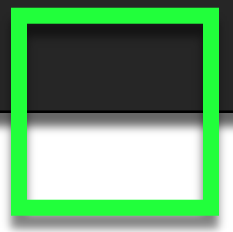
#수치 해석
#프로젝트 1

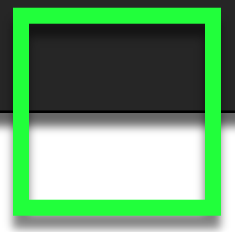
# EigenFace

컴퓨터소프트웨어학부
2018008395 박정호



#0

# Dataset과 라이브러리



# 사용한 Dataset - Eigenface

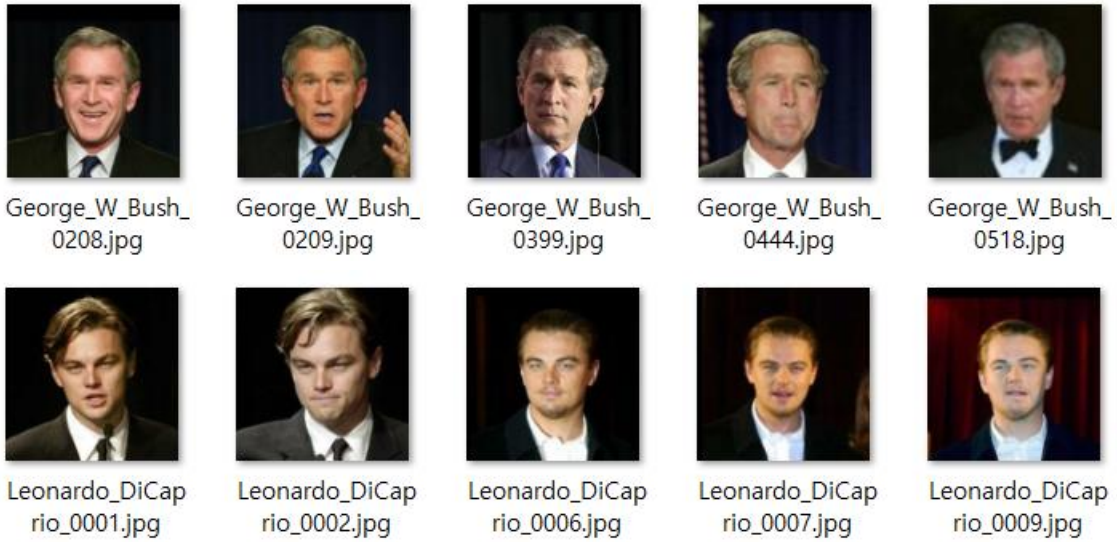
MIT Faces Recognition Project database

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

# 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.



# 사용한 라이브러리

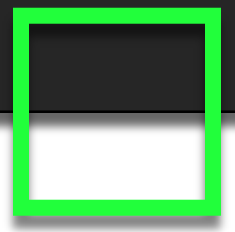
필자가 사용한 오픈 라이브러리는 이 두가지이고,
이외에는 python 3.8에서 기본적으로 제공하는 라이브러리를 사용했다.

1. PIL (python image library)
   이미지 프로세싱 관련 클래스와 메소드를 모아둔 라이브러리이다.
   이미지에서 각 픽셀의 정보를 읽어오거나, 구한 벡터를 이미지로 저장할 때 주로 사용했다.

2. numpy
   수학적 연산, 행렬 연산 등에 관련된 클래스와 메소드를 모아둔 라이브러리이다.
   벡터 관련 연산과 SVD 등, 수학적으로 연산이 필요한 경우와, 이미지의 벡터화 등에 사용했다.

#1

# Eigenface 구현



# 기본적 과정

1. 이미지의 resize와 벡터화
   필자가 구한 이미지는 32*32 픽셀이 아니라 128*128 픽셀의 이미지였다.
   이 이미지를 그대로 사용하기에는 벡터의 차원이 너무 크고, 그로 인해 구해야 할 이미지의 양도 많아지기에, 우선 32*32로 크기를 조정한 다음, 이를 1024 차원의 벡터로 나타내었다.

2. 각 이미지 벡터들의 행렬화
   위에서 구한 벡터를 column vector 로 하는 행렬을 만들었다.
   이 과정에서 모든 column vector 의 평균 벡터를 구해서, 각 column vector 에서 빼 주었다.
   약 4000 개의 이미지를 사용했으니, 행렬의 차원은 1024*4000 정도이다.

3. SVD 적용
   위에서 구한 행렬에 SVD를 적용했다.
   자세한 내용은 다음 슬라이드에서 설명하겠다.

# Eigenface 구하기

필자가 SVD를 쓰기 위해 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 eigenvector 의 행렬로 사용했다. 기존에 이미지 벡터를 column vector로 두었기 때문에, U가 1024*1024의 크기를 갖고 있었기 때문이다.

eigenface를 시각화해본 결과, 초반 70개 정도는 어느정도 사람의 얼굴 형체를 갖고 있었고, 그 이후에는 노이즈라고 판단될 정도로 형체를 알아보기 힘든 이미지를 볼 수 있었다. 따라서, face recognition에서 사용할 eigenface는 70개로 결정했다.

eigenface의 시각화에 대한 설명은 다음 슬라이드에서 계속하겠다.

# Eigenface의 시각화

앞에서 필자가 eigenface 를 선정한 기준은 "얼굴의 형체를 갖고 있는가?" 였다. 이 기준은 벡터를 이미지로 시각화 하지 않는 이상 확인할 수 없는 것이었다. 이 슬라이드에서는 어떻게 eigenface 를 시각화했는지 설명하고자 한다.

벡터를 이미지로 표현하기 위해서는 eigenface의 element 가 모두 0~255 사이의 값을 가져야 한다. Eigenface 는 결국 1024 차원의 벡터이고, 1024의 element 가 0~255의 범위에 있지 않았기 때문에, 이 값을 보정하는 작업이 필요했다. 따라서 필자는 모든 eigenface 에 다음과 같은 과정을 수행했다.

1. eigenface를 이루는 가장 작은 element 와 가장 큰 element 를 구한다. 이를 $min$ 과 $max$ 라 한다.

2. Eigenface의 모든 element 에 대해, 다음과 같은 함수를 적용한다.

$$f(x) = \frac{x - min}{max - min} \times 255$$

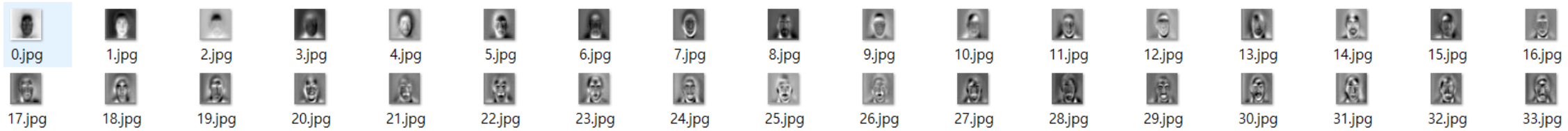
   이 함수를 거치면 벡터의 각 elemen가 0~255의 값으로 scaling 된다.

이 과정을 앞에서 이미지 벡터를 행렬화하는 과정에서 구한 벡터의 평균값에도 적용해서 mean face도 구했다.

# Eigenface의 시각화

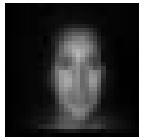
그러나 이렇게 시각화한 벡터는 강제로 0~255 사이의 값으로 scaling 한 것으로 인해 face recognition 에는 사용할 수 없었다. 따라서, face recognition에는 시각화하기 이전의 벡터를 사용했다.

시각화한 eigenface와 mean face는 다음과 같다.




- 초반 51개의 eigenface, 대체로 사람의 얼굴의 형상을 띄고 있는 것을 확인할 수 있다.

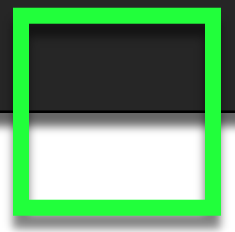

- 중반 34개의 eigenface, 사실상 노이즈라고 보일 정도로 형체가 없는 이미지를 볼 수 있다.



- 약 4000개의 얼굴 이미지에서 도출된 mean face

#2

# Face Recognition 구현



# 기본적 과정

딥러닝과 같은 방식은 사용하지 않고, 수학적으로 face recognition을 구현해보았다.

1. 이미지의 resize와 벡터화
   face recognition 을 위해 필자가 구한 이미지는 32*32 픽셀이 아니라 250*250 픽셀의 이미지였다.
   이미 eigenvector 가 1024 차원으로 구해져 있기 때문에 테스트용 이미지도 32*32로 resize 후 1024 차원의 벡터로 나타내었다.

2. 얼굴 인식을 위한 기준 벡터 생성
   주어진 face 이미지가 특정인인 것을 인식하기 위해서는 그 특정인에 대한 identity가 필요하다고 생각했다.
   따라서 테스트로 사용할 사람의 이미지 여러 장에 대해서 coefficient vector를 미리 구하고, 같은 사람의 coefficient vector 들의 평균을 내서 특정인의 identity를 나타내는 coefficient vector를 만들었다.

3. 얼굴 인식
   주어진 이미지에 대해서 coefficient vector를 구한 뒤, 위에서 구해둔 10 명에 대한 coefficient vector와 각각 비교해서 얼굴 인식을 수행했다.

# coefficient vector 구하기

각 이미지에 대해서 coefficient vector 를 구하는 과정은 과제 명세에서 언급된 것과 같다.

각 eigenvector 를 대상 이미지에 곱하는 것으로 각 coefficient를 구했고 그 coefficient를 모아서 vector 로 만들었다. 앞에서 언급했듯이 eigenface 를 70개 사용했기에 coefficient vector 도 70차원이 될 것이다.

이렇게 구한 coefficient vector 는 위의 2번 과정 (얼굴 인식 과정을 위한 기준 벡터 생성)에서 각 인물의 identity 를 구축하는데 쓰이고, 3번 과정에서 입력된 이미지의 identity를 구하는데도 쓰인다.

자세한 내용은 다음 슬라이드에서 설명하겠다.

# 인물의 identity 구하기

아무리 같은 사람이라 하더라도, 이미지에 따라 조금씩 coefficient vector 가 다를 수 있다. 따라서, 특정인을 구별할 기준이 되기 위해서는 이 coefficient vector 를 어느 정도 대표할 수 있는 기준을 만드는 것이 필요했다. 그래야 face recognition 을 하는 과정에서 이미지의 coefficient vector 와 비교해서 어떤 인물의 이미지인지 확인 할 수 있기 때문이다.

이 대표값을 어떻게 구할지 고민한 끝에, 단순하게 coefficient vector의 평균값으로 잡기로 했다. 각 인물 10명의 이미지 5장씩에서 구해진 coefficient vector 5개씩을 각 인물 별로 평균을 내서, 그 평균 벡터 10개를 각 인물의 identity로 설정했다.

이 방식이 확실히 단점은 있겠지만, 단순한 implementation이 어떤 성능을 낼지 궁금해서 이러한 시도를 해보았다.

# Face recognition 과정

앞 슬라이드에서 설명한 인물의 identity vector를 구하는 과정을 마치면, face recognition 의 준비가 끝난다.

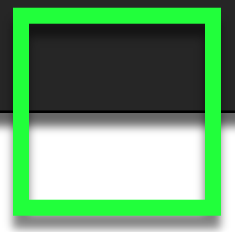
이제 이미지를 선택하면 각 이미지가 어떤 사람의 이미지인지 구별하는 face recognition을 진행할 수 있다. 이 과정은 각 이미지의 coefficient vector 즉 identity 를 구한 뒤, 각 인물들의 identity 를 선택한 이미지의 identity 와 비교해서 가장 가까운 identity 를 찾는 것으로 진행된다.

가장 가까운 identity를 찾는 과정은 결국 주어진 벡터와 가장 가까운 벡터를 찾는 것과 같기 때문에 vector distance 를 비교하는 것으로 진행했다. vector distance 가 가장 작은 인물이 주어진 이미지의 인물이라고 보는 것이다.

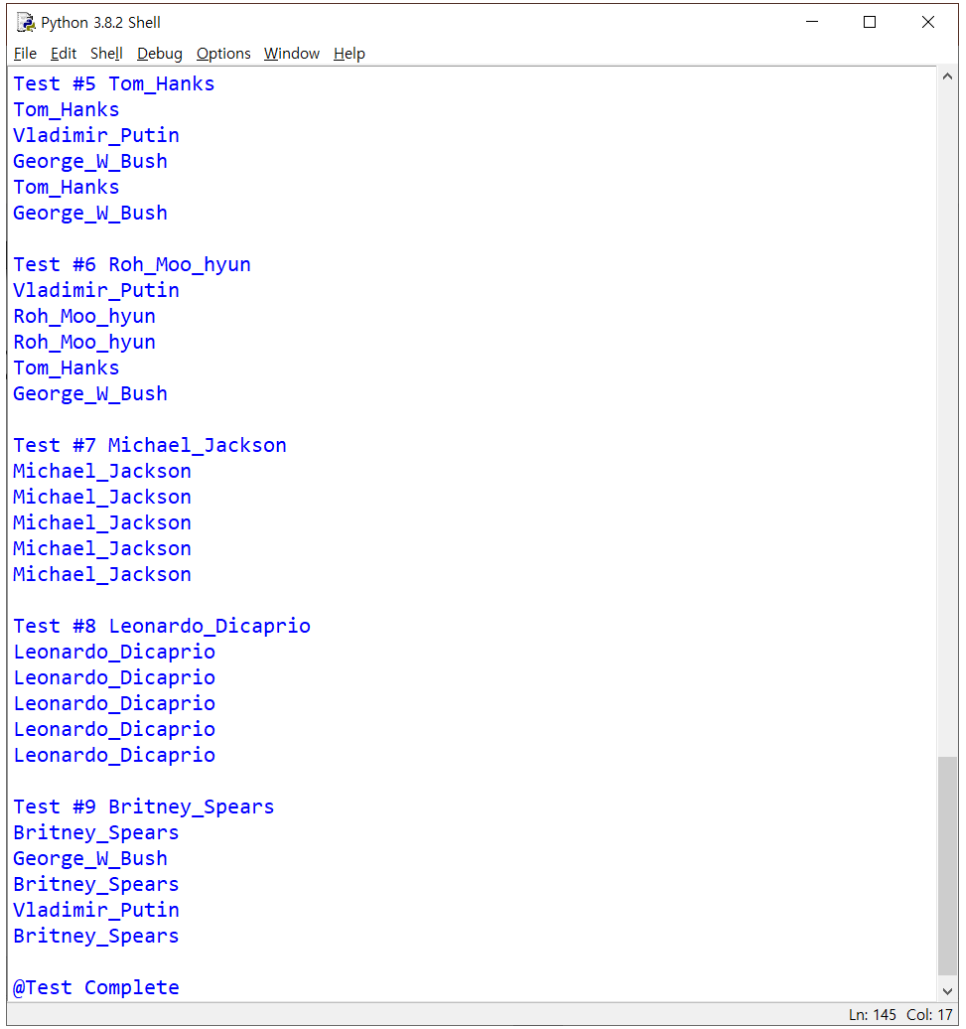
이 다음 섹션에서 보여줄 테스트는 face recognition 명세에 나온 50개의 이미지를 모두 입력해서 테스트를 수행한 결과이다.

#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.