
python

추천 시스템

- 시나리오
  - 영화 추천 시스템
    - 무비렌즈(https://movielens.org/) 영화 평점 데이터 사용
  - 2006년 넷플리스 프라이즈 경진대회
    - 추천 시스템의 정확도를 10% 이상 향상시킨 참가자에게 상금

추천 시스템

- 추천 시스템이란?
  - 아마존: '이 상품을 구매한 고객이 구매한 다른 상품'
  - 5단계의 별점에 따라 음악을 추천해주는 앱

- 응용분야 *(Application of Data Mining to Electronic Commerce, Springer 2001)*
  - 개요 추천
  - 사용자 평가
  - 알림 서비스
  - 연관 아이템 추천
  - 개인화

추천 시스템

1) 개요 추천
   - '이번 주의 인기 상품'
   - 통계 활용, 편집자가 선택한 아이템 추천
   - 개인화 되지 않은 추천
   - 시스템 처음 이용하거나 가끔 이용하는 사용자에게 효과
2) 사용자 평가
   - 다른 상용자의 별점이나 댓글을 보여주고 평균 평점과 같은 통계정보 제공
   - 다른 사람의 정보로부터 판단 근거를 얻음
3) 알림 서비스
   - 푸시 알림, 이메일로 사용자의 흥미 아이템 추천 → 사이트 재방문 유도
4) 연관 아이템 추천
   - 원래 아이템과 함께 연관된 아이템이나, 그 정도 제시 → 동시 구매 유도, 비교
5) 개인화
   - 인기 아이템 목록, 편집자 추천 목록을 통해 사용자가 흥미를 느낄 만한 아이템 노출
     ex) 검색 결과를 개인별로 사용자화하는 방법

추천 시스템

- 데이터 설계와 데이터 입수
  - ***선호 데이터***: 사용자의 아이템 선호도
  - ***검색 쿼리***: '50,000원 이하 한식집'
  - ***비평***: 상품이나 업체에 대한 댓글평
  - ***아이템 특징***: 상품 설명에 쓰인 단어 등에 대한 정보
  - ***인구적 특징***: 사용자의 성별이나 연령 같은 정보
  - ***맥락적 특징***: 추천 받은 아이템을 사용한 날짜나 위치 정보 or 재고 현황 등

추천 시스템

- 추천 시스템이 어려운 이유 1
  - 데이터의 희소성
  - 일부 아이템에만 집중된 정보

    ex) 인기 영화에는 평가 정보가 많지만, 그렇지 못한 영화는 정보가 없음

	영화A	영화B	영화C
사용자1	5		2
사용자2	4		1
사용자3		4	5

- 추천 시스템이 어려운 이유 2
  - 아이템의 평가 비용이 다름

    ex) 음악의 평가는 짧은 시간에 여러 곳을 평가하는 가능,

    ex) 예식장 예약, 주택 구입 → 일생에 몇 번 일어나지 않는 이벤트 → 비용이 큼

    → 예식장 웹 사이트의 페이지 뷰와 같은 간접적인 지표로 부족한 선호를 보충

추천 시스템

- 선호 데이터 수집 방법
  - 명시적 데이터

    ex) 사용자에게 직접 선호도나 관심도를 물어 봄
  - 묵시적 데이터

    ex) 사용자가 상품을 구매하거나 상품 정보를 열람하는 등 아이템에 흥미가 있다고 해석

종류	명시적	묵시적
데이터양	X	O
데이터의 정확성	O	X
미평가와 부정적 평가 구별	O	X
사용자의 인지	O	X

추천 시스템

- 추천 시스템 알고리즘
  - ***협업 필터링 (Collaborative Filtering)***
    - 영화 취향이 비슷한 사람에게 추천 영화 물어보기
    - 비슷한 사람을 찾거나 비슷한 평가를 받은 영화 찾는 방법
  - ***내용 기반 필터링 (Content-based Filtering)***
    - 감독, 장르, 제목에 포함된 단어나 내용이 비슷한 영화 찾는 방법

- 협업 필터링
  - 메모리 기반 협업 필터링 (시스템이 가진 데이터 사용)
  - ***사용자 기반 협업 필터링 (User-based** collaborative filtering)*
  - ***아이템 기반 협업 필터링 (Item-based** collaborative filtering)*

추천 시스템

- ***사용자 기반 협업 필터링***
  - '당신과 비슷한 상품을 산 고객은 이런 상품도 샀다'
  - 사용자와 아이템의 각 쌍에 대한 평점 행렬이 있을 때, 행렬의 누락된 요소에 해당하는 평점을 예측

  1) 사용자의 정보를 벡터로 나타낸다. → (5, 4, 4, 3, -)
  2) 사용자 간의 ***유사도***를 평가한다. → 두 사용자가 공통으로 평가한 항목에 대해서만 계산
  3) 유사도를 기반으로 평점을 계산한다.

- user[i] = [***rating[i][1], rating[i][2], …, rating[i][m]***]
- **유사도**는 두 대상이 비슷할수록 값이 커지고, 다를수록 작아지는 척도
  - ***피어슨 상관계수***
  - ***코사인 유사도***
  - ***자카드 계수***
  - ***유클리안 유사도***

추천 시스템

- **평균제곱차이 유사도 (Mean Squared Difference Similarity**
  - **Mean Squared Difference**

$$\frac{\text{사용자 } u\text{와 사용자 } v\text{가 평가한 상품들의 평점간의 차의 제곱}}{\text{사용자 } u\text{와 사용자 } v\text{가 모두 평가한 상품들의 수}}$$

$$\text{msd}(u, v) = \frac{1}{|I_{uv}|} \cdot \sum_{i \in I_{uv}} (r_{ui} - r_{vi})^2$$

- I_{uv}는 사용자 u와 사용자 v 모두에 의해 평가된 상품의 집합
- $|I_{uv}|$는 사용자 u와 사용자 v 모두에 의해 평가된 상품의 수

  - **Mean Squared Difference Similarity**
    - **Mean Squared Difference(msd)의 역수로 계산**
    - **차이가 클수록 Similarity 값은 작아짐**
    - **msd가 0이 되는 경우에 대비해서 1의 값을 더함**

$$\text{msd\_sim}(u, v) = \frac{1}{\text{msd}(u, v) + 1}$$

추천 시스템

- 사용자 기반 협업 필터링

코사인 유사도

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$, where A_i and B_i are components of vector A and B respectively.

ex) A=(0,1,4), B=(2,1,3)

$$similarity = \frac{(0,1,4)\cdot(2,1,3)}{\sqrt{0^2+1^2+4^2} \times \sqrt{2^2+1^2+3^2}} = \frac{13}{\sqrt{17} \times \sqrt{14}} \cong 0.84$$

추천 시스템

ex) i번째 사용자 U의 평점 벡터 u = user[i]와 j번째 사용자 V의 평점 벡터 v = user[j]의 유사도 정의

1) 피어슨 상관계수 (Pearson product-moment correlation coefficient)
   - 두 벡터의 상관계를 의미
   - -1 ~ 1 사이의 값을 가짐
   - 코사인 유사도의 단점 보안(사용자 개인의 평가 성향 반영 못함)

$$\text{pearson\_sim}(u, v) = \frac{\sum_{i \in I_{uv}} (r_{ui} - \mu_u) \cdot (r_{vi} - \mu_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)^2} \cdot \sqrt{\sum_{i \in I_{uv}} (r_{vi} - \mu_v)^2}}$$

μ_u는 사용자 u의 평균 평점

```python
import numpy as np

def pearson_coefficient(u, v):
    u_diff = u - np.mean(u)
    v_diff = v - np.mean(v)
    numerator = np.dot(u_diff, v_diff)
    denomicator = np.sqrt(sum(u_diff ** 2)) * np.sqrt(sum(v_diff ** 2))

    return numerator / denomicator
```

추천 시스템

ex) i번째 사용자 U의 평점 벡터 u = user[i]와 j번째 사용자 V의 평점 벡터 v = user[j]의 유사도 정의

2) 코사인 유사도 (Cosine similarity)
- 두 특성 벡터간의 유사 정보를 코사인 값으로 표현
- 텍스트 문장 간의 거리를 측정하는 철도
- -1 ~ 1 사이의 값 (-1: 완전히 반대, 0: 서로 독립적, 1: 서로 완전히 같음)

$$x \cdot y = |x||y| \cos\theta$$

$$\cos\theta = \frac{x \cdot y}{|x||y|}$$

$$\text{cosine\_sim}(u, v) = \frac{\sum_{i \in I_{uv}} r_{ui} \cdot r_{vi}}{\sqrt{\sum_{i \in I_{uv}} r_{ui}^2} \cdot \sqrt{\sum_{i \in I_{uv}} r_{vi}^2}}$$

두 사용자가 모두 평가한 상품의 평점을 사용해서 계산

```
np.dot(u, v) / (np.sqrt(sum(u ** 2)) * np.sqrt(sum(v ** 2))
```

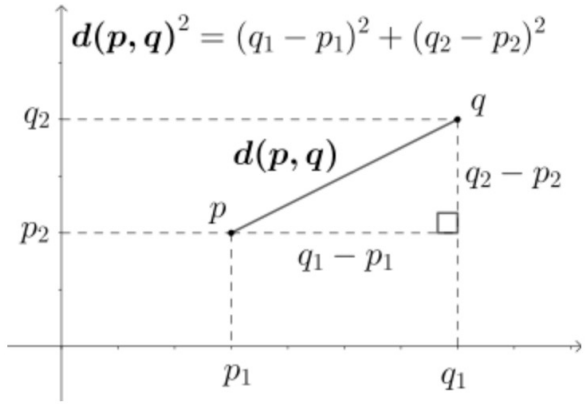
추천 시스템

3) 자카드 계수 (Jaccard index, Jaccard similarity coefficient)
   - 집합과 집합 사이의 거리 계산
   - 0 ~ 1 사이의 값을 가짐

```
np.dot(u, v) / (sum(np.absolute(u)) + sum(np.absolute(v)) - np.dot(u, v))
```

4) 유클리디안 거리 공식 (Euclidean distance)
   - 기본적인 이해를 위해 사용
   - 비교대상 간의 차이인 (x_i-y_i)의 제곱을 모두 구해 더한 다음 제곱근을 취하는 방식
   - 두 점 사이의 거리를 계산할 때 주로 사용
   - ex) 직교 좌표계 $p=(p_1, p_2, \cdots p_n)$, $q=(q_1, q_2, \cdots, q_n)$

$$\sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\ \ldots\ +(q_n-p_n)^2}=\sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}$$



다차원 공간

-	바나나	사과	저는	좋아요
문서1	2	3	0	1
문서2	1	2	3	1
문서3	2	1	2	2

-	바나나	사과	저는	좋아요
문서Q	1	1	0	1

문서Q와 유사한 문서는?

추천 시스템

- 사용자 기반 협업 필터링 → ***성향이 비슷한 사용자가 선호하는 아이템을 제시하는 방법***

- 평점을 예측하는 가장 단순한 방법 → ***가장 비슷한 k명의 사용자 평가를 평균***

ex) 어떤 사용자 U가 영화 M에 내렸던 평가 예측

np.mean(nearest_user_ratings) / k

추천 시스템

- ***아이템 기반 협업 필터링***
  - 피어슨 상관계수나 코사인 유사도를 이용 → 높은 유사도의 아이템 검색
  - 개선된 코사인 유사도를 사용하기도 함
  - 분모로 영화 M에 대한 평점 평균을 이용
  - 영화 M의 평점 평균 대신 사용자 평점 평균으로 나누면 개선된 코사인 유사도

- ***내용 기반 필터링***
  - 영화제목, 감독, 장르, 배우, 평판 등 아이템을 나타내는 정보에 주목
    → 이들과의 정보를 이용하여 추천 항목 선택
  - 사용자 취향을 뜻하는 단어를 알 수 있다면 그 정보를 통해 취향에 맞는 영화 제시

추천 시스템

- ***협업 필터링***과 ***내용 기반 필터링***의 장단점
  - ***협업 필터링***
    - 장점)
      - 장르나 텍스트에 포함된 단어에 유사점이 없어도 되기 때문에 비교적 다양한 추천 결과 얻을 수 있음
      - 도메인 지식 필요 없음
    - 단점)
      - 데이터가 충분히 쌓이기 전까지는 신규 사용자나 새로운 아이템에 대한 추천 어려움
      - 사용자가 적으면 적절한 추천 불가능 → 사용자가 늘지 않는 악순환에 빠질 수 있음
  - ***내용 기반 필터링***
    - 장점)
      - 새로운 서비스처럼 누적된 행동 데이터가 없는 상황에서도 비교적 적절한 추천 가능
    - 단점)
      - 한국에 적용하려면 형태소 분석, 사전 유지보수 등 도메인에 특화된 정보를 다뤄야 함

추천 시스템 구현

- 협업 필터링에 기반한 추천 알고리즘
  - ***유사도(Similarity) 분석***: 넷플릭스, 왓챠 등에서 사용
  - 두 사람간의 유사도를 구해 ***예상 평점을 구하기*** → ***영화 추천***

	캡틴 마블	라스트 미션	돈
사용자1	5	4	2
사용자2	4	2	1
ME		4	3
사용자3	3.5	4	4

```
ratings = {
    'user1':{'captain marvel':5, 'last mission':4, 'money':2},
    'user2':{'captain marvel':4, 'last mission':2, 'money':1},
    'me':{'last mission':4, 'money':3},
    'user3':{'captain marvel':3.5, 'last mission':4, 'money':4}
}
```

추천 시스템 구현

- ex1) me의 영화 평점 구하기

  >>> ratings[‘me’]

  {'last mission': 4, 'money': 3}

- ex2) me의 ‘money ' 영화에 대한 평점 구하기

  >>> ratings[‘me’][‘money’]

  3

추천 시스템 구현

1) 피타고라스 정리를 이용한 두 사람간의 거리 구하기

```
def similarity(i, j):
    return sqrt(pow(i, 2) + pow(j, 2))
```

- ex3) user1과 me사이의 두 영화 (‘last mission’, ‘money’)의 평점 거리 구하기

```
var1 = ratings[‘me']['last mission'] - ratings[‘user3']['last mission']
var2 = ratings[‘me']['money'] - ratings[‘user3']['money']
print(similarity(var1, var2))
```

추천 시스템 구현

2) 2차원에서 피타고라스 정리를 이용한 모든 사용자와의 거리 구하기

```
for u in ratings:
    if u != 'me': #자기자신제외
        num1 = ratings['me’]['last mission’] - ratings[u]['last mission′]
        num2 = ratings['me’]['money’] - ratings[u]['money′]
        print(u, " : ", similarity(num1, num2))
```

user1 : 1.0
user2 : 2.828427124746 1903
user3 : 1.0

추천 시스템 구현

3) 정규화(Normalization)

- 평균화(Standardization) 사용
- 보기 쉽게 하기 위해 값이 큰 것을 유사한 것으로 처리
- 0 ~ 1 사이의 범위로 변경

$$\tilde{d}_i := \frac{d_i - \min\{d\}}{\max\{d\} - \min\{d\}}$$

```
for u in ratings:
    if u != 'me': #자기자신제외
        num1 = ratings['me']['last mission'] - ratings[u]['last mission']
        num2 = ratings['me']['money'] - ratings[u]['money']
        print(u, " : ", 1 / (1 + similarity(num1, num2)))
```

user1 : 0.5
user2 : 0.2612038749637414
user3 : 0.5

문제점은?

추천 시스템 구현

4) 다차원의 유사도 구하기

- ***유클리디안 유사도(Euclidean distance)***
- 코사인 유사도 (Cosine distance)
- 자카드 유사도 (Jaccard's distance)

- ***유클리디안 거리*** (Euclidean distance)
  - i번째 비교대상 아이템 간의 값인 $(x_i - y_i)^2$의 합을 구한 다음 ***제곱근*** 처리

$$= \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

추천 시스템 구현

4) 다차원의 유사도 구하기

```
def similarity_distance(data, name1, name2):
    sum = 0
    for u in data[name1]:
        if u in data[name2]: #같은 영화를 봤다면
            sum += pow(data[name1][u]- data[name2][u], 2)

    return 1/(1+sqrt(sum))
```

```
>>> similarity_distance(ratings, 'user1', 'user2')
0.2898794855663564
```

추천 시스템 구현

5) 전체 데이터에서 유사도가 가장 가까운 사람 구하기

```
def nearest_user(data, name, index=2, sim_function=similarity_distance):
    nearest_user_list = []
    for u in data:
        if name != u: #자기 자신은 제외한다
            nearest_user_list.append(similarity_distance(data, name, u), u)

    nearest_user_list.sort() #오름차순 정렬
    nearest_user_list.reverse() #내림차순 정렬

    return nearest_user_list[:index]
```

```
>>> nearest_user(ratings, 'user2')
[
 (0.2898979485566356, 'user1'),
 (0.2612038749637414, 'me')
]
```

추천 시스템 구현

6) 평점과 이름 분리

```
def nearest_user(data, name, index=2, sim_function=similarity_distance):
    score, names = []
    for u in data:
        if name != u: #자기 자신은 제외한다
            score.append(similarity_distance(data, name, u))
            names.append(u)

    score.sort()
    score.reverse()
    names.sort()
    names.reverse()

    return score, names
```

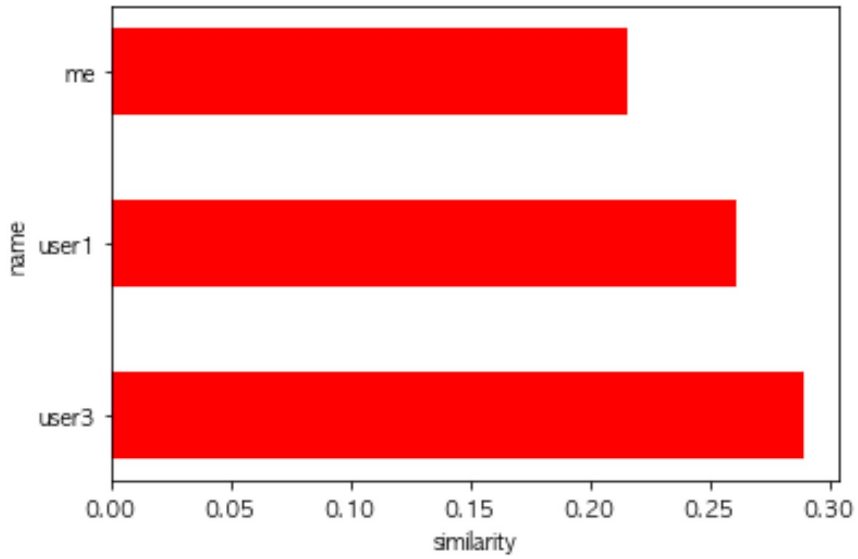
```
>>> score, names = nearest_user(ratings, 'user2')
>>> score
[0.2898979485566356 4, 0.2612038749637414, 0.2155146893583885 2]
>>> names
['user3', 'user1', 'me']
>>> draw_chart(score, names)
```

추천 시스템 구현

7) 데이터 시각화 함수 구현

- matplotlib.pyplot 사용

```
def draw_chart(data, labels): # data, labels는 list형태로 사용
    positions = range(len(data))
    plt.barh(positions, data, height=0.5,color='r') #가로
    plt.yticks(positions, labels)
    plt.xlabel('similarity') #x축
    plt.ylabel('name') #y축
    plt.show() #출력
```



추천 시스템 구현

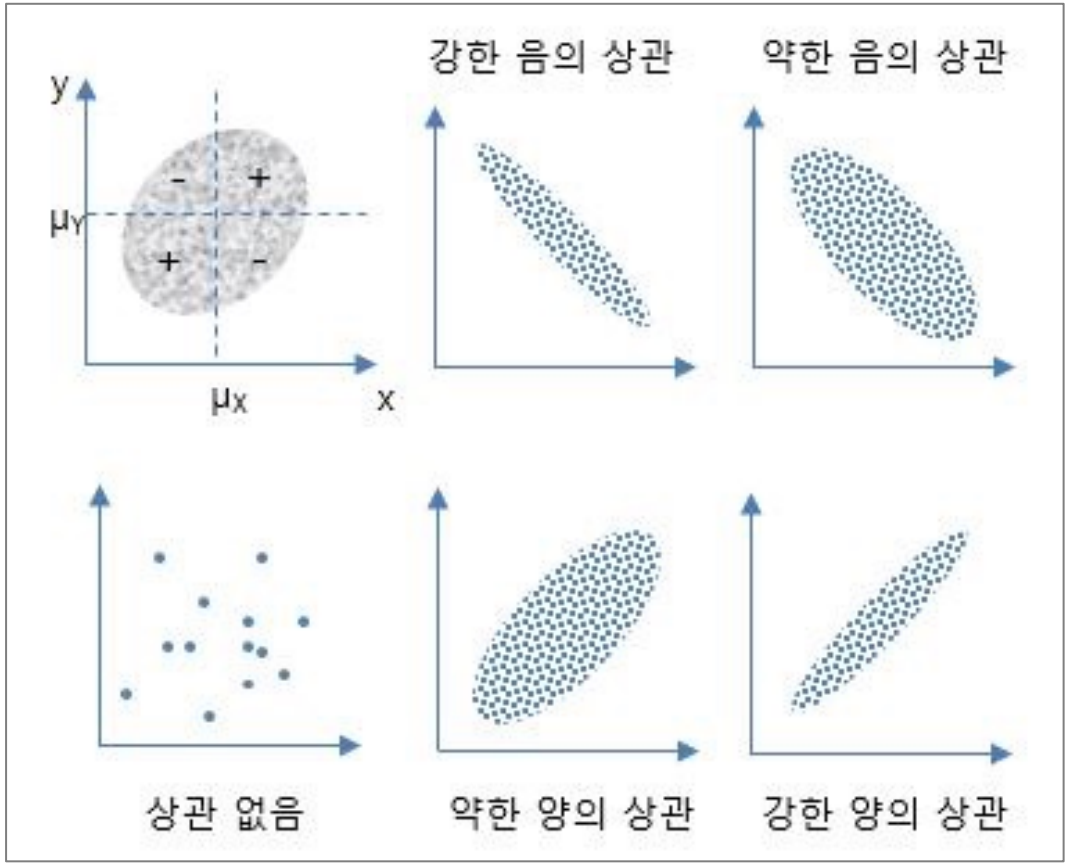
- 유클리디안 거리공식의 문제점

  → 특정 인물의 점수기준이 너무 낮거나 높으면 제대로 된 결과가 나오지 못함

  ex) 재미없는 영화 1, 재미있는 영화 5

8) 상관분석 (Correlation Analysis)

- 두 변수간의 선형적 관계에 대한 분석
- 점수간 관계에 따라 점을 찍은 후 그 점이 분포한 모양에 따라 상관관계 도출



추천 시스템 구현

8) 상관분석 (Correlation Analysis)

- 데이터 변경 → 영화의 개수 및 각 사용자가 본 영화의 수도 다름

```
ratings = {
    'user1': {
        '돈': 2.5,
        '캡틴마블': 3.5,
        '보헤미안 랩소디': 3.0,
        '극한직업': 3.5,
        '이스케이프 룸': 2.5,
        '증인': 3.0,
    },
    'user2': {
        '돈': 1.0,
        '캡틴마블': 4.5,
        '보헤미안 랩소디': 0.5,
        '극한직업': 1.5,
        '이스케이프 룸': 4.5,
        '증인': 5.0,
    },
    …
    'user9': {
        '돈': 3.7,
        '캡틴마블': 4.0,
        '이스케이프 룸': 3.0,
        '극한직업': 4.9,
        '보헤미안 랩소디': 4.1,
    },
}
```

추천 시스템 구현

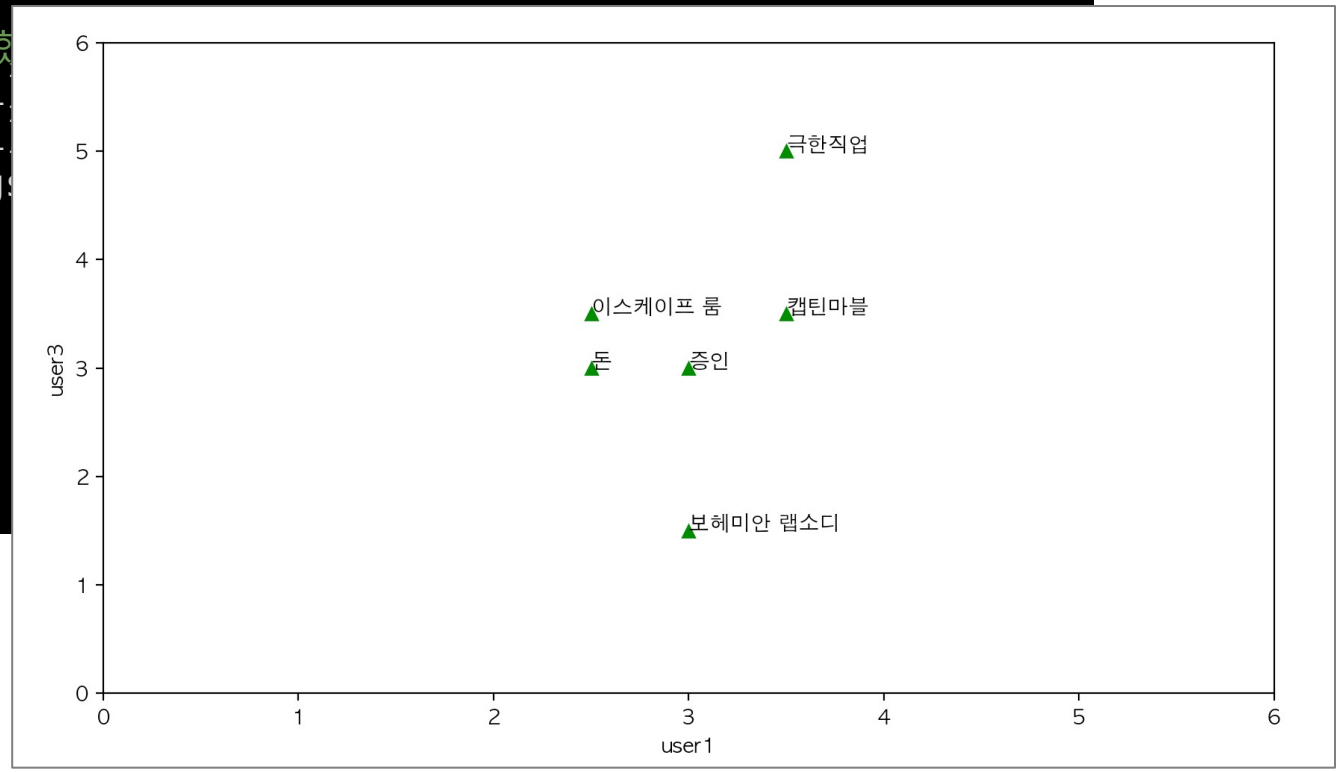
8) 상관분석 (Correlation Analysis)

- scatter plot

```
def draw_graph(data, name1, name2):
    plt.figure(figsize=(14,8))
    x_ratings, y_ratings = []
    for i in ratings[name1]: # i = 키 값
        if i in data[name2]: # 같은 영화를 평가했
            x_ratings.append(ratings[name1][i]
            y_ratings.append(ratings[name2][i]
            plt.text(ratings[name1][i],ratings

    plt.plot(x_ratings, y_ratings,'g^')
    plt.axis([0, 6, 0, 6])
    plt.xlabel(name1)
    plt.ylabel(name2)

    plt.show()
```



>>> draw_graph(ratings, 'user1', 'user3')

추천 시스템 구현

9) 두 사용자간의 상관계수 구하기

- 피어슨 상관계수 (Peason Correlation Coefficient) 사용
- -1 ~ 1 사이의 값 (1에 가까울수록 양의 상관관계)

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2 \sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

$$\text{피어슨 상관계수} = \frac{\text{x, y의 공분산}}{\text{x의 표준편차} * \text{y의 표준편차}}$$

$$= \frac{-20}{\sqrt{2.5} * \sqrt{10}}$$

$$= -1$$

추천 시스템 구현

9) 두 사용자간의 상관계수 구하기

```
def pearson_correlation_coefficient(data, name1, name2):
    sumX=0 # X의 합
    sumY=0 # Y의 합
    sumPowX=0 # X 제곱의 합
    sumPowY=0 # Y 제곱의 합
    sumXY=0 # X*Y의 합
    count=0 #영화 개수

    for movie in data[name1]: # i = key
        if movie in data[name2]: # 같은 영화를 평가했을때만
            sumX += data[name1][movie]
            sumY += data[name2][movie]
            sumPowX += pow(data[name1][movie], 2)
            sumPowY += pow(data[name2][movie], 2)
            sumXY += data[name1][movie] * data[name2][movie]
            count += 1

    return ( sumXY- ((sumX*sumY)/count))
                / sqrt((sumPowX - (pow(sumX,2) / count)) * (sumPowY - (pow(sumY,2) / count)))
```

```
>>> pearson_correlation_coefficient(ratings, 'user1', 'user3')      0.3960590171906697
>>> pearson_correlation_coefficient(ratings, 'user5', 'user8')      0.8934051474415647
```

← 더 높은 유사성

추천 시스템 구현

10) 전체 인원과의 상관계수 구하기

- 전체 데이터를 기준으로 기준이 되는 사람과 나머지 전체 사용자간의 상관계수 구하기

```
def nearest_user(data, name, index = 3):
    result = []
    for u in data:
        if name != u:
            result.append((pearson_correlation_coefficient(data, name, u), u))

    result.sort()
    result.reverse()
    return result[:index]
```

```
>>> nearest_user(ratings, 'user9', 5)
 [
  (0.9607689228305331, 'user4'),
  (0.9281909617845113, 'user5'),
  (0.8180002591461302, 'user7'),
  (0.8105674873765092, 'user8'),
  (0.7997092494906686, 'user1')
 ]
```

추천 시스템 구현

11) 실제 영화 추천, 예상 평점 구하기

- 상관관계 → 공동으로 내린 평가 기준
- 영화추천 → 사용자가 평가를 내리지 않은 영화
  - 유사도 값을 근거로 일정 기준을 충족하는 모든 사용자의 예상 평점과 추천영화 참고

① 모든 사람들의 영화 평점과 유사도 계산

② 예상 평점 구하기

③ 예상 평점의 합, 예상 평균 구하기

④ 예상 평균을 보지 않은 영화에 대해 모두 계산 ***→ 가장 높은 영화 → 추천영화***

추천 시스템 구현

11) 실제 영화 추천, 예상 평점 구하기

- 추천 함수 → getRecommendation()

```
def getRecommendation (data, user):
    result = nearest_user(ratings, user ,len(data))
    score = 0
    rcmmdList =[]
    score_dic = {} # 유사도 총합을 위한 dic
    similarity_dic = {} # 평점 총합을 위한 dic

    for similarity, name in result:
        if similarity < 0 :
                continue

        for movie in data[name]:
            if movie not in data[user]: #name이 평가를 내리지 않은 영화
                score += similarity * data[name][movie] # 영화평점 * 유사도
                score_dic.setdefault(movie, 0) # 기본값 설정
                score_dic[movie] += score # 합계 구함
                # 조건에 맞는 사람의 유사도의 누적합을 구한다
                similarity_dic.setdefault(movie,0)
                similarity_dic[movie] += similarity

            score = 0 #영화가 바뀌었으니 초기화한다
```

추천 시스템 구현

11) 실제 영화 추천, 예상 평점 구하기

- 추천 함수 → getRecommendation()

```
    …
    for key in score_dic:
        score_dic[key] = score_dic[key] / similarity_dic[key] # 평점 총합/ 유사도 총합
        rcmmdList.append( score_dic[key], key )

    rcmmdList.sort()
    rcmmdList.reverse()

    return rcmmdList
```

>>> getRecommendation(ratings, 'user8')

(3.467750847406967, '증인')

추천 시스템 구현

- 무비렌즈 데이터 분석
  - http://files.grouplens.org/papers/ml-100k.zip
  - 사용자 정보, 평점 정보, 영화 정보의 내용 살펴 보기

```
import pandas as pd

u_cols = ['user_id', 'agr', 'gender', 'occupation', 'zip_code']
users = pd.read_csv('/Users/dowon/Desktop/Step1/pr-test/Day3/ml-100k/u.user', sep='|', names=u_cols)
users_head = users.head()
print(users_head)
```

```
   user_id  agr gender  occupation zip_code
0        1   24      M  technician    85711
1        2   53      F       other    94043
2        3   23      M      writer    32067
3        4   24      M  technician    43537
4        5   33      F       other    15213
```

추천 시스템 구현

- 무비렌즈 데이터 분석
  - rating 정보

```
r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']
ratings = pd.read_csv('/Users/dowon/Desktop/Step1/pr-test/Day3/ml-100k/u.data', sep='\t', names=r_cols)
ratings['date'] = pd.to_datetime(ratings['unix_timestamp'],unit='s')
rating_head = ratings.head()
print(rating_head)
```

```
   user_id  movie_id  rating  unix_timestamp                 date
0      196       242       3       881250949 1997-12-04 15:55:49
1      186       302       3       891717742 1998-04-04 19:22:22
2       22       377       1       878887116 1997-11-07 07:18:36
3      244        51       2       880606923 1997-11-27 05:02:03
4      166       346       1       886397596 1998-02-02 05:33:16
```

추천 시스템 구현

- 무비렌즈 데이터 분석
  - 평점 정보 보기

```
m_cols = ['movie_id', 'title', 'release_date', 'video_release_date', 'imdb_url']
movies = pd.read_csv('/Users/dowon/Desktop/Step1/pr-test/Day3/ml-100k/u.item', sep='|', names=m_cols,
usecols=range(5), encoding='latin1')
moview_head = movies.head()
```

	movie_id	title	release_date	video_release_date	imdb_url
0	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...
1	2	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...
2	3	Four Rooms (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Four%20Rooms%...
3	4	Get Shorty (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Get%20Shorty%...
4	5	Copycat (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Copycat%20(1995)

추천 시스템 구현

- 무비렌즈 데이터 분석
  - 전체 데이터 중 평점을 가장 많이 받은 25개 작품의 제목

```
movie_rating = pd.merge(movies, ratings)
lens = pd.merge(movie_rating, users)
top25 = lens.title.value_counts()[:25]
print(top25)
```

```
Star Wars (1977)                            583
Contact (1997)                              509
Fargo (1996)                                508
Return of the Jedi (1983)                   507
Liar Liar (1997)                            485
English Patient, The (1996)                 481
Scream (1996)                               478
Toy Story (1995)                            452
Air Force One (1997)                        431
Independence Day (ID4) (1996)               429
```

추천 시스템 구현

- 무비렌즈 데이터 분석
  - rating 수와 평균 계산, 평균값 정렬

```
movie_stats = lens.groupby('title').agg({'rating': [np.size, np.mean]})
movie_stats.sort_values(by=[('rating', 'mean')], ascending=False).head()
print(movie_stats)
```

	rating	
	size	mean
title		
They Made Me a Criminal (1939)	1	5.0
Marlene Dietrich: Shadow and Light (1996)	1	5.0
Saint of Fort Washington, The (1993)	2	5.0
Someone Else's America (1995)	1	5.0
Star Kid (1997)	3	5.0

추천 시스템 구현

- 무비렌즈 데이터 분석
  - 100건 이상 평가가 된 영화의 수에 대해서 상위 15건만 체크

```
movie_stats = lens.groupby('title').agg({'rating': [np.size, np.mean]})
movie_stats.sort_values(by=[('rating', 'mean')], ascending=False).head()
print(movie_stats)
```

	rating	
	size	mean
title		
Close Shave, A (1995)	112	4.491071
Schindler's List (1993)	298	4.466443
Wrong Trousers, The (1993)	118	4.466102
Casablanca (1942)	243	4.456790
Shawshank Redemption, The (1994)	283	4.445230
Rear Window (1954)	209	4.387560
Usual Suspects, The (1995)	267	4.385768
Star Wars (1977)	583	4.358491