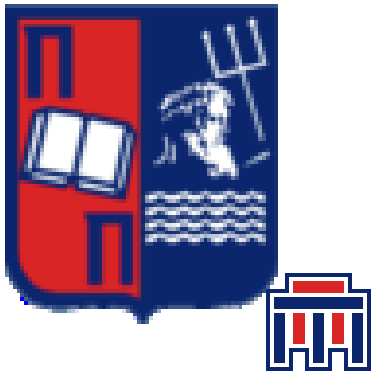
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΜΣ «ΠΛΗΡΟΦΟΡΙΚΗ»



Εργασία Μαθήματος

«Αναγνώριση Προτύπων»

ΣΑΡΑΝΤΙΔΗΣ ΡΩΜΑΝΟΣ ΜΠΠΛ2327

Πειραιάς, 2025

Περιεχόμενα

Προεπεξεργασία Δεδομένων 3

Εύρεση μοναδικών χρηστών U και αντικειμένων I 3

Περιορισμός στα σύνολα U και I βάσει αριθμού αξιολογήσεων 4

Ιστογράμματα για το πλήθος και το χρονικό εύρος αξιολογήσεων ανά χρήστη 5

Αναπαράσταση των αξιολογήσεων ως διανύσματα προτιμήσεων Rj 7

Αλγόριθμοι Ομαδοποίησης Δεδομένων (Clustering) 8

Ομαδοποίηση χρηστών σε συστάδες με χρήση K-Means 8

Χρήση διαφορετικών μετρικών απόστασης 10

Ευκλείδεια Απόσταση 11

Συνημιτονοειδής Απόσταση (Cosine) 11

Γραφική αναπαράσταση 12

Ανάλυση αποτελεσμάτων 15

Παραγωγή συστάσεων με χρήση Τεχνητών Νευρωνικών Δικτύων 16

Συσταδοποίηση χρηστών με χρήση k-means και μετρικής Jaccard distance 16

Ανάλυση της μετρικής 17

Υπολογισμός πλησιέστερων γειτόνων 18

Εκπαίδευση Νευρωνικών Δικτύων 20

Ανάλυση αποτελεσμάτων Νευρωνικών Δικτύων 22

Λειτουργία εκτελούμενου 26

Προεπεξεργασία Δεδομένων

Εύρεση μοναδικών χρηστών U και αντικειμένων I

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

Η αρχική συλλογή των μοναδικών χρηστών και αντικειμένων πραγματοποιείται μέσω της συνάρτησης extract_users_and_movies() στον αρχείο user_matrix.py. Η συνάρτηση αυτή:

- Επεξεργάζεται όλα τα αρχεία .csv που βρίσκονται στον φάκελο Dataset/2_reviews_per_movie_raw.
- Κάθε αρχείο αντιστοιχεί σε μία ταινία και περιέχει εγγραφές με username, rating, και date.
- Οι χρήστες εξάγονται από τα πεδία username, ενώ τα αντικείμενα (ταινίες) εξάγονται από τα ονόματα των αρχείων με μετατροπή σε movie_name.

Απόσπασμα κώδικα:

```
def extract_users_and_movies(folder_path="Dataset/2_reviews_per_movie_raw",
output_path="ModifiedData"):
    os.makedirs(output_path, exist_ok=True)
    files = [f for f in os.listdir(folder_path) if f.endswith('.csv')]

    # Extract movie names
    movie_names = [re.sub(r'.csv$', '', f).replace('_', ':') for f in files]
    movie_df = pd.DataFrame(movie_names, columns=['movie_name'])
    movie_df.to_csv(f"{output_path}/movies.csv", index=False)

    # Combine all reviews
    all_reviews = []
    for f in files:
        df = pd.read_csv(os.path.join(folder_path, f))
        df = df[df['rating'] != 'Null']
        all_reviews.append(df)
    combined_df = pd.concat(all_reviews)

    user_counts = combined_df['username'].value_counts().reset_index()
    user_counts.columns = ['username', 'count']
    users_df = pd.DataFrame(sorted(combined_df['username'].dropna().unique()),
columns=['username'])
    users_df = pd.merge(users_df, user_counts, on='username', how='left')
    users_df.to_csv(f"{output_path}/users.csv", index=False)

    print(f"✓ Extracted {len(users_df)} users and {len(movie_df)} movies.")
```

Η έξοδος αποθηκεύεται στα αρχεία ModifiedData/users.csv και ModifiedData/movies.csv.

*__Σημείωση__: Οι διπλότυπες κριτικές (χρήστης έχει βαθμολογήσει 2+ φορές μια ταινία) απορρίπτονται και κάθε φορά χρησιμοποιείται μόνο η τελευταία κριτική του χρήστη για την εν λόγω ταινία.

Περιορισμός στα σύνολα U και I βάσει αριθμού αξιολογήσεων

"Θεωρείστε την συνάρτηση Φ: $U \rightarrow P(I)$ (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 U και I έτσι ώστε $\forall\, u \in U$, $Rmin \leq |\varphi(u)| \leq Rmax$ όπου $Rmin$ και $Rmax$ ο ελάχιστος απαιτούμενος και ο μέγιστος επιτρεπτός αριθμός αξιολογήσεων ανά χρήστη. Θεωρήστε προφανώς ότι $|U| = n < N$ και $|I| = m < M$."

Η επιλογή των χρηστών που θα ληφθούν υπόψη γίνεται με βάση το πλήθος των αξιολογήσεων που έχουν πραγματοποιήσει. Ο περιορισμός αυτός υλοποιείται στη συνάρτηση filter_users_by_range():

```
    filtered = df[(df['count'] >= min_ratings) & (df['count'] <= max_ratings)]
```

Το φίλτρο αυτό εφαρμόζεται στο users.csv, και κρατά μόνο χρήστες με αριθμό αξιολογήσεων εντός του διαστήματος [Rmin, Rmax] (π.χ., 150–300 αξιολογήσεις). Ο αριθμός των επιλεγμένων χρηστών μειώνεται σε $|U| = n$.

Παράλληλα, κατά την κατασκευή της τελικής μήτρας χρήστη-ταινίας (user_matrix.csv), εξαιρούνται οι ταινίες που έχουν συγκεντρώσει λιγότερες από 50 αξιολογήσεις, για να επιτευχθεί μεγαλύτερη πυκνότητα μετρήσεων και αξιοπιστία του αποτελέσματος:

```
    movie_rating_counts = (matrix_df != 0).sum(axis=0)
    filtered_columns = movie_rating_counts[movie_rating_counts >= 50].index
    matrix_df = matrix_df[filtered_columns]
```

Το αποτέλεσμα είναι ένα πίνακας διαστάσεων n × m, όπου n = $|U|$ και m = $|I|$.

Ιστογράμματα για το πλήθος και το χρονικό εύρος αξιολογήσεων ανά χρήστη

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

Για την ανάλυση της δραστηριότητας των χρηστών, κατασκευάζονται ιστογράμματα που απεικονίζουν:

- Το πλήθος αξιολογήσεων ανά χρήστη.
- Το χρονικό διάστημα μεταξύ πρώτης και τελευταίας αξιολόγησης.

Αυτό γίνεται μέσω της συνάρτησης create_histogramms() στο αρχείο histogramms.py. Αρχικά γίνεται συλλογή όλων των ημερομηνιών των κριτικών ανά χρήστη και κατασκευάζεται ένα DataFrame της ακόλουθης μορφής:

```
    final_df = pd.DataFrame({
      'username': list(user_dates.keys()),
      'count': [len(dates) for dates in user_dates.values()],
      'range': [(max(dates) - min(dates)).days if dates else 0 for dates in user_dates.values()],
      'mean_year': [int(sum(d.year for d in dates) / len(dates)) if dates else 0 for dates in
user_dates.values()]
    })
```

Ακολουθεί η δομή οργάνωσης των στοιχείων για την αποτύπωση του πλήθους και του χρονικού εύρους των βαθμολογήσεων των χρηστών:

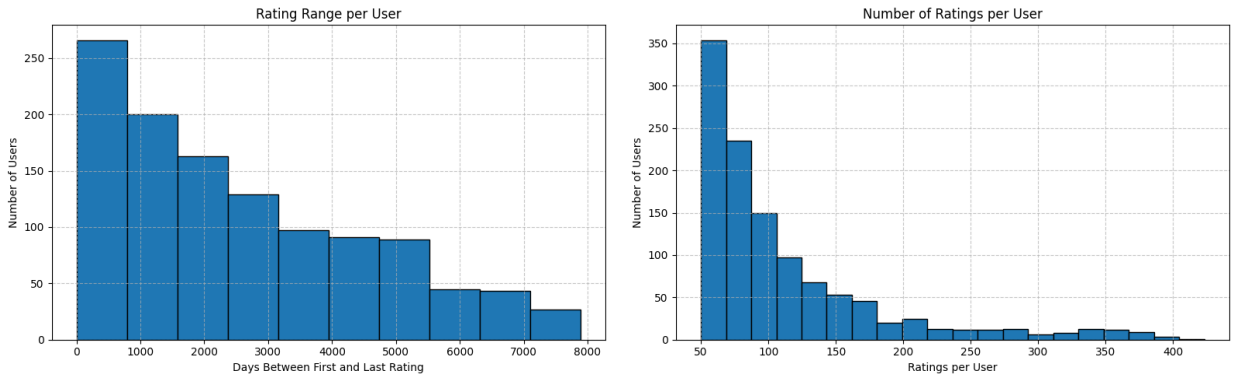
username	count	range	*mean_year*
RickHarvey	56	1241	*2010*
coreyjdenford	69	2031	*2016*
sunznc	54	4664	*2011*
ketgup83	121	3315	*2013*
MickeyTheConstant	53	503	*2019*
...			
vitaleralphlouis	82	4198	*2007*
rgkarim	138	3235	*2015*
rcolgan	58	2295	*2015*
richieandsam	102	685	*2013*
HotToastyRag	103	1109	*2018*
mario_c	59	3225	*2008*

Τα παραπάνω δεδομένα αποθηκεύονται στο user_dates.csv και στη συνέχεια παράγονται δύο ιστογράμματα μέσω matplotlib:

```python
def plot_histograms(df, range_bins=10, count_bins=20, show=True):
    # Plot rating range histogram
    plt.figure(figsize=(8, 5))
    plt.hist(df['range'], bins=range_bins, edgecolor='black')
    plt.ylabel('Number of Users')
    plt.xlabel('Days Between First and Last Rating')
    plt.title('Rating Range per User')
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.tight_layout()
    if show:
        plt.show()

    # Plot count histogram
    plt.figure(figsize=(8, 5))
    plt.hist(df['count'], bins=count_bins, edgecolor='black')
    plt.ylabel('Number of Users')
    plt.xlabel('Ratings per User')
    plt.title('Number of Ratings per User')
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.tight_layout()
    if show:
        plt.show()
```

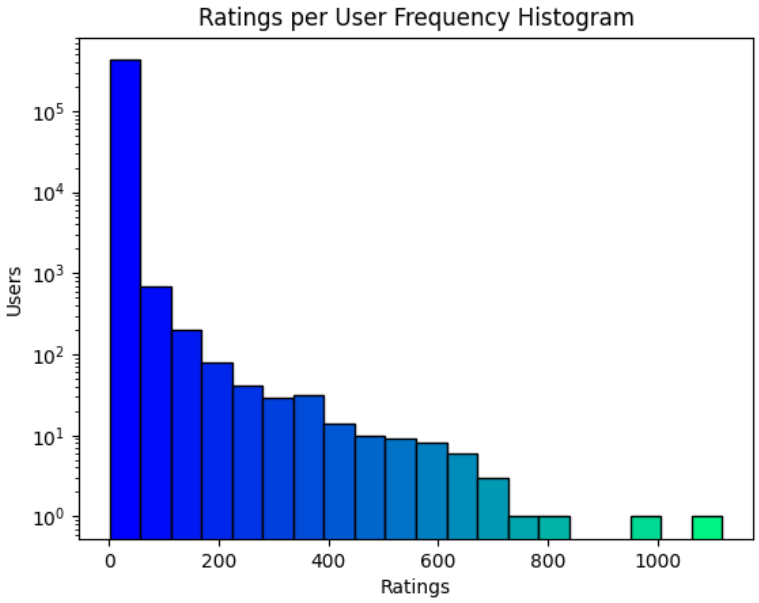
Ακολουθούν τα παραπάνω ιστογράμματα:



Ιστογράμματα συχνοτήτων για την χρονικό εύρος (σε ημέρες) κριτικών των χρηστών (αριστερα) – πλήθος κριτικών ανά χρήστη (δεξιά)

Όπως φαίνεται από τα διαγράμματα, έχει πραγματοποιηθεί περιορισμός των δεδομένων για χρήστες με τουλάχιστον 50 κριτικές και λιγότερες από 400, για να απορριφθούν τα άκρα του συνόλου. Αυτό μειώνει δραστικά το σύνολο των εγγραφών, όπως φαίνεται και από το

ακόλουθο διάγραμμα η συντριπτική πλειοψηφία έχει βαθμολογήσει μόνο 1 ταινία. Ωστόσο είναι απολύτως απαραίτητο για την αξιοπιστία του συνόλου.



Αναπαράσταση των αξιολογήσεων ως διανύσματα προτιμήσεων *Rj*

"Δημιουργήστε μια εναλλακτική αναπαράσταση του συνόλου των δεδομένων ως ένα σύνολο διανυσμάτων προτιμήσεων $\boldsymbol{R} = \{\boldsymbol{R1}, \boldsymbol{R2}, \cdots, \boldsymbol{Rn}\}$ *με* $\boldsymbol{Rj} = \boldsymbol{R}(uj) \in \boldsymbol{Ro\ m}$, $\forall j \in [n]$. *Συγκεκριμένα, μπορούμε να γράψουμε ότι:"*

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

Η τελική αναπαράσταση του συνόλου δεδομένων είναι ένας πίνακας διανυσμάτων προτιμήσεων $\boldsymbol{R} = \{\boldsymbol{R}_1, \boldsymbol{R}_2, \ldots, \boldsymbol{R}_n\}$, όπου κάθε ***Rj*** αντιστοιχεί στις αξιολογήσεις ενός χρήστη για τα m φιλτραρισμένα αντικείμενα. Ο πίνακας αυτός δημιουργείται με τη συνάρτηση build_user_movie_matrix():

```
    users = pd.read_csv(users_file)
movies = pd.read_csv(movies_file)
user_list = users['username'].tolist()
```

```
    movie_list = movies['movie_name'].tolist()
    user_index_map = {user: i for i, user in enumerate(user_list)}
    movie_index_map = {movie: j for j, movie in enumerate(movie_list)}
    matrixR = np.zeros((len(user_list), len(movie_list)), dtype=np.float32)
    for file in os.listdir(reviews_folder):
        movie = re.sub(r'.csv$', '', file).replace('_', ':')
        if movie not in movie_index_map:
            continue
        try:
            ratings = pd.read_csv(os.path.join(reviews_folder, file))
        except Exception as e:
            print(f"Error reading file {file}: {e}")
            continue
        ratings = ratings[ratings['username'].isin(user_index_map)]
        for _, row in ratings.iterrows():
            if pd.notna(row['rating']) and row['rating'] != 'Null':
                matrixR[user_index_map[row['username']], movie_index_map[movie]] =
float(row['rating'])
    matrix_df = pd.DataFrame(matrixR, index=user_list, columns=movie_list)
    matrix_df.index.name = 'username'
    # Remove movies with too few ratings
    movie_rating_counts = (matrix_df != 0).sum(axis=0)
    filtered_columns = movie_rating_counts[movie_rating_counts >= 50].index
    matrix_df = matrix_df[filtered_columns]
    matrix_df.to_csv(output_file)
```

Η προεπιλεγμένη τιμή 0 για μη αξιολογημένα αντικείμενα καθιστά τον πίνακα αραιό (sparse), καθώς η πλειονότητα των θέσεων παραμένει μηδενική — γεγονός που αντανακλά την πραγματικότητα, όπου κάθε χρήστης αξιολογεί μόνο ένα μικρό υποσύνολο των διαθέσιμων ταινιών.

```
    for _, row in ratings.iterrows():
        if pd.notna(row['rating']) and row['rating'] != 'Null':
            matrixR[user_index_map[row['username']], movie_index_map[movie]] = loat(row['rating'])
```

Αλγόριθμοι Ομαδοποίησης Δεδομένων (Clustering)

Ομαδοποίηση χρηστών σε συστάδες με χρήση K-Means

"Να οργανώσετε το περιορισμένο σύνολο των χρηστών $\boldsymbol{U}$ *σε* L *συστάδες (clusters) της μορφής* $\boldsymbol{U} = \boldsymbol{U_{G1}} \cup \boldsymbol{U_{G2}} \cup \cdots \cup \boldsymbol{U_{GL}}$ *έτσι ώστε* $\boldsymbol{U_{Ga}} \cap \boldsymbol{U_{Gb}} \neq \emptyset,\ \forall\ \boldsymbol{a} \neq \boldsymbol{b}$ *βασιζόμενοι στην διανυσματική αναπαράσταση των προτιμήσεών τους μέσω του συνόλου* $\boldsymbol{R}$*. Η διαδικασία ομαδοποίησης των*

διανυσμάτων προτιμήσεων των χρηστών $\boldsymbol{R_j} = \boldsymbol{R}(u_j) \in \boldsymbol{R_o^m}$, $\forall\, j \in [n]$ *επάνω στο περιορισμένο σύνολο των αντικειμένων* $\boldsymbol{I}$, *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* $\boldsymbol{dist}(\boldsymbol{R_u}, \boldsymbol{R_v})$.”

Η ομαδοποίηση των χρηστών πραγματοποιείται με βάση τα διανύσματα προτιμήσεων $\boldsymbol{R_j} \in \mathbb{R}^m$, όπου m ο αριθμός των φιλτραρισμένων ταινιών. Η διαδικασία υλοποιείται από τη συνάρτηση run_kmeans_clustering() του αρχείου KMeans.py.

Αρχικά διαβάζεται το αρχείο user_matrix.csv και εξάγονται τα διανύσματα των χρηστών:

```
    df = pd.read_csv("ModifiedData/user_matrix.csv")
    usernames = df['username']
    users = df.drop('username', axis=1).to_numpy()
```

Η ομαδοποίηση πραγματοποιείται με παραλλαγή του αλγορίθμου **KMeans++**:

```
def kmeans_plus_plus(users, k):
    centroids = [users[np.random.randint(users.shape[0])]]
    for _ in range(1, k):
        dist_sq = np.min([np.sum((users - c) ** 2, axis=1) for c in centroids], axis=0)
        prob = dist_sq / dist_sq.sum()
        centroids.append(users[np.random.choice(users.shape[0], p=prob)])
    return np.array(centroids)
```

Για την αρχικοποίηση των κέντρων επιλέγεται το πρώτο στην τύχη. Ακολούθως τα $k - 1$ επιλέγονται με σταθμισμένη δειγματοληψία, όπου ο βαθμός βαρύτητας των κέντρων υπολογίζεται από την απόσταση τους από τα υπάρχοντα κέντρα, με χρήση Ευκλείδειας απόστασης. Έτσι διασφαλίζεται καλύτερη κατανομή των κέντρων δίνοντας προβάδισμα στον αλγόριθμο συσταδοποίησης.

Σε κάθε επανάληψη του αλγόριθμου k-means:

- Υπολογίζονται οι αποστάσεις όλων των χρηστών από κάθε κέντρο συστάδας.
- Επαναϋπολογίζονται τα κέντρα (centroids) ως μέσος όρος των διανυσμάτων της συστάδας, αγνοώντας τα μηδενικά (δηλαδή μη διαθέσιμες αξιολογήσεις):

```
    centroids = kmeans_plus_plus(users, k)
    distances = pd.DataFrame(0.0, index=usernames, columns=[f"Cluster {i+1}" for i in range(k)])
    previous_clusters = None
    max_iterations = 50

    for iteration in range(max_iterations):
        for i in range(k):
            for j in range(users.shape[0]):
                distances.iloc[j, i] = metric_func(users[j], centroids[i])
        cluster_assignments = distances.idxmin(axis=1)

        new_centroids = []
        for m in range(k):
            cluster_users = users[cluster_assignments == f"Cluster {m+1}"]
            if len(cluster_users) > 0:
                mask = cluster_users != 0
                sums = cluster_users.sum(axis=0)
                counts = mask.sum(axis=0)
                avg = np.divide(sums, counts, out=np.zeros_like(sums), where=counts != 0)
                new_centroids.append(avg)
            else:
                new_centroids.append(users[np.random.randint(users.shape[0])])
        centroids = np.array(new_centroids)
```

Η διαδικασία επαναλαμβάνεται για 50 επαναλήψεις ή μέχρι να μην πραγματοποιείται αλλαγή στις συστάδες σημείων.

```
        if previous_clusters is not None and previous_clusters.equals(cluster_assignments):
            print(f"✓ Converged at iteration {iteration}")
            break
        previous_clusters = cluster_assignments.copy()
```

Χρήση διαφορετικών μετρικών απόστασης

Ο υπολογισμός της απόστασης μεταξύ δύο διανυσμάτων προτιμήσεων γίνεται με τρεις εναλλακτικές μετρικές, όπως ορίζονται στην εκφώνηση. Οι υπάρχουσες βιβλιοθήκες για τον αλγόριθμό k-means δεν υποστηρίζουν παραμετροποίηση της εφαρμοζόμενης μετρικής για τον υπολογισμό της απόστασης. Οι λύσεις σε αυτό το πρόβλημα είναι πιθανότατα είτε χρήση πίνακα αποστάσεων με προηγουμένως εφαρμογή των αντίστοιχων μετρικών ή κατασκευή αλγορίθμου εκ νέου. Στα πλαίσια της εργασίας επέλεξα να υλοποιήσω το δεύτερο. Όπου δυναμικά επιλέγεται κάθε φορά η μετρική προς εφαρμογή (`metric_func(users[j], centroids[i])`).

Ευκλείδεια Απόσταση

"$\boldsymbol{dist_{euclidean}}(\boldsymbol{R_u}, \boldsymbol{R_v}) = \sqrt{\sum_{k=1}^{m} |\boldsymbol{R_u}(k) - \boldsymbol{R_v}(k)|^2 \lambda_u(k)\, \lambda_v(k)}$ *(2) έτσι ώστε:*

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

Η συνάρτηση λj() αποτιμά το αν ο χρήστης uj έχει αξιολογήσει ή όχι το αντικείμενο ίk. Προφανώς, μέσω της συγκεκριμένης συνάρτηση μπορούμε αν υπολογίσουμε την τιμή του |φ(u)| που αποτιμά το πλήθος των αξιολογήσεων που παρείχε συνολικά ο χρήστης u ως:

$|\varphi(u)| = \sum_{k=1}^{n} \lambda_u(k)$"

Η παραπάνω μετρική υλοποιείται ως ακολούθως:

```
def masked_euclidean(u, v):
    mask = (u != 0) & (v != 0)
    return np.sqrt(np.sum((u[mask] - v[mask]) ** 2)) if np.any(mask) else np.inf
```

Ενώ το ζητούμενο "**one-hot encode**", ούτως ώστε να υπολογίζεται μόνο η απόσταση σε περίπτωση που έχουν βαθμολογήσει και οι δυο χρήστες την εν λόγω ταινία, πραγματοποιείται με τη βοήθεια μάσκας (`mask = (u != 0) & (v != 0)`).

Συνημιτονοειδής Απόσταση (Cosine)

" $\boldsymbol{dist_{cosine}}(\boldsymbol{R_u}, \boldsymbol{R_v}) = 1 - \left| \frac{\sum_{k=1}^{m} \boldsymbol{R_u}(k) \boldsymbol{R_v}(k) \lambda_u(k) \lambda_v(k)}{\sqrt{\sum_{k=1}^{m} \boldsymbol{R_u}(k)^2 \lambda_u(k) \lambda_v(k)} \sqrt{\sum_{k=1}^{m} \boldsymbol{R_v}(k)^2 \lambda_u(k) \lambda_v(k)}} \right|$ *(4)*"

Η μετρική στης συνημιτονοειδούς απόστασης υλοποιείται, όπως φαίνεται παρακάτω:

```
def masked_cosine(u, v):
    mask = (u != 0) & (v != 0)
    if not np.any(mask): return 1.0
    u_m, v_m = u[mask], v[mask]
    denom = np.linalg.norm(u_m) * np.linalg.norm(v_m)
    return 1 - (np.dot(u_m, v_m) / denom) if denom else 1.0
```

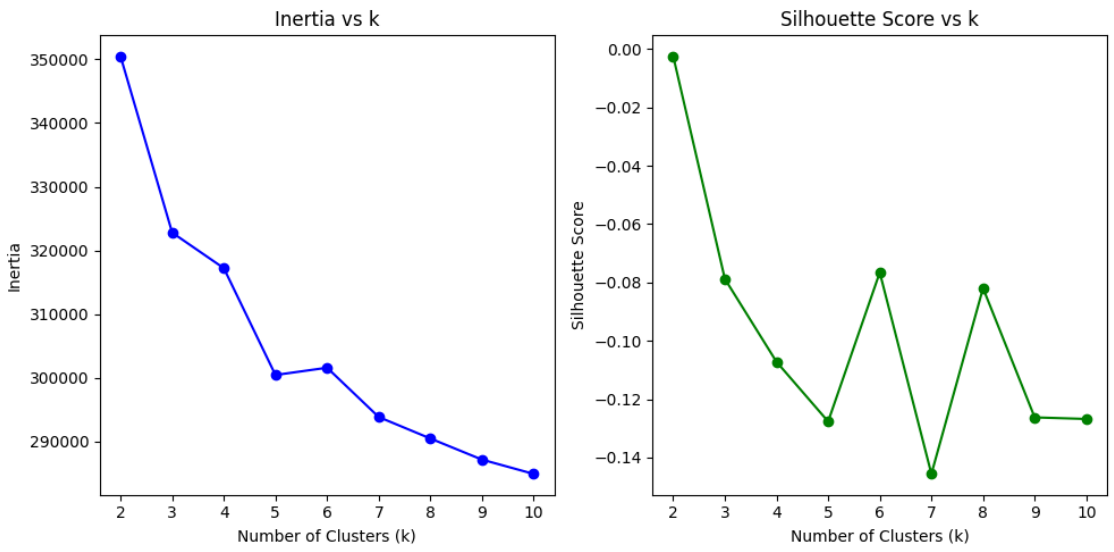
Ενώ χρησιμοποιείται ξανά μάσκα για την αποτύπωση του "**one-hot encode**".

Γραφική αναπαράσταση

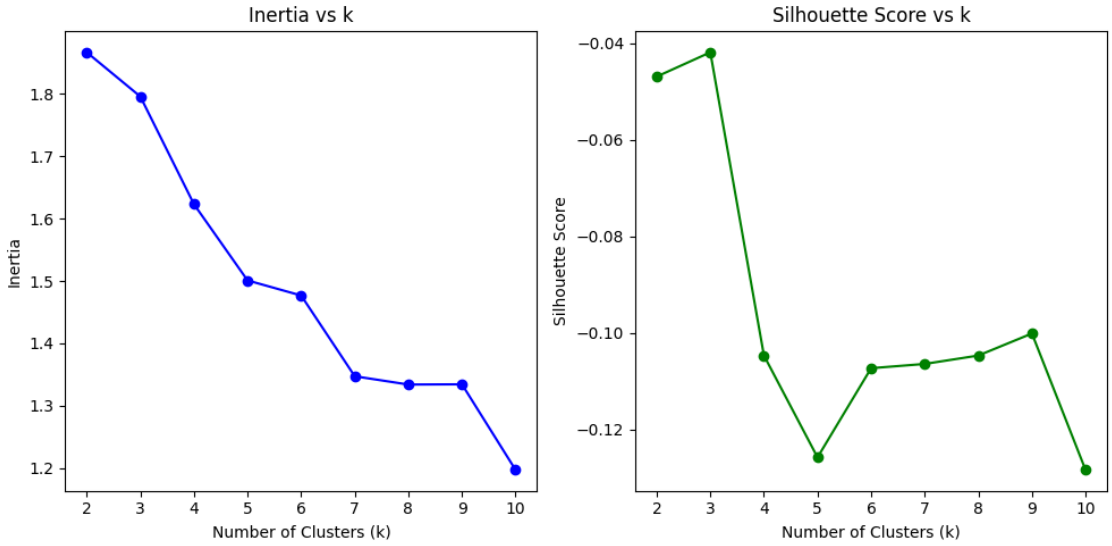
"*Να αναπαραστήσετε γραφικά τις συστάδες των χρηστών που αναγνωρίστηκαν από τον αλγόριθμο k-means για κάθε μία από τις παραπάνω μετρικές για διάφορες τιμές της παραμέτρου L.*"

Κατά την εφαρμογή του αλγορίθμου k-means μετρώντας την απόδοση με τις μετρικές inertia και silhouette score, παρατηρήθηκαν τα εξής αποτελέσματα για τις αποστάσεις Euclidean και Cosine.

Για την απόσταση **Euclidean**, η καλύτερη τιμή του *silhouette score* εντοπίστηκε στο k=2, με τιμή -0.0025, που αν και είναι αρνητική, είναι η λιγότερο αρνητική συγκριτικά με τις υπόλοιπες τιμές του k. Η τιμή της *inertia* σε αυτή την περίπτωση ήταν 350485.29. Καθώς αυξανόταν το k, η inertia μειωνόταν όπως είναι αναμενόμενο, ωστόσο το silhouette score επιδεινωνόταν, υποδεικνύοντας χειρότερη ποιότητα συστάδων. Αυτό δείχνει ότι οι συστάδες δεν είναι ξεκάθαρα διαχωρισμένες και υπάρχει σημαντική επικάλυψη μεταξύ των χρηστών ως προς τα χαρακτηριστικά τους, πιθανότατα λόγω της αραιότητας των δεδομένων. Ωστόσο βάση της *elbow method* από το "Inertia vs k" διάγραμμα, προκύπτει ότι η βέλτιστη τιμή είναι k = 3.



Για την απόσταση **Cosine**, η καλύτερη τιμή του *silhouette score* παρατηρήθηκε στο k=3, με τιμή -0.0419, η οποία επίσης είναι αρνητική. Η *inertia* σε αυτή την περίπτωση ήταν 1.80. Όπως και στην περίπτωση της Euclidean απόστασης, όσο αυξανόταν το k η inertia μειωνόταν, αλλά το silhouette score δεν παρουσίαζε σαφή βελτίωση. Αντίθετα, υπήρχε επιδείνωση ή μικρές διακυμάνσεις που παρέμεναν σε αρνητικές τιμές. Με βάση την elbow method βέλτιστο το βέλτιστο k εκτιμάται ότι είναι k = 5. Ωστόσο αυτό αποτελεί τοπικό ελάχιστο στο διάγραμμα “Silhouette Score vs k”, επομένως αποτελεσματικότερη συσταδοποίηση θεωρείται ότι θα επιτευχθεί για τιμές του k μεταξύ 3 και 5.



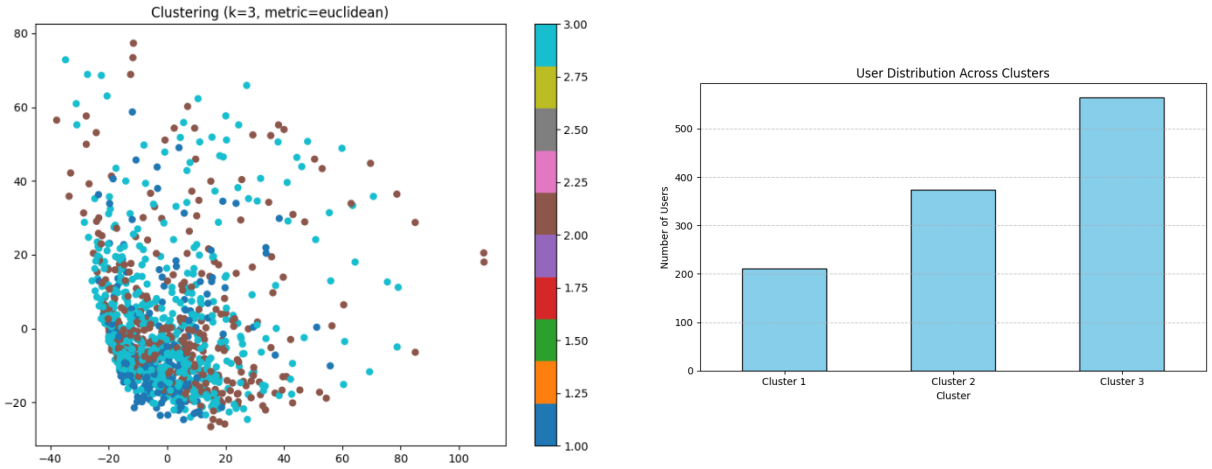
Για να επιτευχθεί γραφική αναπαράσταση της ομαδοποίησης, απαιτείται περαιτέρω επεξεργασία. Τα διανύσματα των χρηστών είναι πολυδιάστατα, καθώς αποτελούνται από βαθμολογία ή απουσία της (0) στο σύνολο των ταινιών του Dataset. Για να επιτευχθεί η αποτύπωση σε δισδιάστατο χώρο γίνεται χρήση της μεθόδου **PCA** (Principal Component Analysis), όπου μειώνει τις διαστάσεις των δεδομένων (2D) αναλύοντας τις κύριες συνιστώσες. Ακολουθεί δείγμα κώδικα για την εφαρμογή της μεθόδου και την γραφική αναπαράσταση, με τη χρήση των βιβλιοθηκών `matplotlib.pyplot`, `sklearn.decomposition`:

```
# PCA visualization
pca = PCA(n_components=2)
users_2d = pca.fit_transform(users)
plt.figure(figsize=(8, 6))
scatter = plt.scatter(users_2d[:, 0], users_2d[:, 1], c=labels, cmap='tab10', s=30)
plt.title(f"Clustering (k={k}, metric={distance_name})")
plt.colorbar(scatter)
plt.tight_layout()
plt.show()
```

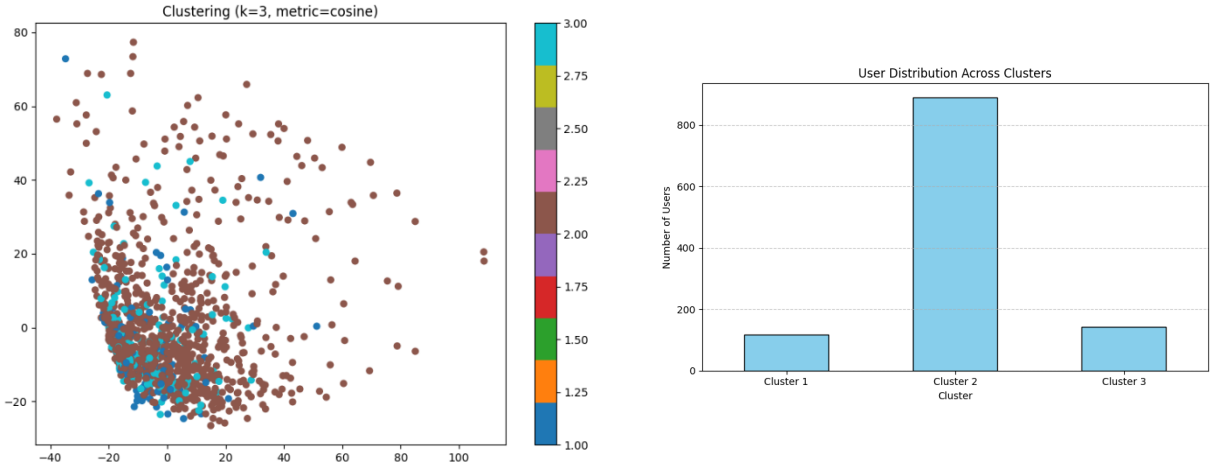
Ενώ παράλληλα υλοποιήθηκε και ραβδόγραμμα για την αναπαράσταση της ποσοτικής κατανομής των χρηστών στις ομάδες. Ακολουθεί η υλοποίηση του διαγράμματος:

```python
# Cluster distribution
cluster_counts = cluster_assignments.value_counts()
plt.figure(figsize=(8, 5))
cluster_counts.sort_index().plot(kind='bar', color='skyblue', edgecolor='black')
plt.xlabel("Cluster")
plt.ylabel("Number of Users")
plt.title("User Distribution Across Clusters")
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

Με βάση τα παραπάνω ακολουθούν τα διαγράμματα των μετρικών Euclidean και Cosine Distance:



Διαγράμματα Διασποράς με χρήση μετρικής Ευκλείδειας Απόστασης (k=3)
(αριστερά – διάγραμμα διασποράς σε 2D χώρο, δεξιά – ραβδόγραμμα κατανομής χρηστών στις συστάδες)



Διαγράμματα Διασποράς με χρήση μετρικής Συνημιτονοειδούς Απόστασης (k=3)
(αριστερά – διάγραμμα διασποράς σε 2D χώρο, δεξιά – ραβδόγραμμα κατανομής χρηστών στις συστάδες)

Ανάλυση αποτελεσμάτων

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

Η αποτελεσματικότητα κάθε μετρικής για την αποτίμηση της ομοιότητας μεταξύ δύο διανυσμάτων προτιμήσεων χρηστών Ru και Rv εξαρτάται από τη φύση των δεδομένων και την πληροφορία που θεωρείται σημαντική για τη σύγκριση χρηστών. Εξετάστηκαν δύο μετρικές: η ευκλείδεια απόσταση και η απόσταση συνημιτόνου.

Η ευκλείδεια απόσταση μετρά το απόλυτο μέγεθος της διαφοράς μεταξύ δύο διανυσμάτων. Λαμβάνει υπόψη τόσο τις ομοιότητες όσο και τις διαφορές στις τιμές των αξιολογήσεων. Σε περιβάλλοντα με αραιά ή ελλιπή δεδομένα, όπως χρήστες που έχουν αξιολογήσει διαφορετικές ταινίες, η ευκλείδεια απόσταση επηρεάζεται από μη κοινούς όρους, οδηγώντας σε μη αξιόπιστες αποστάσεις. Οι τιμές silhouette που προέκυψαν ήταν όλες αρνητικές, με μικρή διακύμανση για τιμές k από 2 έως 10, υποδεικνύοντας ότι οι ομάδες που σχηματίστηκαν δεν ήταν καλά διαχωρισμένες.

Η απόσταση συνημιτόνου εστιάζει στη γωνία μεταξύ των διανυσμάτων, δηλαδή στο σχήμα των προτιμήσεων, και όχι στο απόλυτο μέγεθος των αξιολογήσεων. Είναι ιδιαίτερα χρήσιμη σε περιπτώσεις αραιών διανυσμάτων, όπου η κατεύθυνση των προτιμήσεων έχει μεγαλύτερη σημασία από το μέγεθός τους. Οι τιμές silhouette ήταν επίσης αρνητικές, αλλά

λιγότερο έντονα σε σχέση με την ευκλείδεια απόσταση, ιδιαίτερα για μικρές τιμές του k, υποδηλώνοντας ελαφρώς καλύτερη συσχέτιση των ομάδων.

Καμία από τις δύο μετρικές δεν οδήγησε σε καλά διαχωρισμένες ομάδες, όπως φαίνεται από τις αρνητικές τιμές silhouette. Ωστόσο, η απόσταση συνημιτόνου φαίνεται ελαφρώς πιο κατάλληλη για τα δεδομένα προτιμήσεων χρηστών, καθώς εστιάζει στη σχετική μορφή των αξιολογήσεων και είναι πιο ανθεκτική σε αραιά ή μη ομοιόμορφα δεδομένα. Η ευκλείδεια απόσταση, αντίθετα, είναι πιο ευαίσθητη σε μεγάλες αποκλίσεις τιμών, κάτι που μπορεί να μην είναι επιθυμητό σε αυτό το πλαίσιο.

Παραγωγή συστάσεων με χρήση Τεχνητών Νευρωνικών Δικτύων

Συσταδοποίηση χρηστών με χρήση k-means και μετρικής Jaccard distance

"Να δημιουργήσετε μια εναλλακτική οργάνωση του περιορισμένου συνόλου των χρηστών σε L συστάδες $\boldsymbol{U} = \boldsymbol{U_{G1}} \cup \boldsymbol{U_{G2}} \cup \cdots \cup \boldsymbol{U_{GL}}$ έτσι ώστε $\boldsymbol{U_{Ga}} \cap \boldsymbol{U_{Gb}} \neq \emptyset$, $\forall \boldsymbol{a}, \boldsymbol{b} \in [\boldsymbol{L}]$ με $\boldsymbol{a} \neq \boldsymbol{b}$ κάνοντας χρήση της παρακάτω μετρικής:

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

Η χρήση της παραπάνω μετρικής έγινε ξανά στα πλαίσια του προσαρμοσμένου αλγορίθμου k-means, όπως και οι προηγούμενες μετρικές:

```
def jaccard_distance(u, v):
    u_bin, v_bin = u != 0, v != 0
    inter, union = np.sum(u_bin & v_bin), np.sum(u_bin | v_bin)
    return 1 - (inter / union) if union else 1.0
```

Ο χρήστης καλείται να επιλέξει μετρική μέσω terminal:

```
    print("Choose a distance metric:")
    print("  1. Euclidean")
    print("  2. Cosine")
    print("  3. Jaccard")
    choice = input("Enter your choice (1-3): ")
    distance_name = {"1": "euclidean", "2": "cosine", "3": "jaccard"}.get(choice)
```

Ενώ η επιλογή του εφαρμόζεται στον αλγόριθμο k-means με την βοήθεια της συνάρτησης `get_distance_function`:

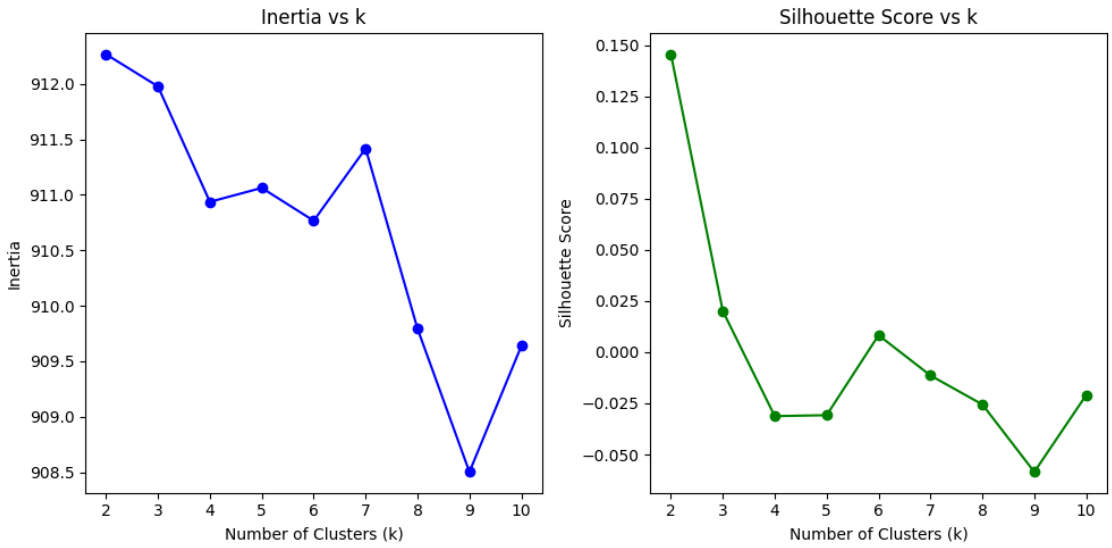
```python
def get_distance_function(name):
    return {
        "euclidean": masked_euclidean,
        "cosine": masked_cosine,
        "jaccard": jaccard_distance
    }.get(name)
```

Αφού ολοκληρωθεί η ομαδοποίηση, εφαρμόζεται ξανά **PCA** για να μειωθεί η διάσταση των διανυσμάτων σε **2D** και να παραχθεί οπτική απεικόνιση των συστάδων.

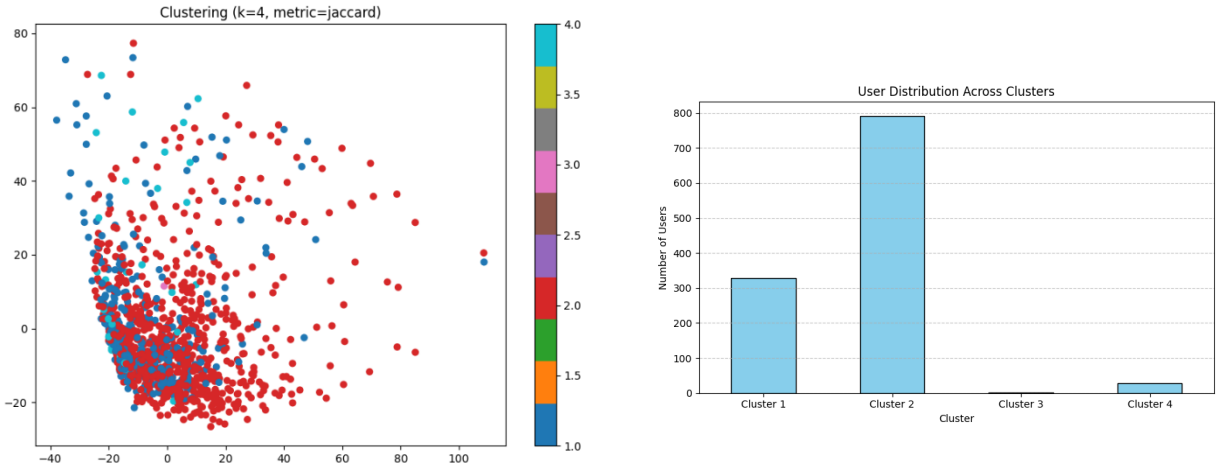
Ανάλυση της μετρικής

"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 **Jaccard** αξιολογεί την ομοιότητα ως προς το *σύνολο* αξιολογημένων αντικειμένων, αγνοεί τις τιμές των αξιολογήσεων και εστιάζει στην ύπαρξη/απουσία τους. Είναι χρήσιμη όταν η πληροφορία προέρχεται κυρίως από το ποια ταινία έχει δει κάποιος, όχι πώς την αξιολόγησε. Μπορεί να ενισχύσει την ανάλυση δεδομένων σε πειράματα με υψηλή διασπορά των στοιχείων και χωρίς εμφανή σύνδεση. Το σημαντικό της μειονέκτημα είναι ότι δεν λαμβάνει υπόψη την τιμή της αξιολόγησης. Παραδείγματος χάριν δυο χρήστες που έχουν βαθμολογήσει τις ίδιες ταινίες διαφορετικά (χαμηλά-υψηλά) μπορούν να συνδεθούν, παρόλο που έχουν εκτιμήσει εντελώς αντίθετα τις κοινές τους ταινίες. Έτσι εξηγείται και το γεγονός ότι η μετρική Jaccard σημείωσε τις υψηλότερες τιμές σε silhouette score σε σχέση με τις υπόλοιπες. Ακολουθούν τα αντίστοιχα διαγράμματα και για αυτή τη μετρική. Η υψηλότερη τιμή σε silhouette score εντοπίζεται για k = 2 (0.0963 – αποτελεί και τη μόνη περίσταση όπου εντοπίζεται θετική σε σχέση με τις υπόλοιπες μετρικές). Η τιμή της ωστόσο, παραμένει πολύ χαμηλά σε σχέση με το επιθυμητό για ευδιάκριτες συστάδες χρηστών (κοντά στην μονάδα). Στο διάγραμμα "Inertia vs k" εντοπίζεται η βέλτιστη τιμή για των αριθμό ομάδων ως 4, με βάση την "elbow method", καθώς εκεί παρατηρείται η μεγαλύτερη επιβράδυνση μείωσης του εσωτερικού τετραγωνικού αθροίσματος των ομάδων (WCSS – Within Clusters Sum of Squares).



Ακολουθούν τα διαγράμματα διασποράς των συστάδων με βάση την απόσταση Jaccard:



Διαγράμματα Διασποράς με χρήση μετρικής Απόστασης Jaccard (k=4)
(αριστερά – διάγραμμα διασποράς σε 2D χώρο, δεξιά – ραβδόγραμμα κατανομής χρηστών στις συστάδες)

Υπολογισμός πλησιέστερων γειτόνων

"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* $u_a \in \boldsymbol{U}_{Ga}$, $\forall \boldsymbol{a} \in [\boldsymbol{L}]$. *Επομένως, για τον εκάστοτε χρήστη* u_a *της εκάστοτε συστάδας* $\boldsymbol{U}_{Ga}$ *μπορούμε να το διάνυσμα των προσωπικών του προτιμήσεων* R_{ua} *καθώς και τα διανύσματα των* k

πλησιέστερων γειτόνων του $R_{ua}^{(1)}$, $R_{ua}^{(2)}$,..., $R_{ua}^{(\kappa)}$ *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:*

$$R_{u_a} = f_a\left(R_{u_a}{}^{(1)}, R_{u_a}{}^{(2)}, \dots R_{u_a}{}^{(k)}\right), \forall\, u_a \in U_{G_a}, \forall\, \alpha \in [L] \ \textbf{(6)"}$$

Για να υλοποιηθεί η συνάρτηση f_a απαιτούνται οι αποστάσεις και η κατανομή των χρηστών σε κάθε συστάδα. Ο παρακάτω αλγόριθμος φορτώνει αυτά τα στοιχεία από την συσταδοποίηση που ήδη έχει πραγματοποιηθεί με kmeans και μετρική την απόσταση Jaccard. Για κάθε χρήστη υπολογίζει ένα σύνολο κοντινότερων γειτόνων (προεπιλογή – 6) στην αντίστοιχη συστάδα. Και τους αποθηκεύει σε νέο αρχείο "`nearest_neighbors.csv`".

```python
def generate_nearest_neighbors(k=6,
                               distances_path="Prediction/distances.csv",
                               clusters_path="ModifiedData/clusters.csv",
                               output_path="Prediction/nearest_neighbors.csv"):
    if not os.path.exists(distances_path):
        print(f"✗ Missing: {distances_path}")
        return
    if not os.path.exists(clusters_path):
        print(f"✗ Missing: {clusters_path}")
        return

    print(f"✓ Loading distances from: {distances_path}")
    distances_df = pd.read_csv(distances_path, index_col=0)
    distances_df.index = distances_df.index.astype(str).str.strip()
    distances_df.columns = distances_df.columns.astype(str).str.strip()

    # Filter to valid usernames in both rows and columns
    valid_usernames = distances_df.index.intersection(distances_df.columns)
    distances_df = distances_df.loc[valid_usernames, valid_usernames]

    clusters_df = pd.read_csv(clusters_path)
    clusters_df.columns = ["username", "cluster"]
    clusters_df["username"] = clusters_df["username"].astype(str).str.strip()
    cluster_series = clusters_df.set_index("username")["cluster"]

    valid_users = distances_df.index.intersection(cluster_series.index)

    neighbors_dict = {
        "username": [],
        "cluster": [],
        **{f"neighbor_{i+1}": [] for i in range(k)}
    }
```

```
    for username in valid_users:
        cluster_label = cluster_series[username]
        same_cluster_users = cluster_series[cluster_series == cluster_label].index
        same_cluster_users = same_cluster_users.intersection(distances_df.columns)

        try:
            user_distances = distances_df.loc[username, same_cluster_users]
            user_distances = user_distances.drop(username, errors='ignore')
            nearest_neighbors = user_distances.nsmallest(k).index.tolist()

            neighbors_dict["username"].append(username)
            neighbors_dict["cluster"].append(cluster_label)
            for i in range(k):
                neighbors_dict[f"neighbor_{i+1}"].append(
                    nearest_neighbors[i] if i < len(nearest_neighbors) else None
                )

        except KeyError as e:
            print(f"Skipping {username} (missing data): {e}")

    os.makedirs(os.path.dirname(output_path), exist_ok=True)
    neighbors_df = pd.DataFrame(neighbors_dict)
    neighbors_df.to_csv(output_path, index=False)

    print(f"[✓] Nearest neighbors saved to: {output_path}")
```

Εκπαίδευση Νευρωνικών Δικτύων

"Το σύνολο των χρηστών της κάθε συστάδας μπορεί να διαμεριστεί περεταίρω σε ένα υποσύνολο χρηστών για εκπαίδευση του νευρωνικού δικτύου U_{Ga}^{train} *και σε ένα υποσύνολο χρηστών για τον έλεγχο της επίδοσης του νευρωνικού δικτύου* U_{Ga}^{test} *έτσι ώστε* $U_{Ga} = U_{Ga}^{train} \cup U_{Ga}^{test}$ *με* $U_{Ga}^{train} \cap U_{Ga}^{test} = \emptyset$*. 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_{Ga} *δίνεται από την σχέση:* $U_{Ga}=\{u_{a,1}, u_{a,2}, \cdots, u_{a,na}\}$ *με* $n_a = |U_{Ga}|$*. Τότε το σύνολο των διανυσμάτων χαρακτηριστικών και το αντίστοιχο σύνολο των ετικετών θα μπορούσε να οργανωθεί σε έναν πίνακα της μορφής:*

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

Το νευρωνικό δίκτυο κατασκευάστηκε με τη βοήθεια της βιβλιοθήκης `tensorflow`. Χρησιμοποιήθηκε ένα απλό Feedforward πολυστρωματικό νευρωνικό δίκτυο. Το στρώμα εισόδου είναι ισομεγέθης με το σύνολο των επιλεγμένων κοντινότερων γειτόνων. Καθώς δέχεται ως είσοδο διάνυσμα με τις βαθμολογίες των γειτόνων για κάποια ταινία, με τις τιμές να ανήκουν στο [0,1] αφού οι βαθμολογίες έχουν διαιρεθεί με 10. Το δίκτυο διαθέτει 3 κρυφά στρώματα με μέγεθος 128, 64 και 32 perceptrons αντίστοιχα, ενώ και τα τρία έχουν ως συνάρτηση ενεργοποίησης ReLU. Στα δύο πρώτα στρώματα εφαρμόζεται dropout 30%, ούτως ώστε να αποφευχθεί η εξάρτηση από συγκεκριμένους νευρώνες ανά στρώμα. Τέλος το στρώμα εξόδου περιέχει ένα μόνο perceptron το οποίο επιστρέφει μια τιμή μεταξύ [0,1], η οποία πολλαπλασιασμένη επί 10 αντιπροσωπεύει την εκτίμηση του δικτύου για την κριτική του χρήστη. Το μοντέλο εκπαιδεύεται για 50 εποχές σε κάθε εκπαίδευση, με μέγεθος batch size 32 ανά βήμα εκπαίδευσης. Η ενημέρωση των βαρών πραγματοποιείται με βελτιστοποιητή Adam και στατικό ρυθμό μάθησης 0.001. Ως σύνολο ελέγχου έχει επιλεχθεί το 10% των παραγόμενων δεδομένων από τις βαθμολογίες των γειτόνων. Ενώ εφαρμόζεται early stopping, σε περίπτωση που δεν σημειώνεται βελτίωση στο σύνολο ελέγχου.

```python
# Train a model for each cluster
for cluster in clusters_df['cluster'].unique():
    print(f"\n[✓] Training model for {cluster}...")
    cluster_users = clusters_df[clusters_df['cluster'] == cluster].index
    cluster_neighbors_df = neighbors_df[neighbors_df['username'].isin(cluster_users)]
    if cluster_neighbors_df.empty:
        print(f"⚠ No users in {cluster}. Skipping.")
        continue
    X, y, usernames = [], [], []
    # Feature extraction for the cluster
    for idx, row in cluster_neighbors_df.iterrows():
        user = row['username']
        neighbors = row[[f'neighbor_{i+1}' for i in range(K)]]
        if user not in ratings_df.index:
            continue
        user_ratings = ratings_df.loc[user].values
        for movie_idx, target_rating in enumerate(user_ratings):
            if target_rating == 0:
                continue
            neighbor_ratings = []
            valid = True
            for neighbor in neighbors:
                if neighbor in ratings_df.index and neighbor in cluster_users:
                    neighbor_rating = ratings_df.loc[neighbor].values[movie_idx]
                    neighbor_ratings.append(neighbor_rating)
                else:
                    valid = False
                    break
            if not valid or np.all(np.array(neighbor_ratings) == 0):
                continue
```

```
            X.append(neighbor_ratings)
            y.append(target_rating)
            usernames.append(user)
    if not X:
        print(f"⚠ No valid data for {cluster}. Skipping.")
        continue
    X = np.array(X)
    y = np.array(y)
    usernames = np.array(usernames)
    # Train-test split
    X_train, X_test, y_train, y_test, users_train, users_test = train_test_split(
        X, y, usernames, test_size=0.2, random_state=42)
    # Build neural network
    model = Sequential([
        Input(shape=(K,)),
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(32, activation='relu'),
        Dense(1)])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error',
        metrics=[custom_metric])
    # Train model
    early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
    history = model.fit(
        X_train, y_train,
        epochs=50,
        batch_size=32,
        validation_split=0.1,
        callbacks=[early_stop],
        verbose=0)
```

Ανάλυση αποτελεσμάτων Νευρωνικών Δικτύων

"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."

Για το παραπάνω νευρωνικό δίκτυο καταγράφηκαν και αποτυπώθηκαν οι τιμές του μέσου απόλυτου σφάλματος (MAE) και μέσου τετραγωνισμένου σφάλματος (MSE) για κάθε συστάδα. Ενώ επίσης δημιουργήθηκαν και τα διαγράμματα απώλειας και "ευστοχίας" των αντίστοιχων νευρωνικών. Ο υπολογισμός του MAE και MSE πραγματοποιείτο κατά την ολοκλήρωση της εκάστοτε εκπαίδευσης:

```
        # Evaluate model
        y_pred_train = model.predict(X_train).flatten() * 10
        y_true_train = y_train * 10
        y_pred_test = model.predict(X_test).flatten() * 10
        y_true_test = y_test * 10
        mae_train = mean_absolute_error(y_true_train, y_pred_train)
        mse_train = mean_squared_error(y_true_train, y_pred_train)
        mae_test = mean_absolute_error(y_true_test, y_pred_test)
        mse_test = mean_squared_error(y_true_test, y_pred_test)
```

Για τον υπολογισμό της “ευστοχίας” (accuracy) των νευρωνικών δικτύων, το μοντέλο ρυθμίστηκε με προσαρμοσμένη μετρική:

```
        model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error',
            metrics=[custom_accuracy])
```

Η μετρική έλεγχε κατά πόσον η εκτίμηση βρισκόταν σε απόσταση ±1 μονάδας από την πραγματική τιμή:

```
def custom_accuracy(y_true, y_pred):
    diff = K_backend.abs(y_true - y_pred)
    return K_backend.mean(K_backend.cast(diff <= 0.1, dtype='float32'))
```

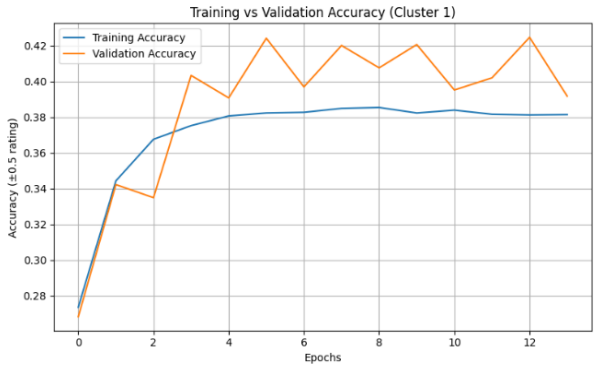
Και υπολογιζόταν κατά την ολοκλήρωση του κύκλου της εκπαίδευσης και ελέγχου του κάθε νευρωνικού δικτύου:

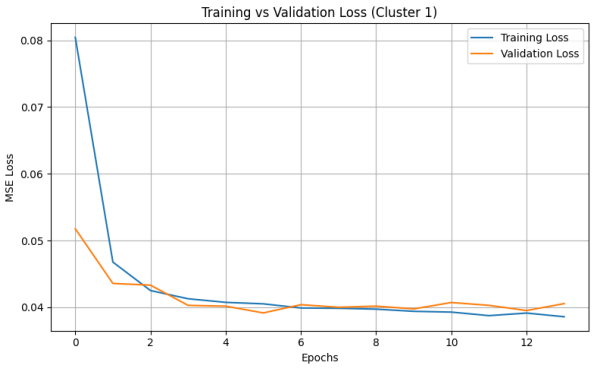
```
        # Evaluate model
        y_pred_train = model.predict(X_train).flatten() * 10
        y_true_train = y_train * 10
        y_pred_test = model.predict(X_test).flatten() * 10
        y_true_test = y_test * 10
        train_accuracy = np.mean(np.abs(y_pred_train - y_true_train) <= 1)
        test_accuracy = np.mean(np.abs(y_pred_test - y_true_test) <= 1)
```

Ακολουθεί ενδεικτικός πίνακας των υπολογισθέντων σφαλμάτων για kmeans με μετρική Jaccard σε σύνολο 3 συστάδων:

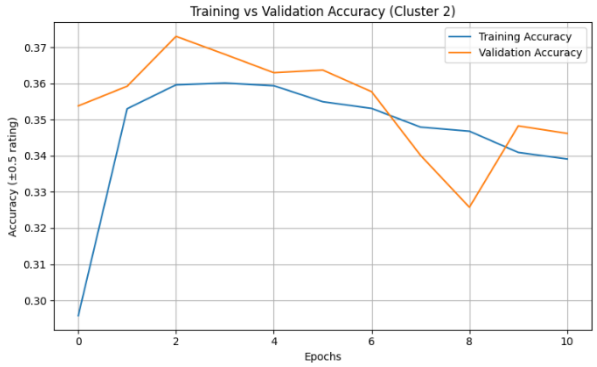
	Train			Test		
Cluster	MAE	MSE	Accuracy	MAE	MSE	Accuracy
Cluster 1	1.5370	3.8866	0.4154	1.5661	4.0619	0.4107
Cluster 2	1.5333	3.8327	0.4003	1.5302	3.8309	0.4047
Cluster 3	1.5333	3.9478	0.4152	1.5805	4.1837	0.4042

Ακολουθούν τα διαγράμματα για τις παραπάνω τιμές:

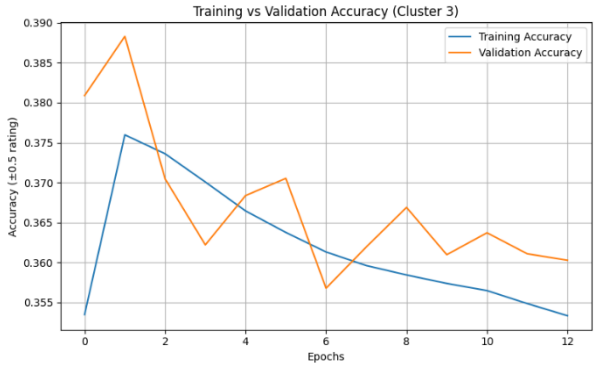


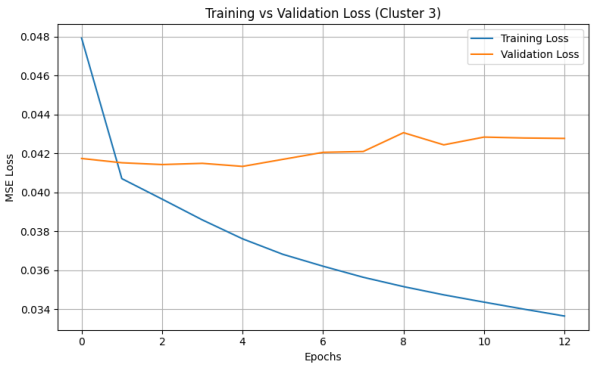


Διαγράμματα ευστοχίας (αριστερά) – απώλειας (δεξιά) για τη συστάδα 1



Διαγράμματα ευστοχίας (αριστερά) – απώλειας (δεξιά) για τη συστάδα 2

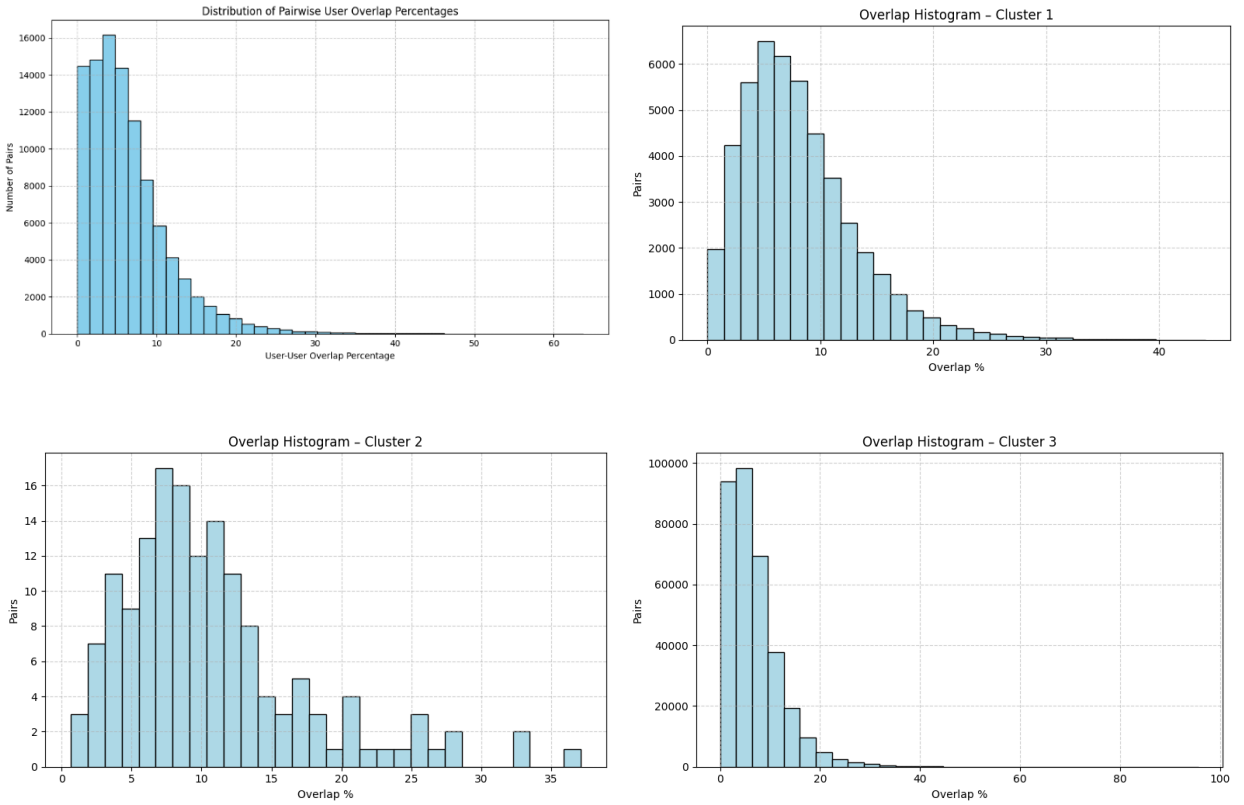




Διαγράμματα ευστοχίας (αριστερά) – απώλειας (δεξιά) για τη συστάδα 3

Όπως γίνεται αντιληπτό η εκπαίδευση των δικτύων είναι ανεπιτυχής. Τα αποτελέσματα του ελέγχου δεν είναι ικανοποιητικά. Χαρακτηριστικό είναι ότι οι εκτιμήσεις έχουν ποσοστό ευστοχίας χαμηλότερο του 50%, ενώ οι τιμές της ευστοχίας χαρακτηρίζονται από έντονη αστάθεια και όχι ουσιαστική βελτίωση. Τα διαγράμματα απώλειας παρουσιάζουν καλύτερη εικόνα με πιο σταθερή πορεία, ωστόσο και σε αυτή την περίπτωση οι συστάδες 2 και 3

παρουσιάζουν αύξηση σφάλματος μετά τις 2-3 πρώτες εποχές. Ακόμη εντύπωση προκαλεί και η διάρκεια των κύκλων. Τα μοντέλα ήταν ρυθμισμένα σε 50 εποχές, αλλά διακόπτουν στις 10-12 εποχές. Γεγονός που υποδεικνύει ότι τα δίκτυα παύουν να εκπαιδεύονται από αυτό το σημείο και αποστηθίζουν/φθίνουν. Όλα αυτά αιτιολογούνται απ' την υπερβολικά αραιή εικόνα του συνόλου δεδομένων (πάνω από το 80% μηδενικές τιμές). Αυτό καθιστά πολύ δύσκολη την συσταδοποίηση αλλά και την επιτυχή εκπαίδευση ενός νευρωνικού δικτύου πάνω σε ένα τέτοιο σύνολο, τουλάχιστον όχι με αυτές τις παραμέτρους. Η υπερβολικά αραιή κατανομή πληροφορίας στο user matrix οδηγεί σε ασθενή συσταδοποίηση. Ακολούθως όταν εκπαιδεύεται το δίκτυο για να εκτιμήσει τις βαθμολογίες του χρήστη, αναγκαστικά βασίζεται σε γείτονες με τους οποίους έχει πολύ μικρή αλληλοκάλυψη. Συνεπώς το αποτέλεσμα του δικτύου καταλήγει να είναι περισσότερο μαντεψιά, παρά εκτίμηση. Τα παραπάνω επιβεβαιώνονται και από τα ακόλουθα ιστογράμματα για την αλληλοκάλυψη (σημείωση: με τον όρο αλληλοκάλυψη, εννοείται η πιθανότητα 2 χρήστες να έχουν βαθμολογήσει την ίδια ταινία).



Ιστογράμματα αλληλοκάλυψης χρηστών
Πάνω: αλληλοκάλυψη στο σύνολο των δεδομένων (αριστερά) - στο Cluster 1* (δεξιά)
Πάνω: αλληλοκάλυψη στο Cluster 2* (αριστερά) - στο Cluster 3* (δεξιά)
*K-means με χρήση Jaccard distance για ομαδοποίηση σε 3 συστάδες

Τα ιστογράμματα υποδεικνύουν ότι παρά την συσταδοποίηση, τα δεδομένα παραμένουν υπερβολικά αραιά. Από την μέση αλληλοκάλυψη κοντά στο 5% για το συνολικό dataset η σημαντικότερη βελτίωση εμφανίζεται στο Cluster 2 με μόλις 10% αλληλοκάλυψη. Θεωρώντας ότι ο αλγόριθμος συσταδοποίησης είναι αξιόπιστος και παράγει αξιόλογες ομάδες καταλήγουμε στο συμπέρασμα ότι τα δεδομένα παραμένουν υπερβολικά αραιά για την επίτευξη ουσιαστικής εκπαίδευσης μοντέλου πάνω σε αυτά.

Λειτουργία εκτελούμενου

Για την λειτουργία του απαραίτητες είναι οι βιβλιοθήκες `pandas`, `numpy`, `matplotlib`, `scikit-learn` και `tensorflow`. Ακολουθεί η εντολή εγκατάστασής τους:

```
pip install pandas numpy matplotlib scikit-learn tensorflow
```

Τα δεδομένα που χρησιμοποιήθηκαν είναι τα raw data από τον IEEE αποθηκευμένα σε CSV. Κατά την έναρξη εκτέλεσης, το πρόγραμμα κατεβάζει από διαδικτυακό χώρο το dataset (πρόσβαση σε διαδικτυακό χώρο – dropbox – όπου έχει ήδη αποθηκευτεί το dataset). Η διαδικασία πραγματοποιείται από το `setup.py` αρχείο και είναι η ακόλουθη:

```
# setup.py
def ensure_dataset_ready():
    import os
    import zipfile
    import requests
    from io import BytesIO
    dataset_url =
"https://www.dropbox.com/scl/fi/67ke65f78o2oz7mc65fhm/Dataset.zip?rlkey=nvjngju1o2iii3h3mo4lvh2d4&s
t=yiapt1ml&dl=1"
    extract_to = "Dataset"
    if os.path.exists(extract_to):
        print(f"✓ Dataset already exists at: {extract_to}")
        return
    print("↓ Downloading dataset from Dropbox...")
    headers = {"User-Agent": "Mozilla/5.0"}

    response = requests.get(dataset_url, headers=headers, stream=True)
    if response.status_code != 200:
        raise Exception(f"✗ Failed to download dataset (HTTP {response.status_code})")
    if 'html' in response.headers.get('Content-Type', '').lower():
        raise Exception("✗ Dropbox returned HTML, not a ZIP file.")
    print("✓ Extracting dataset...")
```

```
    with zipfile.ZipFile(BytesIO(response.content)) as zip_ref:
        zip_ref.extractall(extract_to)
    print(f"✓ Dataset extracted to: {extract_to}")
if __name__ == "__main__":
    ensure_dataset_ready()
```

Η εφαρμογή είναι console-based και αλληλεπιδρά με τον χρήστη ο οποίος μπορεί να ζητήσει οποιοδήποτε τμήμα της εργασίας, να διενεργηθεί πάνω στο dataset. Μόλις τρέξει το αρχείο `computational_assignment.py`, με την παρακάτω εντολή:

```
python computational_assignment.py
```

Εμφανίζεται το ακόλουθο μενού από το οποίο ο χρήστης πλοηγείται σε όλες τις λειτουργίες:

```
=== Movie Recommender System ===
1. Build user-movie matrix
2. Create histograms
3. Run KMeans clustering
4. Generate nearest neighbors
5. Train and evaluate neural network
6. Exit
Select an option (1-6):
```