# AI_project1


**Professor** : 최상호 교수님

**Student ID** : 2020202037

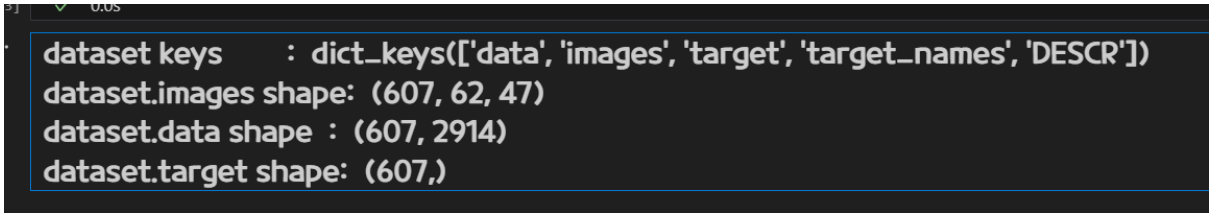
**Name** : 염정호


**Date** : 2024.10.08

# 프로젝트 개요

본 프로젝트에서는 feture extration 을 하는 PCA 을 구현하고 두 방법을 비교하는 프로젝트이다. feture extration 란 원본 데이터에서 유용한 정보를 나타내는 특징 또는 변수를 추출하는 작업을 의미하며 본 프로젝트에선 LWF dataset 을 이용하며 이는 파이썬에서 제공하는 데이터셋으로 여러 유명인의 얼굴이 담긴 이미지를 포함하며 이 데이터 셋을 이용해 PCA 를 적용한 feture extration 을 수행하고 추출한 feature 을 이용한 KNN 모델을 이용해 결과를 비교한다.

본 프로젝트 에서는 LWF 데이터 셋을 이용하는데 해당 이미지는 3 차원 데이터 이므로 pca 를 활용해 데이터 차원을 줄이기 위해 이미지를 평탄화 하는 과정이 필요하다. 3 차원 데이터가 평탄화 과정을 수행하게 되면




다음과 같이 2 차원 dataset 으로 변환된 것을 확인 할 수 있다.

PCA 주성분분석은 데이터의 차원 축소 기법으로 데이터의 구조를 이해하고 시각화 하면 노이즈를 줄이는데 유용하다. PCA 는 고차원 데이터의 변수를 몇 개의 주성분으로 변환하여 데이터의 본질적인 구조를 보존하면서 데이터의 차원을 줄이는 방법이다.
PCA 의 주요 요소는 데이터의 중심화, 공분산행렬의 계산, 고유값 분해, 그람슈미트 알고리즘, 데이터 변환의 방법으로 수행 가능하다.

1.공분산 행렬의 계산
공분산 행렬의 데이터의 각 특성관계를 이해하는데 주요한 역할을 하며 두 변수 간의 변동이 어떻게 함께 발생하는지 보여준다. 공분산 행렬의 계산식은

$$C = \frac{1}{N-1} X_{\text{centered}}^T X_{\text{centered}}$$

열의 평균을 계산하여 데이터의 각 특성에서 평균을 뺀 중심화된 데이터를 이용해 위와 같이 공분한 행렬 C 를 계산 가능하다.

Gram-schmidt process

그람슈미트 프로세스는 주어진 벡터 집합을 직교 또는 직교단위 집합으로 변환하는 방법이다. 보통 첫번째 열 벡터를 baisis 로 두 번재 열벡터부터 이전 열벡터들에 정사여한 벡터를 그 크기만큼 빼는 과정을 반복한다.

$$\text{proj}_u(v) = \frac{< v, u >}{< u, u >} u$$

위와 같은 식으로 나타낼 수 있으며 여기서 <v,u>는 내적을 의미한다. 직교화 하고 싶은 행렬 V 와 그 벡터들을 v 라고 할 때 직교화된 행렬 U 와 그 벡터들을 u 라고 할 때 그랑슈미트 과정을 수식으로 나타내면

$$u_1 = v_1$$

$$u_2 = v_2 - \text{proj}_{u_1}(v_2)$$

$$u_3 = v_3 - \text{proj}_{u_1}(v_3) - \text{proj}_{u_2}(v_3)$$

$$u_4 = v_4 - \text{proj}_{u_1}(v_4) - \text{proj}_{u_2}(v_4) - \text{proj}_{u_3}(v_4)$$

...

$$u_k = v_k - \sum_{j=1}^{k-1} \text{proj}_{u_j}(v_k)$$

$$e_k = \frac{u_k}{\|u_k\|}$$

정규직교 기저벡터 e 를 얻을 수 있게된다. 이제 공분산 행렬을 C 라 하고 그 벡터들을 c 라 할 때 아래 수식과 같이 QR 분해가 가능하다.

$$[c_1\ c_2\ c_3] = [e_1\ e_2\ e_3]\begin{bmatrix} c_1^T e_1 & c_2^T e_1 & c_3^T e_1 \\ 0 & c_2^T e_2 & c_3^T e_2 \\ 0 & 0 & c_3^T e_3 \end{bmatrix}$$

고유값 고유 벡터 추출

공분산 행렬을 R * Q 로 치환한 후 공분산 행렬이 대각 행렬에 수렴할 때까지 반복한다면 대각 성분을 공분산 행렬의 고유값으로 해석 가능하고 Q 행렬을 고유 벡터로 해석 가능하다. 이로써 고유값과 고유 벡터를 구하는 것이 가능하다.

PCA 는 고유값의 크기를 해당 파라미터의 중요도로 해석하는 것을 골자로 한다. 고유값과 고유 벡터를 구했으니, 고유값을 크기 순으로 나열하고, 해당하는 고유벡터를 따로 얻어 원본 데이터 를 선형 변환시킬 행렬을 얻는다. 해당 선형 변환이 완료된 데이터를 출력하면 PCA 를 이용한 Feature extraction 이 완료된다.

PCA with whitening

pca 화이트닝은 차원 축소와 특성 출에 사용되는 기술로 데이터의 특성이 단위 분산을 가지며 상관관계가 없도록 정규화 하는 것을 목표로 한다.

먼저 원본 데이터에서 공분산 행렬을 계산하고 공분산이 0 인 경우를 처리해 원본 데이터를 반환한다. 이제 각 특성의 평균을 빼 중심화를 수행하고 공분산의 제곱근으로 나누어 표준화를 수행 할 수 있다.

$$\text{Whitening}(x) = \frac{x - \text{mean}}{\text{std}}$$

KNN

K-최근접 이웃(KNN, K-Nearest Neighbors) 알고리즘은 주어진 데이터 포인트와 가장 가까운 K 개의 이웃을 찾아 그들의 레이블을 기반으로 분류하는 간단하고 효과적인 머신러닝 기법이다. K 값은 이웃의 수를 결정하며, 작으면 노이즈에 민감하고, 크면

과소적합이 발생할 수 있다. 새로운 데이터 포인트에 대해 학습 데이터의 각 포인트와 거리를 계산하고, K 개의 가장 가까운 이웃을 선택하여 가장 많이 등장하는 레이블을 새 데이터 포인트의 레이블로 지정한다. Python 의 scikit-learn 라이브러리를 이용해 Iris 데이터를 분할하고 KNN 모델을 훈련시키며 예측 후 성능을 평가 가능하다.

F1 score

정밀도는 모델이 양성으로 예측한 사례 중 실제로 양성인 비율을 의미하고, 재현율은 실제 양성인 사례 중에서 모델이 양성으로 정확하게 예측한 비율을 나타낸다. F1 스코어는 두 지표의 균형을 고려하여 모델의 성능을 평가하는 데 유용하며, 특히 클래스 불균형이 있는 데이터셋에서 더 신뢰할 수 있는 평가를 제공한다

$$\text{F1 score} = 2 * \frac{\text{recall} * \text{precision}}{\text{recall} + \text{precision}}$$

# 알고리즘

```python
def covariance(self,data): # clear
    mean = np.mean(data,axis = 0)

    centered_data = data - mean

    cov = np.dot(centered_data.T, centered_data) / (centered_data.shape[0] - 1)

    return cov
```

위 covariance 함수는 입력 받은 데이터에 대한 공분산 행렬을 구하는 함수이다 mean 함수를 이용해 데이터의 각 열에 대한 평균을 계산한 뒤 각 데이터에서 평균을 뺀 결과를 구한다. 이를 통해 데이터의 중심화가 가능하며 각 열의 평균이 0 이 되도록 할 수 있다. 전치행렬과 원래 중심화된 데이터를 곱하는 것으로, 공분산 행렬을 계산하는 것이 가능하다.

```
Custom covariance matrix:
[[ 0.1081907   0.01696511  0.0015109  -0.00648182  0.00887824]
 [ 0.01696511  0.08415406 -0.01441346  0.00241296 -0.00499052]
 [ 0.0015109  -0.01441346  0.08592277 -0.00196287 -0.00454427]
 [-0.00648182  0.00241296 -0.00196287  0.08743794 -0.00272508]
 [ 0.00887824 -0.00499052 -0.00454427 -0.00272508  0.07988535]]

Numpy covariance matrix:
[[ 0.1081907   0.01696511  0.0015109  -0.00648182  0.00887824]
 [ 0.01696511  0.08415406 -0.01441346  0.00241296 -0.00499052]
 [ 0.0015109  -0.01441346  0.08592277 -0.00196287 -0.00454427]
 [-0.00648182  0.00241296 -0.00196287  0.08743794 -0.00272508]
 [ 0.00887824 -0.00499052 -0.00454427 -0.00272508  0.07988535]]

Are the covariance matrices nearly identical? True
```

실행결과 해당 데이터에 대한 공분산행렬 값이 구해진 것을 확인했고 이는 numpy 를 이용한 연산 결과와 동일했다.

```python
def gram_schmidt(self, matrix):
    """Gram-Schmidt 과정을 통해 Q와 R을 동시에 구하는 함수"""
    n_vectors = matrix.shape[1]  # 열 벡터의 개수
    Q = np.zeros_like(matrix)  # 직교 기저를 저장할 행렬
    R = np.zeros((n_vectors, n_vectors), dtype=np.float64)  # R 행렬

    for i in range(n_vectors):
        # 처음에는 주어진 벡터를 그대로 사용
        q_i = matrix[:, i]

        # 이전 기저들에 대한 정사영을 빼줌
        for j in range(i):
            q_j = Q[:, j]
            R[j, i] = np.dot(q_j, matrix[:, i])  # R[j, i]를 정사영으로 저장
            q_i -= R[j, i] * q_j  # q_i에서 정사영을 빼줌

        # 직교화된 벡터를 Q 행렬의 i번째 열에 추가
        R[i, i] = np.linalg.norm(q_i)  # R[i, i]는 q_i의 노름
        Q[:, i] = q_i / R[i, i]  # 정규화하여 단위 벡터로 만듦

    return Q, R
```

그람슈미트 함수 구현

matrix 의 열 개수를 n_vectors 로 저장하고, 동일 크기의 0 으로 초기화된 행렬 Q 와 상삼각 행렬 R 을 생성한다. 각 열(벡터)을 순차적으로 선택하여 직교화 한다. 이때 원본 데이터를 수정하지 않기 위해 복사본을 이용한다. 계산된 직교 벡터들과의 내적을 통해 정사영을 계산하고, 이를 원래 벡터에서 제거하여 직교성을 확보한다. 정사영을 제거한 벡터를 정규화하여 Q 의 열에 추가하고, 관련 정보를 R 에 저장합니다. 직교화된 벡터로 이루어진 행렬 Q 와 상삼각 행렬 R 을 반환한다.

```python
def eig(self, data, max_iter=100):
    """QR 알고리즘을 통해 고유값과 고유벡터를 추출하는 함수"""
    # 공분산 행렬 계산
    cov_matrix = self.covariance(data)

    # QR 분해 초기화
    A = cov_matrix.copy()
    n = A.shape[0]
    Q_total = np.eye(n)  # 고유벡터들을 저장할 행렬

    for i in range(max_iter):
        # QR 분해 수행
        Q, R = self.gram_schmidt(A)

        # 행렬 갱신 (R * Q)
        A = np.dot(R, Q)

        # 고유벡터 계산을 위해 Q 행렬을 누적
        Q_total = np.dot(Q_total, Q)

        max_off_diag = self.max_off_diagonal(A)
        print(f"Iteration {i}, 대각 성분 제외 최대값: {max_off_diag}")

    # 대각 성분들이 고유값, Q_total이 고유벡터
    eigenvalues = np.diag(A)
    eigenvectors = Q_total

    return eigenvalues, eigenvectors
```

eig

QR 분해 알고리즘을 기반으로 주어진 데이터의 고유값과 고유벡터를 계산하는 함수이다. QR 분해 기반의 방법은 반복적으로 행렬을 갱신하여 최종적으로 고유값과 고유 벡터를 얻을 수 있다.

입력 data 로부터 공분산행렬을 계산하여 QR 분해 알고리즘을 적용할 준비를 하고 그람 슈미트 함수를 이용해 직교행렬 Q 를 얻고 직교행렬 Q 의 전치행렬과 원본 행렬 A 를 곱해 R 을 계산한다. A = np.dot(R,Q)를 반복적으로 수행하면 A 는 점점 대각화 되면서 고유 값이 대각 성분에 모이게 할 수 있다.

```python
def whitening(self, data):
    """데이터를 whiten 변환 (평균 제거 후 표준편차로 나누기)"""
    mean = np.mean(data, axis=0)  # 각 특징의 평균 계산
    std_dev = np.std(data, axis=0)  # 각 특징의 표준편차 계산

    # 표준편차가 0인 경우 1로 설정 (0으로 나누는 것을 방지하기 위해)
    std_dev[std_dev == 0] = 1

    # Whitening 변환 적용
    whitened_data = (data - mean) / std_dev

    return whitened_data
```

whitening 함수

데이터를 정규화 하여 평균이 0 이고 분산이 1 이 되도록 하는 과정인 화이트닝 함수로 공분산을 계산하고 주어진 데이터의 평균을 계산하여 각 특성의 평균값을 변수에 저장한다. 원본 데이터에서 평균 값을 빼서 평균 중심화 된 데이터를 생성한다. 평균 중심화 된 데이터에 대해 각 특성의 공분산의 제곱근으로 나누어 화이트닝 처리를 수행할 수 있다.

```
def fit(self, data):
    # Step 1: 고유값 및 고유벡터 계산
    eigenvalues, eigenvectors = self.eig(data)

    # Step 2: 고유값에 따라 내림차순 정렬
    sorted_indices = np.argsort(eigenvalues)[::-1]
    eigenvalues = eigenvalues[sorted_indices]
    eigenvectors = eigenvectors[:, sorted_indices]

    # Step 3: 주성분 선택
    self.components_ = eigenvectors[:, :self.n_components]

    # Step 4: 데이터 평균 중심화
    mean = np.mean(data, axis=0)
    centered_data = data - mean

    # Step 5: PCA 변환 (데이터를 주성분에 투영)
    pca_output = np.dot(centered_data, self.components_)

    # Step 6: Whitening 적용 (선택적)
    if self.whiten:
        pca_output /= np.sqrt(eigenvalues[:self.n_components])

    return pca_output
```

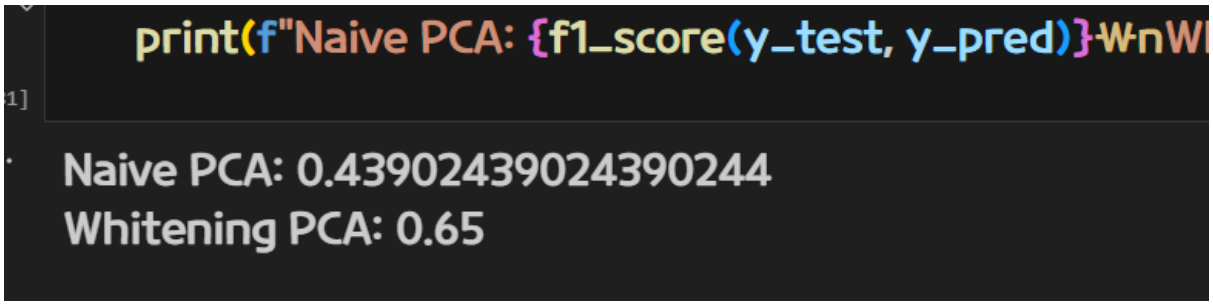
fit 함수

fit 함수는 주어진 데이터를 기반으로 PCA 를 수행하여 주성분을 선택하고 변환된 데이터를 반환하는 역할을 수행한다.

주어진 데이터로부터 고유값과 고유벡터를 하고 고유값을 내림차순으로 정렬하여 가장 중요한 주성분이 먼저 오도록 순서를 변경한다. 정렬된 고유벡터 중 주성분의 개수에 해당하는 것만 선택하여 저장한다. PCA 를 수행하기 전 데이터의 평균을 계산하고 이 평균을 데이터에서 빼 평균 중심화를 수행한다. 중심화 된 데이터에 주성분을 곱해 PCA 의 연산 결과를 계산한다.

필요에 따라 whitening 를 적용해 주성분의 분산을 조정할 수 있다.

# 실행결과




resize 를 0.3 으로 적용하고 n 회의 처리 과정을 실행한 결과

| N | 1 | 10 | 100 | 200 |
|---|---|---|---|---|
| Naive PCA | 0.4285 | 0.4390 | 0.439 | 0.439 |
| Whitening PCA | 0.6829 | 0.6976 | 0.65 | 0.65 |

화이트닝을 적용하지 않은 pca 의 경우 0.44, 화이트닝을 적용한 경우 0.65 로 예측 값이 더 높았는데 이는 화이트닝을 통해 각 특성간의 상관관계를 제거하여 독립적으로 만든 결과이고 모델의 특성을 개별적으로 분석할 수 있게 하여 더 나은 학습이 가능했다.

# Improvement

PCA 의 성능 향상을 시키는 방법으로는 스케일링 방법의 변화가 있다 PCA 데이터는 분산을 최대화 하는 방향으로 주성분을 찾기 때문에, 데이터의 스케일이 다르면 결과에 큰 영향을 미칠 수 있다. 따라서 데이터의 값을 0 과 1 사이의 범위로 변환하는 방법인 MIN-MAX 스케일링 방법이나 Z-점수 정규화 스케일링 방법을 이용할 수 있다. 또한 모디파이트 그람슈미츠 방법을 사용하는 것 또한 성능을 향상 시키는 방법이 될 수 있는데 해당 알고리즘은 기존 그람슈미츠 방법의 개선으로 고유벡터를 계산할 대 때 발생하는 수치적 오류를 줄이고 고유 벡터의 계산의 정확성을 개선하는데 도움을 줄 수 있기 때문이다.

# Consideration

이번 과제에서는 파이썬을 라이브러리를 이용한 PCA 구현했다. 평소 다루지 않던 파이썬을 다루다 보니 많은 어색함을 느꼈던 과제이다. 문법구조가 달랐지만 코드를 한줄씩 실행하다. 보니 오히려 코드를 수정하는 측면에선 더 편안함을 느꼈던 것 같다. pca 를 제작하면서 eig 를 통한 고유값 분해를 수행했을 때 대각성분을 제외한 나머지 행렬의 절대값의 최대값을 구하면서 해당 코드들의 동작 여부를 확인했으나 0 으로 수렴하지 않는 것처럼 보이는 결과가 나왔었다 해당 결과의 이유로는 부동소수점 연산으로 인한 오류 또는 낮은 값들을 처리할 때 생기는 문제로 확인했다. 본 프로젝트를 통해 주성분 분석 기법인  PCA 의 동작을 구현하고 실험하는 과정을 통해 해당 알고리즘에 대한 이해를 높일 수 있었다.

## 고찰