CS485 CW1

20220352 손유호, 20210256 박주현

Q1-1-a

편향된 data가 되지 않도록, 같은 라벨을 가진 face data를 2:8의 비율로 testing set과 training set으로 나눈다. 공분산 행렬의 고유벡터와 고유값을 구한 후 0이 아닌 고유값을 필터링하면 2576개의 고유값이 나온다. 하지만, 10^(-10)보다 작은 고유값을 0으로 보고 그 보다 큰 고유값만 계산한다면 415개의 고유값을 얻을 수 있다.
face recognition에 사용할 고유벡터의 수를 구하기 위해 scree plot을 그려보면 아래와 같다. 대략적으로 50~60의 component number에서 완만해지는 모습을 볼 수 있다.

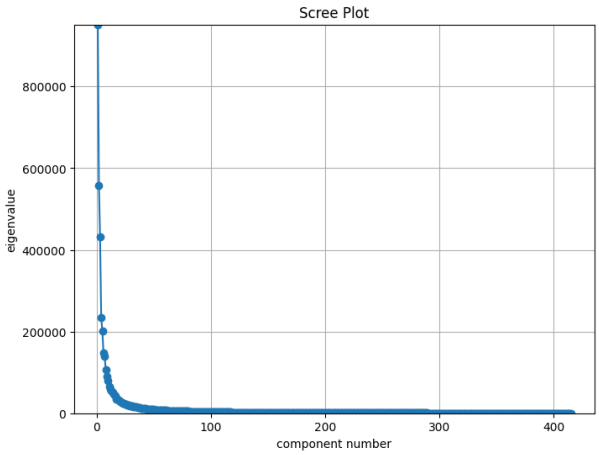

Fig1. eigenvalue plot

그리고 cumulative proportion을 95%와 90%로 두고 각각의 고유값 개수를 구해보면 95%의 경우에는 121개의 고유값이 해당되고, 90%의 경우에는 63개의 고유값이 해당된다. scree plot과 cumulative proportion의 결과를 종합적으로 생각해보았을 때, 시간 복잡도의 개념으로 생각해 본다면, 유효한 features의 수가 2배 더 줄어들어 속도가 8배 빨라짐으로써 얻을 수 있는 편리함이 5%의 추가적인 정보 손실을 감안할 만큼의 빠르기라고 생각했다. 따라서 63개의 고유값에 대응하는 고유벡터를 face recognition에 사용하는 것으로 결론내릴 수 있다. 이는 training set의 data의 개수보다 1개가 작은 고유값이다.

Q1-1-b.

$(1/N)A^TA$의 고유값과 고유벡터를 계산해보면 총 416개의 고유값이 나오는데 a에서 한 것처럼 10^{-10}보다 작은 고유값을 0으로 본다면 415개의 고유값을 얻을 수 있다. 415개의 고유값은 a에서 구한 0이 아닌 고유값과 동일하고, 고유벡터는 $A*((1/N)A^TA$의 고유벡터) = $(1/N)AA^T$의 고유벡터이다. (이때, 고유벡터는 크기가 1로 정규화된 고유벡터이다.)
따라서 $(1/N)A^TA$와 $(1/N)AA^T$는 0이 아닌 고유값이 동일하다. 그렇기에 계산의 상황에 따라 $(1/N)A^TA$와 $(1/N)AA^T$를 알맞게 선택해 연산 속도가 빠른 것을 사용하면 된다. 데이터의 수가 data의 차원보다 훨씬 작을때, $(1/N)A^TA$을 통해 고유 벡터와 고유값을 구한다면 $(1/N)A^TA$은 N X N 행렬이기에, D X D 행렬인 $(1/N)AA^T$을 이용하는 것보다 계산이 간단하다. 하지만 data의 차원이 데이터의 수보다 훨씬 작다면, $(1/N)AA^T$으로 고유 벡터와 고유값을 구하는 것이 $(1/N)A^TA$을 통해 계산하는 것보다 간단하다.

Q1-2-a

face image을 reconstruction하였을 때 reconstruction error를 feature의 수를 다르게 하며 구하였다. 아래의 image는 feature의 수가 50일 때의 reconstruction image이다

Fig2. feature=50 recognition image

3가지 이미지 모두, 이론과 동일하게 선택된 feature, 즉, 주성분의 수가 많을 수록 reconstruction error가 작게 나왔다. 이미지마다 reconstruction error의 값이 달랐다. reconstruction error가 세자리수 초반때로 작지는 않은 값이 나왔다. 이는 같은 인물의 사진이 training data에 8개 밖에 없어 표본 수의 부족과 적절하지 않은 이미지 사용으로 표본의 feature를 충분히 재현해내지 못해 발생한 reconstruction error라고 생각할 수 있다. 또한 test data인 2개의 사진의 상태가 좋지 않다면, 이 상황에 기여했을 수 있다.

Q1-2-b

NN classification에서 변수는 neighbor의 수와 투영될 principal subspace 차원의 크기이다. 변수의 값을 바꿔가며 계산하였을 때, neighbor의 수가 1일 때 인식정확도가 가장 높았고 1보다 클 때 오히려 정확도가 감소하였다. 투영될 principal subspace 차원의 크기는 neighbor의 수와 관계없이 인식 정확도와 비례하는 모습을 보였다. 하지만 차원의 크기가 클 수록 실행시간이 오래 걸리고 메모리를 많이 차지하기에 적절한 크기를 선택할 때 가장 최적의 결과가 나왔다. 그 외에도 weights='distance',metric= 'manhattan',p=1 일때 정확도가 더욱더 많이 올라갔다. 자료의 특성상 가까운 이웃에 더 비중을 두고,맨하탄 거리를 이용했을 때 정확도가 더 올라감을 확인했다.맨하탄 거리는 항상 유클리드 거리보다 크거나 같은데,정확도가 올라간 것으로 보아 자료가 퍼져있는 정도가 커서 맨하탄 거리를 이용했을 때 정확도가 올라갔을 것이라 추측했다.

아래는 이미지 인식의 실패와 성공의 예시이다.

인식에 실패한 이미지들을 보면, PCA로 인해 정보들이 많이 누락되어 다른 이미지와 구별하기 힘들어 분류에 실패하였다고 볼 수 있다.

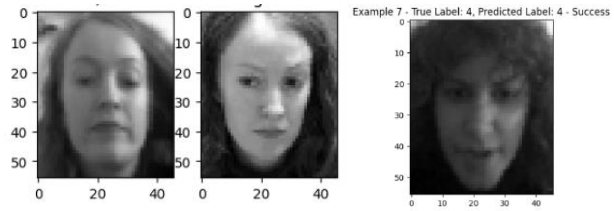

Fig3. Failure, success case(1)

NN classification 외에도 reconstruction error를 최소화하는 방식으로 PCA 기반 얼굴 인식을 수행하였다.

M이 6 이상일때부터 인식 정확도는 커지지 않았고 5일 때 제일 좋았다. 이는 같은 라벨의 이미지가 총 8개가 있어 유효한 고유값의 수가 7개이기 때문에 이러한 결과가 나온 것으로 볼 수 있다. 또한, 계산된 혼동행렬을 통해 reconstruction error를 최소화하는 방법이 NN classification 보다 더 높은 인식 정확도를 보임을 알 수 있었다.

Q2

```
PCA의 주성분 수:  63
NN classification의 neighbor 수: 1
PCA for train_set1:
Training Time: 0.043375492095947266
Reconstruction Error: 68.28978948157464
Accuracy: 0.3557692307692308

Incremental PCA:
Training Time: 0.20839476585388184
Reconstruction Error: 177.84727778791304
Accuracy: 0.7403846153846154

PCA for Full Data:
Training Time: 0.13840603828430176
Reconstruction Error: 176.56564994978132
Accuracy: 0.7403846153846154
```

```
PCA의 주성분 수:  63
NN classification의 neighbor 수: 2
PCA for train_set1:
Training Time: 0.06675052642822266
Reconstruction Error: 68.07854543931612
Accuracy: 0.2307692307692308

Incremental PCA:
Training Time: 0.22140121459960938
Reconstruction Error: 177.84727778791304
Accuracy: 0.5480769230769231

PCA for Full Data:
Training Time: 0.13174319267272295
Reconstruction Error: 176.58140753977042
Accuracy: 0.6057692307692307
```

Fig4. result(2)

위 사진은 결과창이다. 기본적으로 주성분의 수(0-121)에 따라 정확도값과, reconstruction error, training time이 달라지는 것을 볼 수 있다. 공통적으로 reconstruction error는 주성분의 수가 커지면 커질수록 모든 PCA에서 크게 작아졌다. 또한 reconstrucion error에서의 특징이 있었는데, 항상 incremental PCA의 reconstruction error가 조금씩 전체 PCA보다 컸다. 하지만 거의 동일한 값을 가졌다. set 1의 reconstruction error는 전체 PCA에서보다 값이 항상 작았다. 전체 PCA와 set1 PCA와의 reconstruction error는 차원축소의 비율이 set 1이 현저하게 작았기 때문에, set1PCA가 작았다고 설명할 수 있다. incremental PCA의 reconstruction error가 일반 PCA보다 크게 나온 이유는, 각각의 set의 관계를 일반적인 PCA보다 고려하지 않고 처리하기 때문에, 반복적인 특징에서 나오는 중요한 패턴이 누락될 수 있다. 또한 IPCA의 특성상 노이즈가 더 많이 포함되어, 원래 데이터셋의 변동성을 정확히 파악하지 못해 재구성 오차가 커질 수 있다.

training time의 경우, 뚜렷한 증감성을 주성분 수에 따라 판단하기 힘들었다. 하지만, 대부분의 주성분에서 incremental PCA's training time/ PCA's training time의 값이 1을 넘었다. 하지만, 주성분의 수가 커질수록 비율이 줄어들었다. 이 결과로 보아, 계산량이 복잡해지면 복잡해 질수록, IPCA의 특성상 효율적이며, IPCA를 사용할 만큼 모델의 계산량이 많지 않았음을 알 수 있다. 실제로, 더 큰 주성분값을 넣어볼 수는 없었지만, 모델의 크기가 커지고 주성분 값이 더 커졌다면, 충분히 IPCA의 training time이 작아졌을 것이다. accuracy의 경우 PCA, set1PCA, IPCA모두 대부분 주성분이 커질수록 점점더 커졌다. 주성분수가 많아지면 당연히 사람을 구분할 수 있는 features가 많아지므로 accuracy가 올랐다 추측할 수 있다. 다만 효율성의 측면에서 바라보았을 때, 주성분의 수가 63일때 set1은 오히려 104보다 accuracy가 0.8%정도 컸고, 다른 PCA들도 1%밖에 차이가 나지 않는 것으로 보아, Q1에서 논의했던 63의 값이 합리적이였다는 점을 이 문제에서도 확인할 수 있었다. 다음은 KNN의 neighbor 수(1-5)를 구분했을 때의 결과이다. training time에서는 별다른 특징을 찾지 못했다. 다만, reconstruction error의 값들은 KNN의 neighbor 값의 차이에도 굉장히 유사한 값들을 가졌으며, 1이상 차이나지 않았다. accuracy는 neighbor 수=1이 가장 컸고, 숫자가 커질수록 줄어들었다. 이는 노이즈의 숫자가 상당히 적었던 것을 의미한다고 생각한다. 노이즈수가 적으면, 클래스가 비교적 명확하게 구분된다 . 따라서 1일때 정확도가 제일 클 수 있다. 이는 초반 reconstruction error가 IPCA와 PCA사이에서 매우 적은 차이를 낸다는 점에서 신빙성을 얻는다. 노이즈 수가 많았을 경우 그 차이가 커졌을 것이기 때문이다.

Q3

PCA-LDA는 generative이고 discriminative인 특징을 동시에 학습한다. PCA-LDA 기반 얼굴 인식은 within-class scatter 행렬과 between-class scatter 행렬 사이의 비율을 최대화하는, M개의 가장 큰 고유값에 대응되는 고유벡터의 집합에 의해 해결된다.

Sw의 rank는 최대 N-c = 416 - 52 = 364이고 Sb의 rank는 c-1=52-1=51이다. 이것들의 rank를 줄이기 위해서 PCA를 사용하여 차원을 줄이고 LDA를 적용한다. PCA에서 선택된 고유값 M개를 M_pca, LDA에서 선택된 고유값 M개를 M_lda라고 하자. M_pca<=364, M_lda<=51 범위 내에서 계산을 해보면 인식정확도가 M_pca = 150이고, M_lda = 50일때 81.7%로 가장 높았다. 그리고 M_lda 값이 높을 수록 더 좋은 결과가 나오는 것을 볼 수 있었다. 값이 클 수록 차원 축소의 정도가 줄어들기에 더 좋은 결과가 나오는 것으로 생각할 수 있다. M_pca의 경우에는 값이 높은 것보다는 150과 같은 어느정도 적절한 수치에서 인식정확도가 높았다. 이는 M_pca의 값이 클수록 과대적합이 일어날 가능성이 높고, 값이 작으면 인식에 필요한 정보가 부족해지기 때문일 것이다.

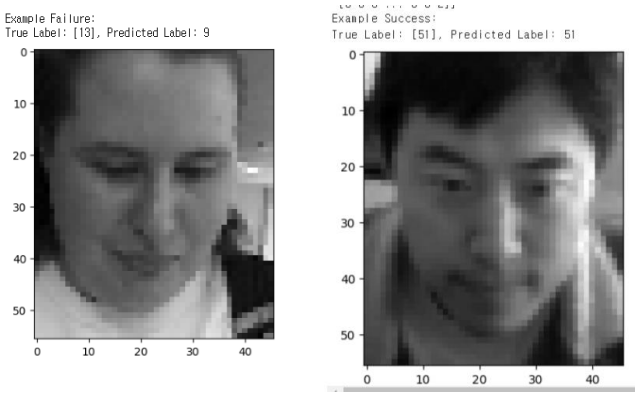

Fig5. Failure, success case(3)

퓨전 룰은 가장 간단한 majority voting을 사용하도록 하자. 먼저, 데이터에 PCA를 적용하여 차원을 N-1차원으로 낮춘다. 그리고 랜덤으로 데이터를 뽑아 training set을 생성하고 각 set마다 PCA-LDA-NN 추정을 적용한다. 이때 선택된 이미지의 수를 나타내는 무작위성 매개변수 ρ의 값이 높으면 랜덤성이 낮고 모델 간 상관 관계가 높다. ρ의 값이 낮으면 랜덤성이 높고 모델 간 상관 관계가 낮다. 따라서 ρ값을 낮추면 개별 모델의 정확도가 낮아졌다. ρ값을 바꿔가며 실험하면 ρ가 N-1과 유사할 때 좋은 결과가 나왔다. 하지만 ρ가 낮을때에도 앙상블 모델의 정확도는 70% 이상을 유지하였고 개별 모델의 성능보다 뛰어난 모습을 보였다.

PCA가 적용된 데이터에서 feature를 무작위로 추출하여 부분집합을 만든다. 이때, 고유값이 큰 순서대로 M0개의 고유얼굴은 고정적으로 포함하고 나머지 feature 중에서 M1 개의 차원을 랜덤으로 선택한다. M0와 M1을 바꿔가면서 실험한 결과, M1의 값이 너무 크지도 작지도 않을 때 좋은 성능을 보였다. 그리고 랜덤 data set보다 더 일관된 결과를 보였는데 이는 feature 무작위 추출이 더 넓은 범위에서 추출하기에 무작위성이 커져 이러한 결과가 나온 것으로 생각된다.

base 모델의 수는 20개 이상을 사용하는 것이 정확도 범위 기준에서 적절해보였다.

Q4

PCA는 데이터의 분산이 최대가 되는 방향을 찾는 것이 목표이기에 비용 함수를 주성분이 될 벡터 e와 scatter 행렬로 <e, S*e> = cost_pca(e)의 형태로 정의할 수 있다. LDA는 between-class scatter matrix를 최대화 하고 within-class scatter matrix를 최소화하는 것이 목표이기에 비용함수를 $\frac{<e, Sb*e>}{<e, Sw*e>}$ = cost_lda(e)로 정의할 수 있다. e는 feature subspace을 뜻하는 단위 벡터이다.

우리의 목표는 PCA와 LDA를 모두 충족하거나 둘 사이의 균형을 조절하는 문제를 공식화하는 것이기에 PCA와 LDA의 비용함수를 더한 값을 최대화하는 것이 우리의 문제이다. 이때, PCA와 LDA의 비용함수를 더하는 비율을 ρ라고 하면 우리의 목표 함수는 아래과 같다

$$J(e) = \frac{1-\rho}{2} cost\_pca(e) + \frac{\rho}{2} cost\_lda(e)$$

이때 ρ는 PCA와 LDA 사이의 균형을 제어하는 매개변수이므로 0에서 1사이의 값을 가지고 0에 가까우면 PCA에 중점을 두고 1에 가까우면 LDA에 중점을 둔다.

목표함수를 최적화하는 해를 도출하기 위해 Lagrange multiplier formulation을 적용하자. 이때, 제약 조건은 e가 단위벡터라는 것, 즉 $||e^2||$ =1이다.

$$L(e, \lambda) = J(e) + \lambda(||e^2|| - 1)$$

새롭게 얻은 목표 함수를 e로 편미분할 수 있고 그렇게 얻은 편미분된 식을 기울기로 해석할 수 있다. 목표함수를 최적화하는 해를 얻기위해서 목표 함수의 기울기를 e방향으로 최대화하기 위한 단위벡터로 만들어야 한다. 그렇기에 목표함수의 기울기를 기울기의 크기로 나누어 단위 벡터로 만들어준다. 단위벡터 e값의 갱신을 값이 더 이상 변하지 않을 때까지 계속해서 반복하여 기울기가 0이 되는 고정점을 찾는다. 이렇게 찾은 고정점이 우리의 목표함수를 최적화하는 해이다.

이렇게 찾아낸 해를 통해 우리는 PCA와 LDA를 동시에 수행할 수 있고 데이터를 효과적으로 재구성하여 분류 또는 판별 작업에 유용한 특성을 추출할 수 있습니다.

또한, 데이터 분석에 대한 깊은 이해와 연구를 위한 중요한 도구중 하나로 사용될 수 있다. 하지만 목표 함수를 최적화하려면 계산적으로 복잡할 수 있다. 매개 변수 ρ를 조절해야 하는데 적절한 균형을 찾는 것이 어려울 수 있다.

Q5

트리의 깊이는 10으로, max_features_values는 sqrt로 고정하면 트리의 수가 증가함에 따라 모델을 train하고 test하는 것에 훨씬 더 많은 시간이 소요됨을 확인할 수 있다. 하지만 Q1과 Q3에서 train하고 test할 때 소요된 시간보다는 적게 소요되었다. 따라서 랜덤 포레스트는 다른 알고리즘보다 train, test 속도가 짧음을 알 수 있다. 또한, 트리의 수가 증가할 수록 인식 정확도가 올라가는 경향을 보였다.

트리의 수는 10으로, max_features_values는 sqrt로 고정하고 트리의 깊이를 바꿔보면, 트리가 깊어질 수록 소요 시간이 커졌고 과대 적합의 영향인지 인식 정확도가 올라가지 않는 모습을 보였다. 그러나 랜덤 포레스트는 여러개의 트리를 사용하여 종합적인 결과를 도출하기에 개별 예측이 과대적합이 되더라도 준수한 인식 정확도를 보였다.

랜덤성을 늘릴수록 정확도가 떨어지는 모습을 보였다.

Weak learner를 다르게 하며 비교하였을 때, 정확도 측면에서는 axis-aligned과 two-pixel test의 차이가 없거나 two-pixel test의 정확도가 낮았다. 주로 트리의 수가 작을 때 two-pixel test의 정확도가 axis-aligned보다 낮은 경향이 있었다.

이는 트리의 수가 적으면 two-pixel test가 data의 특징을 구별할 수 없기 때문이라 예측된다.

```
Number of estimators: 10, Max depth: None, Max features: sqrt
Accuracy: 0.5576923076923077
Confusion matrix:
[[2 0 0 ... 0 0 0]
 [0 2 0 ... 0 0 0]
 [0 0 2 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 0]]
Time taken: 0.37186241149902344 seconds
------------
Example Success:
True Label: [29], Predicted Label: 29
```

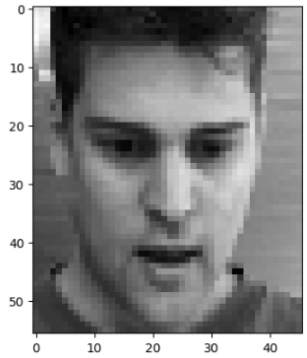

Fig6. Result of (5)

MATERIALS

Sourcecode:https://colab.research.google.com/drive/1VYImLH5Ms2oaBQJ4oIz2EXtgwK_1ROba?usp=sharing