2023-04-05

Πανεπιστήμιο Πειραιώς

Προηγμένα Συστήματα Πληροφορικής – Ανάπτυξη Λογισμικού και Τεχνητής Νοημοσύνης

Τελική Εργασία για το Μάθημα

Αναγνώριση Προτύπων και Μηχανική Μάθηση

των

Αποστόλου Αθανάσιου (mpsp2203@unipi.gr)

Μπιρμπάκου Γεώργιου (mpsp2220@unipi.gr)

Ευαγγέλου Αλέξανδρου (mpsp2210@unipi.gr)

Θέμα: Αλγόριθμοι Σύστασης με Χρήση Τεχνητών Νευρωνικών Δικτύων και Τεχνικών Ομαδοποίησης Δεδομένων

Εισαγωγή

Στα πλαίσια της τελικής εργασίας του μαθήματος “Αναγνώριση Προτύπων και Μηχανική Μάθηση” αναπτύξαμε κώδικα όπου διαβάζει ένα αρχικό Dataset (το οποίο βρίσκεται στην διεύθυνση https://www.dropbox.com/s/6bcfscyexabgy0w/Dataset.npy?dl=0), αναλύει τα δεδομένα του και εξάγει τα επιθυμητά αποτελέσματα.

Ο κώδικας έχει γραφεί σε γλώσσα Python ως ένα jupyter notebook και βρίσκεται στο αρχείο “**IMDB.ipynb**”.

Ο κώδικας και αυτή η αναφορά βρίσκονται επίσης στο git repository
https://github.com/ThanosApostolou/aics-pattern-recognition

Προ-επεξεργασία Δεδομένων

1. Να βρείτε το σύνολο των μοναδικών χρηστών U και το σύνολο των μοναδικών αντικειμένων I.

Αρχικά ξεκινάμε φορτώνοντας τα δεδομένα που έχουμε σε ένα numpy array.
Ελέγχοντας το αρχικό dataset που μας δίνεται, βλέπουμε ότι είναι της ακόλουθης μορφής:

```
[['ur4592644' 'tt0120884' '10' '16 January 2005']
 ['ur3174947' 'tt0118688' '3' '16 January 2005']
 ['ur3780035' 'tt0387887' '8' '16 January 2005']
 ...
 ['ur4592639' 'tt0107423' '9' '16 January 2005']
 ['ur4581944' 'tt0102614' '8' '16 January 2005']
 ['ur1162550' 'tt0325596' '7' '16 January 2005']]
```

Όπου:

- urxxxxxx ένα πρόθεμα ur ακολουθούμενο από το ID κάθε χρήστη
- ttxxxxxx ένα πρόθεμα tt ακολουθούμενο από το ID κάθε ταινίας
- Η αξιολόγηση της ταινίας από το χρήστη
- Η ημερομηνία της αξιολόγησης

Επιθυμούμε στη συνέχεια να δημιουργήσουμε ένα καινούριο Pandas Dataframe από τα δεδομένα που μας δίνονται το οποίο θα έχει τη μορφή:

User	Movie	Rating	Date

```python
def calculate_ratings_matrix_df_W_CommonRatings():
    dataset: np.ndarray = np.load(DATASET_FILE_PATH)
    display('dataset.size', dataset.size)

    #### CELL 1
    #Define the splitter lambda function in order to tokenize the initial string data.
    splitter = lambda s: s.split(",")
    #Apply the splitter lambda function on the string np array
    dataset = np.array([splitter(x) for x in dataset])
    #Set the pickle file for storing the initial dataframe
    pickle_file = os.path.join(DATAFOLDER_PATH, "dataframe.pkl")
    #Check the existence of the specified file.
    if os.path.exists(pickle_file):
        #Load the pickle file
        dataframe = pd.read_pickle(pickle_file)
        print(dataframe)
    else:
        #Create the dataframe object.
        dataframe = pd.DataFrame(dataset, columns=['User','Movie','Rating','Date'])
        #Convert the string elements of the "Users" series into integers
        dataframe["User"] = dataframe["User"].apply(lambda s:np.int64(s.replace("ur","")))
        #Convert the string elements of the "Movies" series into integers
        dataframe["Movie"] = dataframe["Movie"].apply(lambda s:np.int64(s.replace("tt","")))
        #Convert the string elements of the "Ratings" series into integers
        dataframe["Rating"] = dataframe["Rating"].apply(lambda s:np.int64(s))
        #Convert the string element of "Dates" series into datetime Object
        dataframe["Date"] = pd.to_datetime(dataframe["Date"])
        dataframe.to_pickle(pickle_file)
```

Το αποτέλεσμα μετά τη δημιουργία του dataframe μας είναι το ακόλουθο:

```
             User    Movie  Rating       Date
0         4592644   120884      10 2005-01-16
1         3174947   118688       3 2005-01-16
2         3780035   387887       8 2005-01-16
3         4592628   346491       1 2005-01-16
4         3174947    94721       8 2005-01-16
...           ...      ...     ...        ...
4669815    581842   107977       6 2005-01-16
4669816   3174947   103776       8 2005-01-16
4669817   4592639   107423       9 2005-01-16
4669818   4581944   102614       8 2005-01-16
4669819   1162550   325596       7 2005-01-16

[4669820 rows x 4 columns]
```

Από αυτό το dataframe βρίσκουμε το σύνολο των μοναδικών χρηστών U και το σύνολο των μοναδικών ταινιών I:

```
#### CELL 2
#Get the unique users in the dataset.
users = dataframe["User"].unique()
#Get the number of unique users
users_num = len(users)
#Get the unique movie items in the dataset.
movies = dataframe["Movie"].unique()
#Get the number of unique movies
movies_num = len(movies)
#Get the total number of existing ratings.
ratings_num = dataframe.shape[0]
#Report the number of unique Users and Movies in the dataset
display("INITIAL DATASET: {0} number of unique users and {1} of unique movies".format(users_num, movies_num))
#Report the total number of existing ratings in the dataset
display("INITIAL DATASET: {} total number of existing ratings".format(ratings_num))
```

Όπου λαμβάνουμε τα αποτελέσματα:

INITIAL DATASET: 1499238 number of unique users and 351109 of unique movies

INITIAL DATASET: 4669820 total number of existing ratings

2. Θεωρείστε την συνάρτηση $\Phi: U \to P(I)$ (όπου $P(I)$ το δυναμοσύνολο του I) η οποία $\forall u \in U$ επιστρέφει το σύνολο $\varphi(u) \subset I$ των αντικειμένων που αξιολογήθηκαν από τον χρήστη u. Μπορούμε να γράψουμε, επομένως, ότι $\varphi(u) = \{i \in I : R(u,i) > 0\}$. Να περιορίσετε τα σύνολα των μοναδικών χρηστών U και μοναδικών αντικειμένων I στα αντίστοιχα σύνολα $\hat{U}$ και $\hat{I}$ έτσι ώστε $\forall u \in \hat{U}, R_{min} \leq \vee \varphi(u) \vee \leq R_{max}$ όπου R_{min} και R_{max} ο ελάχιστος απαιτούμενος και ο μέγιστος επιτρεπτός αριθμός αξιολογήσεων ανά χρήστη. Θεωρήστε προφανώς ότι $|\hat{U}| = n < N$ και $|\hat{I}| = m < M$.

Θέλουμε αρχικά να δημιουργήσουμε τη συνάρτηση Φ η οποία για κάθε χρήστη επιστρέφει το σύνολο των αντικειμένων I που αυτός έχει αξιολογήσει.

```
# CELL 3
#Define the pickle file that will store the number of ratings per user dataframe
pickle_file = os.path.join(DATAFOLDER_PATH, "ratings_num_df.pkl")
#Check the existence of the previously defined pickle file
if os.path.exists(pickle_file):
    #Load the pickle file
    ratings_num_df = pd.read_pickle(pickle_file)
else:
    ratings_num_df = dataframe.groupby("User")["Rating"].count().sort_values(ascending=False).reset_index(name="ratings_num")
    #Save the previously created dataframe to pickle
    ratings_num_df.to_pickle(pickle_file)
```

Τα αποτελέσματα αυτής της συνάρτησης είναι τα ακόλουθα:

	User	ratings_num
0	2467618	24145
1	20552756	16817
2	2483625	16715
3	482513	13213
4	2898520	12677
...	...	...
1499233	23252608	1
1499234	23252585	1
1499235	23252584	1
1499236	23252533	1
1499237	126872318	1

Τα σύνολα αυτά τα αποθηκεύουμε σε ένα pickle file.

Στη συνέχεια, δημιουργούμε το dataframe που περιέχει το χρονικό εύρος των αξιολογήσεων κάθε χρήστη σε ημέρες (θα χρησιμοποιηθεί για να δημιουργήσουμε το ιστόγραμμα συχνοτήτων που απαιτείται σε επόμενο ερώτημα).

```
# CELL 4
#Set the pickle file that will store the time span per user dataframe
pickle_file = os.path.join(DATAFOLDER_PATH, "ratings_span_df.pkl")
if os.path.exists(pickle_file):
    ratings_span_df = pd.read_pickle(pickle_file)
else:
    ratings_span_df = dataframe.groupby("User")["Date"].apply(lambda date: max(date)-min(date)).sort_values(ascending=False).reset_index(name="ratings_span")
    ratings_span_df.to_pickle(pickle_file)
```

Χρησιμοποιώντας τώρα, αυτά τα δύο dataframe, δημιουργούμε ένα καινούριο, το οποίο περιέχει τόσο τον αριθμό των αξιολογήσεων κάθε χρήστη αλλά και το χρονικό εύρος των αξιολογήσεών του.

Περιορίζουμε τώρα το συγκεκριμένο dataframe έτσι ώστε να περιέχει μόνο τους χρήστες των οποίων ο αριθμός των αξιολογήσεων είναι μεταξύ των 150 και 300.

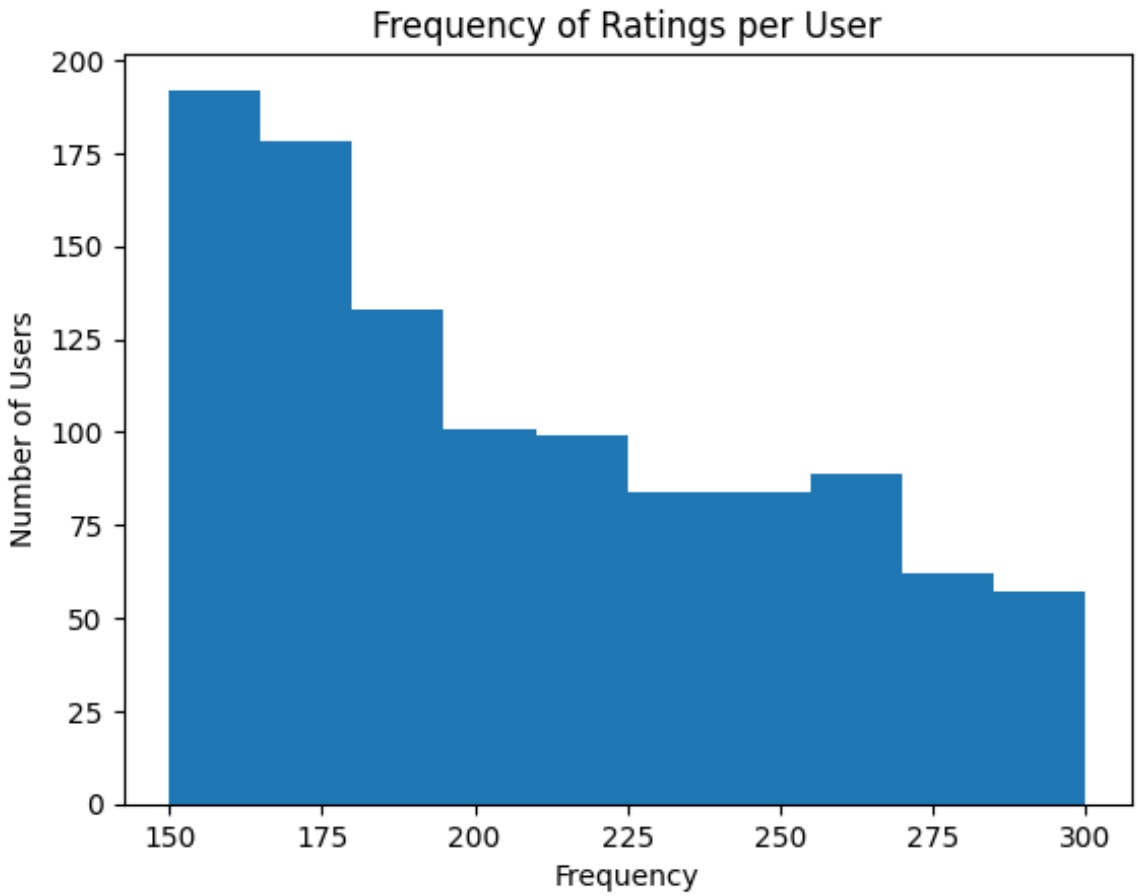
```
#Create a new ratings dataframe by joining the previously defined dataframe
ratings_df = ratings_num_df.join(ratings_span_df.set_index("User"),on="User")
ratings_df["ratings_span"]=ratings_df["ratings_span"].dt.days
#Set the threshold values for the minimum and maximum number of Ratings per user
minimum_ratings = 150
maximum_ratings = 300
#Discard all users that do not pertain to the previous range of ratings
reduced_ratings_df = ratings_df.loc[(ratings_df["ratings_num"] >= minimum_ratings) & (ratings_df["ratings_num"] <= maximum_ratings)]
```

3. **Να δημιουργήσετε και να αναπαραστήσετε γραφικά τα ιστογράμματα συχνοτήτων για το πλήθος αλλά και για το χρονικό εύρος των αξιολογήσεων του κάθε χρήστη.**

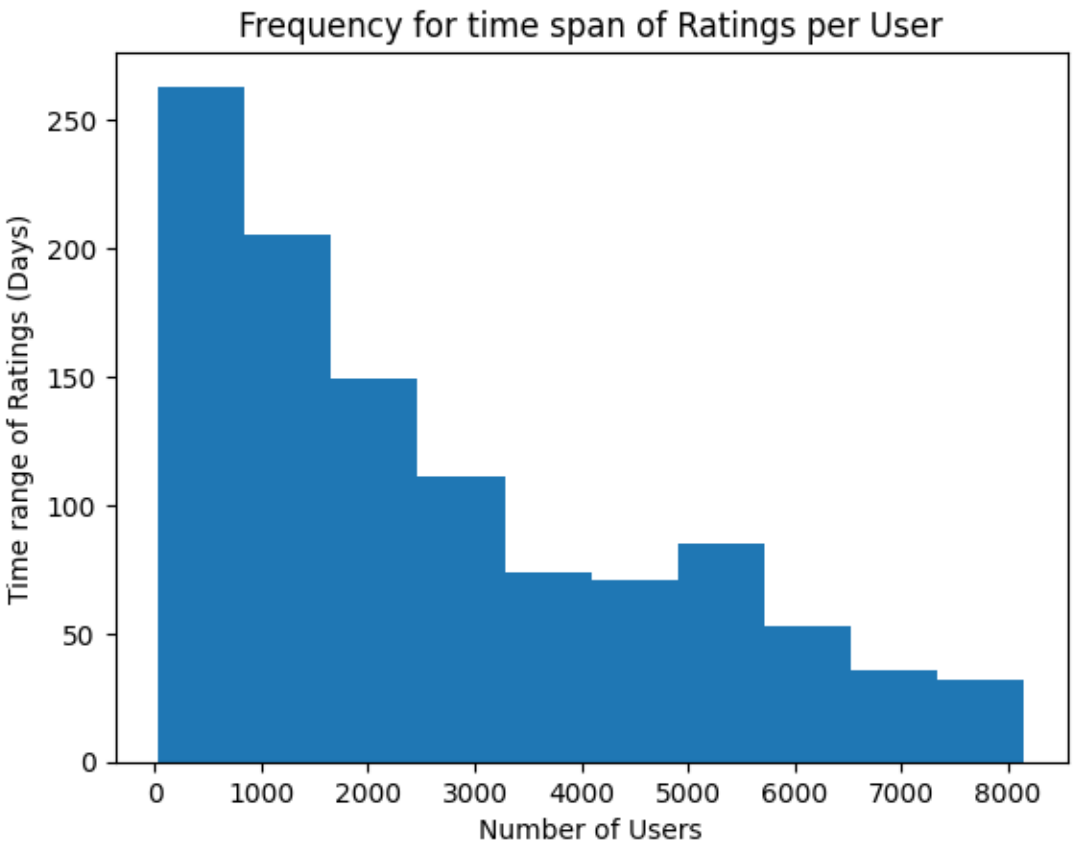
Από τα dataframe που δημιουργήσαμε στο προηγούμενο ερώτημα, δημιουργούμε τα ιστογράμματα των συχνοτήτων για το πλήθος και το χρονικό εύρος των αξιολογήσεων κάθε χρήστη χρησιμοποιώντας τη βιβλιοθήκη matplotlib.

```
#Generate the frequency histogram for the number of ratings per user
reduced_ratings_df["ratings_num"].plot(kind='hist', title='Frequency of Ratings per User', xticks=range(minimum_ratings, maximum_ratings+1, 25))
plt.xlabel('Frequency')
plt.ylabel('Number of Users')

plt.show()

#Generate the frequency histogram for the time span of ratings per user
reduced_ratings_df["ratings_span"].plot(kind='hist', title='Frequency for time span of Ratings per User')
plt.xlabel('Number of Users')
plt.ylabel('Time range of Ratings (Days)')

plt.show()
```

Ιστόγραμμα συχνοτήτων για το πλήθος των αξιολογήσεων:



Ιστόγραμμα συχνοτήτων για το χρονικό εύρος μεταξύ των αξιολογήσεων κάθε χρήστη:



4. Δημιουργήστε μια εναλλακτική αναπαράσταση του συνόλου των δεδομένων ως ένα σύνολο διανυσμάτων προτιμήσεων $R=\{R_1, R_2, \cdots, R_n\}$ **με** $R_j=R(u_j)\in R_o^m, \forall\, j\in[n]$**. Συγκεκριμένα, μπορούμε να γράψουμε ότι:**

$$R_j(k)=\begin{cases} R(u_j,i_k), R(u_j,i_k)>0; \\ 0, R(u_j,i_k)>0. \end{cases} \quad (1)$$

```
#### CELL 5
#Get the final dataframe by excluding all users whose ratings fall outside the prespecified range
final_df = dataframe.loc[dataframe["User"].isin(reduced_ratings_df["User"])].reset_index()
#Drop the links (indices) to the original table
final_df = final_df.drop("index", axis=1)
#Get the unique users and items in the final dataframe along with the final number of ratings
final_users = final_df["User"].unique()
final_movies = final_df["Movie"].unique()
final_users_num = len(final_users)
final_movies_num = len(final_movies)
final_ratings_num = len(final_df)
```

Αφού περιορίσουμε το dataframe στους χρήστες που ορίσαμε στο προηγούμενο ερώτημα, θέλουμε να μετασχηματίσουμε το dataframe έτσι ώστε οι γραμμές του να αναπαριστούν τους χρήστες, οι στήλες τις ταινίες και τα στοιχεία του dataframe να είναι η αξιολόγηση του χρήστη στην κάθε ταινία. Εάν ο χρήστης δεν έχει αξιολογήσει κάποια ταινία, το στοιχείο θα είναι μηδέν.

```
# Create a pivot table of user-movie ratings
ratings_matrix_df = final_df.pivot_table(index='User', columns='Movie', values='Rating')
ratings_matrix_df = ratings_matrix_df.fillna(0)
```

Αλγόριθμοι Ομαδοποίησης Δεδομένων

1. Να οργανώσετε το περιορισμένο σύνολο των χρηστών $\hat{U}$ σε L συστάδες (clusters) της μορφής $\hat{U}=U_{G_1}\cup U_{G_2}\cup\cdots\cup U_{G_L}$ έτσι ώστε $U_{G_a}\cap U_{G_b}\neq\varnothing,\forall\, a\neq b$ βασιζόμενοι στην διανυσματική αναπαράσταση των προτιμήσεών τους μέσω του συνόλου R. Η διαδικασία ομαδοποίησης των διανυσμάτων προτιμήσεων των χρηστών $R_j=R(u_j)\in R_o{}^m,\forall\, j\in[n]$ επάνω στο περιορισμένο σύνολο των αντικειμένων $\hat{I}$, μπορεί να πραγματοποιηθεί με κατάλληλη παραμετροποίηση του αλγορίθμου k-means μεταβάλλοντας την μετρική που αποτιμά την απόσταση μεταξύ δύο διανυσμάτων προτιμήσεων για ένα ζεύγος χρηστών u και v ως $dist(R_u,R_v)$. Συγκεκριμένα, μπορείτε να χρησιμοποιήσετε τις παρακάτω μετρικές (*):

a. $dist_{euclidean}(R_u,R_v)=\sqrt{\sum_{k=1}^{n}|R_u(k)-R_v(k)|^2\,\lambda_u(k)\,\lambda_v(k)}\,(2)$ έτσι ώστε:

$$\lambda_j(k)=\begin{cases}1, R(u_j,i_k)>0;\\ 0, R(u_j,i_k)>0.\end{cases}\,(3)$$

Η συνάρτηση $\lambda_j()$ αποτιμά το αν ο χρήστης u_j έχει αξιολογήσει ή όχι το αντικείμενο i_k. Προφανώς, μέσω της συγκεκριμένης συνάρτηση μπορούμε αν υπολογίσουμε την τιμή του ¿$\varphi(u)\vee$¿ που αποτιμά το πλήθος των αξιολογήσεων που παρείχε συνολικά ο χρήστης u ως: $|\varphi(u)|=\sum_{k=1}^{n}\lambda_u(k)$.

Αρχικά δημιουργούμε τον πίνακα των «βαρών» που περιέχει τις τιμές του λ για τον κάθε χρήστη. Η τιμή των στοιχείων σε αυτόν τον πίνακα θα είναι 1 εάν ο χρήστης έχει αξιολογήσει την ταινία και 0 εάν δεν υπάρχει αξιολόγηση.

```
# Threshold
threshold = 1

# Transform to binary
binary_matrix = np.where(ratings_matrix_df >= threshold, 1, 0)
display('binary_matrix', binary_matrix)
```

Θέλουμε τώρα να κατασκευάσουμε τον πίνακα των Ευκλείδειων αποστάσεων μεταξύ των αξιολογήσεων των χρηστών χρησιμοποιώντας την ανωτέρω μετρική:

```
# Calculate the pairwise weighted Euclidean distance matrix

def create_euclidean_distance_matrix_cached(ratings_matrix: pd.DataFrame, binary_matrix: np_typing.NDArray) -> np_typing.NDArray[np.float64]:
    #Set the npy file that will store the Euclidean distance matrix
    npy_file = os.path.join(DATAFOLDER_PATH, "euclidean_distance_matrix.npy")
    if os.path.exists(npy_file):
        Dist_euclidean: np_typing.NDArray[np.float64] = np.load(npy_file, allow_pickle=True)
        return Dist_euclidean
    else:
        n = ratings_matrix.shape[0]
        Dist_euclidean = np.zeros((n, n))
        for i in range(n):
            for j in range(i, n):
                d = np.sqrt(np.sum(binary_matrix[i,:]*binary_matrix[j,:] * (ratings_matrix.iloc[i,:] - ratings_matrix.iloc[j,:])**2))
                Dist_euclidean[i,j] = d
                Dist_euclidean[j,i] = d
        np.save(npy_file, Dist_euclidean, allow_pickle=True, fix_imports=True)
        return Dist_euclidean


Dist_euclidean = create_euclidean_distance_matrix_cached(ratings_matrix_df, binary_matrix)
Dist_euclidean
```

Για λόγους επίσπευσης των υπολογισμών, αναθέτουμε κατευθείαν την απόσταση μεταξύ των χρηστών [i, j] = [j, i] = d.

Το αποτέλεσμα που λαμβάνουμε είναι της εξής μορφής:

	0	1	2	3	4	5	6	7	8	9
0	0.000000	3.464102	1.000000	3.872983	9.000000	0.000000	7.141428	0.000000	4.00000	3.872983
1	3.464102	0.000000	7.071068	2.236068	2.449490	7.874008	6.000000	3.316625	1.00000	2.828427
2	1.000000	7.071068	0.000000	13.266499	6.164414	17.549929	6.082763	4.898979	1.00000	9.949874
3	3.872983	2.236068	13.266499	0.000000	10.099505	9.486833	4.898979	7.810250	4.00000	3.162278
4	9.000000	2.449490	6.164414	10.099505	0.000000	9.219544	5.830952	4.358899	2.44949	10.908712
...	...	...	...	...	...	...	...	...	...	...
1074	0.000000	3.000000	4.242641	1.000000	0.000000	5.000000	1.000000	0.000000	0.00000	0.000000
1075	1.000000	2.828427	5.196152	9.000000	2.449490	9.433981	3.000000	4.242641	1.00000	8.774964
1076	4.000000	5.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.00000	3.000000
1077	4.000000	0.000000	8.062258	2.000000	6.324555	3.000000	0.000000	0.000000	0.00000	6.164414
1078	3.316625	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.44949	2.000000

1079 rows × 1079 columns

Οι γραμμές και οι στήλες αυτού του dataframe είναι οι μοναδικοί χρήστες ενώ το στοιχείο που υπάρχει σε κάθε περίπτωση είναι η μεταξύ τους απόσταση.

Στον πίνακα που έχουμε δημιουργήσει, θα χρησιμοποιήσουμε τον αλγόριθμο k-means έτσι ώστε να ομαδοποιήσουμε τις αποστάσεις μεταξύ των χρηστών. Θα δημιουργήσουμε δηλαδή συστάδες χρηστών (clusters) των οποίων οι μεταξύ τους αποστάσεις ανά δύο θα είναι οι μικρότερες δυνατές.

```
from sklearn.cluster import KMeans
# Cluster the users using K-means
kmeans = KMeans(n_clusters=L_CLUSTERS_NUM).fit(Dist_euclidean)

# Get the cluster labels
labels_euclidean = kmeans.labels_

# Print the labels
print(labels_euclidean)
```

b. $$dist_{cosine}(R_u, R_v) = 1 - \left| \frac{\sum_{k=1}^{n} R_u(k) R_v(k) \lambda_u(k) \lambda_v(k)}{\sqrt{\sum_{k=1}^{n} R_u(k)^2 \lambda_u(k) \lambda_v(k)} \sqrt{\sum_{k=1}^{n} R_v(k)^2 \lambda_u(k) \lambda_v(k)}} \right| \quad (4)$$

Για να δημιουργήσουμε συστάδες χρησιμοποιώντας την ανωτέρω μετρική, θα λειτουργήσουμε με την ίδια μεθοδολογία όπως στο προηγούμενο ερώτημα.

```
# Calculate the pairwise weighted Cosine distance matrix

def create_cosine_distance_matrix_cached(ratings_matrix: pd.DataFrame, binary_matrix: np_typing.NDArray) -> np_typing.NDArray[np.float64]:
    #Set the npy file that will store the Euclidean distance matrix
    npy_file = os.path.join(DATAFOLDER_PATH, "cosine_distance_matrix.npy")
    if os.path.exists(npy_file):
        Dist_cosine: np_typing.NDArray[np.float64] = np.load(npy_file, allow_pickle=True)
        return Dist_cosine
    else:
        n = ratings_matrix.shape[0]
        Dist_cosine = np.zeros((n, n))
        for i in range(n):
            for j in range(i, n):
                d = 1 - np.abs(np.sum(binary_matrix[i,:] * binary_matrix[j,:] * ratings_matrix.loc[i,:] * ratings_matrix.loc[j,:])
                / (np.sqrt(np.sum(binary_matrix[i,:] * binary_matrix[j,:] * ratings_matrix.loc[i,:])* np.sqrt(np.sum(binary_matrix[i,:]
                * binary_matrix[j,:] * ratings_matrix.loc[j,:])))))
                Dist_cosine[i,j] = d
                Dist_cosine[j,i] = d
        np.save(npy_file, Dist_cosine, allow_pickle=True, fix_imports=True)
        return Dist_cosine

Dist_cosine = create_cosine_distance_matrix_cached(ratings_matrix_df, binary_matrix)
Dist_cosine
```

Αρχικά δημιουργούμε τον πίνακα των αποστάσεων μεταξύ των ζευγών των χρηστών. Αυτός θα είναι της ακόλουθης μορφής:

	0	1	2	3	4	5	6	7	8	9
0	-49.304560	-23.171316	-9.142907	-18.187750	-15.393314	0.000000	-16.441067	0.000000	-10.618950	-15.466178
1	-23.171316	-45.588603	-10.246827	-15.527989	-21.414019	-11.470574	-19.850920	-16.154845	-13.270485	-10.767502
2	-9.142907	-10.246827	-51.417993	-15.439397	-21.888741	-4.886134	-11.812998	-24.434338	-18.417969	-10.573472
3	-18.187750	-15.527989	-15.439397	-29.955234	-16.635486	-5.198522	-9.686401	-11.153215	-8.513657	-4.000000
4	-15.393314	-21.414019	-21.888741	-16.635486	-54.692987	-10.648180	-21.480240	-23.180685	-24.333153	-17.758224
...	...	...	...	...	...	...	...	...	...	...
1074	0.000000	-8.457416	-12.677042	-9.541421	0.000000	-2.833659	-5.687403	0.000000	0.000000	-6.476744
1075	-15.744288	-22.252625	-22.851036	-16.316961	-22.237900	-13.830095	-10.541441	-18.857696	-11.172184	-13.445036
1076	-11.123093	-7.575494	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-3.000000	-12.156527
1077	-3.119534	0.000000	-20.342609	-5.928203	-11.975146	-1.828427	0.000000	0.000000	0.000000	-11.116020
1078	-19.049201	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-16.534737	-10.651803

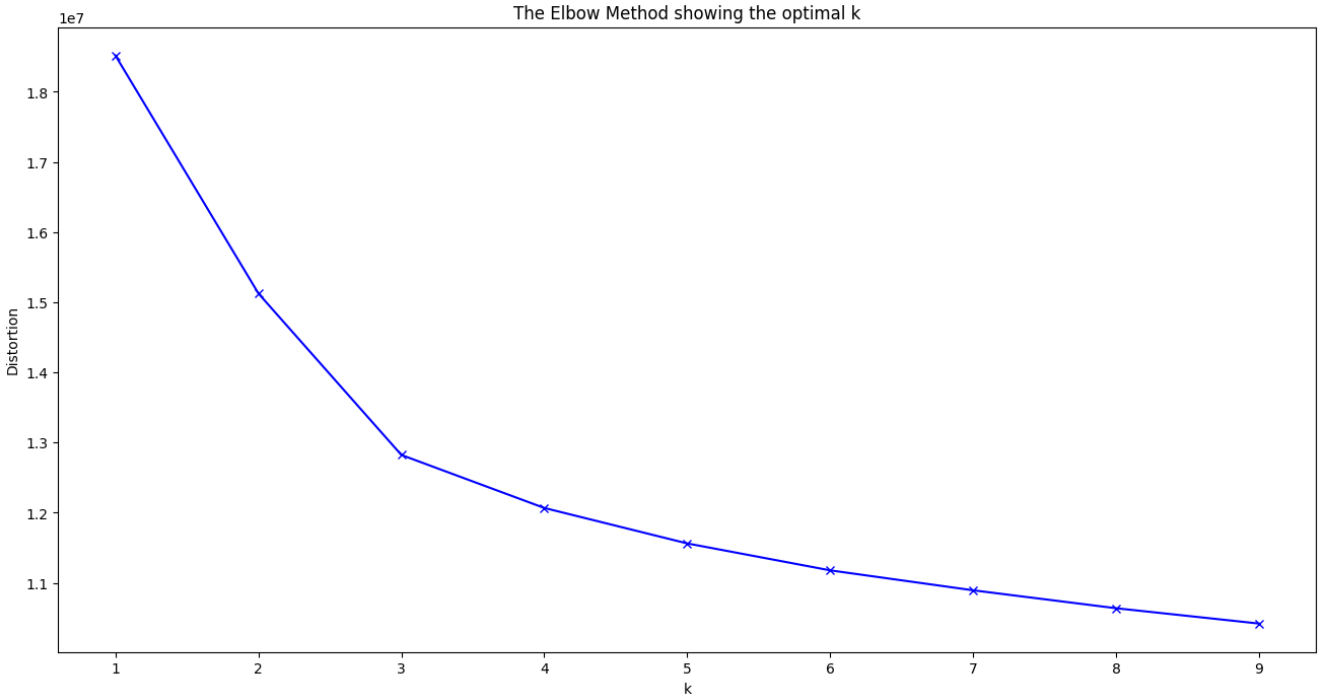
1079 rows × 1079 columns

Από τον ανωτέρω πίνακα θα δημιουργήσουμε τις συστάδες των χρηστών, χρησιμοποιώντας τον αλγόριθμο k-means.

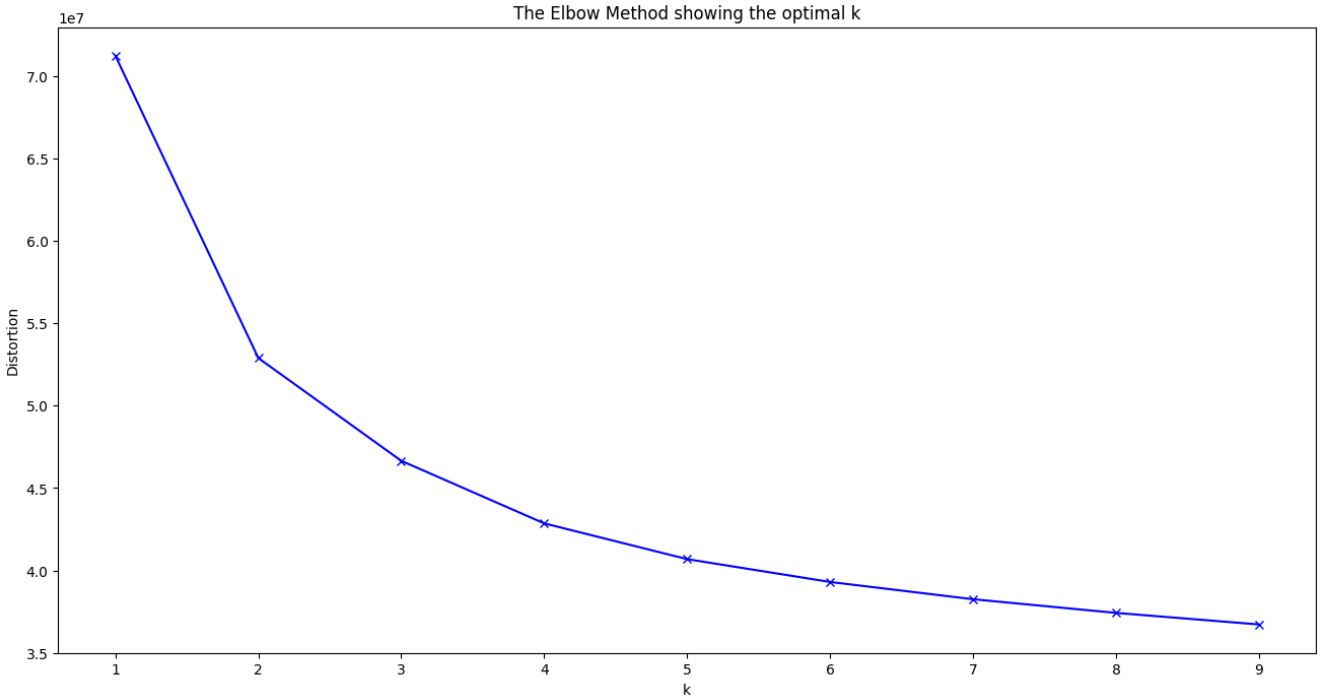
```
# Cluster the users using K-means
kmeans = KMeans(n_clusters=L_CLUSTERS_NUM).fit(df_cosine)

# Get the cluster labels
labels_cosine = kmeans.labels_

# Print the labels
print(labels_cosine)
```

c. Να αναπαραστήσετε γραφικά τις συστάδες των χρηστών που αναγνωρίστηκαν από τον αλγόριθμο k-means για κάθε μία από τις παραπάνω μετρικές για διάφορες τιμές της παραμέτρου L.

Για να επιλέξουμε τον βέλτιστο αριθμό των K διαφορετικών συστάδων χρηστών που θέλουμε να χρησιμοποιήσουμε στον αλγόριθμο k-means χρησιμοποιήσαμε την elbow method. Αυτή απέφερε τα ακόλουθα αποτελέσματα για τη μετρική της Ευκλείδειας απόστασης:



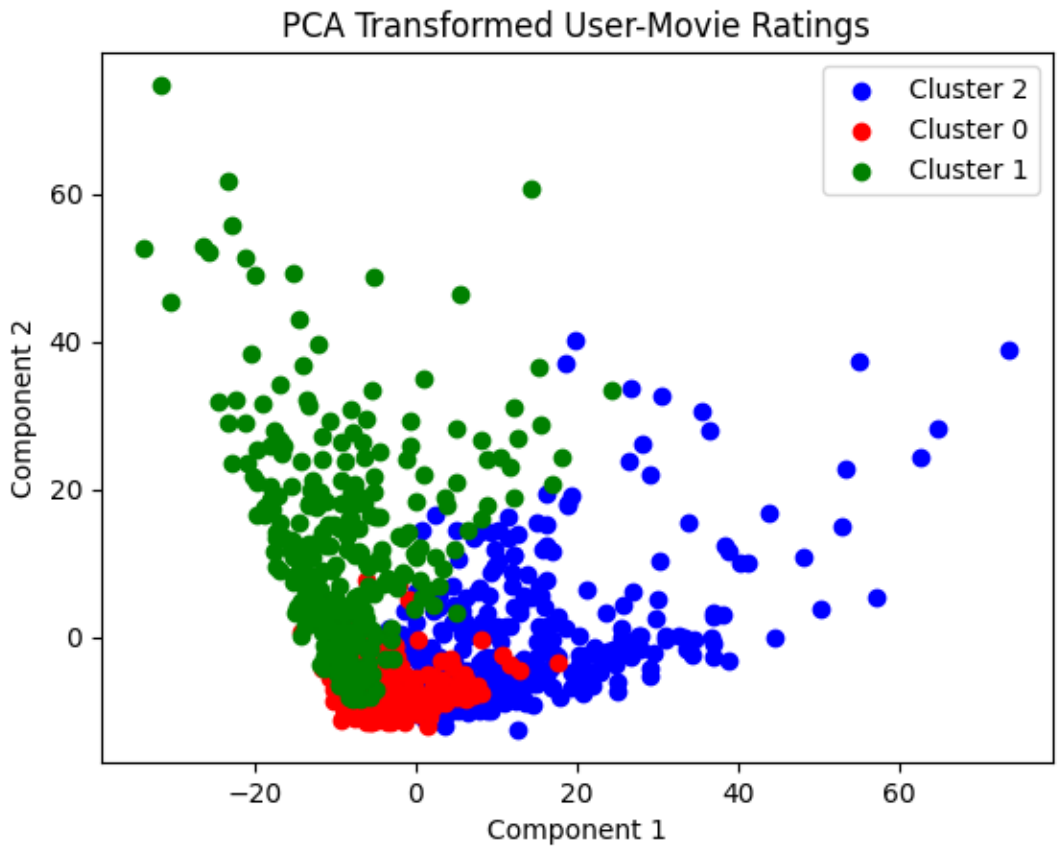
Ενώ για τη μετρική της απόστασης συνημιτόνου:



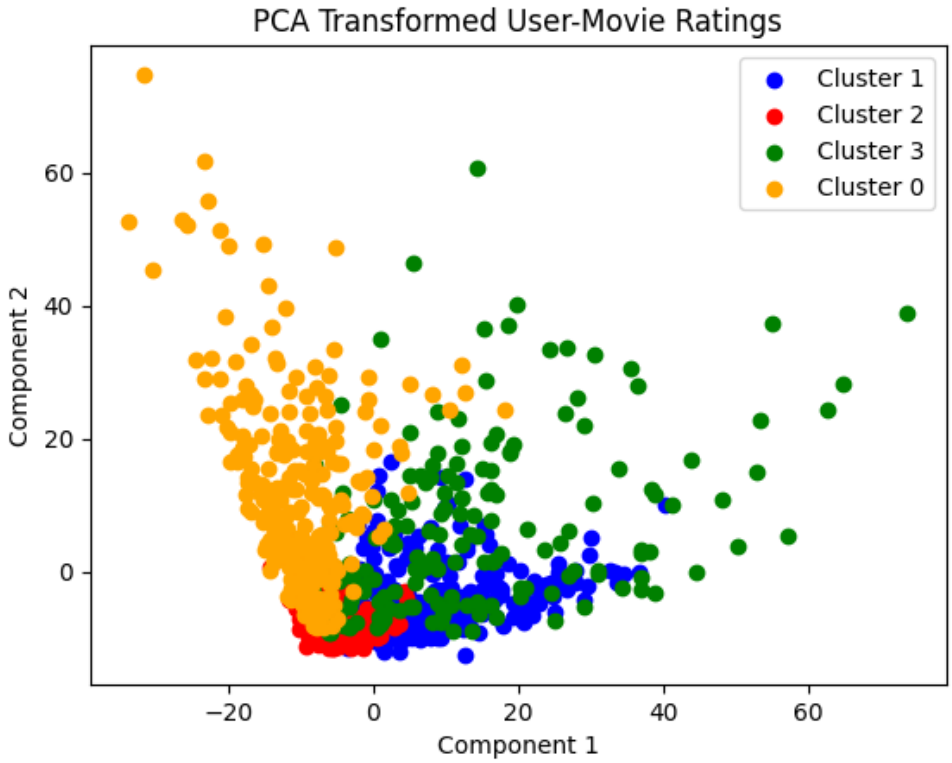
```python
def elbow_method(df: pd.DataFrame, max_iter: int):
  distortions = []
  K = range(1,max_iter)
  for k in K:
    kmeanModel = KMeans(n_clusters=k, n_init=10)
    kmeanModel.fit(df)
    distortions.append(kmeanModel.inertia_)
  plt.figure(figsize=(16,8))
  plt.plot(K, distortions, 'bx-')
  plt.xlabel('k')
  plt.ylabel('Distortion')
  plt.title('The Elbow Method showing the optimal k')
  plt.show()
```

Σε κάθε περίπτωση, εκτιμούμε ότι περισσότερο ενδιαφέρον θα έχουν οι συστάδες χρηστών που θα παραχθούν για 3 < L < 5.

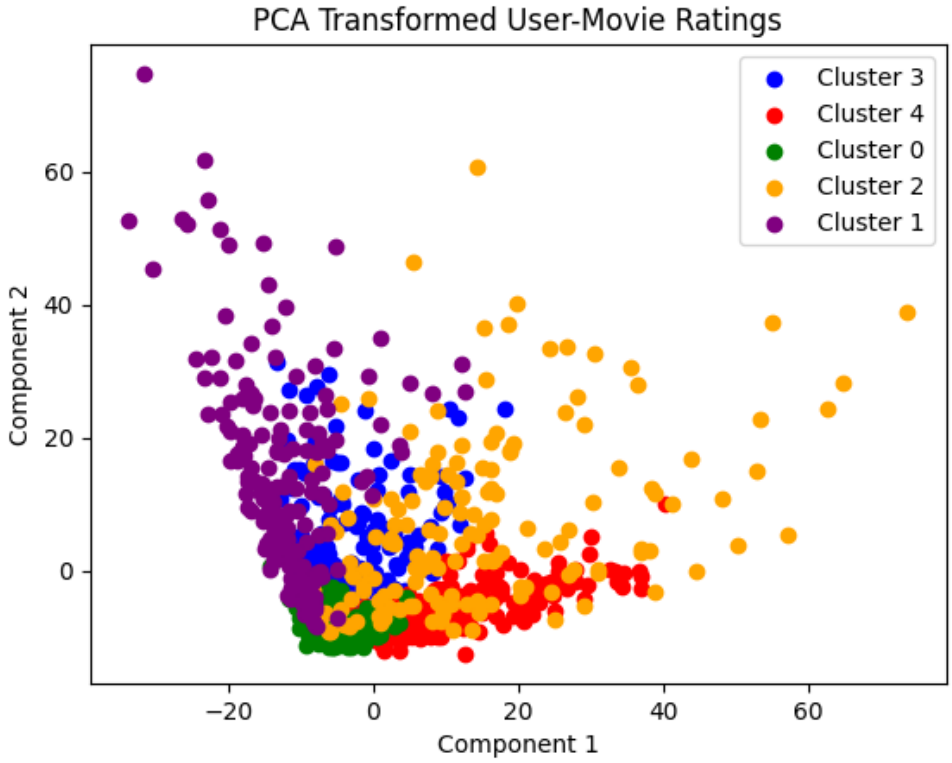
Για L = 3:


PCA Transformed User-Movie Ratings
Cluster 2
Cluster 0
Cluster 1
Component 2
Component 1
60
40
20
0
−20
0
20
40
60

Για L = 4:

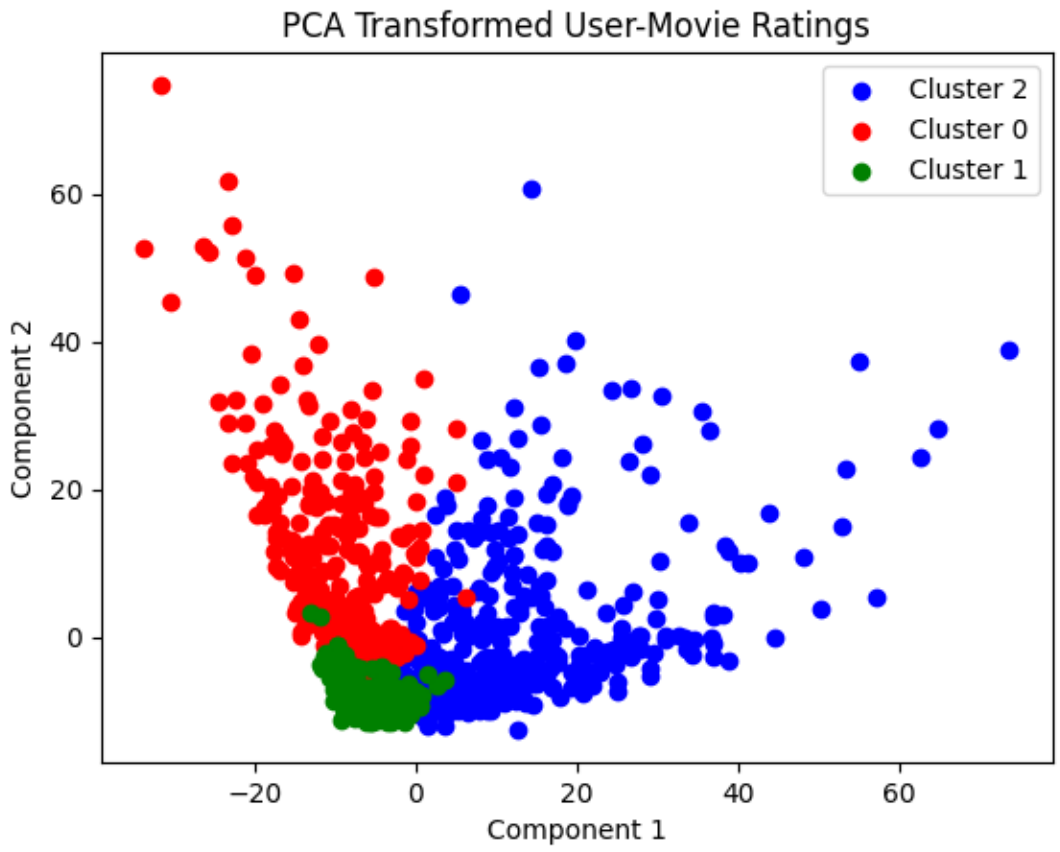

PCA Transformed User-Movie Ratings
Cluster 1
Cluster 2
Cluster 3
Cluster 0
Component 2
Component 1
60
40
20
0
−20
0
20
40
60

Για L = 5:

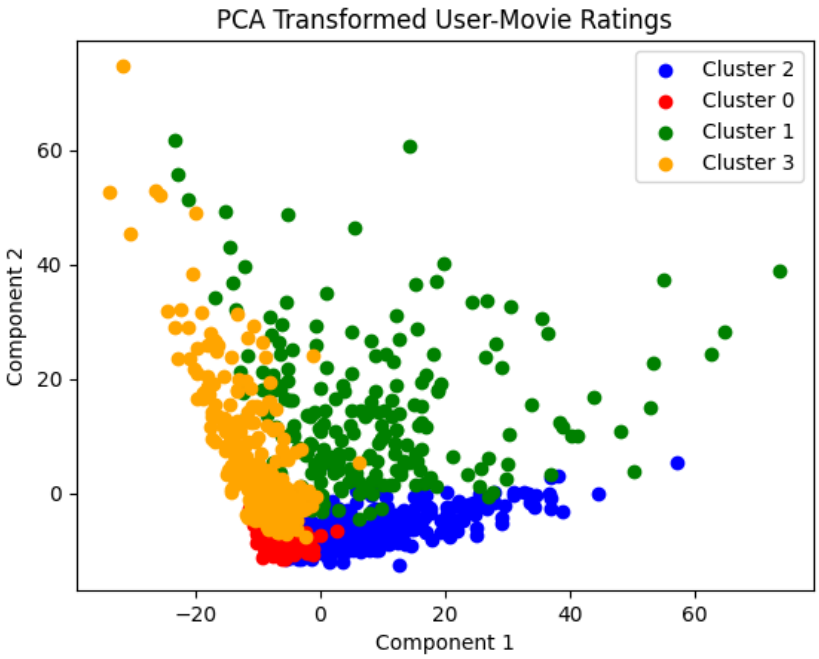


Με χρήση της απόστασης των συνημίτονων έχουμε τα εξής αποτελέσματα:

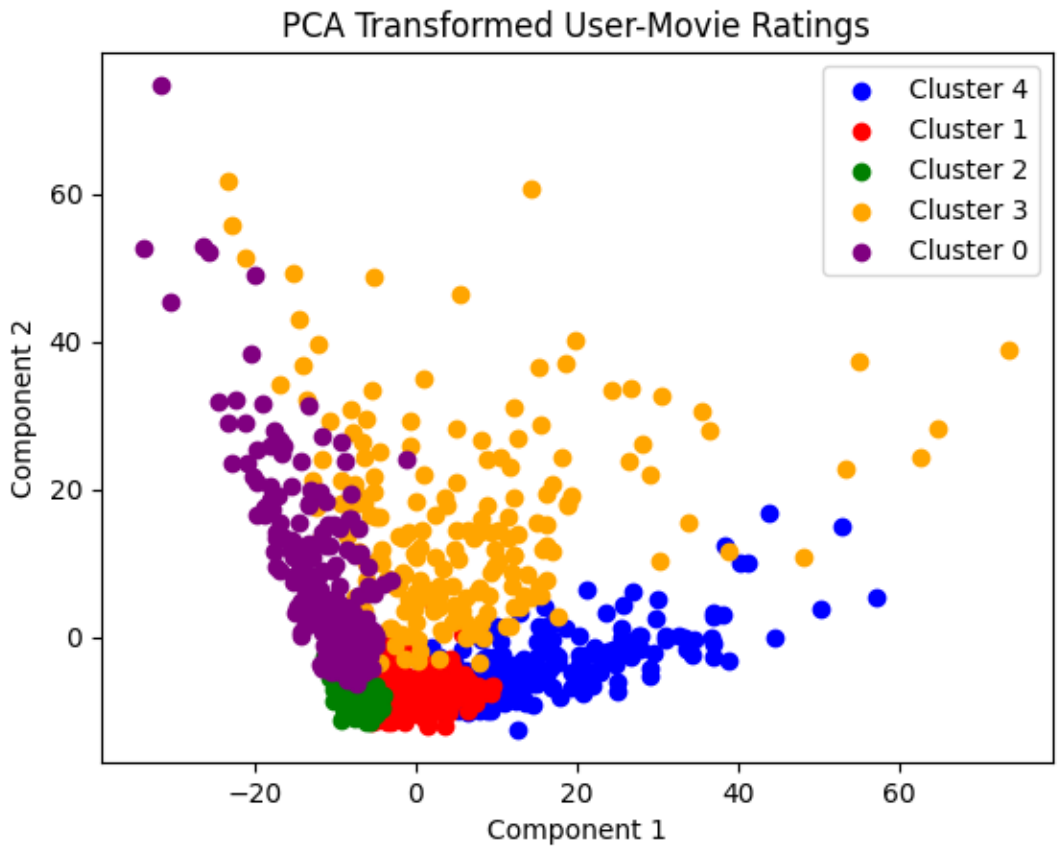
Για L = 3:


PCA Transformed User-Movie Ratings
Cluster 2
Cluster 0
Cluster 1
Component 2
Component 1
60
40
20
0
−20
0
20
40
60

Για L = 4:


PCA Transformed User-Movie Ratings
Cluster 2
Cluster 0
Cluster 1
Cluster 3
Component 2
Component 1
60
40
20
0
−20
0
20
40
60

Για L = 5:



Η μεθοδολογία που ακολουθήσαμε για να απεικονίσουμε τις συστάδες των χρηστών για τις διάφορες τιμές του L ήταν ως εξής:

- Υπολογίσαμε τις συστάδες των χρηστών που παράγονται από τους πίνακες των αποστάσεων για τις δύο διαφορετικές μετρικές (Ευκλείδεια απόσταση και απόσταση συνημίτονου).

- Πήραμε τον πίνακα Χρηστών – Ταινιών και εφαρμόσαμε τη μέθοδο PCA ώστε να μετασχηματίσουμε τον πίνακα από [1079 rows × 68084 columns] σε [1079 rows x 2 columns] ώστε να μπορέσουμε να τους απεικονίσουμε στον δυσδιάστατο χώρο.

- Στον πίνακα που έχουμε δημιουργήσει, δημιουργούμε τη στήλη ‘Clusters’ έτσι ώστε να αναθέσουμε διαφορετικό χρώμα σε κάθε χρήστη αναλόγως τη συστάδα στην οποία ανήκει.

d. Να σχολιάσετε την αποτελεσματικότητα των συγκεκριμένων μετρικών στην αποτίμηση της ομοιότητας μεταξύ ενός ζεύγους διανυσμάτων προτιμήσεων χρηστών R_u και R_v.

Και στις δύο περιπτώσεις, η αποτελεσματικότητα της ομοιότητας των χρηστών βασίζεται σε μεγάλο βαθμό στην ύπαρξη ή όχι κοινά αξιολογημένων ταινιών από τους δύο χρήστες καθώς έχουμε τροποποιήσει τις δύο μετρικές ώστε να λαμβάνουν των πίνακα «βαρών» λ.

Για τη μετρική της Ευκλείδειας απόστασης:

- Η ομοιότητα μεταξύ των χρηστών είναι αντιστρόφως ανάλογη της απόστασης μεταξύ τους. Τα αριθμητικά στοιχεία των αξιολογήσεων δηλαδή παίζουν ρόλο κατά την επιλογή της συστάδας για τον χρήστη.

Για τη μετρική της απόστασης συνημίτονου:

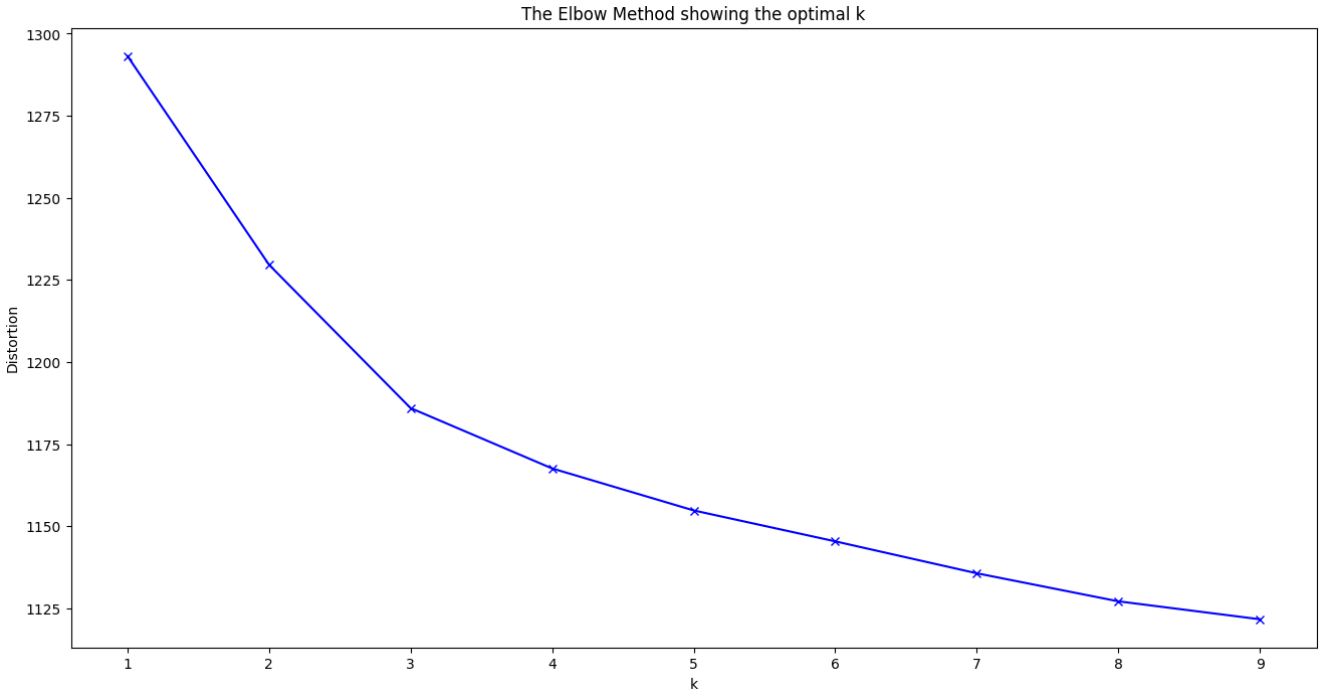
- Είναι πιο χρήσιμη όταν δε μας ενδιαφέρει τόσο η τιμή αυτή καθεαυτή των αξιολογήσεων του χρήστη, καθώς εξαρτάται από τη γωνία που σχηματίζουν τα διανύσματα τους.

Αλγόριθμοι Παραγωγής Συστάσεων με Χρήση Τεχνητών Νευρωνικών Δικτύων

2. Να δημιουργήσετε μια εναλλακτική οργάνωση του περιορισμένου συνόλου των χρηστών σε L συστάδες $\widehat{U} = U_{G_1} \cup U_{G_2} \cup \cdots \cup U_{G_L}$ έτσι ώστε $U_{G_a} \cap U_{G_b} \neq \varnothing, \forall\, a, b \in [L]$ με $a \neq b$ κάνοντας χρήση της παρακάτω μετρικής (**):

$$dist(u, v) = 1 - \frac{|\varphi(u) \cap \varphi(v)|}{|\varphi(u) \cup \varphi(v)|} \quad (5)$$

Έχουμε βρει μέσω της **elbow method** στον πίνακα των jaccard distances ότι ο καλύτερος αριθμός clusters είναι ο 5. Έτσι ορίζουμε την global μεταβλητή **L_CLUSTERS_NUM = 5** για να δημιουργήσουμε 5 clusters.



Έχουμε υπολογίσει στην μεταβλητή **jaccard_dist** (np array) και αντίστοιχα στην **jaccard_df** τον ίδιο πίνακα σε μορφή DataFrame, τις Jaccard Distances όλων των χρηστών με όλους τους χρήστες (δηλ. **jaccard_dist.shape** == (#Users, #Users)).

'jaccard_df'

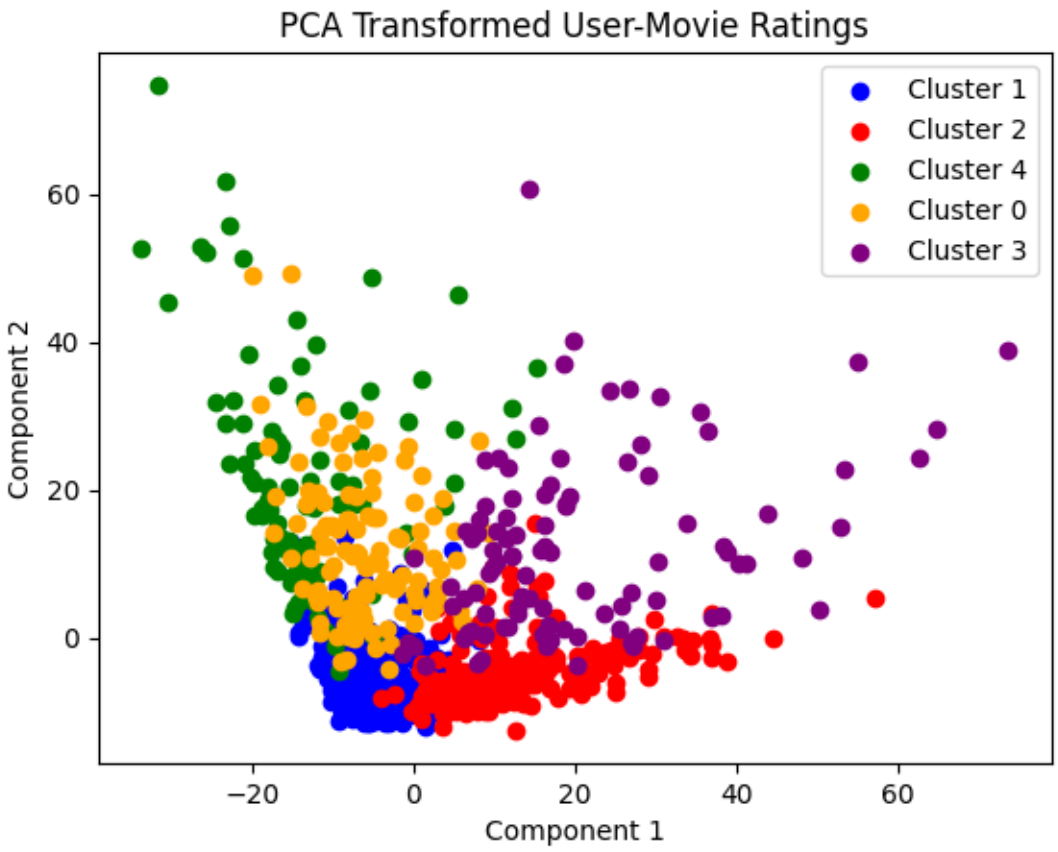
	0	1	2	3	4	5	6	7	8	9	...	1069	1070	1071	1072	1073	1074	1075	1076	1077	1078
0	0.000000	0.983721	0.997717	0.991031	0.990971	1.000000	0.991266	1.000000	0.997738	0.989455	...	0.977186	1.0	0.994444	0.997753	0.993421	1.000000	0.995614	0.997717	0.998267	0.990020
1	0.983721	0.000000	0.993421	0.993651	0.983819	0.990826	0.978261	0.990741	0.996764	0.995455	...	0.984962	1.0	0.995098	1.000000	0.974843	0.996815	0.984375	0.993421	1.000000	1.000000
2	0.997717	0.993421	0.000000	0.964286	0.983923	0.978462	0.993921	0.968652	0.993548	0.993197	...	0.977387	1.0	0.977667	0.990385	0.990769	0.993651	0.984472	1.000000	0.986395	1.000000
3	0.991031	0.993651	0.964286	0.000000	0.984472	0.988201	0.988166	0.985075	0.996894	0.995585	...	0.995192	1.0	0.995249	0.990712	1.000000	0.996942	0.984985	1.000000	0.997812	1.000000
4	0.990971	0.983819	0.983923	0.984472	0.000000	0.994083	0.981982	0.978788	0.980892	0.984270	...	0.967662	1.0	0.985507	0.990625	0.984894	1.000000	0.987915	1.000000	0.995585	1.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1074	1.000000	0.996815	0.993651	0.996942	1.000000	0.997059	0.997050	1.000000	1.000000	0.997788	...	0.995169	1.0	0.968137	0.996904	0.987988	0.000000	0.975610	0.990446	0.993377	1.000000
1075	0.995614	0.984375	0.984472	0.984985	0.987915	0.991379	0.994253	0.991304	0.996970	0.989083	...	0.975962	1.0	0.985882	0.993976	0.954819	0.975610	0.000000	0.993846	0.973568	0.994898
1076	0.997717	0.993421	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.996785	0.995475	...	0.990074	1.0	1.000000	1.000000	0.987654	0.990446	0.993846	0.000000	0.997758	0.997326
1077	0.998267	1.000000	0.986395	0.997812	0.995585	0.997872	1.000000	1.000000	1.000000	0.991349	...	0.994475	1.0	0.994526	0.997792	1.000000	0.993377	0.973568	0.997758	0.000000	1.000000
1078	0.990020	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.992021	0.998039	...	0.978448	1.0	0.993697	1.000000	0.997462	1.000000	0.994898	0.997326	1.000000	0.000000

1079 rows × 1079 columns

Με τον παρακάτω κώδικα βρίσκουμε τα jaccard_labels με τα οποία κατηγοριοποιούμε τους χρήστες σε L_CLUSTERS_NUM clusters.

```
def kmeans_clustering(jaccard_dist, L):
    # Initialize k-means object
    kmeans = KMeans(n_clusters=L)

    # Fit the k-means object to the Jaccard distance matrix
    kmeans.fit(jaccard_dist)

    return kmeans.labels_


def create_jaccard_labels_cached(jaccard_dist, L: int):
    npy_file = os.path.join(DATAFOLDER_PATH, "L_K_DEPEND_jaccard_labels.npy")
    if os.path.exists(npy_file):
        jaccard_labels: np_typing.NDArray = np.load(npy_file, allow_pickle=True)
        return jaccard_labels
    else:
        jaccard_labels = kmeans_clustering(jaccard_dist, L)
        np.save(npy_file, jaccard_labels, allow_pickle=True, fix_imports=True)
        return jaccard_labels


jaccard_labels = create_jaccard_labels_cached(jaccard_dist, L_CLUSTERS_NUM)
```

Με την μέθοδο PCA απεικονίζουμε τους χρήστες χρωματίζοντας τα clusters στα οποία ανήκουν:



a. **Να εξηγήσετε τι εκφράζει η συγκεκριμένη μετρική και να προσδιορίσετε τα μειονεκτήματά της σε σχέση με τις μετρικές που περιγράφονται στις σχέσεις (2) και (4) υπό το πρίσμα της ιδιαίτερης οργάνωσης των χρηστών που μπορεί να επιφέρει.**

Η μετρική της σχέσης 5 περιγράφει την Jaccard distance μεταξύ των χρηστών u και v. Η σχέση $J(u,v)=¿\varphi(u)\cap\varphi(v)\vee\frac{¿}{¿\varphi(u)\cup\varphi(v)\vee¿¿}¿$ ορίζει την Jaccard index ή αλλιώς jaccard similarity coefficient μεταξύ των χρηστών u και v. Αυτός ο συντελεστής ομοιότητας στην περίπτωση μας δείχνει κατά πόσο 2 χρήστες έχουν βαθμολογήσει τις ίδιες ταινίες και είναι μια ποσότητα μεταξύ 0 και 1 όπου οι πιο όμοιοι χρήστες θα έχουν συντελεστή ομοιότητας κοντά στο 1. Η Jaccard απόσταση που χρησιμοποιούμε θα είναι πάλι μεταξύ 0 και 1, αλλά οι πιο κοντινοί χρήστες θα έχουν απόσταση κοντά στο 0.

Εμείς την Jaccard απόσταση την υπολογίζουμε στον χώρο των ταινιών, αλλά με μέτρο την φ(u) η οποία θα έχει τιμή 0 ή 1 και θα δείχνει μόνο αν ένας χρήστης έχει βαθμολογήσει ή όχι την συγκεκριμένη ταινία, αγνοώντας τον πραγματικό βαθμό του rating.

Μειονεκτήματα:

1. Μειονέκτημα αυτής της μετρικής είναι ότι δεν λαμβάνει καθόλου υπόψιν την βαθμολογία του rating ενός χρήστη παρά μόνο τις κοινές βαθμολογήσεις και το πλήθος της ένωσης των βαθμολογήσεων και των 2 χρηστών. Μπορούν δηλαδή 2 χρήστες να έχουν βαθμολογήσει σχεδόν τις ίδιες ταινίες, ο πρώτος με 10.0 και ο δεύτερος με 1.0 αλλά η Jaccard απόσταση τους θα είναι κοντά στο 0.
2. Διαφορετικά sets με τον ίδιο αριθμό κοινών βαθμολογήσεων θα έχουν διαφορετική Jaccard similarity και Jaccard distance.

Άλλα γενικά μειονεκτήματα της Jaccard distance (όπως ότι δεν λαμβάνει υπόψιν την συχνότητα που εμφανίζονται κάποιοι όροι παρά μόνο το πλήθος των κοινών όρων) δεν μας αφορούν καθώς όπως έχουμε πει στην περίπτωσή μας έχουμε δυαδικές τιμές (0 ή 1) λόγω της συνάρτησης φ(u).

b. Η μετρική που περιγράφεται μέσω της σχέσης (5) μπορεί να χρησιμοποιηθεί προκειμένου να προσδιοριστεί το σύνολο $N_k(u_a)=\{u_a^{(1)},u_a^{(2)},\cdots,u_a^{(k)}\}$ των k πλησιέστερων γειτόνων ενός χρήστη $u_a \in U_{G_a}, \forall a \in [L]$. Επομένως, για τον εκάστοτε χρήστη u_a της εκάστοτε συστάδας U_{G_a} μπορούμε να το διάνυσμα τον προσωπικών του προτιμήσεων R_{u_a} καθώς και τα διανύσματα των k πλησιέστερων γειτόνων του $R_{u_a^{(1)}}$, $R_{u_a^{(2)}}$,..., $R_{u_a^{(k)}}$ εντός της συστάδας U_{G_a}. Στόχος του συγκεκριμένου ερωτήματος είναι να αναπτύξετε ένα πολυστρωματικό νευρωνικό δίκτυο για κάθε συστάδα χρηστών U_{G_a} με $a \in [L]$ το οποίο θα προσεγγίζει τις αξιολογήσεις του κάθε χρήστη εντός αυτής μέσω των αξιολογήσεων των k πλησιέστερων γειτόνων του μέσω μιας συνάρτησης της μορφής:

$$R_{u_a}=f_a\left(R_{u_a^{(1)}},R_{u_a^{(2)}},\cdots R_{u_a^{(k)}}\right), \forall u_a \in U_{G_a}, \forall a \in [L] \quad (6)$$

Έπειτα από πολλές δοκιμές καταλήξαμε ότι ο βέλτιστος αριθμό κοντινότερων γειτόνων είναι >=4 και <=10. Διαλέγουμε και θέτουμε την μεταβλητή **K_NEIGHBORS_NUM = 6** ώστε να επιλέξουμε τους 6 κοντινότερους (κατά απόσταση Jaccard) χρήστες.

Χρησιμοποιώντας την συνάρτηση **calculate_clusters_ratings_jaccard_distances_users_indexes()** φτιάχνουμε για κάθε cluster τους πίνακες **cluster_ratings** (**cluster_ratings.shape == (#cluster users, #total movies)**) με τα ratings κάθε χρήστη του cluster και **cluster_jaccard_distances** (**cluster_jaccard_distances.shape == (#cluster users, #total users)**) με τα Jaccard distances κάθε χρήστη του cluster ως προς όλους τους άλλους χρήστες. Χρησιμποποιούμε όλους τους άλλους χρήστες και όχι μόνο αυτούς του cluster για λόγους performance γιατί τις έχουμε ήδη υπολογίσει στα προηγούμενα ερωτήματα.

```
 def calculate_clusters_ratings_jaccard_distances_users_indexes(L_CLUSTERS_NUM: int,
jaccard_labels: np_typing.NDArray, ratings_matrix_array: np_typing.NDArray, jaccard_dist:
np_typing.NDArray):
    clusters_ratings_dict = { i: [] for i in range(L_CLUSTERS_NUM)}
    clusters_jaccard_distances_dict = { i: [] for i in range(L_CLUSTERS_NUM)}
```

```
    clusters_users_indexes_dict: dict[int, set[int]] = { i: set() for i in range(L_CLUSTERS_NUM)}

    for i in range(jaccard_labels.shape[0]):
        label = jaccard_labels[i]
        cluster_ratings = clusters_ratings_dict[label]
        cluster_jaccard_distances = clusters_jaccard_distances_dict[label]
        cluster_users_indexes = clusters_users_indexes_dict[label]

        cluster_ratings.append(ratings_matrix_array[i])
        cluster_jaccard_distances.append(jaccard_dist[i])
        cluster_users_indexes.add(i)


    clusters_ratings_list: list[np_typing.NDArray] = []
    for key in clusters_ratings_dict:
        cluster_ratings = np.array(clusters_ratings_dict[key])
        clusters_ratings_list.append(cluster_ratings)


    clusters_jaccard_distances_list: list[np_typing.NDArray] = []
    for key in clusters_jaccard_distances_dict:
        cluster_jaccard_distances = np.array(clusters_jaccard_distances_dict[key])
        clusters_jaccard_distances_list.append(cluster_jaccard_distances)

    clusters_users_indexes_list: list[set[int]] = []
    for key in clusters_users_indexes_dict:
        clusters_users_indexes_list.append(clusters_users_indexes_dict[key])
```

Με την συνάρτηση **find_nearest_neighbors_using_jaccard()** βρίσκουμε τους **K_NEIGHBORS_NUM** κοντινότερους γείτονες κάθε user για κάθε cluster, εξαιρώντας τον εαυτό του καθώς και τους γείτονες που δεν ανήκουν στο cluster. Σημείωση όπως αναφέραμε πριν τα indexes των γειτόνων είναι ως προς τον πίνακα με τους συνολικούς χρήστες (όχι μόνο του cluster) για λόγους performance, οπότε αργότερα θα πάρουμε τα ratings των γειτόνων από τον αρχικό πίνακα των ratings και όχι από το cluster_ratings.

```
def find_nearest_neighbors_using_jaccard(K_NEIGHBORS_NUM: int, cluster_jaccard_distances:
np_typing.NDArray, cluster_users_indexes: set[int]):
    nearest_neighbors_list: list[np_typing.NDArray] = []
    for row_index in range(cluster_jaccard_distances.shape[0]):
        cluster_jaccard_row = cluster_jaccard_distances[row_index]

        ## set the distance of the users that are the same a index to a value higher thatn 1
        for j in range(cluster_jaccard_row.shape[0]):
            if j == row_index or not(j in cluster_users_indexes):
                cluster_jaccard_row[j] = 2

        ## find the k smallest indexes
        k_nearest_indexes = np.argpartition(cluster_jaccard_row, K_NEIGHBORS_NUM)
```

```
        # k_nearest_indexes = k_nearest_indexes[k_nearest_indexes != row_index]
        nearest_neighbors_list.append(k_nearest_indexes[:K_NEIGHBORS_NUM])

    return np.array(nearest_neighbors_list)
```

Για κάθε cluster θα χρησιμοποιήσουμε την συνάρτηση **create_nn_original_df()** για να φτιάξουμε το dataframe **nn_origin_df**. Αυτό θα περιέχει στην στήλη **USER_RATINGS** τα ratings όλων των χρηστών του cluster για κάθε ταινία διαμορφωμένα στην σειρά σε μόνο μια στήλη (διάνυσμα). Στις στήλες **NEIGHBOR_RATINGS_**{**0**,**1**,.., **K_NEIGHBORS_NUM**} θα περιέχει τα ratings των K πλησιέστερων γειτόνων του κάθε χρήστη για κάθε ταινία. Αποθηκεύουμε αυτά τα dataframe σε pickle files και δεν τα κρατάμε στην μνήμη ώστε να μην γεμίζει η μνήμη και να φορτώνουμε κάθε φορά μόνο εκείνο που χρειαζόμαστε.

```
def create_nn_original_df(ratings_matrix_array: np_typing.NDArray, cluster_ratings:
np_typing.NDArray, nearest_neighbors: np_typing.NDArray, NEIGHBOURS_COLUMNS: list[str]):
    # create user_ratings_list
    user_index_list: list[int] = []
    movie_index_list: list[int] = []
    user_ratings_list: list[int] = []
    for i in range(nearest_neighbors.shape[0]):
        user_index_list.extend([i for ratings in cluster_ratings[i]])
        movie_index_list.extend([j for j in range(cluster_ratings[i].shape[0])])
        user_ratings_list.extend(cluster_ratings[i])

    # create neighbors list
    neighbors_list: list[list[int]] = []
    for i in range(nearest_neighbors.shape[1]):
        neighbor_ratings_list: list[int] = []

        for j in range(nearest_neighbors.shape[0]):
            neighbor = nearest_neighbors[j][i]
            # nearest neighbors have indexs for the ratings_matrix_array up to total users
            # so use ratings_matrix_array instead of cluster_ratings.
            # We have previously ensured that all neighbors belong to this clusters
            neighbor_ratings_list.extend(ratings_matrix_array[neighbor])

        neighbors_list.append(neighbor_ratings_list)



    nn_origin_df = pd.DataFrame()
    nn_origin_df['USER_INDEX'] = user_index_list
    nn_origin_df['MOVIE_INDEX'] = movie_index_list
    nn_origin_df['USER_RATINGS'] = user_ratings_list

    for i in range(len(neighbors_list)):
        neighbor_ratings_list: list[int] = neighbors_list[i]
        nn_origin_df[NEIGHBOURS_COLUMNS[i]] = neighbor_ratings_list
```

```
    return nn_origin_df
```

Από αυτά τα δεδομένα θα χρησιμοποιήσουμε την στήλη **USER_RATINGS** ως τα labels μας για το regression πρόβλημα που θα προσπαθήσουμε να λύσουμε με το νευρωνικό μας δίκτυο και τις στήλες **NEIGHBOR_RATINGS_**{**0**,**1**,.., **K_NEIGHBORS_NUM**} ως τα features μας. Ωστόσο για να εκπαιδευτεί σωστά το νευρωνικό δίκτυο δεν μπορούμε να έχουμε 0 στα features μας καθώς αυτές είναι βαθμολογίες όπου δεν υπάρχουν (θυμίζουμε οι υπάρχουσες βαθμολογίες είναι από 1 εώς 10 ενώ τα μηδενικά είναι στην ουσία οι NaN τιμές για τα ratings που δεν ξέρουμε καθόλου). Επίσης δεν μπορούμε να έχουμε γραμμές που να έχουν μόνο μηδενικά στα features μας καθώς αυτό θα σημαίνει ότι κανείς κοντινότερος γείτονας δεν θα έχει βαθμολογήσει την συγκεκριμένη ταινία και ως αποτέλεσμα να μην μπορεί να προβλεφτεί κάποια βαθμολογία που να βγάζει νόημα. Αυτό το επιτυγχάνουμε με την συνάρτηση **create_nn_filtered_normalized_df()**. Διαιρούμε τα labels και τα features μας με κάποιον παράγοντα για normalization, ωστόσο είδαμε χειρότερα αποτελέσματα έτσι και για αυτό έχουμε θέσει τον παράγοντα αυτόν ίσον με 1 αναιρώντας οποιαδήποτε επίδραση που θα μπορούσε να έχει.

```
def create_nn_filtered_normalized_df(nn_origin_df: pd.DataFrame, NEIGHBOURS_COLUMNS: list[str], ratings_normalize_factor: float):
    # filter the rows that have USER_RATINGS == 0
    nn_filtered_df = nn_origin_df.copy()[nn_origin_df['USER_RATINGS'] != 0]

    # filter the rows that have all NEIGHBOR_RATINGS == 0
    filter_neighbors_query: str = f'{NEIGHBOURS_COLUMNS[0]} != 0'
    for neihbor_column in NEIGHBOURS_COLUMNS[1:]:
        filter_neighbors_query += f' | {neihbor_column} != 0'

    nn_filtered_df = nn_filtered_df.query(filter_neighbors_query)

    # create filtered normalized df
    nn_filtered_normalized_df = nn_filtered_df.copy()
    columns_to_normalize = ['USER_RATINGS']
    columns_to_normalize.extend(NEIGHBOURS_COLUMNS)
    nn_filtered_normalized_df[columns_to_normalize] = nn_filtered_normalized_df[columns_to_normalize] / ratings_normalize_factor
    display('Dataframe with filter user ratings (non zero) and neighbors ratings scaled by maximum rating')
    display(nn_filtered_normalized_df)
    display(nn_filtered_normalized_df.describe())
    return nn_filtered_df, nn_filtered_normalized_df
```

Π.χ. για το Cluster0 το τελικό dataframe θα είναι:

'Dataframe with filter user ratings (non zero) and neighbors ratings scaled by maximum rating'

	USER_INDEX	MOVIE_INDEX	USER_RATINGS	NEIGHBOR_RATINGS_0	NEIGHBOR_RATINGS_1	NEIGHBOR_RATINGS_2	NEIGHBOR_RATINGS_3	NEIGHBOR_RATINGS_4	NEIGHBOR_RATINGS_5
428	0	428	7.0	0.0	0.0	7.0	0.0	0.0	0.0
7518	0	7518	4.0	0.0	0.0	4.0	0.0	0.0	0.0
8140	0	8140	7.0	0.0	0.0	7.0	0.0	0.0	0.0
8872	0	8872	7.0	0.0	0.0	7.0	10.0	0.0	0.0
9612	0	9612	4.0	0.0	0.0	4.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...
8983410	131	64406	7.0	7.0	0.0	0.0	0.0	0.0	0.0
8983546	131	64542	7.0	7.0	0.0	0.0	0.0	7.0	9.0
8984148	131	65144	4.0	4.0	0.0	0.0	0.0	0.0	0.0
8985977	131	66973	1.0	1.0	0.0	0.0	0.0	0.0	0.0
8986135	131	67131	5.0	5.0	0.0	0.0	0.0	0.0	0.0

28189 rows × 9 columns

Με την συνάρτηση **create_model()** δημιουργούμε το νευρωνικό μας δίκτυο μέσω της κλάσης Sequential του tensorflow. Αυτό θα περιέχει ένα Input Layer, 3 κρυφά Dense layers με διπλασιαζόμενα units το καθένα από το προηγούμενο και ένα τελικό Output layer με ένα unit όπου θα προβλέπει την τελικό rating για μια ταινία ενός χρήστη από τα ratings των γειτόνων του.

```
def create_model(my_learning_rate: float, input_shape: tuple):
    """Create and compile a simple linear regression model."""
    # Most simple tf.keras models are sequential.
    model = tf.keras.models.Sequential()

    # Add the layer containing the feature columns to the model.
    model.add(tf.keras.layers.InputLayer(input_shape=input_shape))

    model.add(tf.keras.layers.Masking(
        mask_value=0
    ))

    # Implement L2 regularization in the first hidden layer.
    model.add(tf.keras.layers.Dense(units=12,
                                    activation='relu',
                                    kernel_regularizer=tf.keras.regularizers.l2(0.01),
                                    name='Hidden1'))

    # Implement L2 regularization in the second hidden layer.
    model.add(tf.keras.layers.Dense(units=24,
                                    activation='relu',
                                    kernel_regularizer=tf.keras.regularizers.l2(0.01),
                                    name='Hidden2'))

    # Implement L2 regularization in the third hidden layer.
    model.add(tf.keras.layers.Dense(units=48,
                                    activation='relu',
                                    kernel_regularizer=tf.keras.regularizers.l2(0.01),
                                    name='Hidden3'))

    # Define the output layer.
    model.add(tf.keras.layers.Dense(units=1,
```

```
                        name='Output'))

    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=my_learning_rate),
                  loss="mean_squared_error",
                  metrics=[tf.keras.metrics.MeanSquaredError(), tf.keras.metrics.MeanAbsoluteError()])
    return model


my_model = create_model(learning_rate, input_shape)
```

c. **Το σύνολο των χρηστών της κάθε συστάδας μπορεί να διαμεριστεί περεταίρω σε ένα υποσύνολο χρηστών για εκπαίδευση του νευρωνικού δικτύου $U_{G_a}^{train}$ και σε ένα υποσύνολο χρηστών για τον έλεγχο της επίδοσης του νευρωνικού δικτύου $U_{G_a}^{test}$ έτσι ώστε $U_{G_a}=U_{G_a}^{train}\cup U_{G_a}^{test}$ με $U_{G_a}^{train}\cap U_{G_a}^{test}=\varnothing$. Η καταλληλότερη διαμόρφωση των δεδομένων για την εκπαίδευση των νευρωνικών δικτύων αυτού του ερωτήματος θα μπορούσε να αναπαρασταθεί ως εξής:**

   Έστω ότι το σύνολο των χρηστών που μετέχουν στην συστάδα U_{G_a} δίνεται από την σχέση: $U_{G_a}=\{u_{a,1},u_{a,2},\cdots,u_{a,n_a}\}$ με $n_a=\left|U_{G_a}\right|$. Τότε το σύνολο των διανυσμάτων χαρακτηριστικών και το αντίστοιχο σύνολο των ετικετών θα μπορούσε να οργανωθεί σε έναν πίνακα της μορφής:

$$\begin{bmatrix} R_{u_{a,1}^{(1)}} & R_{u_{a,1}^{(2)}} & \cdots & R_{u_{a,1}^{(k)}} \\ R_{u_{a,2}^{(1)}} & R_{u_{a,2}^{(2)}} & \cdots & R_{u_{a,2}^{(k)}} \\ \vdots & \vdots & \vdots & \vdots \\ R_{u_{a,n_a}^{(1)}} & R_{u_{a,n_a}^{(2)}} & \cdots & R_{u_{a,n_a}^{(k)}} \end{bmatrix} \text{ και } \begin{bmatrix} R_{u_{a,1}} \\ R_{u_{a,2}} \\ \vdots \\ R_{u_{a,n_a}} \end{bmatrix}$$

Όπως βλέπουμε στον προηγούμενο κώδικα έχουμε επιλέξει ως loss function της εκπαίδευσης το mean_squared_error αντί για του mean_absolute_error καθώς είδαμε ότι παρουσιάζει μεγαλύτερη ακρίβεια και καλύτερα αποτελέσματα.

Από το φιλτραρισμένο **filtered_normalized_df** παίρνουμε το 80% των δεδομένων ως train dataset και το υπόλοιπο 20% ως test dataset.

```
train_df, test_df = train_test_split(nn_filtered_normalized_df, test_size=0.2, random_state=42)
```

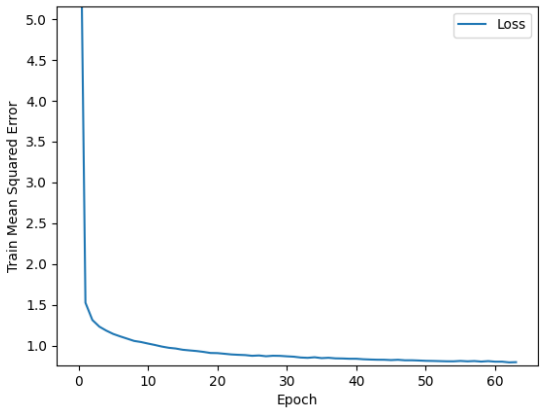
Από το train_df παίρνουμε τα features X_train και τα labels Y_train ως numpy arrays κι εκπαιδεύουμε το νευρωνικό δίκτυο με την **train_model()**. Κρατάμε τα διανύσματα των epochs, καθώς και των mean_squared_error και mean_absolute_error και τα αναπαριστάμε με την **plot_the_loss_curve()**. Διαλέγουμε μικρό **learning_rate = 0.001** και λίγα **epochs = 64** επειδή είδαμε ότι πέρα από αυτά δεν αλλάζουν ουσιαστικά οι γραφικές των losses, για να αποφύγουμε το overfitting.

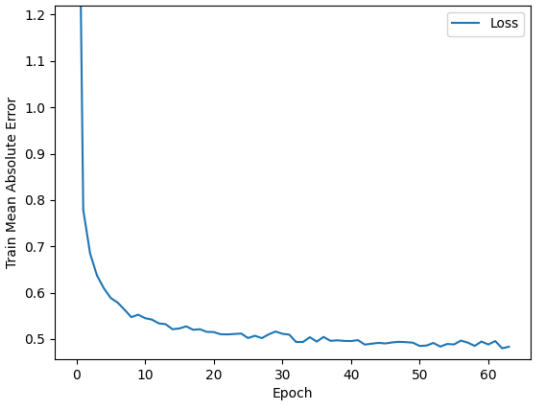
```
def plot_the_loss_curve(epochs, mse_or_mae, is_mse: bool, filename: str):
  """Plot a curve of loss vs. epoch."""
```

```
    plt.figure()
    plt.xlabel("Epoch")
    ylabel = 'Train Mean Squared Error' if is_mse else 'Train Mean Absolute Error'
    plt.ylabel(ylabel)

    plt.plot(epochs, mse_or_mae, label="Loss")
    plt.legend()
    plt.ylim([mse_or_mae.min()*0.95, mse_or_mae.max() * 0.6])
    plt.savefig(os.path.join(RESULTS_PATH, filename))
    plt.show()


def train_model(model: tf.keras.models.Sequential, X_train: np_typing.NDArray, Y_train:
np_typing.NDArray, epochs: int, batch_size: int=1):
    """Train the model by feeding it X_train."""

    # Features as a numpy array
    history = model.fit(x=X_train, y=Y_train, batch_size=batch_size, epochs=epochs, shuffle=True)

    # The list of epochs is stored separately from the rest of history.
    epochs = history.epoch

    # To track the progression of training, gather a snapshot
    # of the model's mean squared error at each epoch.
    hist = pd.DataFrame(history.history)
    mse = hist["mean_squared_error"]
    mae = hist["mean_absolute_error"]

    return epochs, mse, mae

learning_rate = 0.001
epochs = 64
batch_size = 128

X_train = train_df[NEIGHBOURS_COLUMNS].to_numpy()
Y_train = train_df['USER_RATINGS'].to_numpy()

epochs, train_mse_series, train_mae_series = train_model(my_model, X_train, Y_train, epochs,
batch_size)
train_mse = train_mse_series.iloc[-1]
train_mae = train_mae_series.iloc[-1]
plot_the_loss_curve(epochs, train_mse_series, True, f'cluster{cluster_index}_mse.png')
plot_the_loss_curve(epochs, train_mae_series, False, f'cluster{cluster_index}_mae.png')
```

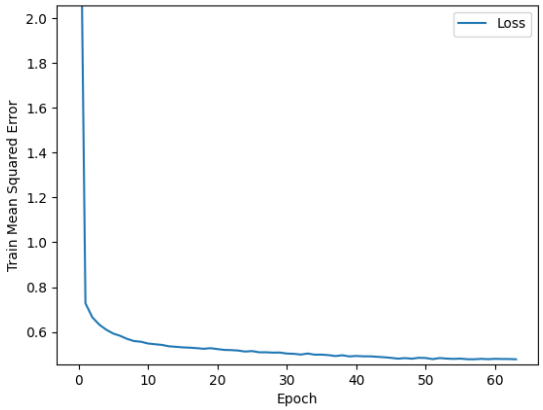
Παρουσιάζουμε τα train plots των mean_squared_error και mean_absolute_error σε σχέση με τα epochs για κάθε cluster:

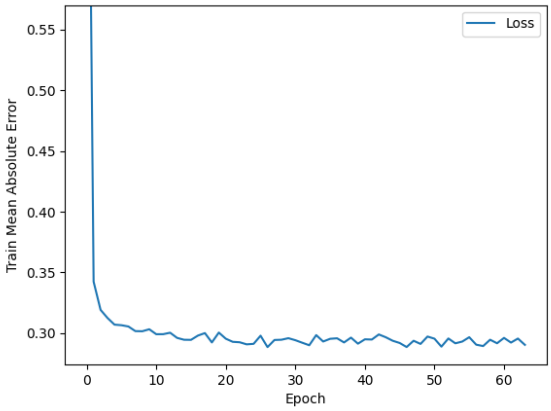
Cluster0 train plots:



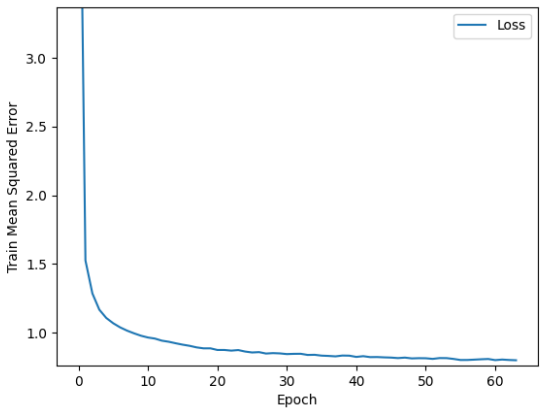


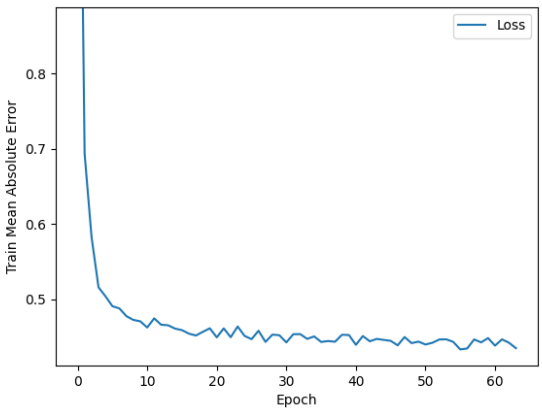
Cluster1 train plots:



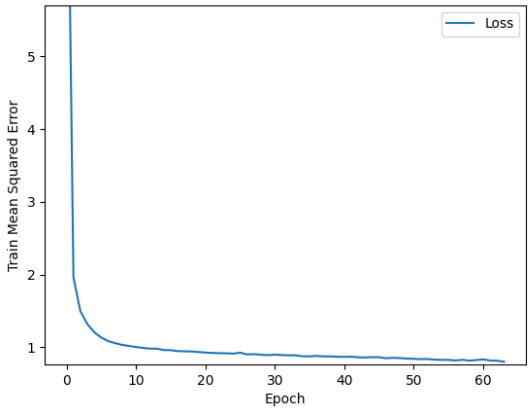


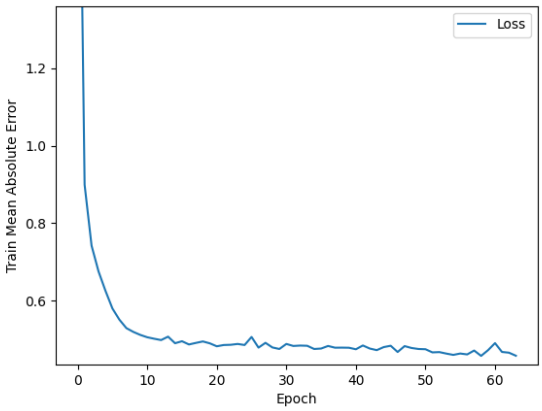
Cluster2 train plots:



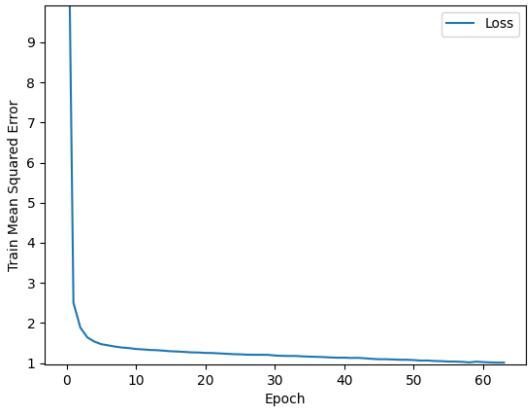


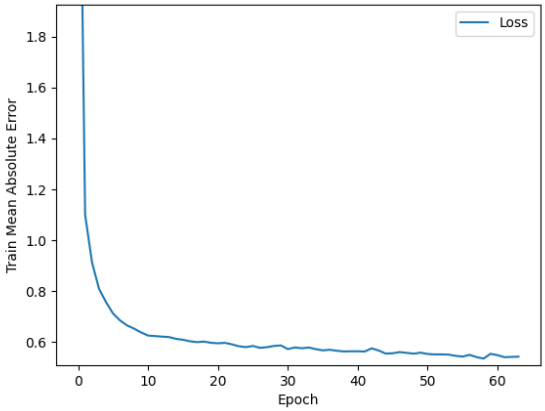
Cluster3 train plots:





Cluster4 train plots:





d. Η προσεγγιστική ακρίβεια των παραπάνω νευρωνικών δικτύων μπορεί να μετρηθεί μέσω της μετρικής του μέσου απόλυτου σφάλματος ανάμεσα στις πραγματικές και τις εκτιμώμενες αξιολογήσεις των χρηστών. Να παρουσιάσετε πίνακες των αποτελεσμάτων σας τόσο για την ακρίβεια εκπαίδευσης όσο και για την ακρίβεια ελέγχου για κάθε συστάδα χρηστών.

Από το test_df παίρνουμε τα features X_test και τα labels Y_test ως numpy arrays και κάνουμε evaluate το νευρωνικό δίκτυο με την **evaluate_model()**.

```
def evaluate_model(model: tf.keras.models.Sequential, X_test: np_typing.NDArray, Y_test: np_typing.NDArray, batch_size: int=1):
    """Evaluate the model against the X_test"""
```

```
    # Features as a numpy array
    return model.evaluate(x = X_test, y = Y_test, batch_size=batch_size)


X_test = test_df[NEIGHBOURS_COLUMNS].to_numpy()
Y_test = test_df['USER_RATINGS'].to_numpy()
test_loss, test_mse, test_mae = evaluate_model(my_model, X_test, Y_test, batch_size)
```

Παίρνουμε ως features X_origin τα ratings από το αρχικό μη φιλτραρισμένο **nn_origin_df** και τα χρησιμοποιούμε για να προβλέψουμε τα USER_RATINGS από τους γείτονες μέσω της **predict_model()**. Υπολογίζουμε τα πραγματικά mean_squared_error και mean_absolute_error από τα δεδομένα μας που δεν είναι 0 σε σύγκριση με τα predictions μέσω της **calculate_real_mse_mae()**.

```
def predict_model(model: tf.keras.models.Sequential, X_origin: np_typing.NDArray, batch_size:
int=1):
    """Predict the model with the X_origin"""

    # Features as a numpy array
    return model.predict(x = X_origin, batch_size=batch_size)


def calculate_real_mse_mae(origin_ratings: np_typing.NDArray, predictions: np_typing.NDArray):
    """Calculate the real mse and mae comparing real ratings and predictions."""

    n = 0
    absolute_sum = 0
    squared_sum = 0
    for i in range(origin_ratings.shape[0]):
        # take into consideration only non 0 ratings
        if origin_ratings[i] != 0.0:
            n += 1
            abs_value = abs(origin_ratings[i] - predictions[i])
            absolute_sum += abs_value
            squared_sum += math.sqrt(abs_value)


    mse = absolute_sum / n
    mae = squared_sum / n
    return mse, mae


X_origin = nn_origin_df[NEIGHBOURS_COLUMNS].to_numpy()
predictions_normalized = predict_model(my_model, X_origin, batch_size)
# turn list of sinle item lists to a sigle list with floats
predictions_normalized = np.array([prediction_normalized[0] for prediction_normalized in
predictions_normalized])
predictions = predictions_normalized * ratings_normalize_factor
```

```
real_mse, real_mae = calculate_real_mse_mae(real_ratings, predictions_normalized)
```

Συγκεντρώνουμε όλα τα αποτελέσματα από όλα τα clusters στο DataFrame **results_df**:

'results_df'

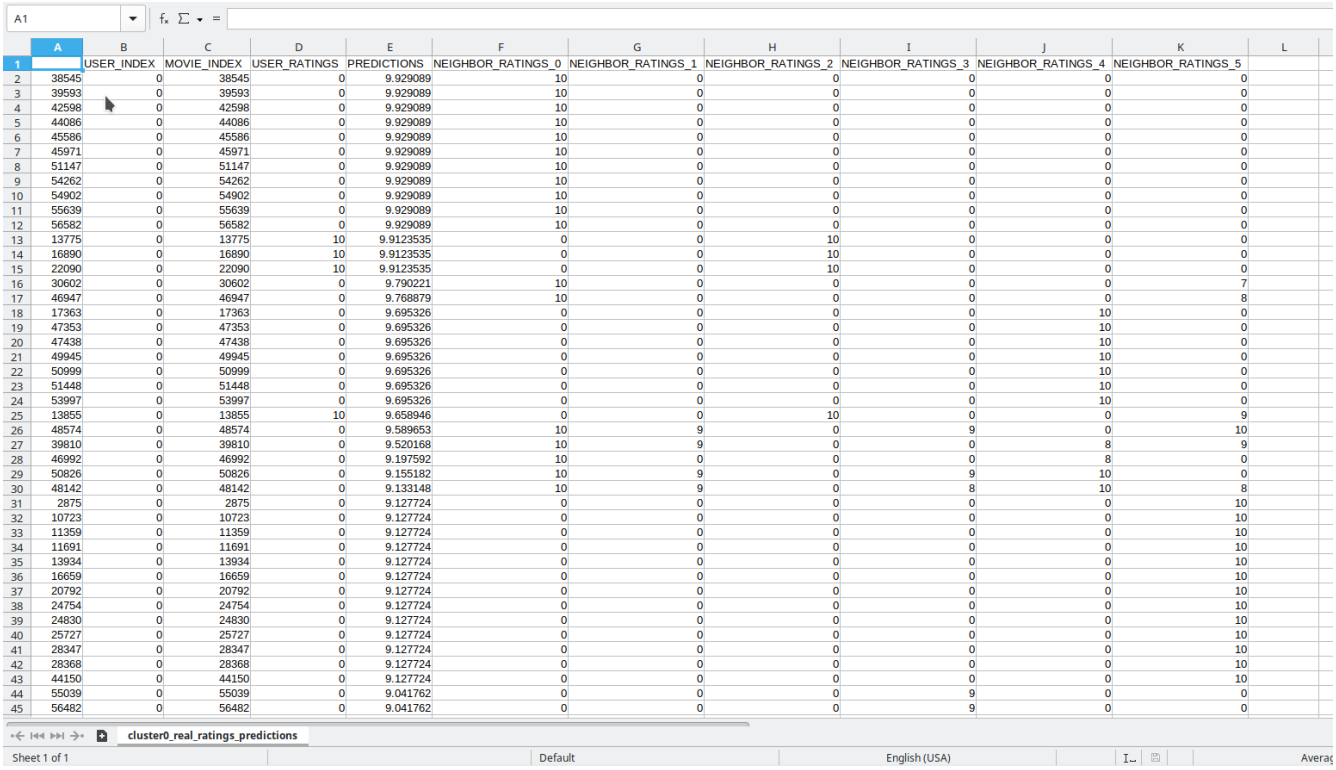
	TRAIN_MSE	TRAIN_MAE	TEST_MSE	TEST_MAE	REAL_MSE	REAL_MAE
CLUSTER_0	0.798611	0.483483	0.831502	0.467930	0.464581	0.528358
CLUSTER_1	0.477400	0.290390	0.493693	0.256128	0.265765	0.334756
CLUSTER_2	0.799668	0.435391	0.817660	0.436078	0.437125	0.519217
CLUSTER_3	0.802961	0.457544	0.877381	0.488667	0.471173	0.545672
CLUSTER_4	1.009662	0.543380	1.085815	0.534414	0.528939	0.584993

	TRAIN_MSE	TRAIN_MAE	TEST_MSE	TEST_MAE	REAL_MSE	REAL_MAE
CLUSTER_0	0.8	0.48	0.83	0.47	0.46	0.53
CLUSTER_1	0.48	0.29	0.49	0.26	0.27	0.33
CLUSTER_2	0.8	0.44	0.82	0.44	0.44	0.52
CLUSTER_3	0.8	0.46	0.88	0.49	0.47	0.55
CLUSTER_4	1.01	0.54	1.09	0.53	0.53	0.58

Για καλύτερη ερμηνεία των προβλέψεων δημιουργούμε το **real_ratings_predictions_df** ως αντίγραφο του **nn_origin_df** και προσθέτουμε το predictions ως στήλη “PREDICTIONS”. Ταξινομούμε ως προς το USER_INDEX ascending και τα PREDICTIONS descending και τα αποθηκεύουμε ως csv.

```
real_ratings_predictions_df = nn_origin_df.copy()
real_ratings_predictions_df['USER_RATINGS'] = real_ratings
real_ratings_predictions_df.insert(3, 'PREDICTIONS', predictions)
real_ratings_predictions_df = real_ratings_predictions_df.sort_values(by=['USER_INDEX',
'PREDICTIONS'], ascending=[True, False])
real_ratings_predictions_df_csv = os.path.join(RESULTS_PATH,
F"cluster{cluster_index}_real_ratings_predictions.csv")
real_ratings_predictions_df.to_csv(real_ratings_predictions_df_csv)
```

Έτσι βλέπουμε για παράδειγμα τα πρώτα rows του **cluster0_real_ratings_predictions.csv**



	A	USER_INDEX	MOVIE_INDEX	USER_RATINGS	PREDICTIONS	NEIGHBOR_RATINGS_0	NEIGHBOR_RATINGS_1	NEIGHBOR_RATINGS_2	NEIGHBOR_RATINGS_3	NEIGHBOR_RATINGS_4	NEIGHBOR_RATINGS_5
2	38545	0	38545	0	9.929089	10	0	0	0	0	0
3	39593	0	39593	0	9.929089	10	0	0	0	0	0
4	42598	0	42598	0	9.929089	10	0	0	0	0	0
5	44086	0	44086	0	9.929089	10	0	0	0	0	0
6	45586	0	45586	0	9.929089	10	0	0	0	0	0
7	45971	0	45971	0	9.929089	10	0	0	0	0	0
8	51147	0	51147	0	9.929089	10	0	0	0	0	0
9	54262	0	54262	0	9.929089	10	0	0	0	0	0
10	54902	0	54902	0	9.929089	10	0	0	0	0	0
11	55639	0	55639	0	9.929089	10	0	0	0	0	0
12	56582	0	56582	0	9.929089	10	0	0	0	0	0
13	13775	0	13775	10	9.9123535	0	0	10	0	0	0
14	16890	0	16890	10	9.9123535	0	0	10	0	0	0
15	22090	0	22090	10	9.9123535	0	0	10	0	0	0
16	30602	0	30602	0	9.790221	10	0	0	0	0	7
17	46947	0	46947	0	9.768879	10	0	0	0	0	8
18	17363	0	17363	0	9.695326	0	0	0	0	10	0
19	47353	0	47353	0	9.695326	0	0	0	0	10	0
20	47438	0	47438	0	9.695326	0	0	0	0	10	0
21	49945	0	49945	0	9.695326	0	0	0	0	10	0
22	50999	0	50999	0	9.695326	0	0	0	0	10	0
23	51448	0	51448	0	9.695326	0	0	0	0	10	0
24	53997	0	53997	0	9.695326	0	0	0	0	10	0
25	13855	0	13855	10	9.658946	0	0	10	0	0	9
26	48574	0	48574	0	9.589653	10	9	0	9	0	10
27	39810	0	39810	0	9.520168	10	9	0	0	8	9
28	46992	0	46992	0	9.197592	10	0	0	0	8	0
29	50826	0	50826	0	9.155182	10	9	0	9	10	0
30	48142	0	48142	0	9.133148	10	9	0	8	10	8
31	2875	0	2875	0	9.127724	0	0	0	0	0	10
32	10723	0	10723	0	9.127724	0	0	0	0	0	10
33	11359	0	11359	0	9.127724	0	0	0	0	0	10
34	11691	0	11691	0	9.127724	0	0	0	0	0	10
35	13934	0	13934	0	9.127724	0	0	0	0	0	10
36	16659	0	16659	0	9.127724	0	0	0	0	0	10
37	20792	0	20792	0	9.127724	0	0	0	0	0	10
38	24754	0	24754	0	9.127724	0	0	0	0	0	10
39	24830	0	24830	0	9.127724	0	0	0	0	0	10
40	25727	0	25727	0	9.127724	0	0	0	0	0	10
41	28347	0	28347	0	9.127724	0	0	0	0	0	10
42	28368	0	28368	0	9.127724	0	0	0	0	0	10
43	44150	0	44150	0	9.127724	0	0	0	0	0	10
44	55039	0	55039	0	9.041762	0	0	0	9	0	0
45	56482	0	56482	0	9.041762	0	0	0	9	0	0

Από αυτά τα αποτελέσματα μπορούμε π.χ. να φτιάξουμε ένα σύστημα προτάσεων ταινιών σε χρήστες βάση των κοντινότερων γειτόνων του. Για παράδειγμα βλέπουμε στο row 2 ότι μπορούμε να προτείνουμε την ταινία με index 38545 (index από τις συνολικές ταινίες) στο χρήστη με cluster index 0 (index από το cluster) καθώς η πρόβλεψη είναι υψηλή (9.93) λόγων των γειτόνων του ενώ ο ίδιος ο χρήστης δεν την έχει βαθμολογήσει.

Παρατηρούμε ότι οι προβλέψεις του νευρωνικού μας δικτύου δεν είναι καλές όταν όλοι οι γείτονες έχουν 0 ratings για μια ταινία (δηλαδή δεν την έχουν βαθμολογήσει), όπως και είναι αναμενόμενο. Σε ένα πραγματικό σύστημα προβλέψεων θα μπορούσαμε να φιλτράρουμε αυτές τις γραμμές όπου όλοι οι γείτονες έχουν 0 ratings ξέροντας ότι δεν μπορούμε να αποφανθούμε σωστό rating σε αυτές τις περιπτώσεις. Μπορούμε να αυξήσουμε τον αριθμό των πλησιέστερων γειτόνων για να έχουμε περισσότερες προβλέψεις όπου τουλάχιστον ένας γείτονας θα έχει κάποιο rating μη μηδενικό, αλλά από τις μετρήσεις μας είδαμε ότι χάνουμε σε ακρίβεια καθώς τα losses αυξάνονται.