추알못의 슬기로운 추천생활 #3
prod. 추천 알고리즘 못 들어보신 분

글. 김희동 kt스카이라이프 AI/DX 팀

이전 『추알못의 슬기로운 추천생활』#1에서 추천이란 어떤 조건에 꼭 들어맞는 사람이나 물건을 책임지고 권하는 것을 뜻하기에 단순 소개와는 다르다 하였습니다. 그래서 머신러닝의 프레임웍에서 문제 정의를 가장 먼저 확실히 해야 하는 것과 같이 추천 또한 대상(User)이 누군지에 대한 명확한 정보(Explicit Data)가 있어야 하고 또 무엇(Item)이 알맞은 것인 지에 대한 근거(Algorithm)의 타당성 또한 따져보아야 한다고 언급한 바 있습니다. 그리고 추천시스템의 1세대 추천방식인 설문조사, 베스트셀러, 스테디셀러 등과 2세대 방식 중 연관관계분석(Association Rule Mining)을 이해하기 위해 지지도(Support), 신뢰도(Confidence), 향상도(Lift) 값의 의미와 Apriori, FPG 등의 알고리즘 대해 알아봤습니다.

『추알못의 슬기로운 추천생활』#2에서는 같은 2세대 방식인 정보필터링(Information Filtering)에는 콘텐츠 기반 필터링(Content-based filtering)과 협업 필터링(Collaborative filtering)이 있고 협업 필터링에는 메모리 기반(Memory-based)과 모델 기반(Model-based)이 있다고, 설명해 드렸습니다. 여기서 메모리 기반은 다시 사용자기반(User-based)과 아이템 기반(Item-based)으로 나뉘고 모델기반(Model-based)은 요즘 가장 연구가 활발히 진행되고 있는 3세대 추천방식에 해당합니다. 다양한 딥러닝을 기반으로 개발된 3세대 기법에는 AE, MLP, CNN, RNN, DSSM, RBM, NADE, GAN 등이 있고 이 가운데 비교적 가장 많은 논문이 나오고 있는 오토인코더 AE에 대해 설명해 드렸습니다. 아래 [Figure 1]은 지금까지 다뤘던 위와 같은 추천시스템을 잘 분류하고 있기에 상기 차원에서 마지막으로 한번 더

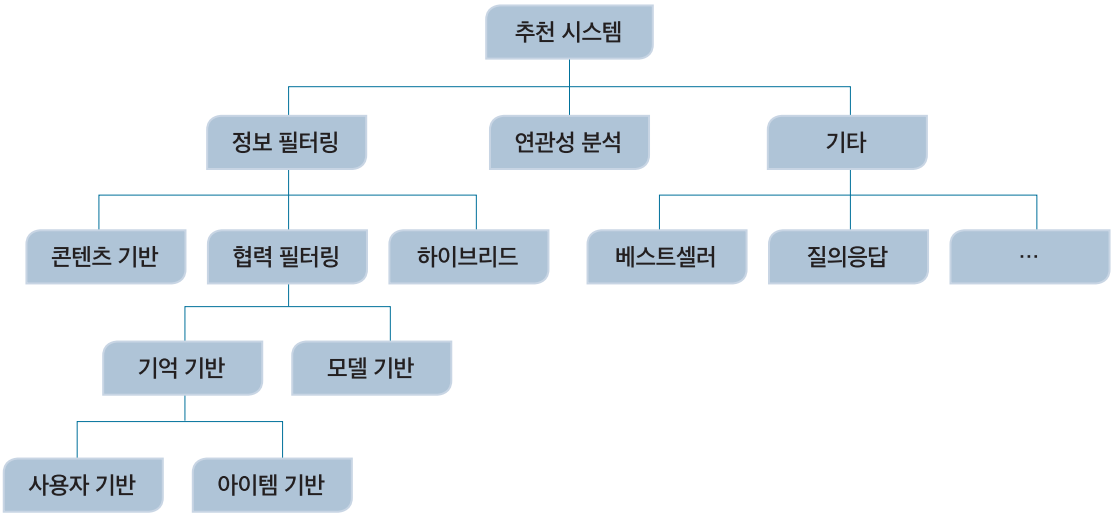


Figure 1. The Categorization of Recommender Systems, Son et al.,2015

관련 사례연구에서 발췌하여 보았습니다.

그밖에 추천시스템의 여러 종류 외에도 이와 함께 사용되는 Euclidean Distance, Cosine Similarity 등과 같은 유사도 알고리즘 그리고 협업필터링에서 추천시스템의 핵심이라 할 수 있는 행렬분해(Matrix Factorization)를 설명드리면서 주성분분석 PCA까지 살펴보았습니다. 그럼 이제 본격적으로 이런 PCA를 비롯한 행렬분해가 추천시스템에 어떤 방식으로 적용되는지 확인해 봐야겠습니다. 앞서 설명해 드렸듯이 협업필터링의 경우 User-Item Matrix의 Density가 매우 낮아서, Cosine Similarity나 Pearson's correlation coefficient를 이용할 경우, 서로 비슷한 User가 없어 유사도 값을 도출할 수 없는 데이터 희소성 문제(Sparseness Problem)가 발생하게 됩니다. 이를 해결하기 위하여 Melville et al.(2002)는 아이템을 선택한 사용자 정보를 통해, 아이템 간의 유사도를 계산하고 이를 이용하여 사용자-아이템 행렬에 각 사용자가 선택한 아이템과 유사한 다른 아이템을 가상으로 선택한 것으로 여기는 유사 사용자 벡터를 정의한 뒤, 밀도가 높아진 User-Item Matrix를 이용하여 이웃 기반(K-Nearest Neighbor, KNN) 협업필터링을 수행하였습니다. Goldberg et al.(2001)은 이런 데이터 희소성 문제를 해결하기 위하여 PCA를 수행한 뒤, 이를 바탕으로 Clustering을 수행하여 유사한 이웃을 찾았습니다. 그럼 이제 데이터 희소성 문제점을 근본적으로 해결하기 위해 데이터의 차원을 축소하는 방법으로 특이값 분해(Singular Value Decomposition, SVD)와 비음수행렬분해(Non-Negative Matrix Factorization, NMF)만 남았습니다. NMF는 PCA와 SVD와의 차이점을 비교하면서 마지막에 설명드리겠습니다.

먼저 SVD는 중요하지 않은 사용자나 아이템을 User-Item Matrix에서 직접 제거하여 행렬의 차원을 축소하는 방법입니다. 우선 특이값 분해의 기하학적 의미를 이해하기 위해 행렬 A의 선형변환을 도식화해보겠습니다. 이전 원고에서 PCA를 설명하면서 여러 의미로 나눠봤었는데 그중에서 SVD 또한 선형변환하는 차원축소 방법론과도 같기에 짚고 넘어가겠습니다.

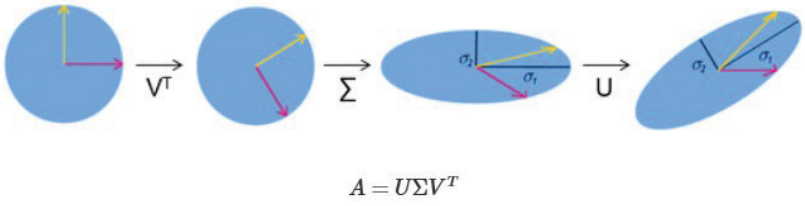


Figure 2. The Understanding of Matrices' Linear-transform, WIKI

[Figure 2]는 행렬 A가 선형 변환되고 있는 과정을 보여 주고 있는데 이는 직교행렬(VT)에 의해 회전되고, 대각행렬(Σ)에 의해 스케일 변환된 후, 다시 직교행렬(U)에 의해 회전되는 과정의 결과와 동일합니다. 다시 말해 선형변환하는 차원축소 방법론이란 모든 행렬은 선형변환(Linear Transform) 가능한데 여기서 직교행렬(Orthogonal)의 선형변환이란 Rotate 변환을 뜻하고 대각행렬(Diagonal, Σ)의 선형변환은 Stretch 변환을 뜻합니다.

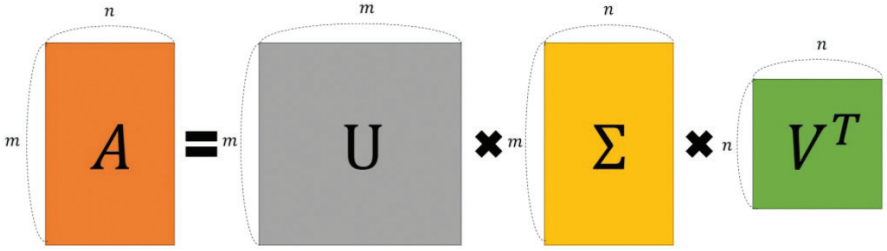


Figure 3. The Decomposition of Matrix A by SVD, Github

SVD의 특이값 분해는 m×m 형태의 정방행렬(Square matrix)만 가능한 PCA의 고유값 분해와 달리 m×n과 같은 모든 직각(Orthogonal) 행렬에 대해 분해가 가능합니다. [Figure 3]에서 보듯 m×n 크기의 행렬 A는 m×m 크기의 행렬 U와 m×n 크기의 Σ, 그리고 n×n 크기의 VT의 형태로 분해될 수 있습니다. 지금부터 특이벡터 U나 V 뒤에 따라붙는 T는 전치행렬(Transpose)을 뜻합니다. 따라서 VT는 V의 전치행렬을 말합니다. 그러면 A = U Σ VT가 될 텐데 여기서 Σ 양옆에 위치한 U와 VT를 각각 U에 속한 Left Singular Vector(좌측 특이 벡터), V에 속한 벡터를 Right Singular Vector(우측 특이 벡터)라고 합니다. 중앙에 위치한 이 Σ가 특이값(Singular Value)을 갖는 대각(Diagonal) 행렬이어서 대각선으로만 0이 아닌 값을 가지며 나머지는 0이 되는 특징을 가지고 있습니다. 또한 양 특이벡터 U와 VT는 서로 직교하는 새로운 축에 투영되는 PCA의 원리와 같이 직교하는 성질을 갖고 있습니다. 각각을 좀 더 자세히 들여다보면 U는 AAT를 고유값 분해해서 얻은 직교 행렬이며, V는 ATA를 고유값 분해해서 얻은 직교 행렬입니다. AAT = U (Σ Σt) Ut로 분해할 수 있는데 증명은 생략도록 하겠습니다. 여기서 (Σ Σt)의 대각 원소 값은 AAT의 고유값(eigen value)이 되며 Σ의 대각 원소값이 바로 행렬 A의 특이값(Singular Value)입니다. Σ는 AAT 혹은 ATA의 고유값의 루트(Square Root) 값과 같은데 AAT와 ATA의 고유값이 같다는 증명 또한 생략도록 하겠습니다. 아마도 처음 접하신 분들은 이해가 잘 안 가실 겁니다. 행렬 분해가 다소 난해한 벡터와 행렬과 같은 선형대수학을 바탕으로 하고 있기 때문입니다. 좀 더 이해가 쉽게 가기 위해 이전의 유사도 알고리즘을 활용한 멘토 찾기와 비슷한 사례를 SVD에도 적용해 보겠습니다.

SVD Example(a.k.a. popularity poll)

Can the value of D1's 4th subject be predicted from the other value of student-faculty matrix? What about Q's 3rd and 4th subject values? So in this case, is the full SVD effective?

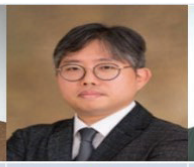
The MOT Faculty → Graduate Student ↓				
D1(9th AI Track)	3	2	1	0
D2(9th IE Track)	1	2	3	0
D3(9th ETM Track)	2	2	2	2
Q(10.5th AI Track)	1	5	0	0

Figure 4. The SVD Example used in Prediction, Author

예 1) 이번에 A는 학생과 과목의 선호도를 나타내는 매트릭스라고 가정합니다. D1은 AI 트랙을 선택하여 인공지능 과목을 담당하는 교수에게는 3점을, 차례대로 선형회귀방정식(Linear Regression)과 STATA 툴을 활용하여 다양한 통계적 기법을 배울 수 있는 연구방법론에는 2점, 끝으로 다기준 의사결정 MCDM 방법론인 AHP/ANP, DEA, Delphi 등을 다루는 기술예측 및 기획과목에는 1점을 주었고 타 트랙 교수의 과목은 아직 듣지 않아 점수가 공란으로 되어있습니다. D2, D3, Q도 각자 트랙 필수 및 선택과목에 따라 점수를 위와 같이 주었다고 했을 때, 아직 점수를 주지 못한 과목의 점수를 어떻게 예측 할 수 있을지에 대하여 이 User-Item 매트릭스를 행렬 분해를 해 보겠습니다.

풀이 1) 우선 A는 4×4 정방행렬입니다. 위 설명과 같이 행렬 A = U Σ VT와 같은 형태로 분해될 수 있습니다. 정의에 의해 U는 AAT를 고유값 분해해서 얻은 직교 행렬입니다. A가 4×4 행렬이고 AT 역시 정방행렬이기에 AAT가 4×4 행렬

이 되긴 되나 고유값을 구하기 위한 행렬식은 4차 방정식이 되기에 수기로 풀기에는 힘이 듭니다. 아래 코드 결과에 'Left Singular Vector'가 바로 U가 됩니다. 마찬가지 방법으로 VT는 ATA를 고유값 분해해서 얻은 직교 행렬로 'Right Singular Vector'가 됩니다. 끝으로 Σ는 AAT 혹은 ATA의 고유값을 분해하여 루트(Square Root)를 취한 특이값으로 대각으로만 값을 갖고 나머지는 0이 되는 대각행렬입니다. 잘 보시면 아시겠지만 PCA의 Σ처럼 Scalar 값이 큰 순서대로 나열되고 있습니다. 이 부분을 눈여겨보셨다면 차원축소에 특이값이 어떻게 이용되는지 알 수 있습니다.

```
import numpy as np
from numpy.linalg import svd

A = np.array([[3,2,1,0]
              [1,2,3,0]
              [2,2,2,2]
              [1,5,0,0]])

U, S, VT = svd(A)
print("Left Singular Vectors:")
print(U)
print("Singular Values:")
print(np.diag(S))
print("Right Singular Vectors:")
print(VT)
print("The Restored matrix:")
print(U @ np.diag(S) @ VT)
```

```
Left Singular Vectors:
[[-0.45887801  0.16325091  0.68579824 -0.54079653]
 [-0.43426205  0.4025392  -0.70050605 -0.39833396]
 [-0.48282551  0.45737034  0.14564233  0.73244809]
 [-0.60640496 -0.77596631 -0.133268    0.1113057 ]]
Singular Values:
[[7.29857799 0.         0.         0.        ]
 [0.         3.23815052 0.         0.        ]
 [0.         0.         1.95134218 0.        ]
 [0.         0.         0.         1.56121882]]
Right Singular Vectors:
[[-0.46350838 -0.7924771  -0.37367761 -0.13230673]
 [ 0.3184121  -0.56622156  0.70583784  0.28248862]
 [ 0.77634018 -0.20727833 -0.57623684  0.149274  ]
 [-0.28472732  0.09169998 -0.1735197   0.93830292]]
The Restored matrix:
[[ 3.00000000e+00  2.00000000e+00  1.00000000e+00 -3.33066907e-16]
 [ 1.00000000e+00  2.00000000e+00  3.00000000e+00  1.94289029e-16]
 [ 2.00000000e+00  2.00000000e+00  2.00000000e+00  2.00000000e+00]
 [ 1.00000000e+00  5.00000000e+00 -1.33226763e-15 -6.66133815e-16]]
```

Code 1. The calculation of Singular-Value Decomposition

흥미로운 점은 User-Item의 분해된 행렬을 곱해서 원래의 행렬로 복원하였더니 점수가 0으로 표기되었던 부분이 어떤 숫자로 채워져 있는 것을 확인하실 수 있습니다. 그러나 실제로 협업필터링에서 이런 방식으로 점수를 예측하지는 않습니다. 위 사례는 차원축소가 없는 Full SVD(4×4 행렬을 4×4 행렬로 분해) 방식이기 때문입니다. 단순히 수학적으로 복원하면 원래의 값이 나온다는 것을 확인하는 정도일 뿐이지 별다른 의미를 주지는 못합니다.

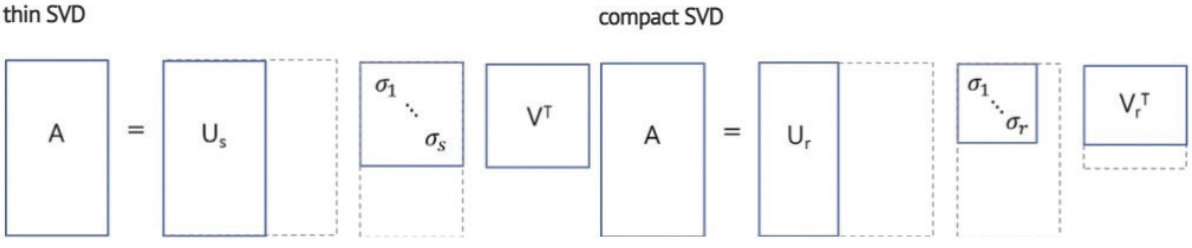


Figure 5. The Reduced SVD, WIKI

그래서 Reduced SVD 방식이 등장하였습니다. Reduced SVD에는 Thin, Compact, Truncated 방식이 있습니다. 위 [Figure 5]에 보는 바와 같이 Thin SVD의 시그마 Σs는 Diagonal에만 특이값을 갖는 대각 행렬이므로 점선 부분을 제거하고 이에 해당하는 왼쪽특이벡터 U의 점선 부분을 뺀 Us로만 원래의 행렬 A를 UsΣsVT 형태로 나타낸 경우입니다. 이와 달리 Compact SVD의 시그마 Σr는 Σs와 같이 Diagonal에만 특이값을 갖는 대각 행렬이지만 차이점이 있다면 Non-zero 특이값만 계산되며 이에 상응하는 양쪽 특이벡터 Ur과 VTr 모두 제거하여 UrΣrVTr로 행렬 A를 복원합니다. 그러나 이 둘 모두는 협업필터링 방식의 차원축소에 활용되지는 않습니다. 이보다 더 효율적인 방식은 남은

Truncated SVD입니다. 그럼 이제 SVD의 상징인 차원축소가 어떻게 이뤄지는지 Truncated SVD에 대해 알아봐야겠습니다.

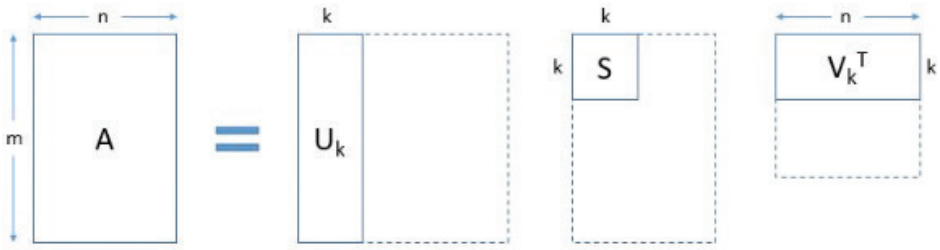


Figure 6. The Principle of Truncated SVD, WIKI

[Figure 6]은 Truncated SVD를 보여주고 있습니다. 원래의 행렬 A는 m×n일 때, Full SVD의 Left Singular Vector인 U의 형태는 m×m이 되어야 하나 m×k의 Uk(where m>k)가 되었고 Right Singular Vector인 VT의 형태는 n×n이 되어야 하나 k×n의 VkT(where n>k)로 줄어들었습니다. 이는 기준이 되는 Σ(S)의 Singular Value 중에서 가장 큰 k개까지만 선택하였기 때문입니다. 이렇게 행렬 분해된 값을 계산하여 복원하게 되면 원래의 A와는 달라지나 A를 가장 잘 설명할 수 있는(scalar 값이 가장 큰) Singular Value를 취해준다면 k만큼의 차원축소 효과를 볼 수 있기 때문입니다. 그럼 k값에 따라 복원되었을 때 얼마나 값이 차이가 있는지 알아보겠습니다. Original 행렬과 Truncated SVD로 분해 후 복원된 값을 한 줄에 비교하기 위해 소수점 8자리로 6열씩 8행을 갖는 임의의 행렬을 불러왔습니다. 아래 [Code 2]에서 보듯 6개의 전체 Singular Value 중에서 1이 넘는 두 개의 k값(3.75와 1.10)을 취하게 되면 6차원에서 2차원으로 줄어들게 됩니다. Truncated SVD로 분해 후 복원된 값을 계산해 보니 대략 67% 정도로 원본을 설명해 주고 있습니다.

```
import numpy as np
from scipy.sparse.linalg import svds
from scipy.linalg import svd
matrix = np.random.random((8, 6))
print('원본 행렬:₩n',matrix)
U, Sigma, Vt = svd(matrix, full_matrices=False)
print('분해 행렬 차원:',U.shape, Sigma.shape, Vt.shape)
print('Sigma값 행렬:', Sigma)
num_components = 2
U_tr, Sigma_tr, Vt_tr = svds(matrix, k=num_components)
print('Truncated SVD 분해 행렬 차원:',U_tr.shape, Sigma_tr.shape, Vt_tr.shape)
print('Truncated SVD Sigma값 행렬:', Sigma_tr)
matrix_tr = np.dot(np.dot(U_tr,np.diag(Sigma_tr)), Vt_tr)
print('Truncated SVD로 분해 후 복원 행렬:₩n', matrix_tr)
```

```
원본 행렬:                                                    Truncated SVD로 분해 후 복원 행렬:
 [[0.30984824 0.89023326 0.29565298 0.76515676 0.63236164 0.44980213] [[0.58854113 0.49622554 0.57453862 0.48912464 0.65837767 0.42422521
 [0.88479226 0.55847796 0.56518144 0.87738044 0.62073661 0.14039268] [0.67035477 0.56638356 0.65437143 0.55527532 0.7476414  0.48228066]
 [0.53217433 0.442152   0.5783206  0.16417528 0.20708072 0.55622985] [0.51616451 0.56268266 0.50016751 0.22940393 0.332908   0.27281247]
 [0.61563445 0.13438533 0.80418877 0.42164405 0.82728703 0.59864263] [0.56399185 0.37198481 0.55359198 0.63081603 0.82950594 0.48713666]
 [0.45423941 0.00412503 0.43902454 0.9408348  0.9026608  0.71410918] [0.42997916 0.05285083 0.42877736 0.84215306 1.07496488 0.55101911]
 [0.37640082 0.57617413 0.2727912  0.04545021 0.89890603 0.55599695] [0.49572196 0.40801568 0.48421786 0.42756084 0.57362801 0.36506628]
 [0.77421672 0.84580574 0.78701761 0.17237908 0.16290954 0.36079129] [0.74181665 0.9264238  0.71539339 0.14535254 0.25259986 0.30040864]
 [0.64464144 0.45024079 0.76002894 0.41671881 0.99805494 0.15683085] [0.63268101 0.50236392 0.61853434 0.57446048 0.76736171 0.48023553]
분해 행렬 차원: (8, 6) (6,) (6, 6)
Sigma값 행렬: [3.75742891 1.10664508 0.81060195 0.7541745  0.61351551 0.22664595]
Truncated SVD 분해 행렬 차원: (8, 2) (2,) (2, 6)
Truncated SVD Sigma값 행렬: [1.10664508 3.75742891]
```

Code 2. The Truncated SVD, where k = 2 out of 6 Singular Value Components

[Code 3]은 6개의 전체 Singular Value 중에서 k=4일 때의 경우입니다. 2차원밖에 줄이지 못하였으나 91%로 정확도가 상승하였습니다. 한눈에 봐도 k=2일 때보다 확연히 오차가 줄어들었음을 확인할 수 있습니다. 모두에서 문제 정의를 확실히 파악해야만 관련 데이터를 수집하고 학습 및 검증을 하기 위한 전처리 과정을 거쳐 최적의 모델을 개발할

수 있다고 말씀드렸습니다. 여기서도 데이터 손실보다 차원축소가 우선시 되는 것이 문제 해결에 유리한지 아니면 그 반대인지에 따라 k값을 선택하게 됩니다. 이제 실제 활용사례 예시만 남았습니다.

```
import numpy as np
from scipy.sparse.linalg import svds
from scipy.linalg import svd
matrix = np.random.random((8, 6))
print('원본 행렬:₩n',matrix)
U, Sigma, Vt = svd(matrix, full_matrices=False)
print('분해 행렬 차원:',U.shape, Sigma.shape, Vt.shape)
print('Sigma값 행렬:', Sigma)
num_components = 4
U_tr, Sigma_tr, Vt_tr = svds(matrix, k=num_components)
print('Truncated SVD 분해 행렬 차원:',U_tr.shape, Sigma_tr.shape, Vt_tr.shape)
print('Truncated SVD Sigma값 행렬:', Sigma_tr)
matrix_tr = np.dot(np.dot(U_tr,np.diag(Sigma_tr)), Vt_tr)
print('Truncated SVD로 분해 후 복원 행렬:₩n', matrix_tr)
```

```
원본 행렬:
 [[0.8279368  0.01892728 0.31586612 0.55114048 0.36524552 0.55931816]
 [0.64131457 0.46593663 0.85157558 0.63303099 0.3770385  0.28353791]
 [0.75956019 0.35165124 0.31298819 0.87920498 0.6645093  0.31683356]
 [0.85610716 0.3788443  0.0375528  0.26046502 0.08258812 0.40282959]
 [0.25530016 0.96430946 0.78468407 0.18799961 0.14339593 0.04176518]
 [0.71779359 0.38294627 0.06925932 0.5350464  0.17891963 0.26789512]
 [0.56827444 0.5349743  0.32938087 0.95606978 0.49941254 0.4324608 ]
 [0.1896294  0.87438686 0.23671037 0.71993535 0.92186006 0.91714062]]
분해 행렬 차원: (8, 6) (6,) (6, 6)
Sigma값 행렬: [3.43366923 1.11844936 0.97443976 0.62149652 0.43383481 0.18670469]
Truncated SVD 분해 행렬 차원: (8, 4) (4,) (4, 6)
Truncated SVD Sigma값 행렬: [0.62149652 0.97443976 1.11844936 3.43366923]
Truncated SVD로 분해 후 복원 행렬:
 [[0.82565851 0.10683524 0.19527951 0.67792158 0.35649482 0.38651716]
 [0.6442167  0.51129368 0.78851378 0.6911595  0.38317598 0.18904851]
 [0.72603283 0.29710426 0.39504326 0.86296668 0.57865836 0.4697368 ]
 [0.85012231 0.39233726 0.02021019 0.28999299 0.06652357 0.38367354]
 [0.24942851 0.95795734 0.79464671 0.18962364 0.12825892 0.06214309]
 [0.7298083  0.32675967 0.1441475  0.43514889 0.21209603 0.36453766]
 [0.60654982 0.47305453 0.40672964 0.80124914 0.60137544 0.50623553]
 [0.18376036 0.90963816 0.18938037 0.77962881 0.90540529 0.8543255 ]]
```

Code 3. The Truncated SVD, where k = 4 out of 6 Singular Value Components

넷플릭스, 아마존, 유튜브 등 내로라하는 굴지의 IT 기업들은 이미 자신만의 사업 특성에 맞는 추천시스템을 개발, 활용하면서 다양한 방식으로 알고리즘을 수정 보완해가고 있습니다. 스트리밍 시장의 경우 추천시스템의 성과에 따라 만족해하는 가입자가 늘고 또 다양한 니즈를 충족시키기 위해 Original 콘텐츠를 제작하거나 다양한 콘텐츠의 저작권을 사들여 제공하면 할수록 User-Item 매트릭스는 늘어나게 됩니다. 그럼 필연적으로 고차원의 데이터를 처리하여야 하는데 차원의 저주(Curse of Dimensionality)라 하여 고차원의 데이터는 처리해야 할 양도 양이지만 일정 차원이 넘으면 Classifier의 성능이 기하급수적으로 떨어지게 됩니다.

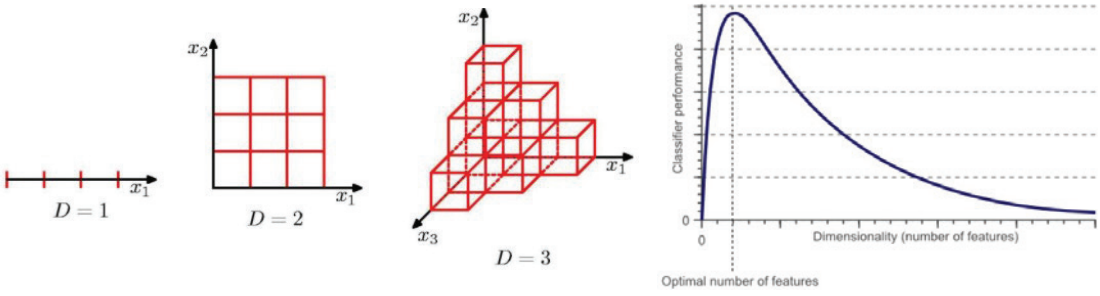


Figure 7. The Curse of Dimensionality, Christopher & Vincent

[Figure 7]은 차원 D가 늘어남에 따라 처리해야 할 데이터가 기하급수적으로 늘어나고 추가 Training Data 없이 상황이 지속되어 임계점(Optimal number of features)에 다다르면 Classifier 성능 또한 기하급수적으로 떨어져 결국에 0에 수렴하게 된다는 것을 보여주고 있습니다. 이런 차원의 저주 이외에도 User-Item 매트릭스가 늘어날수록 새로운 콘텐츠에 대한 선호도를 판단할 수 있는 사용자 점수는 없거나 부족할 수밖에 없고 따라서 전체 Dataset에서 차지하는 비중 또한 줄어들게 됩니다. 이를 협업 필터링의 가장 큰 단인 First Rater 문제라 하였고 이로 인한 데이터 희소성(Sparsity)의 해결 여부가 전체 추천시스템 성능을 좌우하게 됩니다. 머신러닝의 개발 과정을 경험해 보신 분은 아시겠지만 모델

을 개발하는데 있어 가장 까다로운 부분이 데이터 전처리(Pre-processing) 과정입니다. 전처리는 해당 모델이 분석하기 용이하게 데이터를 가공하는 것을 뜻하는데 데이터 희소성은 결측치(Missing Value)에 해당합니다. 이 부분을 뺄 것인지 채운다면 평균값인지 예측값인지 정해야 합니다. 평균이 수월할 수도 있지만 오차는 당연히 발생하고 예측은 별도알고리즘 필요하게 되어 시스템 복잡해지는 단점이 발생하게 되는데 이를 다루는 것이 피처엔지니어링 변수가공입니다. 여기서 변수가공까지 다루기엔 불가능하고 행렬 분해를 통해 결측치를 예측하는 방법에 대해 소개하겠습니다.

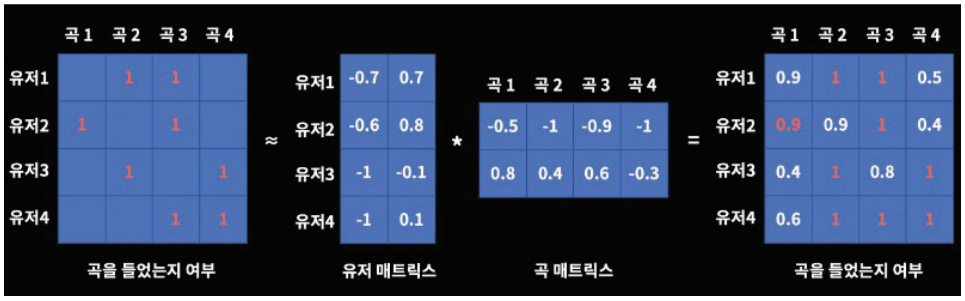


Figure 8. The Example of Matrix Factorization, kakao Tech

[Code 2~3]은 Truncated SVD 알고리즘에서 k값에 따른 차원축소와 이를 바탕으로 복원된 후의 효율에 대한 것으로 랜덤으로 데이터 샘플을 불러오다 보니 모든 행렬의 값이 채워진 경우였습니다. 즉, 결측치가 없는 경우로 실제 필드 사례와는 확연한 차이가 있습니다. 위 [Figure 8]은 멜론을 인수한 카카오가 유저-곡 매트릭스에서 아직 음원을 구매하지 않아 점수를 주지 못한 곡의 점수를 예측하기 위하여 행렬 분해 과정이 어떻게 이뤄지고 있는지를 보여주고 있습니다. 앞서 설명한 바대로 A = U Σ VT의 형태이나 빠른 이해를 돕기 위해 특이값 Σ를 생략하였습니다. 이와 같을 순 없지만 위 예시대로라면 U는 왼쪽 특이벡터로서 4×4에서 4×2로 Truncated 되었고 마찬가지로 VT 또한 2×4로 Truncated 된 오른쪽 특이벡터입니다. 따라서 4개의 특이값 중에서 원래의 유저-곡 매트릭스를 잘 설명해 줄 수 있는 가장 값이 큰 2개의 특이값 만을 선택하였다는 것을 알 수 있고 따라서 2×2의 특이값을 대각으로 갖는 Diagonal Σ 매트릭스가 됩니다. 하나만 예를 들면 유저1의 곡1에 관한 점수가 없었으나 행렬 분해를 통해 복원하니 행렬 곱셈의 법칙에 따라 성분별로 곱해주게 되면 [(-0.7×-0.5)+(0.7×0.8)]=0.91됩니다. 또 곡4에 대해서도 계산해 보면 [(-0.7×-1)+(0.7×-0.3)]= 0.49가 됩니다. 이 점수들이 곧 유저1의 예측된 선호도가 되고 이는 바로 협업필터링에서 추천의 가부를 판단하는 주요 정보가 되는 원리입니다. 만약 추천 Classification 모델로 기준을 0.5로 잡았다면 곡1은 기준보다 크기에 추천이 되고 곡4는 기준미달로 추천에서 제외가 됩니다. 또 곡4의 값이 0.5보다 크다고 하였을 경우에도 Classification이 아니라 Clustering으로 군집화가 가능하고 이때 소비(1)과 가장 가까운 유사도를 보이는 곡1이 곡4보다 먼저 추천되게 됩니다. 그렇다면 이제 행렬분해의 마지막 비음수행렬분해(Non-negative Matrix Factorization, NMF)에 대한 설명을 끝으로 길고 길었던 알고리즘 소개를 끝마칠까 합니다.

NMF는 세 개의 행렬로 분해되는 SVD(A = U Σ VT)와 달리 두 개의 행렬로 분해되는 형태로 어떤 행렬 V가 있을 때, V를 W*H=V인 두 행렬 W와 H로 분해하고 W와 H에 임의의 값을 채워가며 W*H가 V와 유사해질 때까지 증배갱신규칙(Multiplicative Update Rule)을 반복하게 됩니다. 또 차이점은 [Figure 8]과 달리 이름에서 알 수 있듯 음수를 포함하지 않은 행렬 V를 음수를 포함하지 않은 행렬 W와 H의 곱으로 분해하는 알고리즘입니다. 행렬이 음수를 포함하지 않는다는 성질은 분해 결과 행렬을 찾기 쉽게 만들어 주고 일반적으로 행렬 분해는 정확한 해가 없기 때문에 이 알고리즘은 대략적인 해를 구하게 됩니다. 이런 특징 등으로 NMF는 컴퓨터 시각 처리, 문서 분류, 음파 분석, 계량분석화학, 추천시스템 등에 쓰이고 있습니다.

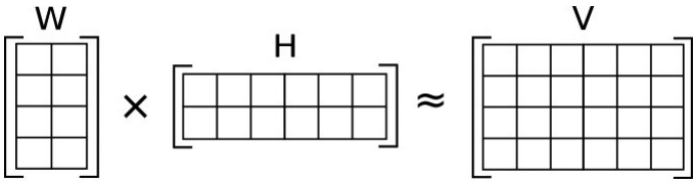


Figure 9. The Matrix Formation of NMF, WIKI

[Figure 9]는 NMF의 분해형태를 잘 보여주고 있는데 여기서 W의 각 열이 V의 원래 특성을 나타내 주고 있으며 위와 같이 열 길이가 2일 때, 특성(Feature) f1과 f2가 존재하게 됩니다. W의 각 행은 V의 같은 행이 '특성에 얼마나 적합'한지 즉, 특성에 대한 가중치를 뜻합니다. H의 경우는 각 행이 특성이 되고 W와 마찬가지로 H의 각 열은 V의 같은 열이 '특성에 얼마나 적합'한지를 나타내 주고 있습니다. PCA는 데이터의 분산을 가장 많이 설명해주는 주성분들의 크기순으로 선택하여 차원을 축소하였고 SVD도 이와 유사하게 유저-아이템 메트릭스를 잘 설명해 줄 수 있는 Σ의 특이값들 중에서 가장 큰 값을 고르는 방식으로 차원을 축소한다고 앞서 설명드린 바 있습니다. 하지만 NMF는 이와 달리 0 이상의 비음수 값을 갖는 특성이기만 하면 크기와 상관없이 데이터의 특성을 분해하여 차원을 축소할 수 있습니다. 이 역시 글로는 설명이 난해할 수 있기에 카카오의 유저-곡 매트릭스인 [Figure 8]의 NMF 알고리즘을 Python으로 구현해 보겠습니다. 아래 [Code 4]는 특성이 두 개(f1~2)일 때와 세 개(f1~3)일 때 W와 V의 형태 및 값과 다시 복원한 후, 얼마나 원래의 값과 비슷해지는지를 비교해 보았습니다. 왼쪽 코드는 특성이 2개(n=2)인 경우로 원래의 유저-곡 4×4 V 행렬이 유저-특성 4×2 W 행렬과 특성-곡 2×4 H 행렬로 분해되었고 오른쪽 코드는 특성이 3개(n=3)인 경우로 원래의 유저-곡 4×4 V 행렬이 유저-특성 4×3 W 행렬과 특성-곡 3×4 H 행렬로 나타내 보았습니다. 복구된 매트릭스를 서로 비교해 보시면 아시겠지만 원래의 데이터를 더 많은 특성으로 분해한 경우가 그렇지 않은 경우보다 더 정교하다는 것을 알 수 있습니다. 하지만 그만큼 차원축소의 효과가 약해지므로 Trade-off 되는 성질을 모델에 따라 알맞게 지정해줘야 합니다.

```python
import numpy as np
from sklearn.decomposition import NMF

V= np.array([[0,1,1,0],[1,0,1,0],[0,1,0,1],[0,0,1,1]])
model=NMF(n_components=2, init='random', random_state=0)
model.fit(V)
nmf_features=model.transform(V)
matrix_restored=np.dot(nmf_features,model.components_)

print('Sahpe:',nmf_features.shape,model.components_.shape)
print('W(Weight Matrix):')
print(nmf_features)
print('H(Feature Matrix):')
print(model.components_)
print('Restored Matrix:')
print(matrix_restored)
```

```
Sahpe: (4, 2) (2, 4)
W(Weight Matrix):
[[0.3722352  0.5348911 ]
 [0.         0.86412811]
 [0.71273072 0.        ]
 [0.37223523 0.53489109]]
H(Feature Matrix):
[[0.         1.38193617 0.24159478 1.38193629]
 [0.65518989 0.         1.39331669 0.        ]]
Restored Matrix:
[[0.35045524 0.51440528 0.83520277 0.51440533]
 [0.566168   0.         1.20400412 0.        ]
 [0.         0.98494836 0.17219202 0.98494845]
 [0.35045523 0.51440532 0.83520277 0.51440537]]
```

```python
import numpy as np
from sklearn.decomposition import NMF

V= np.array([[0,1,1,0],[1,0,1,0],[0,1,0,1],[0,0,1,1]])
model=NMF(n_components=3, init='random', random_state=0)
model.fit(V)
nmf_features=model.transform(V)
matrix_restored=np.dot(nmf_features,model.components_)

print('Sahpe:',nmf_features.shape,model.components_.shape)
print('W(Weight Matrix):')
print(nmf_features)
print('H(Feature Matrix):')
print(model.components_)
print('Restored Matrix:')
print(matrix_restored)
```

```
Sahpe: (4, 3) (3, 4)
W(Weight Matrix):
[[0.         0.75424947 1.37842827]
 [0.         1.2169568  0.        ]
 [0.56867294 0.         1.3337852 ]
 [0.59432685 0.75243151 0.01395543]]
H(Feature Matrix):
[[0.         0.00661503 0.15160011 1.70682908]
 [0.46518449 0.         0.98936102 0.        ]
 [0.         0.73579006 0.06442326 0.01060202]]
Restored Matrix:
[[0.35086515 1.01423382 0.83502787 0.01461413]
 [0.56610943 0.         1.20400962 0.        ]
 [0.         0.98514769 0.17213768 0.98476834]
 [0.35001946 0.01419976 0.83542547 1.0145623 ]]
```

Code 4. The NMF Performance Comparison between n = 2 and 3

지금까지 『추알못의 슬기로운 추천생활』 시즌 1과 2를 마무리하면서 추천시스템과 관련하여 많은 의문점이 듭니다. 추천... 과연 많으면 많을수록 우리 삶에 더 좋은 것일까요? 위 질문과 관련해서 세부적으로 먼저 기술적 한계를 찾아보고 끝으로 윤리적 문제 등에 관한 여러 고찰을 바탕으로 바람직한 추천은 어떤 것일까에 대한 또 다른 화두를 던지며 마쳐볼까 합니다.

먼저 콘텐츠 기반 추천 알고리즘입니다. 솔직히 큰 결함이라기보다는 더 정확한 추천시스템인 협업필터링으로 발전할 수밖에 없는 계기가 되었을 수도 있겠는데요. 예를 들어 콘텐츠를 스트리밍으로 제공하는 OTT 회사에 신규가입자가 들어왔습니다. 그럼 자동으로 회사에서는 가장 인상 깊게 본 영화를 묻게 될 것이고 그래서 봉준호 감독의 '살인의 추억'과 홍상수 감독의 '생활의 발견'이라고 대답하였다고 하죠. 일단은 관찰을 하며 이 두 영화와 비슷한 장르의 영화를 추천하였고 결과는 실패하였습니다. 그러던 중 리모컨 조회정보로 '괴물'과 '설국열차'를 시청한 로그가 남았습니다. 그럼 회사에서는 이제 다시 본격적인 추천시스템을 가동합니다. 공통점을 뽑아보니 '봉준호' 감독이란 메타정보를 가지고 '기생충'을 추천하였습니다. 그랬더니 결과는 성공이었고 이제 이 가입자는 봉준호란 레이블이 붙여졌습니다. 그래서 다음 봉준호 감독의 영화를 추천하였는데 실패하였습니다. 원인 분석을 하던 중 이 고객은 홍상수 감독의 '극장전'을 조회정보를 검색하여 시청하였습니다. 그렇다면 다음에는 과연 어떤 영화를 추천해야 성공률이 높아질까요?
1번) 또 다른 홍상수 감독의 영화 '지금은 맞고 그때는 틀리다'를 추천한다. 2)번 이와 비슷한 장르의 영화 '내가 고백을 하면'을 추천한다. 3번) 살인의 추억과 같은 주연 배우(김상경)의 다른 영화 '화려한 휴가'를 추천한다. 이 새로운 가입자의 영화 취향이 그려지십니까? 가장 가까운 답은 콘텐츠 기반의 영화 추천으로는 빨리 찾기가 힘듭니다. 첫 번째 '살인의 추억'과 '생활의 발견'의 공통점은 유명감독과 유명배우의 '페르소나' 영화라는 것입니다. 이 가입자가 검색하여 본 영화 모두가 페르소나와 관계가 있습니다. 봉준호와 송강호는 '살인의 추억'을 시작으로 '괴물', '설국열차', '기생충'을 함께하였고 홍상수와 김상경은 '생활의 발견'과 '극장전'에서 감독과 배우로 호흡하였습니다. 결론적으로 이 둘이 함께한 또 다른 영화 '하하하'를 추천하였다면 아마도 시청할 확률이 높았을 것입니다. 콘텐츠 기반 추천의 태생적 한계입니다. 그래서 여러 변수를 다루고 이를 다각도로 분석, 사용자 또는 아이템 간 선호도를 바탕으로 비슷한 그룹끼리 유사점을 추출 및 클러스터링으로 추천을 해주는 협업 필터링이 대세로 자리 잡게 되었습니다. 그럼 메모리 기반의 협업필터링은 완벽하니 콘텐츠 기반은 안 쓰이느냐? 그건 아닙니다. 메모리 기반은 머신러닝 바탕의 모델기반으로 흐름이 넘어가고 있고 콘텐츠 기반은 Parsimony 관점에서 가장 단순하고 확실한 알고리즘이기에 여러 알고리즘의 조합으로 지금도 활용되고 있습니다.

두 번째는 유사도 알고리즘인 유클리디안 거리입니다. 유클리디안 관점에서 거리상으로 멀리 떨어져 있어 다른 그룹으로 배제되었으나 코사인 유사도에서는 같은 그룹으로 재편되는 경우도 있습니다. 바로 스케일에 매우 민감하기 때문입니다. 그래서 항상 단위를 통일해야 하며 다른 경우에도 정규화 과정이 꼭 필요합니다. 영화의 예를 들어 설명해 보겠습니다. 이번엔 영화평론가 얘기입니다. 박평식이란 유명한 평론가가 있습니다. 이 분야에서 가장 오랜 경력을 가지고 있으면서 아직도 현업에서 활동하고 있죠. 그런데 이분의 평점은 타 평론가와 너무 확연히 차이가 납니다. 같은 씨네21의 이동진이나 유지나 평론가의 10점 만점 평점은 수두룩하나 박평식의 경우 아직까지 전무합니다. 그나마 9점도 두 손으로 꼽을 정도며 보통 5점이면 볼만한 영화라고 평해집니다. 그래서 "박평식이 7점이다! 무조건 보자"라는 말들도 있을 정도입니다. 그렇기 때문에 아이템 기반으로 필터링된 영화 분류의 경우 같은 수작이라 할지라도 이동진, 유지나는 같이 군집화되어 있어도 박평식은 다른 그룹(평작 또는 졸작)에 있는 것으로 분류될 것입니다. 이게 스케일이 작아서 아직 감이 안 오실지도 모르겠습니다. 또 다른 영화 예를 들어보겠습니다. 이번엔 한석규, 이미연이 나오는 '넘버3'의 한 대사입니다. 이미연이 묻습니다. 참고로 이 둘은 부부 사이입니다. "자기는 나 몇% 믿어?" 한석규가 대답합니다. "나? 51%", "에게~고작 51% 실망이야!" 그도 그럴 것이 이미연은 100% 믿고 있고 또 100%를 기대하였는데 겨우 51%라니... 차이가 2배 정도 나니 이 둘은 유클리디안 거리상 남남이나 다름없습니다. 하지만 한석규의 입장은 다릅니다. 직업이 조폭이니 누군가를 믿으면 믿을수록 그만큼 배신당할 확률이 높다는 것을 몸소 체험해 온 것이죠. 따라서 한석규가 49%면 안 믿는 거고 51%면 믿으면 무조건 믿는다는 걸 이미연은 간과한 겁니다. 이처럼 연인이나 부부 사이에서는 믿음이나 사랑의 절대적 크기보다는 같은 곳을 바라보며 함께 가고 있다면 제일 가까운 사이다는

것을 유클리디안은 몰라도 코사인 유사도가 말해주고 있는지도 모르겠습니다.

세 번째는 주성분 분석의 맹점입니다. 주성분 분석은 원천적으로 데이터 추출(Extraction)이지 데이터 선택(Selection)이 아닙니다. 원리가 데이터 분산이 큰 축을 설명해주는 대리 변수를 활용하여 데이터의 분포를 나타내는 선인 고유벡터와 여기에 투영된 데이터의 분산 값인 고유값으로 변환되었기에 새로운 도메인의 정보라 원래의 정보 손실이 불가피합니다. 따라서 정보의 특성을 설명하는데 분산의 관점에서만 고려하니 변수 간 중요도는 간과될 수 있고 중요한 데이터도 아웃라이어로 판단되어 성능을 현저하게 떨어뜨릴 수도 있습니다. 또한 클래스 간의 특성은 인지하지 못하기 때문에 분류하는데 어려움이 따를 수 있습니다. 그래서 클래스들 사이의 거리(Between-class Distance)는 최대로 하고 클래스 내부의 분산(Within-class Variance)은 최소가 되게 하는 고유값(Eigenvalue)과 고유벡터(Eigenvector)로 클래스들의 분리 축을 찾는 방법인 선형 판별 분석 LDA(Linear Discriminant Analysis)가 고려되었는데요. 하지만 이 LDA 조차도 Linear란 이름에서 알 수 있듯이 Linear한 Decision Boundary만을 사용하여 최적화하기에 새로운 Dataset에 적용 시, 과적합의 문제가 동반될 수 있고 두 클래스 간의 평균은 비슷하나 분산만 차이가 있는 경우 성능이 떨어지게 됩니다. 그래서 이러한 단점을 극복하기 위해 SVM(Support Vector Machine) 기법이 나왔는지도 모르겠습니다.

끝으로 윤리적 문제입니다. 자극은 더한 자극을 원합니다. 중독에서 종전과 같은 만족을 경험하려면 더 강한 강도나, 혹은 더 긴 지속 기간의 자극을 필요로 합니다. 대표적인 물질 중독으로는 마약류가 있고 행동 중독은 도박, 인터넷, 게임 등이 있습니다. 시청물인 콘텐츠 또한 마찬가지라고 생각합니다. 예를 들어 정부의 재난지원금 정책에 불만이 있는 중장년층이 있습니다. 물론 정치적으로는 중립이었지만 준다던 통신비 2만 원을 갑자기 취소하자 오락가락하는 정책에 불만이 생기게 되었습니다. 그래서 너튜브 검색 중 정부 정책을 비판하는 보수 정치평론가의 방송을 시청하니 속이 시원하게 되었습니다. 콘텐츠를 끝까지 소비하니 이제 너튜브에서는 부동산 정책을 비판하는 영상을 추천해줍니다. 그랬더니 별다른 반응이 없자 다른 정부 비판 영상을 추천해줍니다. 그러나 내용은 없고 보수들이 좋아할 만한 단어나 표현들로 가득한 영상에 사이다 같은 느낌을 받아 시청하였습니다. 이렇게 시청-추천-시청-추천이 반복되면서 자신도 모르게 어느샌가 정치적 스펙트럼이 중도에서 점점 우측으로 기울어지게 됩니다. 그러면서 자신에게 만족을 주었던 행동은 반복되고 더 강한 자극을 원하게 되면서 과도한 탐닉으로 이어진다면? 설상가상으로 지나친 시간 소비나 과도한 별풍 등으로 경제적인 균형이 깨지고 그로 인한 역기능이 초래된다면 이는 모두 누구의 책임인 겁니까? 물론 사례가 다소 비약적이긴 하지만 소비만을 목적으로 하는 추천이 없었다면 이 모든 일은 일어나지 않았을 것입니다. 그래서 '추천은 과연 많으면 많을수록 좋기만 한 것일까'라는 생각이 문뜩 들었습니다.

지금은 약한 인공지능 기반의 알고리즘이 주로 연구되고 있지만 먼 훗날 사람의 심정이나 감정까지 파악할 수 있는 강한 인공지능 시대에는 개인에게 만족을 주면서 올바른 방향으로 유도할 수 있는 추천시스템이 더 바람직한 추천상(狀)이 아닐까요?