Collaborative Filtering - Recommend Algorithm

협업 필터링을 사용한 추천 알고리즘에는 크게 사용자 기반 협업 필터링(User Based Collaborative Filtering)과 아이템 기반 협업 필터링(Item Based Collaborative Filtering)이 있습니다. 보편적으로는 추천 시스템으로 사용자 기반 협업 필터링이 사용되는데, 아이템 기반 협업 필터링은 유져수와 아이템이 겹치는 수가 적을 경우에 아이템 기반으로 필터링을 하는것으로 대체로 쇼핑몰 같은 곳에서 상품의 종류는 많은데 사용자가 구매하는 아이템이 많지 않아 사용자간에 구매 아이템이 겹치는경우가 드문경우에 사용됩니다.

여기에서는 사용자 기반 협업 필터링을 학습하고 구현해보겠습니다.

1. Similarity - 유사도

유사도는 두개의 벡터 데이터사이에 얼마나 가까운지를 나타내는 척도로 Euclidean Distance, Cosine, Manhattan Distance, Minkow Distance, Jaccard Similarity등이 있습니다.

여기에서는 Euclidean Distance, Cosine Similarity를 구현해 보고, 실제 테스트 데이터에 적용하여 두개의 차이점을 살펴보겠습니다. 결론부터 이야기하자면 추천 알고리즘은 Euclidean Distance Similarity보다 Cosine Similarity가 더욱 정확한 결과를 낼수 있습니다. 그 이유는 두가지 방법의 유사도 구하는 방법을 학습한 후에 설명하겠습니다.

2. Euclidean Distance Similarity - 유클리드 거리 유사도

두 벡터사이의 거리를 측정하는 방법으로 공간상에서 두개의 벡터가 가까이 있을수록 두 벡터 같은 특징을 가진다는것을 의미 합니다.

벡터의 각 좌표를 뺀후에 제곱한 수를 다 더하고 제곱근을 구해주는 방법으로 수식으로 나타내면 아래와 같습니다. 우리가 예전에 배웠던 직각 삼각형에서 빗변을 구해줄때 사용하는 피타고라스의 정리를 생각하시면 됩니다. 직각 삼각형은 2차원에서 빗변을 구했지만 유클리드는 벡터 좌표의 수만큼 차원이 증가한 벡터 좌표의 거리를 구한다고 생각하면 됩니다.

$$similarity = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}$$

```
import numpy as np


# 샘플 데이터

vector_1 = np.array([1,2,3,4,5])

vector_2 = np.array([5,4,3,2,1])

vector_3 = np.array([2,3,4,5,6])


# 계산과정

print((vector_1 - vector_2))

print((vector_1 - vector_2) ** 2)

print(sum((vector_1 - vector_2) ** 2))

print(np.sqrt(sum((vector_1 - vector_2) ** 2)))


# numpy 함수 사용

np.linalg.norm(vector_1 - vector_2), np.linalg.norm(vector_1 - vector_3)


# 결과
[-4 -2  0  2  4]
[16  4  0  4 16]
40
6.324555320336759
(6.324555320336759, 2.23606797749979)
```

계산과정을 보면 처음에는 각 요소들을 빼주고 뺀 결과를 제곱한후에 각 요소을 다 더하고 제곱근을 해주면 두 벡터 사이의 거리를 구할수 있습니다.

계산 결과를 보면 vector_1과 vector_2의 거리가 6.32정도가 나왔고, vector_1과 vector_3 사이의 거리는 2.23정도로 vector_1과 vector_2 사이의 거리보다 vector_1과 vector_2 사이의 거리가 더욱 가까운것을 확인할 수 있습니다.

3. Cosine Similarity - 코사인 유사도

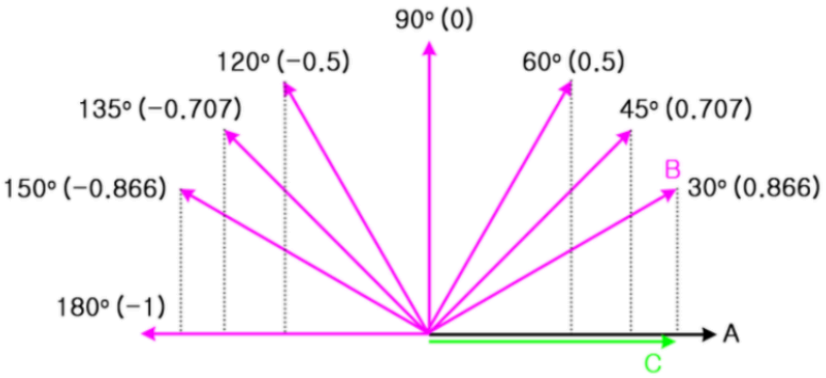
코사인 유사도는 두 벡터 사이의 $\cos\theta$ 값을 구하는 것으로 θ 각이 작다는 것은 두 벡터가 같은 방향성을 갖는것을 의미 합니다. 그러므로 코사인 유사도 결과값이 크다는것(θ가 작음)은 비슷한 방향성을 갖는 비슷한 벡터 데이터라는 의미가 됩니다. 코사인 유사도는 벡터의 내적에서 벡터의 크기를 나눈것으로 수식은 아래와 같습니다.

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta$$

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Cosine Similarity Formula

내적은 아래의 그림에서 A와 B의 내적은 C가 됩니다. 세타 값이 증가 할때마다 내적의 크기는 작아 집니다.



벡터 사이의 거리

import numpy as np

from scipy import spatial

샘플 데이터

vector_1 = np.array([1,2,3,4,5])

vector_2 = np.array([5,4,3,2,1])

vector_3 = np.array([11,19,28,32,47])

분자 - 벡터의 내적

print(sum(vector_1 * vector_2))

print(np.dot(vector_1, vector_2)) # 내적 numpy 함수

분모 - 벡터의 크기

계산과정

print(vector_1 * vector_1, vector_2 * vector_2)

print(sum(vector_1 * vector_1), sum(vector_2 * vector_2))

print(np.sqrt(sum(vector_1 * vector_1)), np.sqrt(sum(vector_2 * vector_2)))

numpy 함수 사용

print(np.sqrt(np.dot(vector_1, vector_1)) * np.sqrt(np.dot(vector_2, vector_2)))

벡터의 내적 / 벡터의 크기

print(np.dot(vector_1, vector_2) / (np.sqrt(np.dot(vector_1, vector_1)) * np.sqrt(np.dot(vector_2, vector_2))))

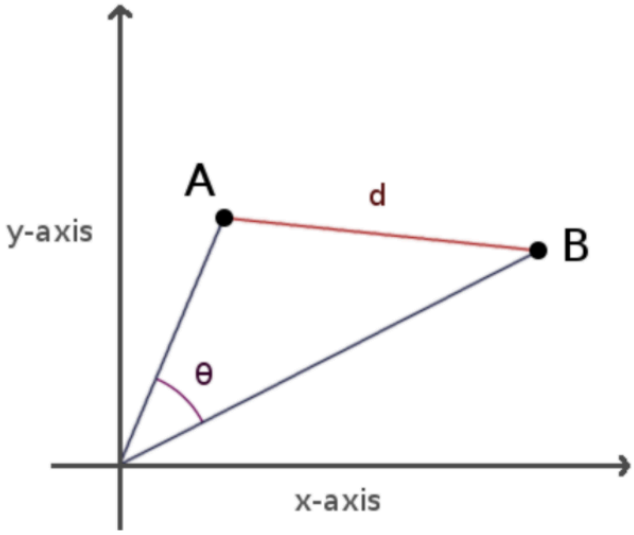
scipy 함수 사용

1 - spatial.distance.cosine(vector_1, vector_2), 1 - spatial.distance.cosine(vector_1, vector_3)

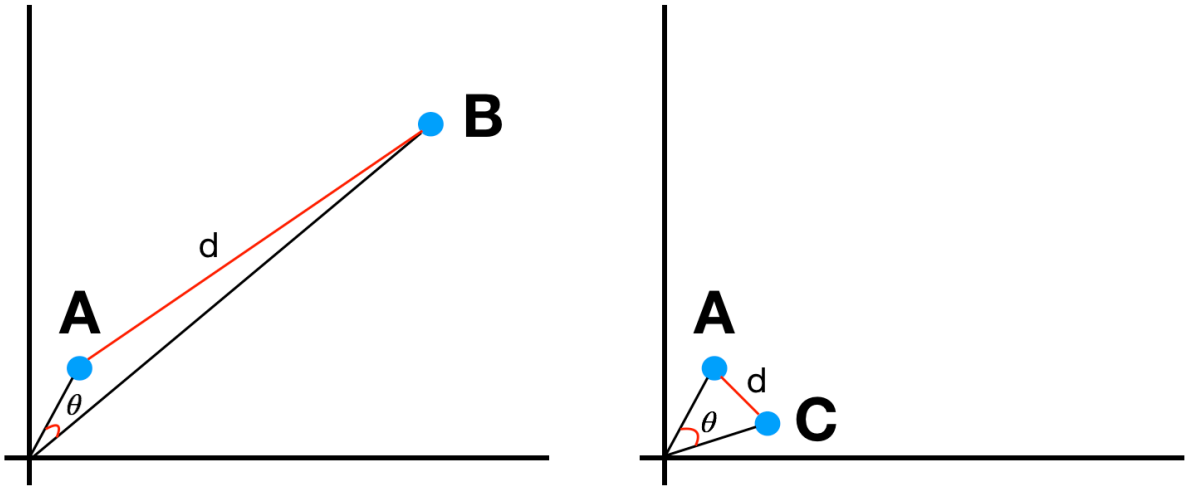
---

4. The Difference between Euclidean Distance Similarity and Cosine Similarity

d는 Euclidean Distance를 의미하며, $\cos\theta$의 θ를 의미합니다. 두개의 유사도는 하나는 거리를 하나는 각도를 의미하는 수치를 나타냅니다. 그러므로 d 값이 작다고해서 θ 값이 작은건 아니며 θ 값이 크다고해서 d 값이 큰것은 아닙니다.



아래의 그래프는 d 값이 작다고해서 θ 값이 작은건 아니며 θ 값이 크다고해서 d 값이 큰것이 아님을 보여주는 그래프 입니다. 왼쪽의 그래프는 θ 값이 작지만 d값이 큰경우이고, 오른쪽의 그래프는 d값이 작지만 θ값이 큰 그래프를 보여줍니다.



위에서 살펴본 Euclidean Distance와 Cosine Similarity의 특성으로 인해 추천 알고리즘에는 Cosine Similarity가 사용된다고 위에서 언급하였습니다.

예를들어 뉴스 컨텐츠를 소비하는데 A 사용자는 서비스에 대한 소비량이 적고 B 사용자는 서비스에 대한 소비량이 많지만 소비하는 컨텐츠는 비슷한 경우 왼쪽과 같은 모양의 그래프가 나옵니다. A와 C 사용자 모두 컨텐츠 소비는 비슷하게 하는데 서로 다른 컨텐츠를 소비하는 경우에는 우측과 같은 모양의 그래프가 나옵니다.

이러한 경우 A에게 컨텐츠를 추천해주기 위해서 어떤 사용자의 데이터를 활용하는게 옳을까요? B 사용자의 데이터를 활용하여 A에게 컨텐츠를 추천해주는것이 더욱 정확합니다.

5. Recommend System - 추천 시스템

위에서 배운 Cosine Similarity를 이용하여 간단한 추천 시스템을 학습하겠습니다. 아래와 같은 순서로 학습하며 추천 시스템을 만들어 보도록 하겠습니다. 또한 시스템을 만들면서 유사도를 통해 구한 데이터에서 mean score 행렬을 만들어 어떻게 추천을 하는지를 구현해 보고, 결과 데이터에 대한 평가 수치까지 학습하겠습니다.

- sample dataset - 간단한 샘플 데이터
- similarity matrix - 사용자 기반의 유사도 행렬
- similarity mean score matrix - 추천할 대상에 대한 평균 스코어 행렬
- recommend - 평균 스코어 행렬로 추천할 컨텐츠 순서
- error - 오차 - MSE, RMSE, MAE

(1) Sample Dataset - 샘플 데이터

간단한 샘플 데이터를 Pandas의 Dataframe으로 만듭니다. 데이터는 (0, 1, 2, 3, 4, 5) 6가지의 수치로 나타내며 수치에 대한 값은 컨텐츠에 가지는 흥미도를 나타냅니다. 흥미도는 컨텐츠를 본 시간, like 클릭, 댓글 작성등의 활동을 조사하여 점수를 구할수 있습니다. 실습할 예제는 이미 이런 데이터를 구한 상태의 데이터를 가지고 특정 사용자에게 컨텐츠를 추천 하도록 하겠습니다.

```
import numpy as np
import pandas as pd
from scipy import spatial
```

```
# sample data set matrix
columns = ["article_1","article_2","article_3","article_4","article_5"]
index = ["user_1", "user_2", "user_3", "user_4"]

data = np.array([
    [5,3,0,0,2],
    [2,0,0,1,4],
    [0,0,4,3,1],
    [4,0,4,5,0],
])

sample_df = pd.DataFrame(data, columns=columns, index=index)
sample_df
```

---

	article_1	article_2	article_3	article_4	article_5
user_1	5	3	0	0	2
user_2	2	0	0	1	4
user_3	0	0	4	3	1
user_4	4	0	4	5	0

(2) Similarity Matrix - 유사도 행렬

사용자 기반 유사도 필터링을 사용하기 때문에 모든 사용자 사이의 유사도를 구합니다.

유사도 행렬을 구할때 주의 해야할 것은 샘플 데이터에서 value가 0인 데이터는 대상이 되는 user가 전혀 활동이 없었다는것을 의미하기 때문에 유사도를 구할때 value가 0인 article은 데이터는 제거하였습니다. 만약에 데이터가 사용자가 제목을 보고도 관심이 없어서 열지 않았다는 가정을 하면 value가 0인 데이터는 그냥 넣어도 됩니다.

```
# 코사인 유사도 구하는 함수
def cosine_smimilarity(vector_1, vector_2):

    # vector_1 데이터가 0인 index를 제거
    idx = vector_1.nonzero()[0] # vector에서 value가 0이 아닌 index를 구함
    # index 값으로 vector의 요소를 필터링 함
    vector_1, vector_2 = np.array(vector_1)[idx], np.array(vector_2)[idx]

    return 1 - spatial.distance.cosine(vector_1, vector_2)
```

샘플 데이터프레임과 유사도 함수를 넣으면 유사도 매트릭스를 구해주는 함수를 작성합니다.

```
# 유사도 행렬 함수
def similarity_matrix(sample_df, similarity_func):

    # index 데이터 저장
    index = sample_df.index

    # 데이터 프레임 전치 (index - article, columns - user)
    df = sample_df.T

    # 모든 user 데이터 사이의 유사도를 구해 행렬 생성
    matrix = []
    for idx_1, value_1 in df.items():
        # row 데이터 저장
        row = []
        for idx_2, value_2 in df.items():
            # 두 user 사이의 유사도 구함
```

```
            row.append(similarity_func(value_1, value_2))
        matrix.append(row)

    return pd.DataFrame(matrix, columns=index, index=index)

sm_df = similarity_matrix(sample_df, cosine_smimilarity)
sm_df
```

---

	user_1	user_2	user_3	user_4
user_1	1.000000	0.652929	0.324443	0.811107
user_2	0.729397	1.000000	0.483046	0.443039
user_3	0.196116	0.332956	1.000000	0.949474
user_4	0.529813	0.770054	0.821210	1.000000

(3) Similarity Mean Score Matrix - 유사도 평균값 행렬

추천 대상을 정하고 그 대상에 맞는 유사도 행렬을 구하는 코드입니다.

```
# 추천할 대상 및 추천 대상과 유사한 몇개의 데이터까지 사용할지에 대해 설정
user, closer_count = "user_1", 2

# 본인 데이터 제거
ms_df = sm_df.drop(user)

# 유사도가 높은 순으로 sorting
ms_df = ms_df.sort_values(user, ascending=False)

# 위의 설정 대로 컨텐츠를 추천할 사용자와 유사도가 높은 사용자 필터링
ms_df = ms_df[:closer_count]
```

ms_df

---

	user_1	user_2	user_3	user_4
user_2	0.729397	1.000000	0.483046	0.443039
user_4	0.529813	0.770054	0.821210	1.000000

user_1과 유사도가 높은 상위 2명 user에 대한 데이터 입니다.

```
# use_1과 가까운 상위 2개 데이터
sample_df.loc[ms_df.index]
```

---

	article_1	article_2	article_3	article_4	article_5
user_2	2	0	0	1	4
user_4	4	0	4	5	0

위의 user 데이터에 대한 컨텐츠별 평균을 구해줍니다.

```
mean = np.zeros(len(sample_df.columns))
for ms_user, sms_value in ms_df[user].items():
    mean += sample_df.loc[ms_user]
mean /= len(ms_df[user])

pred_df = pd.DataFrame(columns=sample_df.columns)
pred_df.loc["user"] = sample_df.loc[user]
pred_df.loc["mean"] = mean
pred_df
```

---

	article_1	article_2	article_3	article_4	article_5
user	5	3	0	0	2
mean	3	0	2	3	2

위에 학습한 내용으로 평균 스코어 행렬 함수를 작성합니다.

```
# 유사도가 높은 user에 대한 평균값 구하는 함수
def mean_score(sample_df, sm_df, target, closer_count):

    # 유사도 행렬에서 추천 user와 가까운 user의 유사도 데이터 프레임
    ms_df = sm_df.drop(target)
    ms_df = ms_df.sort_values(target, ascending=False)
    ms_df = ms_df[target][:closer_count]

    # 유사도가 높은 user를 나타내는 데이터 프레임
    ms_df = sample_df.loc[ms_df.index]

    # 결과 데이터 프레임 생성
    pred_df = pd.DataFrame(columns=sample_df.columns)
    pred_df.loc["user"] = sample_df.loc[target]
    pred_df.loc["mean"] = ms_df.mean()

    return pred_df
```

결과 데이터

```
# 결과 데이터 - sample_df : sample dataframe, sm_df : similarity matrix dataframe
target, closer_count = "user_1", 2
pred_df = mean_score(sample_df, sm_df, target, closer_count)
pred_df
```

---

	article_1	article_2	article_3	article_4	article_5
user	5	3	0	0	2
mean	3	0	2	3	2

(4) Recommend - 추천

위에서 구한 평균값 행렬 데이터 프레임으로 추천 기사를 데이터 프레임으로 확인하고 추천순으로 정렬합니다.

```
# user가 읽지 않은 기사를 순서대로 나열 추천 기사 정렬 및 출력
recommand_df = pred_df.T
recommand_df = recommand_df[recommand_df["user"] == 0]
recommand_df = recommand_df.sort_values("mean", ascending=False)
print(list(recommand_df.index))
recommand_df
```

```
['article_4', 'article_3']
```

	user	mean
article_4	0	3
article_3	0	2

aritcle_1, aritcle_2, aritcle_5는 이미 읽었기 때문에 추천에서 제외하고 읽지 않은 aritcle_3, aritcle_4 중에 aritcle_4를 추천합니다.

(5) Performance Evaluation - 성능 측정

아래의 데이터에서 user는 실제 값에 대한 vector, mean은 예측 값에 대한 vector입니다. 이 데이터로 오차를 구해 모델을 성능을 평가할수 있습니다. 모델의 성능을 측정한 결과로 이전 모델에서 성능을 비교하며 추천 시스템의 성능을 증가시키는 기준 지표로 사용 될수 있습니다.

	article_1	article_2	article_3	article_4	article_5
user	5	3	0	0	2
mean	3	0	2	3	2

MSE - Mean Squrared Error

$$MSE = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}$$ (y_i : 실제 값, $\hat{y}_i$: 예측된 값)

```
def mse(value, pred):
    # user 데이터에서 0인 데이터 제거
    idx = value.nonzero()[0]
    value, pred = np.array(value)[idx], np.array(pred)[idx]

    # 수식 계산후 결과 리턴
    return sum((value - pred)**2) / len(idx)

mse(pred_df.loc["user"], pred_df.loc["mean"])
```

```
4.333333333333333
```

```
# 전체 user에 대한 평가
def evaluate(df, sm_df, closer_count, algorithm):

    # user 리스트
    users = df.index
    evaluate_list = []

    # 모든 user에 대해서 mae 값을 구함
    for target in users:
        result_df = mean_score(df, sm_df, target, closer_count)
        evaluate_list.append(algorithm(result_df.loc["user"], result_df.loc["mean"]))

    # 모든 user의 mae값의 평균을 리턴
    return np.average(evaluate_list)

evaluate(sample_df, sm_df, 2, mse)
```

4.5

RMSE - Root Mean Squrared Error

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$ (y_i : 실제 값, $\hat{y}_i$: 예측된 값)

```
def rmse(value, pred):

    # user 데이터에서 0인 데이터 제거
    idx = value.nonzero()[0]
    value, pred = np.array(value)[idx], np.array(pred)[idx]
```

```
    # 수식 계산후 결과 리턴
    return np.sqrt(sum((value - pred)**2) / len(idx))

rmse(pred_df.loc["user"], pred_df.loc["mean"])
```

```
2.0816659994661326
```

```
evaluate(sample_df, sm_df, closer_count, rmse)
```

```
2.067791827548017
```

MAE - Mean Absolute Error

$e_i = y_i - \hat{y}_i$ (y_i : 실제 값, $\hat{y}_i$: 예측된 값)

$$MAE = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

```
# 한명의 user에 대한 MAE 값
def mae(value, pred):

    # user 데이터에서 0인 데이터 제거
    idx = value.nonzero()[0]
    value, pred = np.array(value)[idx], np.array(pred)[idx]

    # 수식 계산후 결과 리턴
    return np.absolute(sum(value - pred)) / len(idx)

mae(pred_df.loc["user"], pred_df.loc["mean"])
```

```
1.6666666666666667
```

---

evaluate(df, sm_df, closer_count, mae)

```
1.1666666666666667
```

(6) Summary

1. Sample Dataset

	article_1	article_2	article_3	article_4	article_5
user_1	5	3	0	0	2
user_2	2	0	0	1	4
user_3	0	0	4	3	1
user_4	4	0	4	5	0

2. Similarity Matrix

	user_1	user_2	user_3	user_4
user_1	1.000000	0.652929	0.324443	0.811107
user_2	0.729397	1.000000	0.483046	0.443039
user_3	0.196116	0.332956	1.000000	0.949474
user_4	0.529813	0.770054	0.821210	1.000000

3. Closer User Dataset (for user_1)

	article_1	article_2	article_3	article_4	article_5
user_2	2	0	0	1	4
user_4	4	0	4	5	0

4. Prediction Dataset (for user_1)

	article_1	article_2	article_3	article_4	article_5
user	5	3	0	0	2
mean	3	0	2	3	2

5. Performance Evaluation (MAE)

user - mean = [5, 3, 2] - [3, 0, 2] = [2, 3, 0]

sum(user - mean) = sum([2, 3, 0]) = 5

sum(user - mean) / len(user(not 0)) = 5 / 3 = 1.6666...