



UiT The Arctic University of Norway

DTE-2501 AI Methods and Applications

Using the Pearson Correlation for Collaborative Filtering

Andreas Dyrøy Jansson
PhD Candidate
C3190
andreas.d.jansson@uit.no

In this lecture, we will go through an example where we calculate the Pearson correlation between two users and use this to make a simple recommendation engine.

The basics of the Pearson correlation

- Pearson correlation coefficient
 - Tells us something about the relationship between two datasets

$$w(a, i) = \frac{\sum_j (v_{a,j} - \bar{v}_a)(v_{i,j} - \bar{v}_i)}{\sqrt{\sum_j (v_{a,j} - \bar{v}_a)^2 \sum_j (v_{i,j} - \bar{v}_i)^2}}$$

- Gives a number between 1 and -1
 - 1 means we have a strong correlation
 - 0 means no correlation
 - -1 means a strong inverse correlation
- Takes the variations ($v_{i,j}$ - average v) as "argument"

Pearson correlation is a number between -1 and 1 which tells us if there is a correlation between two sets of data. 1 means that there is a strong correlation, 0 means no correlation and -1 is a strong inverse correlation. This means that we can use this to see if there is some sort of relationship between the users, which in turn can be used as a basis for a recommendation system.

An example: product reviews

Index (j)	Product	User 1	User 2	User 3	User 4	User 5	User 6	User 7
0	Crazy Cola	1		3	4	1	4	5
1	Zombo Soda	1	3	4	5	3	3	5
2	Ol Geezer's				2	3	3	
3	Dr. Hiccups			4	5	2	4	3
4	Fantom	2		5	3	2	5	4
5	Sprizz	1	5	5	2	3	2	
6	Chimpus			1	2	4	3	
7	TopChug				3	1	3	

- These are the voting vectors (profiles) of each user

Let's say that we collect some product review data for a set of users. Each user has voted on the products and given them a score from 1 to 5, where 1 is bad and 5 is good. We call this the voting vectors, or profiles, of each user. Notice that not all users have voted on every product.

Recommendation system

- Consider that we get a new user
 - We will refer to them as *a* (active user)
- We observe their votes as such:
- Problem: How can we predict their opinion on "Dr. Hiccups"?

Product	Active user
Crazy Cola	3
Zombo Soda	2
Ol Geezer's	3
Dr. Hiccups	
Fantom	4
Sprizz	1
Chimpus	
TopChug	5

Let's look at a new user with their own voting profile. Let's imagine that we are doing market research for a company. How can we predict what this user's opinion will be on the product Dr. Hiccups? Recall the lecture about the wisdom of crowds. If we can somehow find a way to group users together based on their profiles, we can use collaborative filtering to predict a new user's vote based on the history of similar users. Going forward, we will refer to this new user as active user, or simply *a*.

Memory-based algorithms for Collaborative Filtering (Breese et al, UAI98):

- Predicted vote for “active user” a is weighted sum

$$p_{a,j} = \bar{v}_a + \kappa \sum_{i=1}^n \underbrace{w(a,i)}_{\text{Pearson correlation coefficient of } a \text{ and } i} (v_{i,j} - \bar{v}_i)$$

normalizer

- $v_{i,j}$ = vote of user i on item j (product)
- I_i = items for which user i has voted
- Mean vote for i is

$$\bar{v}_i = \frac{1}{|I_i|} \sum_{j \in I_i} v_{i,j}$$

We can predict the vote for a product for the active user by using a weighted sum. What we have here is that the predicted vote for product j by the active user a is equal to the average vote of the a plus a normalizer multiplied by the sum of all correlations between the active user and the other users multiplied by the variation for product j by each user.

This might seem a bit overwhelming, but we will step through each part of this formula in the next few slides, along with some code examples.

Step 1 – find the mean votes of each user

- Variation: $(v_{i,j} - \bar{v}_i)$
- We start by finding the mean vote for each person using the formula:

$$\bar{v}_i = \frac{1}{|I_i|} \sum_{j \in I_i} v_{i,j}$$

- The formula decoded (User 1):
 - $(1 + 1 + 2 + 1) / 4 = 1,25$

The first step is to find the average vote of each user. Let's look at user 1 for example. Their votes are 1, 1, 2, and 1. We simply sum them together and divide by 4 to get the mean vote. If we do the same for the rest of our users, we get the following table.

Step 1 – find the mean votes of each user

- Variation: $(v_{i,j} - \bar{v}_i)$
- We start by finding the mean vote for each person using the formula:

$$\bar{v}_i = \frac{1}{|I_i|} \sum_{j \in I_i} v_{i,j}$$

- The formula decoded (User 1):
 - $(1 + 1 + 2 + 1) / 4 = 1,25$

User 1	User 2	User 3	User 4	User 5	User 6	User 7	Active user
1,25	4	3,666667	3,25	2,375	3,375	4,25	3

Step 2 – find the variation for each vote

- The variation is the difference between each vote and the user's average

$$(v_{i,j} - \bar{v}_i)$$

- We take the product vote and subtract the mean vote
 - Example - User 1:

Next, we find the variance using this formula. This is simply the vote for product j minus the average vote for the user. If we look at user 1 again, computing the variance gives us the following table.

Step 2 – find the variation for each vote

- The variation is the difference between each vote and the user's average

$$(v_{i,j} - \bar{v}_i)$$

- We take the product vote and subtract the mean vote
 - Example - User 1:

Product	User 1
Crazy Cola	$1 - 1,25 = -0,25$
Zombo Soda	$1 - 1,25 = -0,25$
Ol Geezer's	-
Dr. Hiccups	-
Fantom	$2 - 1,25 = 0,75$
Sprizz	$1 - 1,25 = -0,25$
Chimpus	-
TopChug	-

Step 3 – Compute the correlation matrix

- Using *corrcoef* from Numpy
- Example: find the correlation matrix for two users
- Ignore missing values

Now we move on to the Pearson correlation. Since we are using Python, we can use a function from Numpy to compute the correlation matrix between our users. For this first example, we will simply look at how to compute the correlation between two of the users. In order to use the Numpy function, each variance vector must be the same size. We see that we have some missing values. We ignore these products, and end up with the following table:

Step 3 – Compute the correlation matrix

- Using *corrcoef* from Numpy
- Example: find the correlation matrix for two users
- Ignore missing values

Product	User 1	Active user
Crazy Cola	-0,25	0
Zombo Soda	-0,25	-1
Fantom	0,75	1
Sprizz	-0,25	-2

Now we move on to the Pearson correlation. Since we are using Python, we can use a function from Numpy to compute the correlation matrix between our users. For this first example, we will simply look at how to compute the correlation between two of the users. In order to use the Numpy function, each variance vector must be the same size. We see that we have some missing values. We ignore these products, and end up with the following table:

Step 3 – Compute the correlation matrix

- Using Python (Numpy):
 - Put the variances for all users into arrays:

```
import numpy as np
###
user_1_variance = np.array([-0.25, -0.25, 0.75, -0.25])
active_user_variance = np.array([0, -1, 1, -2])

corr_matrix = np.corrcoef(user_1_variance, active_user_variance)
```

Product	User 1	Active user
Crazy Cola	-0,25	0
Zombo Soda	-0,25	-1
Fantom	0,75	1
Sprizz	-0,25	-2

We put the values from the table into two Numpy-arrays like so. We call the function, and get the correlation matrix.

Step 3 – Compute the correlation matrix

- Using Python (Numpy):

- Put the variances for all users into arrays:

```
import numpy as np
###
user_1_variance = np.array([-0.25, -0.25, 0.75, -0.25])
active_user_variance = np.array([0, -1, 1, -2])

corr_matrix = np.corrcoef(user_1_variance, active_user_variance)
```

Product	User 1	Active user
Crazy Cola	-0,25	0
Zombo Soda	-0,25	-1
Fantom	0,75	1
Sprizz	-0,25	-2

- This gives us the correlation matrix:
- About the correlation matrix
 - Symmetric
 - Diagonal 1's is the self-correlation
 - The other values is the actual correlation value with the other user
 - In this case, the value is 0.77, which indicates a strong correlation between these users
 - We keep the value from [0][1]

```
[[1. 0.77459667]
 [0.77459667 1.  ]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

On the diagonal, we have User 1 correlated with User 1 and Active user correlated with Active user. We see that these values are 1, which is expected, since this is the self-correlation. This is not interesting to us however, so we keep the values in [0][1] = 0.77459667, which is the actual correlation between these two users. This is close to 1, which tells us that there is a somewhat strong correlation between these two users.

Step 4.5 – set up the vectors for all users

- We must compute the correlation r between Active User a and all the other users

```
# pad the vectors with a token so they are of equal length
user_1_variance = [-0.25, -0.25, 'x', 'x', 0.75, -0.25, 'x', 'x']
# other user vectors go here...

active_user_variance = [0, -1, 0, 'x', 1, -2, 'x', 2]

# put all the users' vectors in an array so we can loop over them
all_user_variances = [user_1_variance, user_2_variance, user_3_variance, \
user_4_variance, user_5_variance, user_6_variance, user_7_variance]
```

We must do this for every user 1 to 7 and the active user. To make things easier, we will use arrays and loops. Since most of the users' vectors are incomplete, we will pad them with a placeholder value x so that they all are the same length. When we compute the correlation in the next step, we check if this value is present in the vector. If it is, we simply skip that product. We put the vectors of user 1 to 7 in an array so we can loop over them in the next step.

Step 5 - Set up the for-loops to compute the correlations

```
#we also need to save the results for later. we keep them in this array
correlations = []

# set up vectors of equal size. we ignore the products that have not been voted on, i.e where the variance is 'x'
for other_user_variance in all_user_variances:
    # we create two temporary arrays to keep our variances
    temp_active = []
    temp_other = []
    # we then loop over all the values in both users' vectors
    for i in range(len(active_user_variance)):
        # we can only use the values for products that the user has voted on. We therefore ignore the x's
        if active_user_variance[i] != 'x' and other_user_variance[i] != 'x':
            # we put the values that are not 'x' in our temp arrays
            temp_active.append(active_user_variance[i])
            temp_other.append(other_user_variance[i])

    # now that we have set up vectors of equal length, we can compute the correlation using numpy
    correlation_matrix = np.corrcoef(np.array(temp_active), np.array(temp_other))
    correlation = correlation_matrix[0][1]
    # keep the result in an array
    correlations.append(correlation)
```

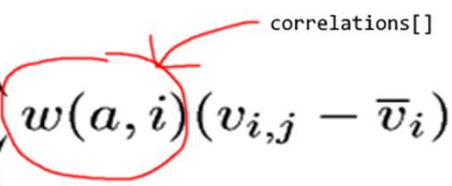
In the first for-loop, we iterate over all the users' variance arrays. We let the code do the work of creating the actual arrays we use for computing the correlations. Using two temporary arrays, we only keep the variances for the products that are not x for both of the users. When we have done this for all variances of one user and the active user, we pass these arrays to the Pearson correlation function of Numpy, like we saw in the first code example. The only difference this time is that the arrays are created dynamically, and their lengths change based

on the variances we ignore for each user.

Remember that we are not interested in the self-correlation, so we only keep the value from $[0][1]$ in the matrix. We store this value in an array called correlations.

Step 6 – Compute the predicted vote

- Recall the formula:
 - $p_{a,j}$ is the predicted vote

$$p_{a,j} = \bar{v}_a + \kappa \sum_{i=1}^n w(a,i) (v_{i,j} - \bar{v}_i)$$


- Simplified:
 - $p_{a,j} = \text{average_vote_for_a} + \text{kappa} * (\text{sum_correlation_variance})$
 - We set $K = 1$

After we have computed all the correlations, we can take a step back and take another look at the formula for predicting the vote. As before, the predicted vote for product j by the active user is equal to the average vote of the active user + kappa multiplied by the sum of all correlations between active user and the other users multiplied by the user's variance for product j . We'll set kappa to 1.

Sum correlation * variance

- Finally, we can compute the predicted vote for the product:
 - Remember, we skip the products that users have not voted on (x)

```
kappa = 1
average_vote_for_a = 3
sum_correlation_variance = 0
j_index = 3 # the index of the product we want to predict the vote for (Dr.Hiccups)
for i in range(len(all_user_variances)):
    if all_user_variances[i][j_index] != 'x':
        sum_correlation_variance += all_user_variances[i][j_index] * correlations[i]

p_a_j = average_vote_for_a + kappa * (sum_correlation_variance)
```

- We find that predicted vote for j=3 is 4.6
 - Conclusion: Active user is likely to give this product a high score

This is easier to show in code. If we write the formula like so, we can compute the sum using a for-loop. Kappa is 1, the average vote for active user we found earlier, and the product we want to predict the vote for has index 3 in the list. We then simply multiply and add together all the correlations and variances that are not x, and put it into the formula like so. The predicted vote for Active User thus becomes their average vote plus 1 multiplied by the sum of correlation variances. When running this calculation on the data presented in the start, we get 4.6 as result for the product Dr.Hiccups. We can

therefore conclude that the user is likely to give this product a high score, probably a 4 or a 5.