선형대수 Term Project – SVD

2017027265 여주안

1. Introduction

한 학기 동안 배운 선형대수의 내용을 바탕으로 SVD(Singular Value Decomposition), PCA(Principal Component Analysis)을 구현한다. 순서는 다음과 같다:

1. Element간의 상관 관계와 규칙을 정의하고, 100-D 벡터 1000개를 생성
2. 데이터를 결합하여 Matrix A를 만들고 A에 대해 SVD 진행
3. SVD로 얻은 Eigen Vector에서 일정 수의 Principal Component를 선택
4. 100개의 100-D 벡터를 동일한 규칙을 적용하여 생성
5. 선택한 Principal Component로 생성한 100-D 벡터에 대한 Representation 생성
6. 생성된 Representation 벡터와 100-D 벡터들 사이의 Average Vector Distance를 계산
7. 3에서 Principal Component의 수를 10, 15, 20, … , rank(A)로 증가시키며 5, 6 반복

이 보고서는 각 단계를 어떻게 정의하고 실행했는지 설명하고 3가지 주제에 대해 논의한다:

1. Why the graph shows the ‘elbow point’?
2. SVD and PCA
3. Data Compression

(모든 코드 설명은 생략 – Python / Numpy / Jupyter Notebook 사용, 엑셀 그래프)

2. Generate Vectors (1)

문제 정의: 100 차원의 벡터 1000개를 일정한 규칙에 따라 생성한다.

해결: Column Vector들을 생성한다. 일정한 규칙을 적용하여 1000 차원의 벡터 100개를 생성한다.

<벡터 생성 규칙>

[V1, V3, V5, V7, V9]: Range -100 ~ 100의 무작위 정수로 이루어진 1000 차원 벡터를 생성한다. V2, V4, V6, V8, V10와 선형 관계를 이룬다.

[V2, V4, V6, V8, V10]: V(N-1)의 모든 요소에 특정한 계수를 곱해서 생성한다 (Ex. V2 = 2 x V1)

[V11 – V55]: Range -20 ~ 20의 무작위 정수로 이루어진 1000차원 벡터를 45개 생성한다.

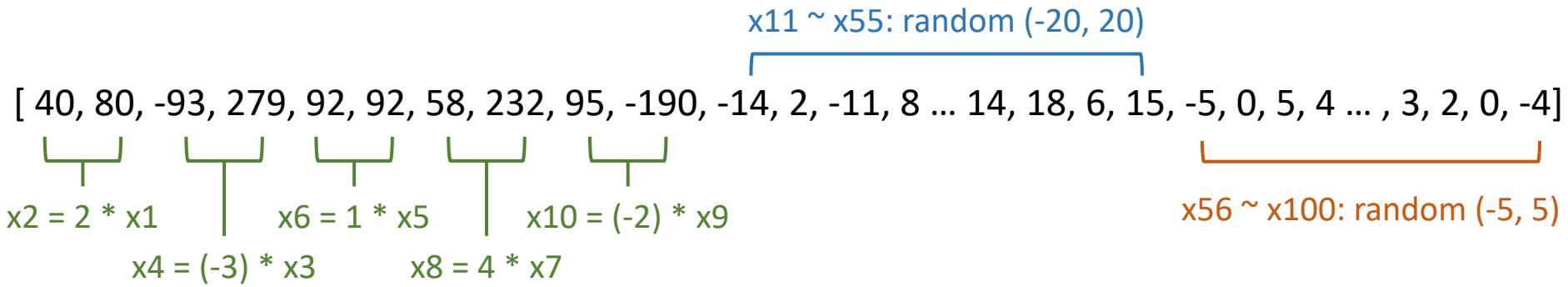
[V56 – V100]: Range -5 ~ 5의 무작위 정수로 이루어진 1000차원 벡터를 45개 생성한다.

Relations: V2 = 2 x V1 V4 = (-3) x V3 V6 = 1 x V5 V8 = 4 x V7 V10 = (-2) x V9

총 5개의 관계를 정의한다. 이 중 (V3, V4), (V9, V10)은 음의 상관관계를 가지며 (V1, V2), (V5, V6), (V7, V8)은 양의 상관관계를 가진다. V2 – V10의 범위는 곱해진 계수만큼 늘어나는데 V8 요소의 최솟값은 -400, 최댓값은 400이다.

3. Generate Vectors (2)

생성된 벡터 예시 (V1~V100 첫번째 element 결합)



벡터 생성 규칙의 의미:

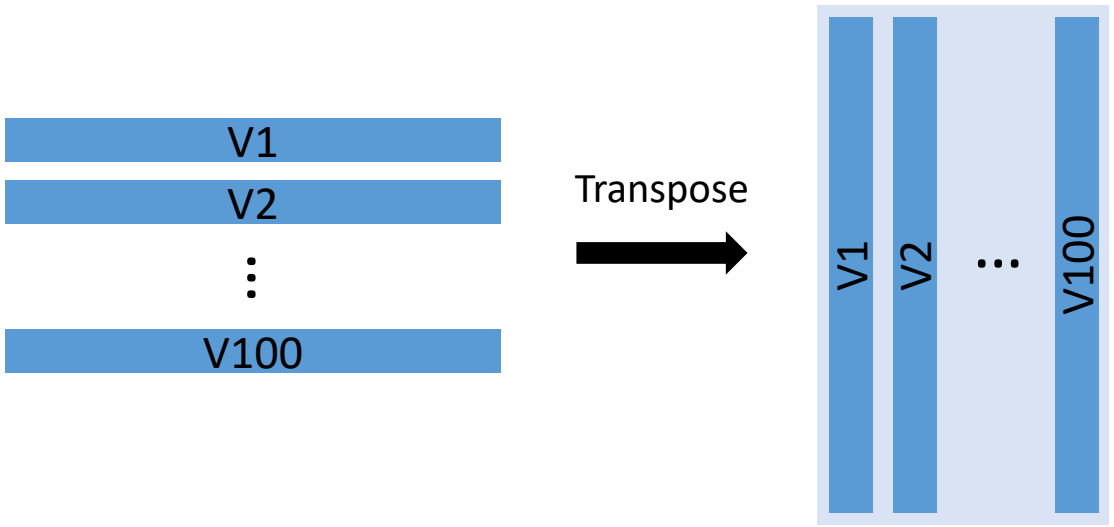
데이터는 항목 간에 상관관계가 있다. 실제 데이터는 X5 = X1 – 2*X3^2 + 11*X4 와 같이 여러 요소가 연관된 복잡한 관계를 가질 수 있지만, 이번 프로젝트에서는 2항목 씩 짝지어 일차식의 관계를 가지도록 정의한다. 이런 항목간의 관계를 나타내기 위해 사용하는 요소가 x1 – x10 이다. x1 – x10 항목을 위해 V1 – V10 벡터를 만든다. 홀수 번째 벡터는 -100 부터 100까지의 임의의 수를, 짝수 번째 벡터는 홀수 번째 벡터에 특정 계수를 곱해서 생성한다.

V11 – V55까지는 -20 부터 20, V56 –V100은 -5 부터 5의 범위로 임의의 수를 생성한다. 범위가 V1 – V10 벡터(-100 ~ 100) 보다 작은 것은 상관관계 없이 랜덤으로 만들어진 벡터(V11-V55) 항목의 크기를 줄여 요소 x1 – x10을 dominant 하게 만들기 위한 것이다. x1 – x10은 절댓값이 큰 데이터가 생성될 확률이 높아 결과적으로 Principal Component 에서 x1 – x10의 상관관계가 더 명확해진다.

4. Create Matrix A, Correlation Matrix

문제 정의: 생성한 1000차원의 벡터 100개로 1000 x 100 행렬 A를 만든다.

해결: V1 – V100을 결합해 100x1000 행렬을 만든 뒤 전치(Transpose) 한다.



V1 – V100을 결합하면 100 x 1000 행렬이 만들어진다. 이 행렬을 Transpose 해서 1000 x 100 행렬 A를 만들었다. V 벡터는 각각의 요소 x와 대응하므로 규칙을 만족하는 행렬 A가 만들어진다.

Correlation Matrix, R

$A^T A$

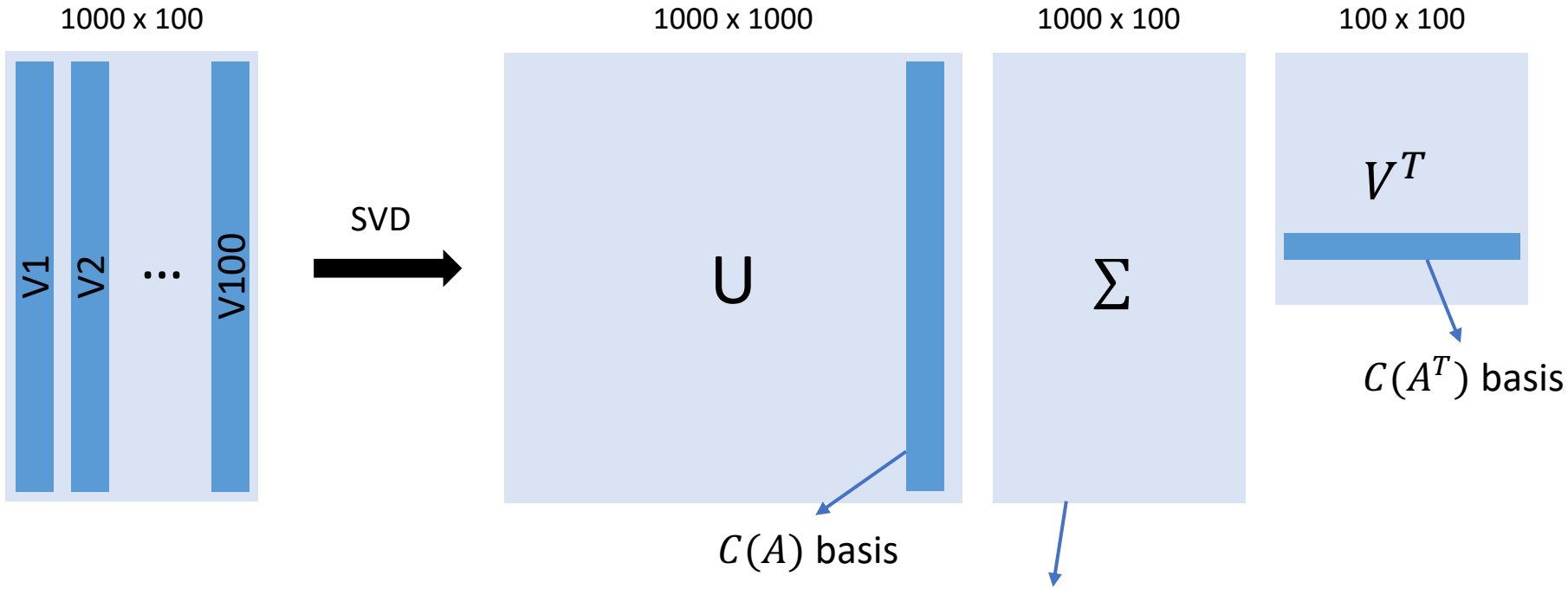
A^T(100 x 1000)와 A(1000 x 100)을 곱한 행렬 A^TA 는 100 x 100 행렬이 된다.

R (100 x 100)

$(A^T A)^T = A^T A$ 이므로 A^TA 는 항상 대칭 행렬이다. 대칭 행렬의 Eigen Vector 모두 서로 linearly independent 하다. 이 성질을 이용해 non-square matrix A 의 basis를 구할 것이다.

5. Extract Basis Vectors, SVD

100-D 벡터들의 basis vector 를 구하기 위해 행렬 A를 SVD 한다.



Non-Zero Singular Values: 7611.915, 5760.479, 4279.866, 4048.688, 2573.776, 449.214, … 80.182, 79.354, 77.642 (95개)

-> Rank = 95

SVD 결과로 나온 V^T의 1 - 95 번째 row vector를 basis vector로 사용한다. Singular value(혹은 eigen value)가 큰 eigen vector 부터 principal component 로 선택한다

6. Represent 100-D Vectors (1)

문제 정의: 100-D 벡터 X와 근사한 벡터를 생성한다.

해결: 선택한 Principal Components(eigen vectors)를 선형 결합하여 근사 벡터 $\hat{X}$ 을 만든다. Principal Component의 개수를 늘려 나가면서 결과를 비교해본다.

$\hat{X} = c_1 e_1 + c_2 e_2 + c_3 e_3 + \ \ldots + c_n e_n$; $\hat{X} \approx X$ (X: 같은 규칙으로 만들어진 100-D 벡터)

$c_k = X \cdot e_k$

두 벡터를 내적한 값은 하나의 벡터에 다른 벡터의 '성질'이 얼마나 포함되었는지 나타낸다. 따라서 X 벡터와 eigen vector를 내적한 값 c는 X 벡터에 eigen vector가 어느정도 포함되었는지 나타낸다. Eigen vector의 성질이 적다면 작은 c값을, 크다면 큰 c값을 가지게 된다.

계산한 c값을 eigen vector에 곱해 더하면 Principal Components의 선형 결합을 통해 근사 벡터 $\hat{X}$을 구할 수 있다.

7. Represent 100-D Vectors (2)

선택한 Principal Component 수를 10, 15, 20, 25, … , 95 (rank A)로 늘리며 $\hat{X}$을 구한다. X 하나를 생성하고 Principal Component 수를 바꿔가면서 근사한 벡터 $\hat{X}$를 구한 결과는 다음과 같다:

생성한 100-D 벡터 X

[34.0, 68.0, 62.0, -186.0, 84.0, 84.0, -53.0, -212.0, 88.0, -176.0, -11.0, -2.0, 5.0, … ,-4.0, 5.0, 0.0, 5.0, 3.0]

$\hat{X}$ - number of P.C. = 10

[33.8, 67.7, 62.0, -185.9, 83.8, 83.8, -53.0, -212.0, 87.8, -175.7, 0.4, -3.3, 4.1, … , 0.5, 0.0, 0.5, 0.0, 0.9]

$\hat{X}$ - number of P.C. = 50

[34.0, 67.9, 62.0, -186.0, 84.0, 84.0, -53.0, -212.0, 88.0, -176.0, -11.0, -1.6, 4.9, … , 0.3, -0.1, 1.0, -0.4, 1.5]

$\hat{X}$ - number of P.C. = 95

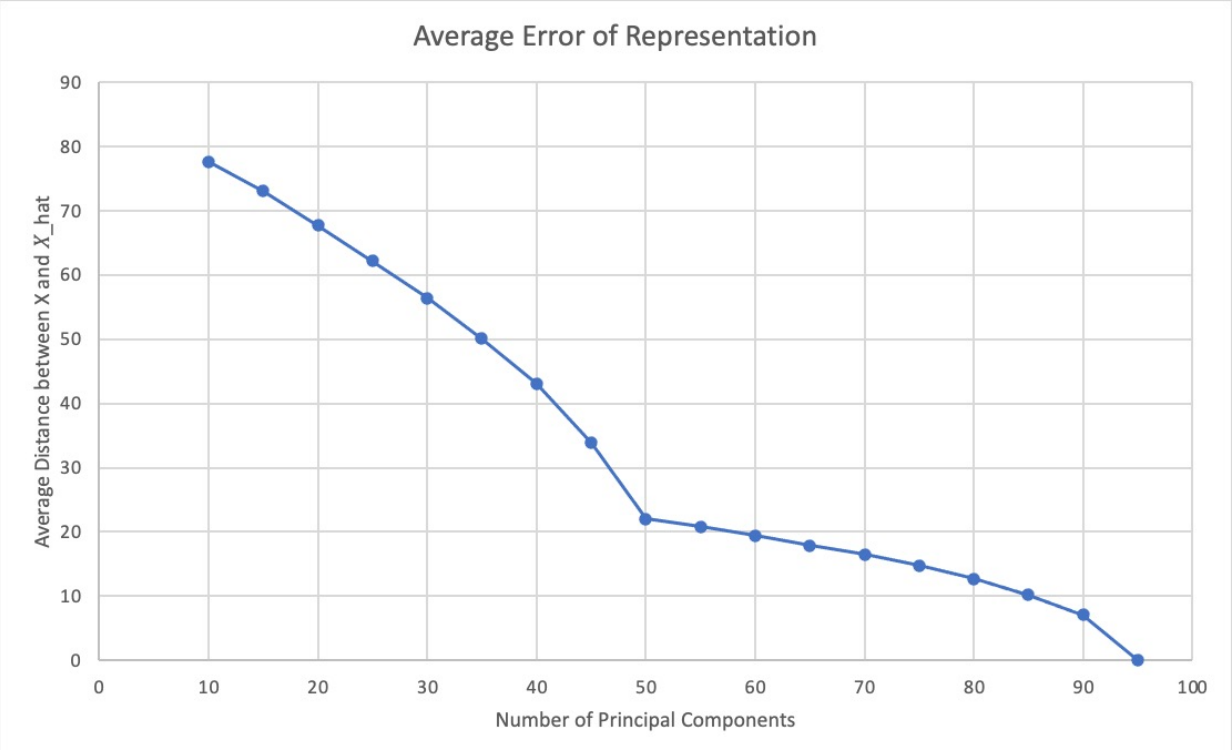
[34.0, 68.0, 62.0, -186.0, 84.0, 84.0, -53.0, -212.0, 88.0, -176.0, -11.0, -2.0, 5.0, … , -4.0, 5.0, 0.0, 5.0, 3.0]

모든 수는 소수점 둘째자리에서 반올림하여 표시

Principal Component 수가 늘어날수록 결과가 벡터 X와 더 비슷해지는 것을 확인할 수 있다. 다음 슬라이드에서는 Vector Distance를 이용해 오차를 정량적으로 평가할 것이다.

8. Check the errors in representation

새로운 100-D 벡터를 100개 생성한다 (동일한 규칙 적용). 생성된 100-D 벡터와 Principle Component를 이용해 만든 벡터의 사이의 평균 Vector Distance($avg \|X - \hat{X}\|$)를 통해 오차를 계산하고 그래프로 표현한다.



Principal Components : 10, 15, 20, 25, 30, 35, … , 80, 85, 90, 95

결과 분석

선택하는 Principal Component의 개수가 증가할수록 Average Distance는 감소하였다. N = 95 에서는 그 값이 0에 수렴했다.

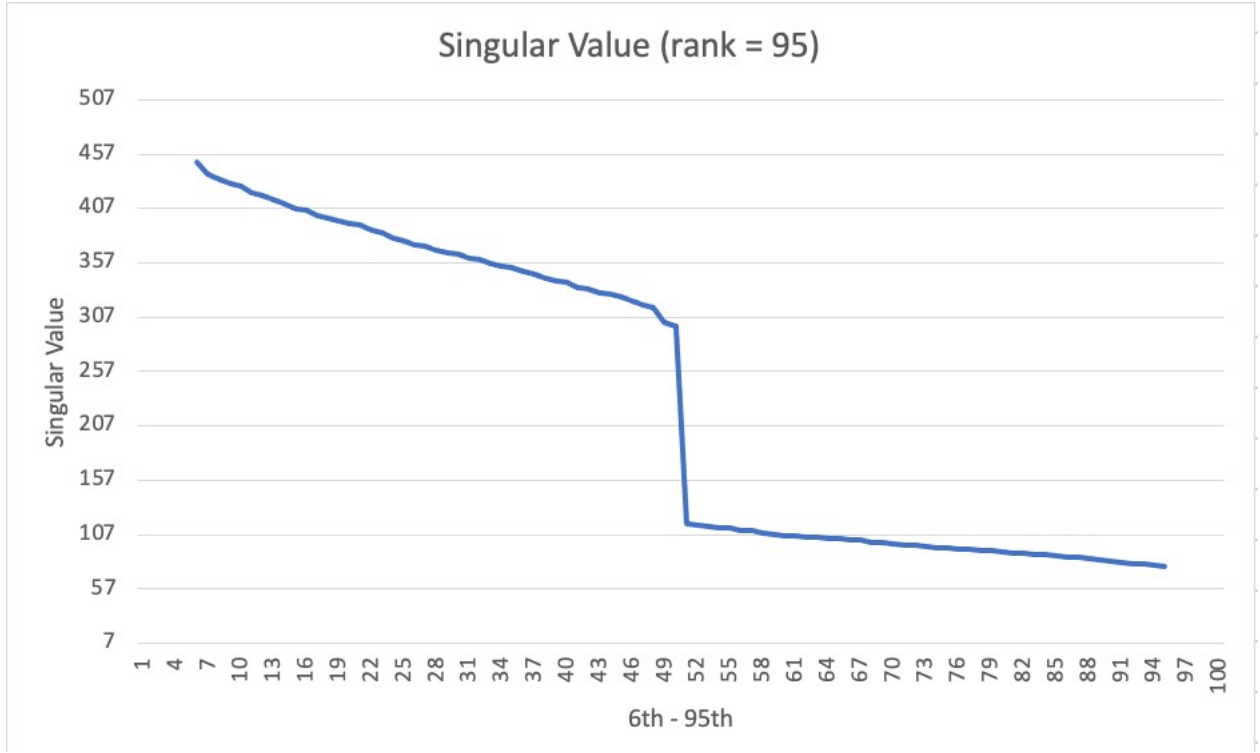
10 – 50 에서는 대체로 가파르게 감소하다가 50 – 90 에서는 완만하게 줄어드는데, 그 이유는:

1. eigen vector가 eigen value 크기로 정렬되었기 때문이다. 처음에는 singular value가 큰 Principal Component가 뽑혀 정확도가 높아질 확률이 크지만 점점 singular value가 작은 Principal Component가 추가되어 개선폭이 작아진다.
2. Principal Component의 개수가 많아질수록 점점 실제값에 근사하기 때문에 Distance가 감소하는 폭도 줄어든다.

※ Number of P.C. = 50 지점에서 기울기가 팔꿈치처럼 꺾어지는 것을 발견했다. Why? => Discussion 1

Discussion 1. Why the graph shows the ‘elbow point’

그래프에서 기울기가 변하는 지점(Number of Principal Component = 50)을 발견했다. 원인을 알아보기 위해 Singular Value(Square root of Eigen Value) 그래프를 그려보았다.



SVD 결과로 얻은 Singular Values

Singular value 값을 확인해보니 N=50인 지점에서 값이 절반 이하로 급격히 감소했다. Average Distance 그래프에서 기울기의 절댓값이 N=50 지점에서 갑자기 낮아진 것의 이유를 Singular Value에서 찾을 수 있었다.

Singular Value가 큰 N=20 Principal Component를 선형 결합하면 N=80 Principal Component보다 큰 가중치를 가질 확률이 줄어들고 그만큼 실제값과의 격차는 더 좁아진다. 따라서 Singular Value의 값이 N=50에서 급격히 감소하면서 Average Distance의 감소 기울기가 갑자기 완만해진 것으로 추측된다.

(N=50 Singular Value = 298.5182)
(N=51 Singular Value = 116.9103)

Discussion 2. SVD and PCA

프로젝트를 통해 SVD(Singular Value Decomposition)와 PCA(Principal Component Analysis)의 의미에 대해 고민해볼 수 있었다. 프로젝트의 데이터는 100-D 벡터 1000개로, non-square matrix의 형태를 가진다. 행렬의 Basis를 구하는 방법은 Gram-Schmidt 방법을 사용하거나 선형 독립인 Eigen Vector를 이용하는 방법이 있다. 하지만 Eigen Vector는 square matrix에 대해서만 정의된다. 따라서 non-square matrix A를 A^TA 꼴의 symmetric 한 square matrix로 변환한 뒤, basis vector가 정렬된 형태로 분해하는 것이 SVD의 본질이다.

프로젝트에서는 SVD의 결과로 생성된 V행렬에서 행렬 A의 row vector에 대한 basis를 얻었다. 그 후, 얻은 basis들을 Singular Value 크기 순으로 정렬해서 Principal Component를 선택했다. Principal Component를 통해 데이터를 재구성할 수도 있고 데이터의 특징을 얻을 수도 있다.

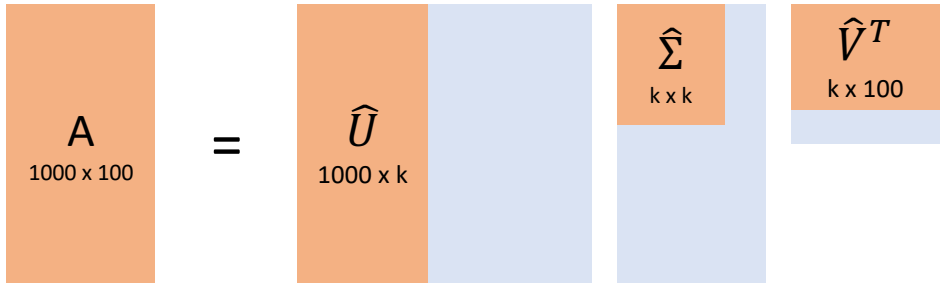
SVD 결과로 얻은 Principal Component 예시

[0.004, 0.007, 0.034, -0.103, 0.003, 0.003, -0.241, -0.964, 0.007, -0.013, 0 , 0.001, -0.001, 0 , 0.002, 0 , 0 , -0.002, 0.001, -0.001, -0.001, 0 , 0.002, 0.001, 0.001, 0 , 0.001, 0 , -0.001, -0.001, 0.001, -0.001, 0. , -0.001, -0. , 0.003, -0.002, 0.002, 0.001, 0. , -0.002, -0.001, -0.001, 0.001, 0.001, -0.002, -0.002, 0.003, 0.001, -0.001, 0.002, -0.001, -0.001, -0.004, 0 , 0 , 0.001, -0.001, 0 , -0.001, 0.001, 0 , 0 , 0 , 0 , -0.001, 0 , -0.001, 0 , 0 , -0.001, 0 , 0 , 0 , 0 , 0 , 0 , 0.001, 0 , 0.001, 0.001, 0 , 0 , 0 , 0 , -0.001, 0 , 0 , 0 , 0 , -0.001, 0 , 0 , 0 , -0.001, 0 , 0 , 0 , 0 , 0]

Principal Component를 자세히 살펴보며 벡터를 생성할 때 사용한 특징이 일부 들어간 것을 알 수 있었다. 예를 들어 x4 = x3 x (-3)의 관계를 0.034, -0.103에서 찾을 수 있다. 또 흥미로운 특징은 값을 크게 설정한 x1-x10(± 100)의 값이 x11-x55 (± 20), x56-x100 (± 5) 보다 대체로 크다는 것이다. 상관관계를 확인하고 싶은 x1-x10의 값을 크게 만들어서 Principal Component에서 성분을 크게 만든 것을 확인할 수 있다.

Discussion 3. Data Compression

마지막으로 SVD의 결과를 데이터 압축의 관점에서 살펴보려고 한다. 앞에서 우리는 eigen vector 중 일부만을 선택해서 실제값과 근접한 데이터를 만들었다. 이 원리를 이용하면 데이터를 압축할 수 있다.



앞에서 25개의 Principal Component를 사용해 데이터를 복원할 수 있었다. 그렇다면 Principal Component로 이루어진 25 x 100 행렬을 기억하면 100-D 벡터 1000개의 특징을 저장할 수 있다.

다음으로 95개의 eigen vector 중, 몇 개를 선택하면 적은 용량에 높은 정확도를 얻을 수 있을지 생각해보았다. Representation의 Vector Distance 그래프를 분석하는 과정에서 N=50 지점에서 기울기가 급격히 변하는 현상을 Discussion 1에서 다뤘다. 그 내용을 응용하여 순서대로 50개의 Principal Component를 선택하면 데이터의 특징을 효율적으로 압축, 복원할 수 있다고 생각한다. 다른 방법으로 Singular Value의 합을 누적한 뒤 “전체 Singular Value의 합에서 80~90% 비율이 되는 N을 결정한다” 와 같은 규칙을 사용할 수 있다고 생각했다.

Conclusion

100 차원의 데이터 1000개를 규칙에 맞춰 생성하고 SVD를 해서 Principal Component를 얻었다. 그리고 테스트 데이터를 만들어서 Principal Component로 만든 Representation과의 오차를 구했다. 보고서에 따로 설명하지는 않았지만 모든 과정을 코드로 구현해서 직접 실행하고 결과를 보았다. 그 과정에서 SVD와 PCA, 한 학기 동안 배운 선형대수 내용에 대해 많은 고민을 하고 더 깊이 이해할 수 있었다. 또 여러 의문점들이 생겼는데 그것들을 스스로 해결해보면서 생각한 내용을 Discussion 1부터 Discussion 3에 정리했다.

평소 데이터 분석, Deep Learning에 관심이 많았는데 여러 데이터 분석 Method들의 수학적 원리를 이해하는게 중요하다고 자주 생각했다. 이번 프로젝트에서 SVD, PCA라는 데이터 분석 방법에 대해 그 원리를 꼼꼼히 살펴보면서 흥미로웠다. 실제 이미지 데이터를 가지고 SVD를 해보지 못했지만 가까운 시일 내로 직접 해보고 싶다. 또 일반적인 숫자 형 데이터에는 어떻게 SVD를 사용할지 관심이 컸다. 그것이 계기가 되어서 이번 프로젝트에 관심을 가지고 진행한 것 같다.

프로젝트에서 데이터를 직접 생성하는 방식은 흥미로웠다. 비록 일차식의 간단한 관계였지만 결과에 맞게 조건을 변화하면서 데이터를 생성하는 과정이 재미있었다. 최근 유행하는 딥러닝 분야인 GAN이 연상되면서(원리는 전혀 다르지만) 미래에는 데이터를 확장하고 생성하는 영역이 더 중요해지지 않을까 생각했다.