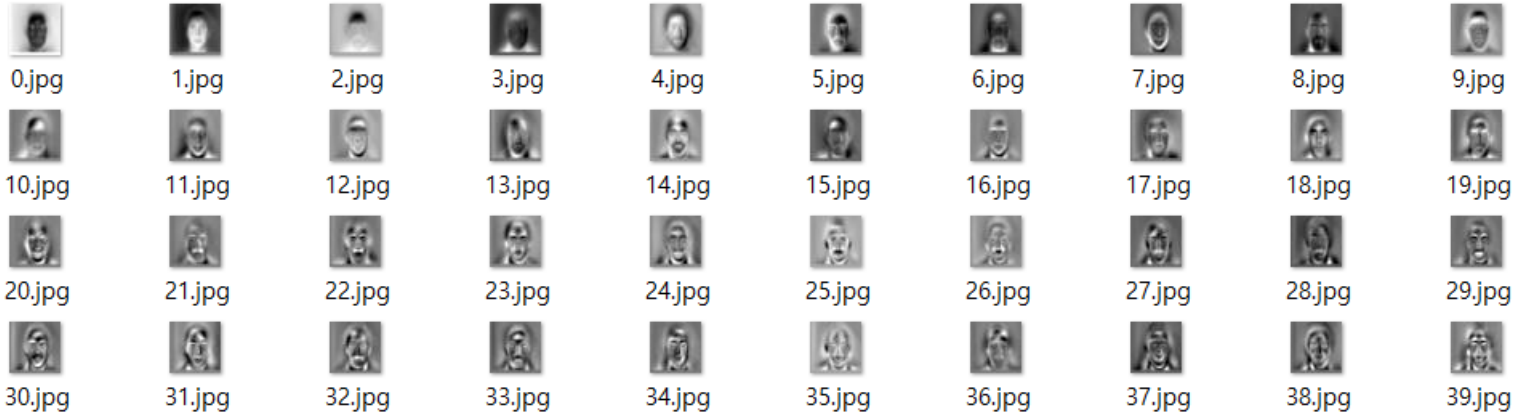
Eigenface 구현

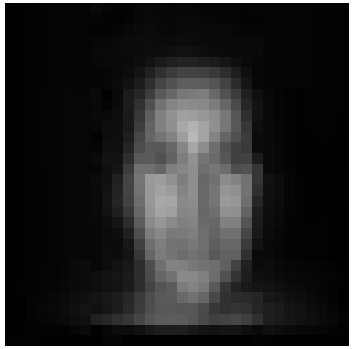


Eigenface 구하기

우선 앞에서 만든 3991개의 1024차원 벡터를 가지고 1024*3991 행렬을 구축했다.
즉 이 행렬의 column vector 하나하나가 필자가 구한 Dataset의 이미지들인 것이다. 이 행렬을 가지고 SVD를 수행했다. 사용한 함수는 numpy.linalg.svd 로, 오픈 라이브러리의 메소드이다. 복잡한 과정을 띄는 SVD를 직접 구현하는 것보다 이렇게 하는 것이 더 정확하고 빠를 것이라 생각해서 라이브러리를 사용했다.

결과로 나온 U, C, V 중 필자는 U를 eigenvector의 행렬로 사용했다. 기존에 이미지 하나하나의 픽셀을 column vecto로 두었기 때문이다. Eigenvector를 시각화하기 때문에 U의 column vector 하나하나를 시각화해서 이미지로 저장했다. 그 이미지 중 일부는 다음과 같다. 또한, 오른쪽의 큰 사진이 우리가 구한 mean vector이다.





mean_face.jpg

Eigenface 구하기

```
def showVectorAsImage(vector, image_name):
    image = Image.new('L', (32, 32))
    image.putdata(vector)
    image.save(image_name)

for i in range(U.shape[0]):
    minimum = min(U[i])
    maximum = max(U[i])
    for j in range(U.shape[1]):
        visualized_U[i][j] =
        int(255*(U[i][j] - minimum)/(maximum-minimum))
    showVectorAsImage(visualized_U[i], 'eigenfaces/'+str(i)+'.jpg')
```

구한 eigenface 들을 이미지로 저장하는 과정이다.
벡터를 이미지의 픽셀로 나타낼 수 있는 범위인
0에서 255 사이의 값으로 스케일링했다.

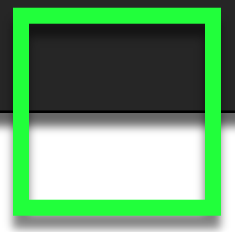
단, 이렇게 구한 이미지는 단순히 시각화를 위한 것으로,
실제 계산은 U의 column vector를 가지고 진행했다.

이렇게 1024개의 eigenface를 구하긴 했으나,
실제로 face recognition에서 사용한 eigenface들은 상위 70개이다.

70개 이후부터는 얼굴의 형체를 알아보기 힘들 정도로 노이즈가 심했기 때문에 굳이 face recognition에 사용할 필요가 없다고 보았기 때문이다.

#2

Face Recognition 구현



특정인의 coefficient vector 구하기

```
def getCoefficient(name, testset, eigenfaces, mean_face):
    result = np.zeros((eigenfaces.shape[1],))
    for i in range(5):
        filename = testset+str(i)+'.jpg'
        image = np.asarray(
            Image.open(filename, 'r').convert('L')
            ).reshape((1024,))

        x = []
        for row in eigenfaces.T:
           x.append((image@row))
        saveResultImage(eigenfaces@np.array(x)+mean_face,
                        testset+'result'+str(i)+'.jpg')
        result += np.array(x)
    result /= 5
    return (name, result)
```

각 사람별로 coefficient vector를 구하는 과정이다.

사람의 이름과 해당 사람의 이미지가 저장된 디렉토리, eigenface 행렬, 그리고 초반에 구한 얼굴들의 mean vector를 인자로 받는다.

디렉토리의 사진 5개를 1024의 벡터로 해석해와서, 각 eigenface와 곱하면서 해당 eigenface의 계수를 구한다.

이렇게 구해진 coefficient vector 5개를 평균내서 해당 사람의 coefficient vector를 구한 뒤,이름과 같이 return했다.

이렇게 모인 (이름, 벡터)의 tuple은 하나의 리스트에 저장되어 다음 함수에서 쓰인다.

Face Recognition Test

```
def processTest(coefficient, eigenfaces, filename):
    image = np.asarray(
        Image.open(filename, 'r').convert('L')
        ).reshape((1024,))
    x = []
    for row in eigenfaces.T:
        x.append((image@row))
    x = np.array(x)

    dist = None
    result = None
    for name, ce in coefficient:
        v = x - ce
        if dist != None:
            if dist > np.dot(v, v):
                dist = np.dot(v, v)
                result = name
        else:
            dist = np.dot(v, v)
            result = name
    return result
```

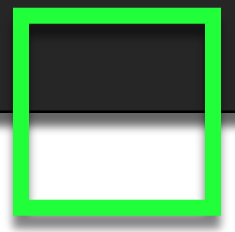
주어진 이미지를 가지고 특정인을 파악하는 함수이다. Coefficient는 앞에서 구한 (이름, 벡터)의 tuple을 저장한 리스트이며, filename은 face recognition을 수행할 이미지이다.

주어진 이미지의 coefficient vector를 구한 뒤, 앞에서 구해 둔 coefficient vector 들과 거리를 비교하면서 가장 가까운 coefficient vector를 찾는다.

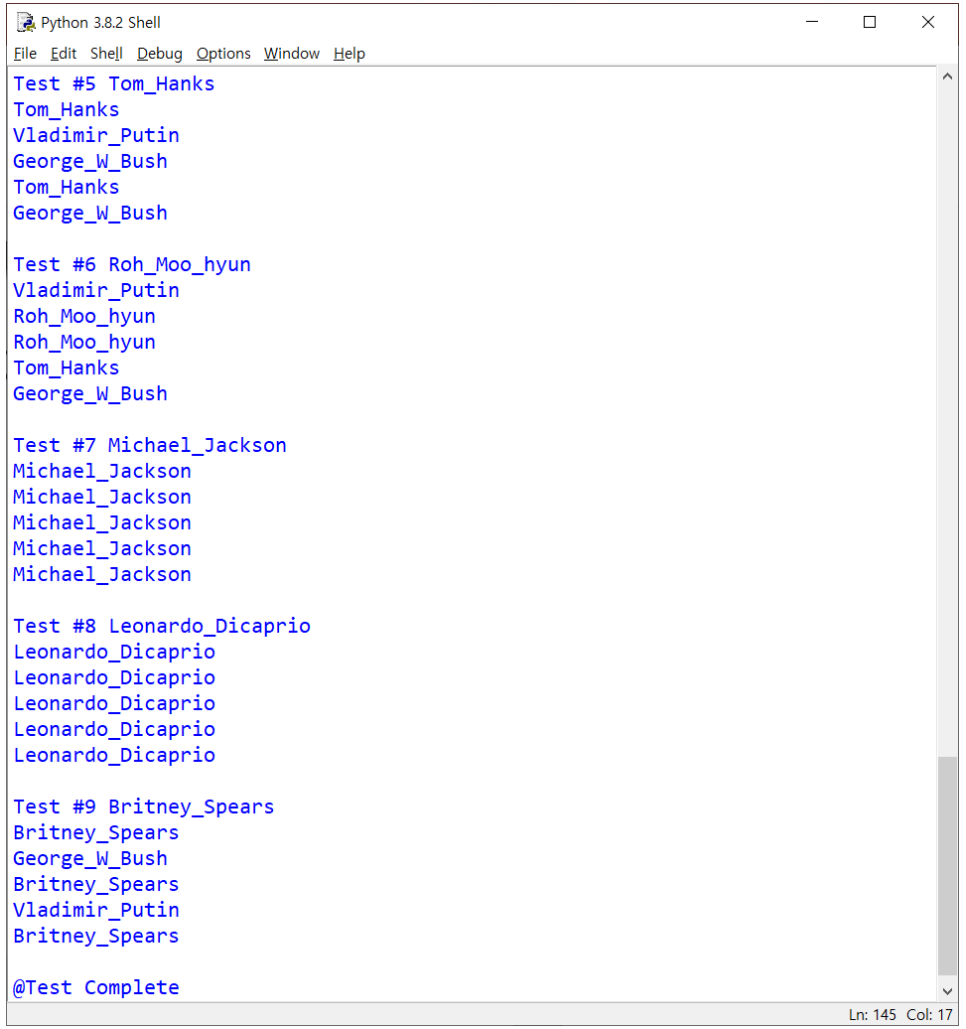
여기서, 가장 가까운 벡터에 대응되는 이름이, 해당 이미지에 해당하는 사람일 것으로 판단했다.

#3

결과와 비교



Eigenface를 이용한 Face Recognition



Eigenface를 이용한 Face Recognition

앞에서 언급했다시피, eigenface 70개로 face recognition test를 진행했다.

Test # 옆의 이름이 해당 테스트에서 쓰이는 이미지의 인물이며, 서로 다른 이미지 5개를 가지고 인식을 시도했다. 여기서 결과로 나오는 이름이 각 이미지가 인식된 인물이다.

좀 더 자세하게 말하자면,

빌 클린턴, 조지 부시, 레오나르도 디카프리오 같은 경우에는 모두 해당 인물을 잘 인식하고 있고, 블라디미르 푸틴 같은 경우는 한 경우에서 잘못된 인식을 보여주었다. 또한, 브리트니 스피어스나 톰 행크스, 실베스터 스탤론은 절반 내외의 정확도를 보여주었다.

전체적으로는 50%~100%의 정확도를 보여주고 있다.
이 다음 슬라이드에서 이렇게 정확도가 갈리는 이유를 분석해보자.

Eigenface를 이용한 Face Recognition

다시 결과를 요약하자면 전체적으로 만족스러웠다.

다만, 실베스터 스탤론, 톰 행크스, 브리트니 스피어스 등에서는 인식의 정확도가 낮은 것을 확인할 수 있었다. 이 슬라이드에서는 그 이유에 대해서 분석해 보았다.


George_W_Bush_0208.jpg


George_W_Bush_0209.jpg


George_W_Bush_0399.jpg


George_W_Bush_0444.jpg


George_W_Bush_0518.jpg


Britney_Spears_0001.jpg


Britney_Spears_0005.jpg


Britney_Spears_0009.jpg


Britney_Spears_0010.jpg


Britney_Spears_0014.jpg

정확도가 높게 나온 조지 부시 전 미국 대통령의 이미지와,
정확도가 낮게 나온 미국 배우 브리트니 스피어스의 이미지이다.

가장 큰 차이는 배경색의 일관성이었다. 아무래도 얼굴만 crop된 이미지를 찾기 힘들어서 배경을 포함한 Dataset을 사용했더니, 배경색으로 인한 정확성 감소를 보게 되었다. 또한, 얼굴 색조라던지, 헤어스타일 등, 얼굴의 특성이 일정하지 못한 경우도 이 얼굴 인식에 큰 영향을 주는 것을 볼 수 있었다.

Eigenface를 이용한 얼굴 복원

얼굴 인식 테스트를 수행하면서, coefficient vector와 eigenface matrix를 곱해서 이미지 복원도 같이 시도를 해보았다. 여기서 비슷한 이미지가 도출된다면, 같은 얼굴로 인식된 근거로 쓰일 수 있을 것이다.

Test 0 Test 8 Test 5 Test 9

옆의 사진들은 eigenface로 복원한 얼굴들인데, Test 0과 8은 얼굴 인식이 성공적이었던 케이스이고, Test 5와 9는 성공적이지 못한 케이스이다.

예상대로, 성공적인 케이스들은 모두 육안으로 보기에 유사한 이미지를 가지고 있었으며, 배경색도 거의 일정하게 이루어짐을 확인할 수 있었다. 또한, 성공적이지 못한 케이스들은 이미지의 모양도 제각각이며, 배경색도 일정치 못한 것을 볼 수 있다.

배경색이나, 얼굴형 등이 인식에 영향을 주는 것을 알 수 있다.

Face Recognition 방식의 문제

필자가 구현한 face recognition은 주어진 사람의 이미지 5개에 대해서 각각 coefficient vector를 구하고, 이 벡터들의 평균값을 구해서 그 값을 그 사람을 나타내는 coefficient vector로 쓰는 방식이었다. coefficient vector를 가지고 특정인을 판단할 수 있는 기준을 만들기 위해서 이렇게 진행했지만, 이 방식에는 문제가 있다.

평균을 구하는 과정에서 데이터의 왜곡이 발생할 수 있다.
같은 사람이면 coefficient vector가 비슷할 것이라는 전제에서 평균을 구한 것이기 때문이다.
앞에서 언급한 배경색, 얼굴 특징의 변화 등이 전혀 고려되지 않았기 때문에 예외적인 데이터가 평균을 크게 움직일 수도 있었다.

특히 이번 과제처럼 기준점을 만드는 데이터의 수가 5개 정도로 작은 경우에는 이 문제는 더욱 크게 작용했다.

옆의 데이터처럼 5개의 이미지의 배경색이 모두 다르고, 헤어스타일이나 표정이 조금씩 다른 이미지의 경우 평균을 낼 경우 그 어느 이미지와도 가깝지 않은, 완전히 새로운 이미지가 결과값으로 도출될 수도 있다.

개선점

1. 데이터의 일관성
   최대한 일관성 있는 데이터를 준비해야 더 성공적인 결과를 도출할 수 있음을 확인했다.

2. 데이터 보정의 필요성
   또한 의외성을 가진 데이터를 어느 정도 보정할 수 있다면 데이터가 좀 많이 퍼져있더라도 결과가 성공적일 것이다.

3. 더 많은 양의 데이터
   또한 얼굴 인식의 기준을 구축할 때도, 많은 양의 데이터를 사용한다면 일부 데이터가 예외적이더라도 평균에 그리 큰 영향을 미치지 못할 것이다.

4. 얼굴 인식 방법의 개선
   단순히 평균을 구하고, 이 평균에 가장 가까운 벡터를 판단하는 것보다 더 정밀한 방법을 찾는 것이 필요하다.