9주차 6장 차원 축소 필사

01. 차원 축소(Dimension Reduction) 개요

- 대표적인 차원 축소 알고리즘 : PCA, LDA, SVD, NMF
- 차원 축소
  - 매우 많은 피처로 구성된 다차원 데이터 세트의 차원을 축소해 새로운 차원의 데이터 세트를 생성하는 것
  - 일반적으로 차원이 증가할수록 데이터 포인트 간의 거리가 기하급수적으로 멀어짐, 희소한 구조를 가짐
  - 수백 개 이상의 피처로 구성된 데이터 세트의 경우
    - 예측 신뢰도가 떨어짐
    - 개별 피처간에 상관관계가 높을 가능성이 큼
    - 선형 회귀와 같은 선형 모델 : 입력 변수 간의 상관 관계가 높을 경우 다중 공선성 문제 → 모델 예측 성능 저하
  - 다차원의 피처를 차원 축소해 피처 수를 줄이면 더 직관적으로 데이터 해석 가능
    - 3차원 이하의 차원 축소를 통해 시각적으로 데이터를 압축하여 표현 가능
  - 차원 축소를 하면 학습 데이터의 크기가 줄어 학습에 필요한 처리 능력도 줄일 수 있음
  - 피처 선택
    - 특정 피처에 종속성이 강한 불필요한 피처는 아예 제거하고 데이터의 특징을 잘 나타내는 주요 피처만 선택
  - 피처 추출
    - 기존 피처를 저차원의 중요 피처로 압축해서 추출
    - 새롭게 추출된 중요 특성은 기존 피처와는 완전히 다른 값
    - 함축적인 특성 추출
      - 단순 압축이 아닌, 피처를 함축적으로 더 잘 설명할 수 있는 또 다른 공간으로 매핑해 추출하는 것
        - 예) 학생 평가 요소 : 모의고사 성적, 종합 내신성적, 수능성적, 봉사활동, 대외활동, 학교 내외 수상경력 → 학업 성취도, 커뮤니케이션 능력,

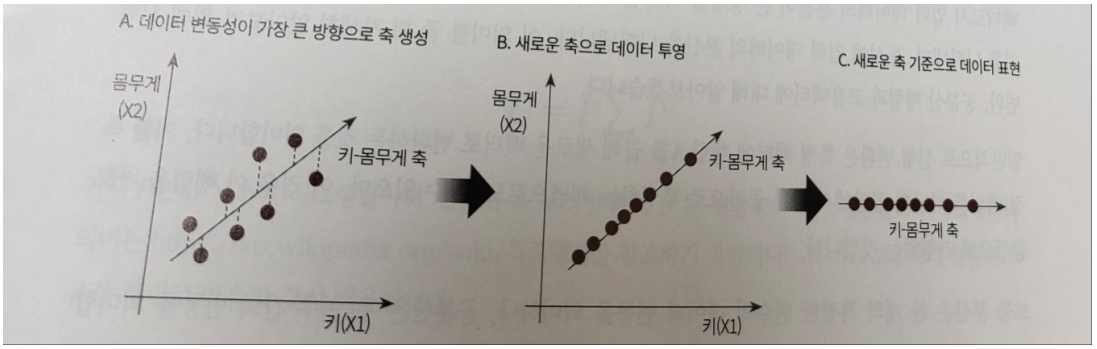
문제 해결력으로 추출 가능

- 기존 피처가 전혀 인지하기 어려웠던 잠재적인 요소를 추출하는 것

- 차원 축소의 중요한 의미 : 차원 축소를 통해 데이터를 더 잘 설명할 수 있는 잠재적인 요소를 추출
- 이미지 데이터에서의 차원 축소 알고리즘
  - 많은 픽셀로 이루어진 이미지 데이터에서 잠재된 특성을 피처로 도출해 함축적 형태의 이미지 변환, 압축 수행 가능
  - 변환된 이미지는 원본 보다 훨씬 적은 차원 → 분류 수행 시 과적합 영향력이 작아짐 → 예측 성능 증대
- 텍스트 문서에서의 차원 축소 알고리즘
  - 텍스트 문서의 숨겨진 의미 추출 가능
  - 문서 내 단어들의 구성에서 숨겨져 있는 시맨틱 의미나 토픽을 잠재 요소로 간주하고 찾아낼 수 있음
  - SVD, NMF는 시맨틱 토픽(Semantic Topic) 모델링을 위한 기반 알고리즘으로 사용됨

02. PCA(Principle Component Analysis)

- 가장 대표적인 차원 축소 기법
  - CV 분야에 더 활발하게 적용됨 (얼굴 인식의 경우 Eigen-face라고 불리는 PCA 변환으로 원본 얼굴 이미지를 변환해 사용하는 경우가 많음)
- 여러 변수 간에 존재하는 상관관계를 이용해 이를 대표하는 주성분을 추출해 차원을 축소하는 기법
- 기존 데이터의 정보 손실이 최소화되는 것이 당연함
- 차원 축소 방법
  - 가장 높은 분산을 가지는 데이터 축을 찾아 이 축으로 차원을 축소함 → PCA의 주성분이 됨
    - 즉, 분산이 데이터의 특성을 가장 잘 나타내는 것으로 간주
  - 키와 몸무게 2개의 피처를 가지고 있는 데이터 세트 → 한 개의 주성분을 가진 데이터 세트로 차원 축소 가능 (데이터 변동성이 가장 큰 방향으로 축을 생성 → 새롭게 생성된 축으로 데이터 투영)



1. 가장 큰 데이터 변동성을 기반으로 첫 번째 벡터 축 생성, 두 번째 축은 이 벡터 축에 직각이 되는 벡터(직교 벡터)를 축으로 함
2. 세 번째 축은 다시 두 번째 축과 직각이 되는 벡터를 설정하는 방식으로 축을 생성
3. 생성된 벡터 축에 원본 데이터를 투영하면 벡터 축의 개수만큼의 차원으로 원본 데이터가 차원 축소됨

- 원본 데이터의 피처 개수에 비해 매우 작은 주성분으로 원본 데이터의 총 변동성을 대부분 설명할 수 있는 분석법
- 선형대수 관점에서 PCA 해석
  - 입력 데이터의 공분산 행렬을 고유값 분해 → 고유벡터에 입력 데이터 선형 변환
  - 고유벡터가 PCA의 주성분 벡터로서 입력 데이터 분산이 큰 방향을 나타냄
  - 고윳값 : 고유벡터의 크기, 입력 데이터 분산을 나타냄
  - 선형 변환 : 특정 벡터에 행렬 A를 곱해 새로운 벡터로 변환하는 것
  - 공분산 : 두 변수 간의 변동, 여러 변수와 관련된 공분산을 포함하는 정방향 행렬

	X	Y	Z
X	**3.0**	−0.71	−0.24
Y	−0.71	**4.5**	0.28
Z	−0.24	0.28	**0.91**

  - 예시
    - 대각선 원소는 각 변수(X,Y,Z)의 분산을 의미
    - 대각선 이외 원소 : 가능한 모든 변수 쌍 간의 공분산을 의미
    - X,Y,Z의 분산 : 3.0, 4.5, 0.91

- X와 Y의 공분산 : -0.71, X와 Z의 공분산은 -0.24, Y와 Z의 공분산은 0.28

- 고유벡터 : 행렬 A를 곱하더라도 방향이 변하지 않고 그 크기만 변하는 벡터 (Ax= ax A는 행렬, x는 고유벡터, a는 스칼라값)
  - 고유벡터는 여러 개가 존재하며, 정방 행렬은 최대 그 차원 수만큼의 고유벡터를 가질 수 있음
    - 2X2 행렬은 2개의 고유벡터, 3X3 행렬은 3개의 고유벡터를 가질 수 있음
  - 고유벡터는 행렬이 작용하는 힘의 방향과 관계가 있어서 행렬을 분해하는 데 사용됨
- 공분산 행렬 : 정방행렬이며 대칭행렬(전치행렬 = 행렬)
  - 개별 분산값을 대각 원소로 하는 대칭행렬 → 항상 고유벡터를 직교행렬로, 고유값을 정방 행렬로 대각화할 수 있음

$$C = P \sum P^T$$

  - P는 nXn의 직교 행렬, 시그마는 nXn 정방행렬, P^T는 행렬 P의 전치 행렬
  - 다음과 같이 대응된다.

$$C = [e_1 \cdots e_n] \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} e_1^t \\ \cdots \\ e_n^t \end{bmatrix}$$

  - 즉, 공분산 C는 고유벡터 직교 행렬 * 고유값 정방 행렬 * 고유벡터 직교 행렬의 전치 행렬로 분해됨
    - e1는 1번째 고유벡터를, 람다i는 i번째 고유벡터의 크기를 의미함
    - e1는 가장 분산이 큰 방향을 가진 고유벡터이며, e2는 e1에 수직이면서 다음으로 가장 분산이 큰 방향을 가진 고유벡터임

- ⇒ 입력 데이터의 공분산 행렬이 고유벡터와 고유값으로 분해될 수 있으며, 이렇게 분해된 고유벡터를 이용해 입력 데이터를 선형 변환하는 방식 = PCA

- PCA 수행 과정
  1. 입력 데이터 세트의 공분산 행렬을 생성함
  2. 공분산 행렬의 고유벡터와 고유값을 계산함
  3. 고유값이 가장 큰 순으로 K개(PCA 변환 차수만큼)만큼 고유벡터를 추출함
  4. 고유값이 가장 큰 순으로 추출된 고유벡터를 이용해 새롭게 입력 데이터를 변환함
- 예제) iris 데이터 세트
  - 위 데이터 세트는 sepal length, sepal width, petal length, petal width 4개의 속성으로 되어 있음 → 2개의 PCA 차원으로 압축하여 원래 데이터 세트와 비교
  1. 사이킷런의 붓꽃 데이터를 load_iris() API를 이용해 로딩한 뒤 시각화를 편하게 하기 위해 DataFrame으로 변환함
  2. 각 품종에 따라 원본 붓꽃 데이터 세트가 어떻게 분포돼 있는지 2차원으로 시각화 (sepal length와 sepal width를 x축과 y축으로 하여 품종 데이터 분포를 나타냄)
     a. setas 품종의 경우 sepal width와 sepal length가 일정하게 분포되어 있지만 versicolor와 virginica는 sepal width와 sepal length 조건만으로는 분류가 어려움
  3. PCA로 4개 속성을 2개로 압축한 후 2개의 PCA 속성으로 붓꽃 데이터 품종 분포를 2차원으로 시각화
     a. PCA를 적용하기 전 개별 속성을 함께 스케일링해야 함 → 각 속성값을 동일한 스케일로 변환하는 것이 필요함
     b. StandardScaler를 이용해 평균이 0, 분산이 1인 표준 정규 분포로 iris 데이터 세트의 속성값들을 변환함
     c. 스케일링이 적용된 데이터 세트에 PCA를 적용해 2차원 PCA 데이터로 변환
        i. 사이킷런은 PCA 클래스 제공함 (생성 파라미터로 n_components를 입력받음)
        ii. n_components는 PCA로 변환할 차원의 수를 의미함 (예제에서는 2로 설정)
        iii. fit과 transform을 호출해 PCA로 변환 수행
  4. 변환된 PCA 데이터 세트를 DataFrame으로 변환, 2차원상에서 시각화

    a. pca_component_1 속성을 X축으로, pca_component_2 속성을 Y축으로 해서 붓꽃 품종 분포 확인

    b. PCA 변환 후에도 x축을 기반으로 setosa 품종은 명확하게 구분 가능, versicolor와 virginica는 x축을 기반으로 서로 겹치는 부분이 일부 존재하지만 비교적 잘 구분됨 → x축이 원본 데이터의 변동성을 잘 반영했기 때문

5. PCA component 별로 원본 데이터의 변동성을 얼마나 반영하고 있는지 알아보기

    a. explained_variance_ratio_ 속성 : 전체 변동성에서 개별 PCA 컴포넌트별로 차지하는 변동성 비율 제공

    b. pca_component_1이 전체 변동성의 약 72.9% 차지, 두번째인 pca_component_2가 약 22.8%를 차지함 → PCA를 2개 요소로만 변환해도 원본 데이터의 변동성을 95% 설명 가능

6. 원본 데이터 세트와 PCA로 변환된 데이터 세트에 각각 분류 적용하여 결과 비교

    a. Estimator는 RandomForestClassifier를 이용하고 cross_val_score()로 3개의 교차 검증 세트로 정확도 결과 비교

    b. 원본 데이터 세트 대비 예측 정확도는 PCA 변환 차원 개수에 따라 예측 성능이 떨어질 수밖에 없음

        i. 붓꽃 데이터의 경우 2개의 변환 속성으로 감소하면서 예측 성능 정확도가 원본 대비 8% 하락함 → 비교적 큰 성능 수치의 감소지만, 속성 개수가 50% 감소한 것 대비 원본 데이터의 특성을 상당 부분 유지하고 있음

- 예제 2) 신용카드 고객 데이터 세트

1. DataFrame으로 데이터 세트 로딩

    a. 위 데이터 세트는 30,000개의 레코드와 24개의 속성을 가짐

    b. target 값 : default payment next month (다음달 연체 여부를 의미, 연체=1, 정상납부=0)

2. 데이터 세트 칼럼 변환

    a. PAY_0 칼럼을 PAY_1로 변환, default payment next month 칼럼도 default로 변경

    b. default 칼럼을 y_target 변수로 별도로 저장, 피처 데이터는 default 칼럼을 제외한 별도의 DataFrmae으로 만듦

3. 데이터 세트 시각화

    a. DataFrame의 corr()를 이용해 각 속성 간의 상관도를 구한 뒤 시본의 heatmap으로 시각화

b. BILL_AMT1~BILL_AMT6 6개 속성끼리의 상관도가 대부분 0.9 이상으로 매우 높음

c. PAY_1~PAY_6까지의 속성 역시 상관도가 높음

d. → 높은 상관도를 가진 속성들은 소수의 PCA만으로도 자연스럽게 속성들의 변동성 수용 가능

4. PCA 변환 및 컴포넌트의 변동성 확인

a. BILL_AMT1~BILL_AMT6 6개 속성을 2개의 컴포넌트로 PCA 변환한 뒤 개별 컴포넌트의 변동성을 explained_variance_ratio_ 속성으로 알아봄

b. 단 2개의 PCA 컴포넌트만으로도 6개 속성의 변동성을 95% 이상 설명 가능

5. 원본 데이터 세트와 PCA 변환 데이터 세트의 분류 예측 결과 상호 비교

a. 원본 데이터 세트 : 랜덤 포레스트를 이용해 타겟 값인 디폴트 값을 3개의 교차 검증 세트로 분류 예측 → 평균 예측 정확도 : 81.7%

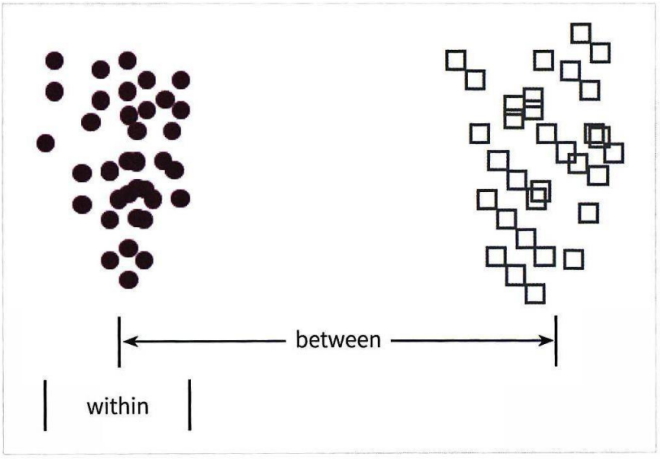
b. PCA 변환한 데이터 세트 : 동일한 분류 예측 적용 → 평균 예측 정확도 : 79.73%

i. 원본 대비 약 1~2%의 예측 성능 저하만 발생

ii. PCA의 뛰어난 압축 능력을 잘 보여줌

03. LDA(Linear Discriminant Analysis)

- 개요
  - LDA는 선형 판별 분석법으로 불리며 PCA와 매우 유사함
  - LDA는 PCA와 유사하게 입력 데이터 세트를 저차원 공간에 투영해 차원을 축소하는 기법
  - 중요한 차이점은 LDA는 지도학습의 분류에서 사용하기 쉽도록 개별 클래스를 분별할 수 있는 기준을 최대한 유지하면서 차원 축소
    - PCA는 입력 데이터 변동성의 가장 큰 축을 찾음 vs LDA는 입력 데이터의 결정 값 클래스를 최대한 분리할 수 있는 축을 찾음
  - LDA는 특정 공간상에서 클래스 분리를 최대화하는 축을 찾기 위해 클래스 간 분산과 클래스 내부 분산의 비율을 최대화하는 방식으로 차원 축소
    - 클래스 간 분산은 최대한 크게 가져가고, 클래스 내부의 분산은 최대한 작게 가져가는 방식



- LDA를 구하는 스텝
  - PCA와 유사하나 가장 큰 차이점은 공분산 행렬이 아니라 클래스 간 분산과 클래스 내 분산 행렬을 생성한 뒤, 이 행렬에 기반해 고유벡터를 구하고 입력 데이터를 투영함

1. 클래스 내부와 클래스 간 분산 행렬을 구합니다. 이 두 개의 행렬은 입력 데이터의 결정 값 클래스별로 개별 피처의 평균 벡터(mean vector)를 기반으로 구합니다.
2. 클래스 내부 분산 행렬을 S_W, 클래스 간 분산 행렬을 S_B라고 하면 다음 식으로 두 행렬을 고유벡터로 분해할 수 있습니다.

$$S_W^T S_B = \begin{bmatrix} e_1 & \cdots & e_n \end{bmatrix} \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} e_1^T \\ \cdots \\ e_n^T \end{bmatrix}$$

3. 고유값이 가장 큰 순으로 K개(LDA변환 차수만큼) 추출합니다.
4. 고유값이 가장 큰 순으로 추출된 고유벡터를 이용해 새롭게 입력 데이터를 변환합니다.

- 붓꽃 데이터 세트에 LDA 적용하기
  - 사이킷런의 LDA를 이용하여 변환, 그 결과를 품종별로 시각화
  - 사이킷런은 LDA를 LinearDiscriminantAnalysis 클래스로 제공함

1. 붓꽃 데이터 세트 로드, 표준 정규 분포로 스케일링
2. 2개의 컴포넌트로 붓꽃 데이터를 LDA 변환
   a. LDA는 실제로는 지도학습이므로 클래스의 결정 값이 변환 시에 필요함
   b. lda 객체의 fit() 메서드를 호출할 때 결정값이 입력됐음에 유의해야 함
3. LDA 변환된 입력 데이터 값을 2차원 평면에 품종별로 표현

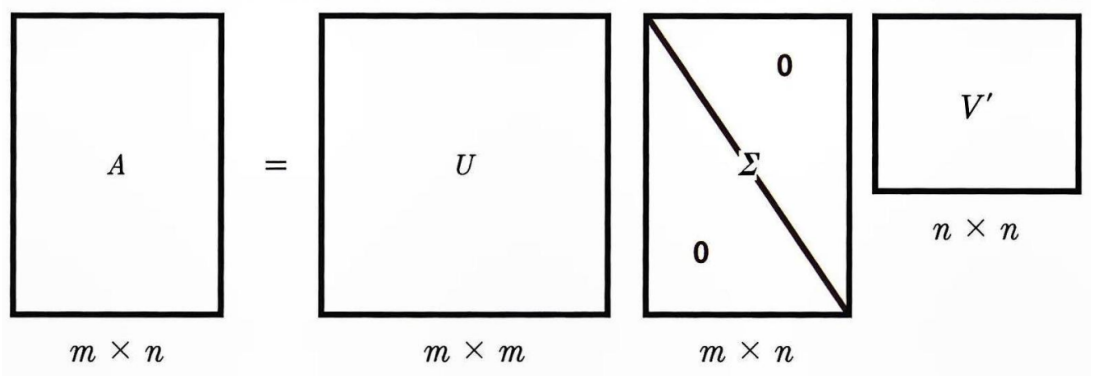
4. LDA로 변환된 붓꽃 데이터 세트 시각화
    a. PCA로 변환된 데이터와 좌우 대칭 형태로 많이 닮아 있음

04. SVD(Singular Value Decomposition)

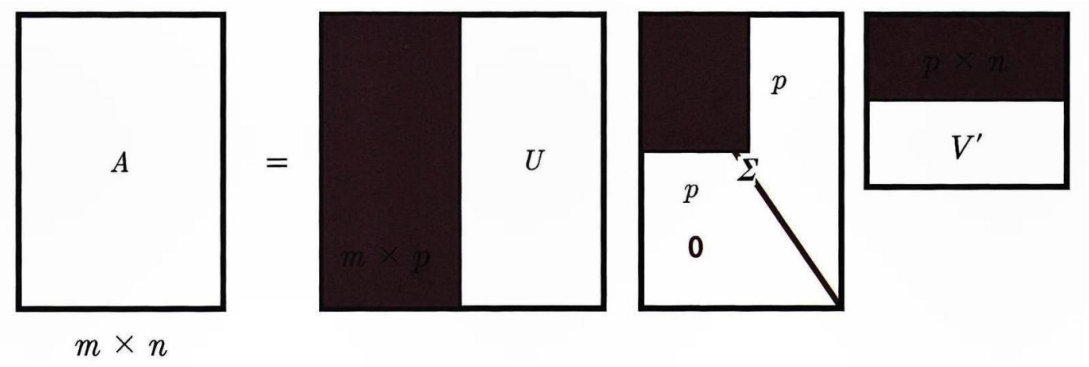
- 개요
    - SVD 역시 PCA와 유사한 행렬 분해 기법 이용
    - SVD는 정방행렬뿐만 아니라 행과 열의 크기가 다른 행렬에도 적용 가능
    - 일반적으로 SVD는 m X n 크기의 행렬 A를 다음과 같이 분해하는 것을 의미함

$$A = U \sum V^T$$

    - SVD는 특이값 분해로 불리며 U와 V에 속한 벡터는 특이벡터(singular vector)임. 모든 특이벡터는 서로 직교하는 성질을 가짐
    - 시그마는 대각행렬이며, 행렬의 대각에 위치한 값만 0이 아니고 나머지 위치의 값은 모두 0임
    - 시그마가 위치한 0이 아닌 값이 바로 행렬 A의 특이값임
    - SVD는 A의 차원이 mXn일 때 U의 차원이 mXm, 시그마의 차원이 mXn, V^T의 차원이 nXn으로 분해함



    - 일반적으로는 다음과 같이 시그마의 비대각인 부분과 대각원소 중에 특이값이 0인 부분도 모두 제거하고 제거된 시그마에 대응되는 U와 V 원소도 함께 제거해 차원을 줄인 형태로 SVD를 적용함
    - 컴팩트한 형태로 SVD를 적용하면 A의 차원이 mXn일 때, U의 차원을 mXp, 시그마의 차원을 pXp, V^T의 차원을 pXn으로 분해함

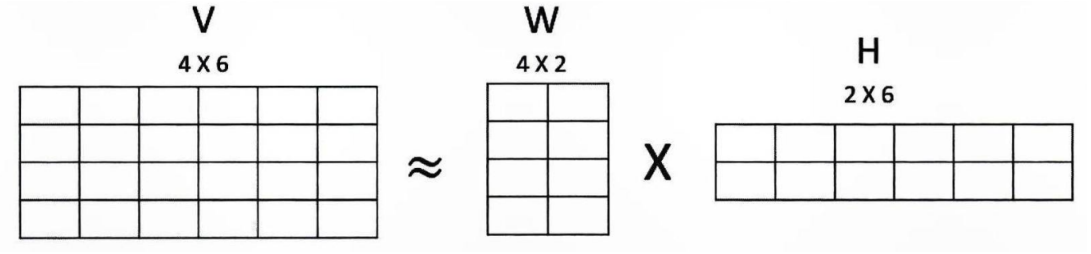


    - Truncated SVD는 시그마의 대각원소 중에 상위 몇 개만 추출해서 여기에 대응하는 U와 V의 원소도 함께 제거해 더욱 차원을 줄인 형태로 분해함
    - 일반적인 SVD는 보통 넘파이, 사이파이 라이브러리를 이용해 수행함
- 예제) 넘파이의 SVD를 이용해 SVD 연산 수행, SVD로 분해가 어떤 식으로 되는가
    - 넘파이 SVD 모듈인 numpy.linalg.svd를 로딩함, 랜덤한 4X4 넘파이 행렬 생성
        - 랜덤 행렬 생성하는 이유 : 행렬의 개별 로우끼리의 의존성을 없애기 위함
    - 생성된 a 행렬에 SVD를 적용해 U, Sigma, Vt 도출
        - SVD 분해는 numpy.linalg.svd에 파라미터로 원본 행렬을 입력하면 U 행렬, Sigma 행렬, V 전치 행렬을 반환함
        - Sigma 행렬의 경우 0이 아닌 값의 경우만 1차원 행렬로 표현함
        - 원본 행렬로의 복원은 이 U, Sigma, Vt를 내적하면 됨 (유의할 부분 : Sigma의 경우 0이 아닌 값만 1차원으로 추출했으므로 다시 0을 포함한 대칭행렬로 변환한 뒤 내적을 수행해야 함)
    - 데이터 세트가 로우 간 의존성이 있을 경우 어떻게 Sigma 값이 변하고 이에 따른 차원 축소가 진행되는가
        - 일부러 의존성을 부여하기 위해 a 행렬의 3번째 row를 ‘첫 번째 row+두 번째 row’로 업데이트, 4번째 row는 첫 번째 row와 같다고 업데이트함 → 로우 간 관계가 매우 높아짐
        - 위 데이터를 SVD로 다시 분해
            - Sigma 값 중 2개가 0으로 변함. ⇒ 선형 독립인 로우 벡터 개수가 2개라는 의미 (Rank=2)
            - U 행렬 중 선형 두 개의 열만 추출하고, Vt의 경우는 선행 두 개의 행만 추출해 복원
        - Truncated SVD를 이용해 행렬 분해

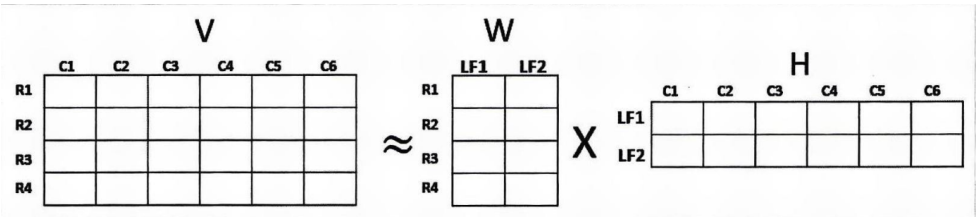
        - 시그마 행렬에 있는 대각원소, 즉 특이값 중 상위 일부 데이터만 추출해 분해하는 방식
        - 인위적으로 더 작은 차원의 U, Sigma, Vt로 분해하기 때문에 원본 행렬을 정확하게 다시 원복할 수는 없음 → 데이터 정보가 압축되어 분해됨에도 불구하고 상당한 수준으로 원본 행렬을 근사할 수는 있음
        - 넘파이가 아닌 사이파이에서만 지원됨
        - 사이파이의 Truncated SVD는 희소 행렬로만 지원되어 spicy.sparse.linalg.svds를 이용해야 함
        - 임의의 원본 행렬 6X6을 Normal SVD로 분해해 분해된 행렬의 차원과 Sigma 행렬 내의 특이값을 확인 → 다시 Truncated SVD로 분해해 분해된 행렬의 차원, Sigma 행렬 내의 특이값, 그리고 Truncated SVD로 분해된 행렬의 내적을 계산 → 다시 복원된 데이터와 원본 데이터 비교
            - (6,6) (6,) (6,6) → (6,4) (4,) (4,6) 으로 분해됨. 즉, 완벽하게 복원되지 않고 근사적으로 복원됨
- 사이킷런 TruncatedSVD 클래스를 이용한 변환
    - 사이킷런 TruncatedSVD 클래스는 사이파이의 svds와 같이 Truncated SVD 연산을 수행해 원본 행렬을 분해한 U, Sigma, Vt 행렬을 변환하지는 않음
    - PCA 클래스와 유사하게 fit()와 transform()을 호출해 원본 데이터를 몇 개의 주요 컴포넌트로 차원을 축소해 변환함
    - 원본 데이터를 Truncated SVD 방식으로 분해된 U*Sigma 행렬에 선형 변환해 생성
    - PCA와 유사하게 변환 후 품종별로 어느 정도 클러스터링이 가능한 정도로 각 변환 속성으로 뛰어난 고유성을 가지고 있음
    - 두 개의 변환 행렬 값과 원본 속성별 컴포넌트 비율값을 비교해보면 서로 거의 같음
        - 즉, 데이터 세트가 스케일링으로 데이터 중심이 동일해지면 사이킷런의 SVD와 PCA는 동일한 변환 수행 → PCA가 SVD 알고리즘으로 구현됐음을 의미함
        - PCA는 밀집 행렬에 대한 변환만 가능, SVD는 희소 행렬에 대한 변환도 가능함
- SVD 사용
    - CV에서 이미지 압축을 통한 패턴 인식과 신호 처리 분야에 사용됨
    - 텍스트의 토픽 모델링 기법인 LSA의 기반 알고리즘

05. NMF(Non-Negative Matrix Factorization)

- 개요
  - Truncated SVD와 같이 낮은 랭크를 통한 행렬 근사 방식의 변형
  - 원본 행렬 내의 모든 원소 값이 모두 양수라는 게 보장되면 다음과 같이 좀 더 간단하게 두 개의 기반 양수 행렬로 분해될 수 있는 기법을 지칭함



  - 행렬 분해 : 일반적으로 SVD와 같은 행렬 분해 기법을 통칭함
    - 행렬 분해 결과 W 행렬과 H 행렬은 일반적으로 길고 가는 행렬과 작고 넓은 행렬로 분해됨
    - 분해된 행렬은 잠재 요소를 특성으로 가짐
      - W는 원본 행에 대해서 이 잠재 요소의 값이 얼마나 되는지에 대응
      - H는 이 잠재 요소가 원본 열(즉, 원본 속성)로 어떻게 구성되었는지를 나타내는 행렬



  - NMF는 SVD와 유사하게 차원 축소를 통한 잠재 요소 도출로 이미지 변환 및 압축, 텍스트의 토픽 도출 등의 영역에서 사용됨
    - 사이킷런에서는 NMF 클래스를 이용해 지원됨
  - 예제) 붓꽃 데이터를 NMF를 이용 → 2개의 컴포넌트로 변환하고 시각화
  - NMF는 SVD와 유사하게 이미지 압축을 통한 패턴 인식, 텍스트의 토픽 모델링 기법, 문서 유사도 및 클러스터링에 잘 사용됨
    - 특히 영화 추천과 같은 추천 영역에 활발하게 적용됨
    - 잠재 요소 기반 추천 방식

        - 사용자의 상품(예: 영화) 평가 데이터 세트인 사용자-평가 순위 데이터 세트를 행렬 분해 기법을 통해 분해 → 사용자가 평가하지 않은 상품에 대한 잠재적인 요소 추출 → 이를 통해 평가 순위 예측 → 높은 순위로 예측된 상품 추천

06. 정리

- PCA
    - 많은 피처로 이뤄진 데이터 세트를 더욱 직관적으로 이해할 수 있음
    - 데이터를 잘 설명할 수 있는 잠재적 요소를 추출하는 데 큰 의미가 있음
    - 입력 데이터 변동성이 가장 큰 축을 구하고, 다시 이 축에 직각인 축을 반복적으로 축소하려는 차원 개수만큼 구한 뒤 입력 데이터를 이 축들에 투영해 차원을 축소하는 방식
    - 입력 데이터의 공분산 행렬을 기반으로 고유 벡터 생성 → 고유 벡터에 입력 데이터를 선형 변환
- LDA
    - PCA와 매우 유사, LDA는 입력 데이터의 결정 값 클래스를 최대한으로 분리할 수 있는 축을 찾는 방식으로 차원 축소
- SVD, NMF
    - 매우 많은 피처 데이터를 가진 고차원 행렬을 두 개의 저차원 행렬로 분리하는 행렬 분해 기법
    - 원본 행렬에서 잠재된 요소를 추출하기 때문에 토픽 모델링이나 추천 시스템에서 활발하게 사용됨