Ανάκτηση Πληροφορίας

**Project 2022-2023**



|  |  |  |
| --- | --- | --- |
| **Στυλιανάκης Στυλιανός** | 1059713 | steliostylian@gmail.com |
| **Κονταράκης Ιωάννης Γεώργιος** | 1067375 | kogi\_20010@hotmail.com |

**[Icon

Description automatically generated](https://github.com/Steliostyl/Projects-kDDs-2022)**[**Link to GitHub**](https://github.com/Steliostyl/Data-Retrieval-Project-2022)

Πίνακας περιεχομένων

[Περιβάλλον υλοποίησης 3](#_Toc126533920)

[Βιβλιοθήκες που χρησιμοποιήθηκαν 3](#_Toc126533921)

[Ερώτημα 1 – ElasticSearch 3](#_Toc126533922)

[Σύνδεση με την Elastic, δημιουργία δείκτη & εισαγωγή βιβλίων 3](#_Toc126533923)

[Πραγματοποίηση ερωτημάτων 6](#_Toc126533924)

[Υπολογισμός συνδυαστικών βαθμολογιών 7](#_Toc126533925)

[Ερώτημα 2 – Clustering 10](#_Toc126533926)

[Προεπεξεργασία του users dataset 10](#_Toc126533927)

[Αλγόριθμος clustering 12](#_Toc126533928)

[Υπολογισμός μέσων βαθμολογιών συστάδων 16](#_Toc126533929)

[Υπολογισμός συνδυαστικών βαθμολογιών με συμπλήρωση κενών βαθμολογιών από το μέσο όρο συστάδων 17](#_Toc126533930)

[Ερώτημα 3 – Νευρωνικό δίκτυο 17](#_Toc126533931)

[Word Embeddings 18](#_Toc126533932)

[NN with Embedding Layer 18](#_Toc126533933)

[Νευρωνικό δίκτυο 20](#_Toc126533934)

[Έξοδος κατά την εκτέλεση 22](#_Toc126533935)

[ElasticSearch 22](#_Toc126533936)

[Clustering 22](#_Toc126533937)

[Νευρωνικό δίκτυο 24](#_Toc126533938)

# Περιβάλλον υλοποίησης

Ως γλώσσα υλοποίησης επιλέχθηκε η **python**, καθώς υπάρχουν πολλές χρήσιμες βιβλιοθήκες για την εργασία.

### Βιβλιοθήκες που χρησιμοποιήθηκαν

#### pandas

Βασική βιβλιοθήκη διαχείρισης δεδομένων. Τα περισσότερα δεδομένα μας φορτώνονται, επεξεργάζονται και ανακτούνται από **DataFrames**, τη βασική δομή δεδομένων των pandas.

#### elasticsearch

Χρησιμοποιήθηκε για τη σύνδεση με την Elasticsearch, καθώς και την πραγματοποίηση των queries, μέσω REST API calls.

#### sklearn

Βιβλιοθήκη με πολλές χρήσιμα εργαλεία για υλοποίηση clustering, καθώς και νευρωνικών δικτύων. Χρησιμοποιήθηκε κυρίως στο ερώτημα 3.

#### kmodes

Βιβλιοθήκη για clustering, η οποία χρησιμοποιεί και την **sklearn** (σε συνδυασμό με άλλες βιβλιοθήκες). Συγκεκριμένα, από αυτήν τη βιβλιοθήκη χρησιμοποιήσαμε την κλάση **KPrototypes** για να υλοποιήσουμε το clustering.

#### matplotlib, pyplot, seaborn

Βιβλιοθήκες για σχεδιασμό και εμφάνιση γραφημάτων.

#### re

Βιβλιοθήκη αναγνώρισης κανονικών εκφράσεων, χρησιμοποιήθηκε στον καθαρισμό των datasets.

# Ερώτημα 1 – ElasticSearch

### Σύνδεση με την Elastic, δημιουργία δείκτη & εισαγωγή βιβλίων

Ξεκινώντας, αρχικοποιούμε ένα instance της κλάσης Elasticsearch με τα απαραίτητα στοιχεία για τη σύνδεση στο cluster μας. Την πρώτη φορά που θα τρέξουμε την Elasticsearch, πρέπει να δημιουργήσουμε ένα index στο οποίο θα προσθέσουμε τα βιβλία, όπως φαίνεται παρακάτω.

def main():

    ############################ ELASTICSEARCH #############################

    # Connect to the ElasticSearch cluster

    es = Elasticsearch(

        "http://localhost:9200",

        ssl\_assert\_fingerprint=CERT\_FINGERPRINT,

        basic\_auth=("elastic", ES\_PASSWORD)

        )

    # Create a new index in ElasticSearch called books

    print("Creating books index...")

    elastic.createIndex(es, idx\_name="books")

    # Insert a sample of data from dataset into ElasticSearch

    book\_count = elastic.insertData(es)[0]["count"]

    print(f"Inserted {book\_count} books into index.")

main.py

Με τη μέθοδο createIndex δημιουργούμε το books index, ορίζοντας τις ιδιότητες των δεδομένων που θα εισάγουμε. Σε σχόλιο υπάρχει η μέθοδος με την οποία μπορούμε να διαγράψουμε το index που έχουμε δημιουργήσει για να το ξαναφτιάξουμε από την αρχή.

def createIndex(es: Elasticsearch, idx\_name: str = "books") -> None:

  """Creates a new book index, deleting

  any pre-existing with the same name."""

  # Define the mappings of the books index.

  mappings = {

    "properties": {

      "isbn": {"type": "text", "analyzer": "keyword"},

      "book\_title": {"type": "text", "analyzer": "english"},

      "book\_author": {"type": "text", "analyzer": "standard"},

      "year\_of\_publication": {"type": "integer"},

      "publisher": {"type": "text", "analyzer": "standard"},

      "summary": {"type": "text", "analyzer": "english"},

      "category": {"type": "text", "analyzer": "standard"}

    }

  }

  # Delete pre-existing index with the same name

  #es.indices.delete(index=idx\_name)

  # Create a new index named idx\_name with the defined mappings

  es.indices.create(index=idx\_name, mappings=mappings)

elastic.py

Με τη μέθοδο insertData εισάγουμε τα βιβλία στη βάση. Επιλέξαμε να τα εισάγουμε όλα μαζί (in bulk) με μία κλήση στην Elastic, η οποία περιέχει λίστα με όλα τα βιβλία. Η μέθοδος επιστρέφει την απάντηση που πήραμε από την Elastic μετά την εισαγωγή των βιβλίων, όπου φαίνεται πόσα βιβλία υπάρχουν.

def insertData(es: Elasticsearch) -> str:

  """Parses the data of a specified csv file and

  inserts the data into Elasticsearch. Returns

  the number of entries after insertion."""

  # Parse the CSV dataset

  dataframe = pd.read\_csv(BOOKS).dropna().reset\_index()

  # Create a list containing the parsed rows from the CSV file

  bulk\_data = []

  for i,row in dataframe.iterrows():

      bulk\_data.append(

          {

              "\_index": "books",

              "\_id": i,

              "\_source": {

                "isbn": row["isbn"],

                "book\_title": row["book\_title"],

                "book\_author": row["book\_author"],

                "year\_of\_publication": row["year\_of\_publication"],

                "publisher": row["publisher"],

                "summary": row["summary"],

                "category": row["category"]

              }

          }

      )

  # Bulk insert the rows into ElasticSearch

  bulk(es, bulk\_data)

  # Refresh the books index and return the number of items in it.

  es.indices.refresh(index="books")

  resp = es.cat.count(index="books", format="json")

  return resp

elastic.py

### Πραγματοποίηση ερωτημάτων

Για την πραγματοποίηση ερωτημάτων, ο χρήστης πρέπει να εισάγει έναν αλφαριθμητικό όρο, τον οποίο θα αναζητήσει η Elastic στα βιβλία που της έχουμε φορτώσει στη βάση, καθώς και το αναγνωριστικό του.

    # Input query parameters and make query to ES

    search\_string = input("Enter search string:")

    user\_id = \_input("Enter your user ID (must be an integer): ", int)

    es\_reply = elastic.makeQuery(es, search\_string)

main.py

Τα ερωτήματα γίνονται μέσω της μεθόδου makeQuery. Για κάθε ερώτημα, ζητάμε στην Elastic να αναζητήσει τους όρους σε πολλαπλά πεδία των εγγραφών, πιο συγκεκριμένα στους τίτλους των βιβλίων, καθώς και στην περιγραφή τους. Όπως φαίνεται στο σώμα του ερωτήματος στη γραμμή των πεδίων, αποφασίσαμε να δώσουμε πιο μεγάλο βάρος στον τίτλο του βιβλίου (1.5 αντί για 1 συγκεκριμένα), παρά στην περίληψη. Αυτήν την απόφαση την πήραμε βασισμένοι στην υπόθεση ότι, αν οι όροι αναζήτησης υπάρχουν στον τίτλο, ουσιαστικά είναι εγγυημένο ότι περιλαμβάνονται στο **βασικό** θέμα του βιβλίου, σε αντίθεση με την περίληψη.

def makeQuery(es: Elasticsearch, search\_string: str) -> tuple:

  """Creates a query and returns Elasticsearch's answer."""

  # Create query body using inputs. Using multi\_match we check multiple fields

  # of an entry for the given search string and the final score is calculated

  # by adding the (weighted) score of the fields.

  query\_body = {

    "query": {

        "multi\_match": {

            "query": search\_string,

            "type": "most\_fields",

            "fields": ["book\_title^1.5", "summary"]

        }

    }

  }

  # Make the query to ElasticSearch

  es\_reply = es.search(

    index = "books",

    body = query\_body,

    size = 10\_000

  )

  return es\_reply["hits"]

elastic.py

### Υπολογισμός συνδυαστικών βαθμολογιών

Μόλις λάβουμε την απάντηση της Elastic, υπολογίζουμε τη συνδυαστική βαθμολογία των βιβλίων συνυπολογίζοντας τις πιθανές βαθμολογίες που έχουν κάνει οι χρήστες στα σχετικά βιβλία.

    # Get the combined scores for all replies

    print("\nCalculating combined scores...")

    combined\_scores = functions.calculateCombinedScores(es\_reply, user\_id)

    combined\_scores.to\_csv(SCORES\_NO\_CLUST, index=False)

main.py

Η μέθοδος calculateCombinedScores χρησιμοποιείται τόσο για το 1ο, όσο και για το 2ο ερώτημα, μετά το clustering. Για αυτόν το λόγο, έχει κάποια προαιρετικά ορίσματα τα οποία μας χρησιμεύουν μόνο στο 2ο ερώτημα. Τα βασικά ορίσματα που χρειάζονται και για τα 2 ερωτήματα, είναι η απάντηση της Elastic και το αναγνωριστικό του χρήστη. Τελικά, η μέθοδος επιστρέφει ένα DataFrame με την ανανεωμένη συνδυαστική βαθμολογία.

def calculateCombinedScores(es\_reply: dict, user\_id: int, use\_cluster\_ratings:\

    bool = False, avg\_clust\_ratings: pd.DataFrame = None,\

        cluster\_assigned\_users: pd.DataFrame = None) -> pd.DataFrame:

functions.py

Αρχικά, η μέθοδος αποθηκεύει το max\_score, όλα τα βιβλία που επέστρεψε η Elastic σε μία λίστα από λεξικά και φορτώνει όλες τις αξιολογήσεις του χρήστη σε ένα DataFrame, χρησιμοποιώντας τη μέθοδο getUserRatings.

    books = es\_reply["hits"]

    max\_score = es\_reply["max\_score"]

    user\_ratings = getUserRatings(user\_id)

    print(f"User {user\_id} has rated {len(user\_ratings)} book(s).")

functions.py

Η μέθοδος getUserRatings απλώς διαβάζει το CSV με όλα τα Book Ratings και επιστρέφει μόνο τις γραμμές που περιέχουν το user\_id που της περνάμε.

def getUserRatings(user\_id: int, filename: str = RATINGS) -> pd.DataFrame:

    """Read ratings CSV and return specified user's ratings."""

    users\_ratings\_df = pd.read\_csv(filename)

    return  users\_ratings\_df.loc[users\_ratings\_df["uid"] == user\_id]

functions.py

Στη συνέχεια, αρχικοποιεί μία άδεια λίστα όπου θα μπουν τα βιβλία αφού υπολογιστεί η συνδυαστική βαθμολογία τους. Έπειτα, εάν η μέθοδος έχει κληθεί με το use\_cluster\_ratings=True (όπως καλείται στο 2ο ερώτημα δηλαδή), ανακτάται από το αντίστοιχο dataframe η συστάδα στην οποία ανήκει ο χρήστης, εάν αυτός υπάρχει στο αρχείο BX-Users.csv. Σε περίπτωση που δεν υπάρχει, εκτυπώνεται αντίστοιχο μήνυμα.

    # List to be filled with book entries

    # and their combined scores

    books\_list = []

    users\_cluster = -1

    # Get user's cluster

    if use\_cluster\_ratings:

        try:

            users\_cluster = cluster\_assigned\_users["Cluster"].\

                loc[cluster\_assigned\_users["User\_ID"] == user\_id].iloc[0]

            print(f"User {user\_id} belongs in cluster {users\_cluster}")

        except:

            print(f"User {user\_id} doesn't exist in database.")

            print("Scores with clustering will be the same as previous scores.")

functions.py

Το κύριο κομμάτι της μεθόδου αυτής είναι η βασική της επανάληψη, στην οποία υπολογίζεται η συνδυαστική βαθμολογία κάθε βιβλίου και δημιουργούνται εγγραφές βιβλίων. Οι εγγραφές αυτές περιέχουν όλες τις πληροφορίες κάθε βιβλίου, καθώς και την υπολογισμένη βαθμολογία τους. Οι εγγραφές αυτές προστίθενται στην books\_list.

Αρχικά, προσπαθούμε να ανακτήσουμε τη βαθμολογία του χρήστη για το συγκεκριμένο βιβλίο. Σε περίπτωση που δεν το έχει βαθμολογήσει και δε χρησιμοποιούμε clustering, πριν υπολογιστεί η συνδυαστική βαθμολογία, θέτουμε τη βαθμολογία του χρήστη ίση με DEFAULT\_RATING, σταθερά η οποία έχει οριστεί στην αρχή του αρχείου.

Ως τιμή για αυτή τη σταθερά επιλέξαμε να είναι το 5, η μέση τιμή των πιθανών βαθμολογιών δηλαδή. Επιλέξαμε την τιμή αυτή επειδή θέλουμε τα βιβλία που έχει βαθμολογήσει αρνητικά ο χρήστης (δηλαδή με βαθμολογία κάτω από 5) να εμφανίζονται πιο κάτω στην κατάταξη από ότι αυτά που δεν έχει βαθμολογήσει ακόμα. Αντίστοιχα, τα βιβλία στα οποία έχει βάλει θετική βαθμολογία, προωθούνται πιο ψηλά στην κατάταξη.

    # Iterate through the documents of the ES reply

    for book in books:

        isbn = book["\_source"]["isbn"]

        norm\_es\_score = 10\*book["\_score"]/max\_score

        # User has rated book

        try:

            # Get user's book rating and typecast it to float (from string)

            user\_rating = float(user\_ratings\_df.\

                loc[user\_ratings ["isbn"] == isbn]["rating"].iloc[0])

        # User has not rated this book

        except:

            # User doesn't exist in the database or function has

            # been called with arg use\_cluster\_ratings = False

            if users\_cluster == -1:

                user\_rating = DEFAULT\_RATING

            else:

                user\_rating = getAvgClusterRating(

                    users\_cluster, isbn, avg\_clust\_ratings)

        # Combined score will be calculated using combinedScoreFunc

        score = combinedScoreFunc(norm\_es\_score, user\_rating)

        # Create a new book entry as a list

        new\_book = [

            score, user\_rating, norm\_es\_score, isbn,

            book['\_source']["book\_title"],

            book['\_source']["book\_author"],

            book['\_source']["year\_of\_publication"],

            book['\_source']["publisher"],

            book['\_source']["summary"],

            book['\_source']["category"]

            ]

        books\_list.append(new\_book)

functions.py

Για τον υπολογισμό της συνδυαστικής βαθμολογίας, χρησιμοποιείται η μέθοδος combinedScoreFunc, η οποία παίρνει σαν είσοδο το score της Elasticsearch (κλιμακωμένο στο διάστημα [0, 10]), καθώς και τη βαθμολογία του χρήστη.

Στη βαθμολογία του χρήστη έχουμε προσθέσει ένα βάρος, ώστε να έχει μεγαλύτερη σημασία από ότι το σκορ της Elastic. Ο λόγος που δε σκεφτήκαμε την αντίθετη περίπτωση είναι επειδή η Elastic επιστρέφει μόνο σχετικά βιβλία στα οποία έχει βρει τους όρους αναζήτησης. Πειραματιστήκαμε λίγο με τη σταθερά αυτή και καταλήξαμε στο βάρος 1.25 για τη βαθμολογία του χρήστη.

Η συνδυαστική βαθμολογία προκύπτει από το σταθμισμένο άθροισμα των 2 εισόδων, κλιμακωμένο στο διάστημα [0, 10] και επιστρέφεται από τη μέθοδο.

def combinedScoreFunc(norm\_es\_score: float, user\_rating: float) -> float:

    """Function that accepts as inputs the elastic search score of a book,

    as well as the user's rating of the book and returns a combined score."""

    # Add the normalized and weighted scores and scale them in the range [0, 10]

    return (USER\_R\_WEIGHT \* user\_rating + norm\_es\_score) / (1 + USER\_R\_WEIGHT)

functions.py

Αφού τελειώσει η επανάληψη και έχουμε προσθέσει όλες τις εγγραφές των σχετικών βιβλίων στην books\_list, δημιουργούμε ένα DataFrame με τις εγγραφές αυτές, προσθέτοντας και τα ονόματα των αντίστοιχων στηλών τους. Τέλος, τα ταξινομούμε και επιστρέφουμε μόνο τις 10% καλύτερες αντιστοιχίσεις, σύμφωνα με τις συνδυαστικές τους βαθμολογίες.

    # Create a new dataframe from books\_list and sort it by score

    best\_matches = pd.DataFrame(data=books\_list, columns=[

        "score", "user\_rating", "es\_scaled\_score", "isbn", "book\_title",

        "book\_author", "year\_of\_publication", "publisher", "summary",

        "category"]).sort\_values(by="score", ascending=False)

    # Only keep the best 10% documents

    return best\_matches.head(len(best\_matches.index)//10)

functions.py

# Ερώτημα 2 – Clustering

### Προεπεξεργασία του users dataset

Πριν ξεκινήσουμε το clustering, πρέπει να γίνει μία προεπεξεργασία του Users dataset. Για αυτόν το σκοπό χρησιμοποιούμε τη μέθοδο processUsersCSV.

    # Create a CSV containing users sorted by their country

    # Entries that have a high chance of being fake are ignored

    print("Creating processed users CSV...")

    processed\_users = functions.processUsersCSV()

    processed\_users.to\_csv(PROC\_USERS, index=False)

main.py

Αρχικά, η μέθοδος διαβάζει το αρχείο BX-Users.csv, το φορτώνει σε ένα dataframe και διατρέχει τις γραμμές του, εξάγοντας το uid του χρήστη, την ηλικία του, τη χώρα του και το μοναδικό id της. Για τον κάθε χρήστη, αυτές οι τιμές αποθηκεύονται σε μία λίστα η οποία με τη σειρά της προστίθεται στη λίστα χρηστών users. Τελικά, δημιουργείται ένα dataframe με τους χρήστες, ορίζοντας τα κατάλληλα ονόματα για την κάθε στήλη.

Σε περίπτωση που ο χρήστης δεν έχει συμπληρώσει ηλικία ή ηλικία που έχει συμπληρώσει δεν ανήκει σε ένα φυσιολογικό διάστημα [0, 120], του αναθέτουμε μία καινούρια τιμή. Επιλέξαμε αυτή η τιμή να είναι η μέση τιμή του διαστήματος, δηλαδή 60. Άλλη ανάθεση που θα μπορούσε να γινόταν θα ήταν να διαλέγαμε τυχαία έναν αριθμό στο διάστημα αυτό. Σε κάθε περίπτωση, δεν μπορούμε να αφήσουμε κενή την ηλικία, καθώς ο αλγόριθμος του clustering δε δέχεται μη αριθμητικές τιμές.

Η χώρα του εξάγεται με τη χρήση μίας κανονικής έκφρασης η οποία ουσιαστικά διαβάζει όλους τους χαρακτήρες μετά το τελευταίο κόμμα του αλφαριθμητικού location.

Το μοναδικό id της κάθε χώρας είναι ουσιαστικά η σειρά εμφάνισής της στο dataset μας, με εξαίρεση της κενές εγγραφές. Για κάθε εγγραφή που έχει κενή χώρα, το id της χώρας υπολογίζεται ως συνάρτηση του uid, αυξημένο κατά 1000. Το 1000 επιλέχθηκε επειδή, τουλάχιστον για το dataset μας, εγγυάται ότι δε θα υπάρξουν συγκρούσεις με τα ids άλλων χωρών.

Η στήλη uid αποθηκεύεται όπως είναι, αφού γίνει μετατροπή του αλφαρηθμιτικού σε ακέραιο.

def processUsersCSV() -> pd.DataFrame:

    """Extract vital information from BX-Users csv and save it to a new CSV."""

    # Create empty users dictionary

    users = []

    countries = []

    # Iterate BX-Users.csv rows

    for entry in pd.read\_csv(USERS).iterrows():

        # Cast age to int

        uid = int(entry[1][0])

        # Get location

        location = entry[1][1]

        # Try to cast age to int. If it's empty or

        # out of bounds, set it to the default age

        try:

            age = int(entry[1][2])

            if age > MAX\_AGE or age < 0:

                age = DEF\_AGE

        except:

            age = DEF\_AGE

        # Extract country from location string

        country = re.findall(r"[\s\w+]+$", location)

        # If country is empty, set country\_id to 1000+uid, so that users with

        # incomplete countries do not belong in the same "country". Essential

        # for clustering! Also, 500 is arbitrarily chosen butis empirically

        # higher than the count of normal countries, so there are no conflicts.

        if not country:

            country = ""

            country\_id = 1000 + uid

        else:

            country = country[0][1:]

        # If country isn't already in the countries list,

        # its index will be equal to the length of countries

        if country not in countries:

            country\_id = len(countries)

            countries.append(country)

        # Otherwise, its index is equal to the index of the country in countries

        else:

            country\_id = countries.index(country)

        users.append([uid, age, country, country\_id])

    users\_df = pd.DataFrame(

        users, columns=["User\_ID", "Age", "Country", "Country\_ID"]

    )

    return users\_df

functions.py

### Αλγόριθμος clustering

#### Το πρόβλημα

Στην εργασία, ζητείται ως αλγόριθμος ομαδοποίησης να χρησιμοποιηθεί ο Kmeans. Παρόλα αυτά, ο απλός Kmeans δέχεται **μόνο** αριθμητικές ιδιότητες (features). Πιο συγκεκριμένα, ο αλγόριθμος αρχικοποιεί τυχαία k κεντροειδή, τα οποία αποτελούν αρχικά το κέντρο της κάθε συστάδας. Στη συνέχεια, υπολογίζει τις ευκλείδειες αποστάσεις των σημείων (δεδομένων μας) από τα κεντροειδή, αναθέτοντας το κάθε σημείο στο κοντινότερο κεντροειδές. Στο τέλος του βήματος, ξαναυπολογίζονται τα κεντροειδή ως το μέσο των σημείων τα οποία ανήκουν σε αυτό. Ο αλγόριθμος τερματίζει όταν τα κεντροειδή δεν μετακινούνται.

#### Λύση #1 – kMeans

Θα μπορούσαμε να χρησιμοποιήσουμε τον kMeans από τη βιβλιοθήκη scikit-learn, κι ας μην θεωρούμε αρμόζει στα δεδομένα μας. Παρόλα αυτά, θεωρήσαμε ότι υπάρχουν πιο σωστές λύσεις για τα ζητούμενα της εργασίας.

#### Λύση #2 – Custom kMeans with geolocation from API

Θα μπορούσαμε να χρησιμοποιήσουμε κάποιο API (πχ το Geocoding της Google) ώστε να μετατρέπουμε τις Πόλεις-Χώρες σε συντεταγμένες και στη συνέχεια να προγραμματίζαμε έναν αλγόριθμο για kMeans χωρίς τη χρήση κάποιας έτοιμης βιβλιοθήκης (πχ scikit-learn). Ο αλγόριθμος αυτός θα εκτελούσε τον kMeans με συνυπολογισμό της χωρικής και ηλικιακής διαφοράς, επαναπροσδιορίζοντας το κέντρο της κάθε συστάδας. Ενώ ίσως είναι η πιο σωστή λύση, είναι εκτός των ορίων της εργασίας.

#### Λύση #3 - kPrototypes

Απλοποιώντας λίγο το πρόβλημα, θεωρούμε πως η ιδιότητα της χώρας δεν αποτελεί αριθμητική ιδιότητα, αλλά κατηγορηματική. Αυτό σημαίνει πως δεν έχει νόημα να χρησιμοποιήσουμε τις αποστάσεις των ids των χωρών ως είσοδο για τον αλγόριθμο ομαδοποίησης. Για παράδειγμα, δεν έχει νόημα να πούμε ότι η χώρα με id 50 έχει μεγαλύτερη απόσταση με τη χώρα με id 1 από ότι έχει με τη χώρα 49. Για το λόγο αυτό, τη χρησιμοποιούμε ως κατηγορηματική ιδιότητα, το οποίο σημαίνει ότι, κατά την ομαδοποίηση, μας ενδιαφέρει μόνο η ισότητα και όχι η απόσταση! Δηλαδή, από τη στιγμή που δεν έχουν το ίδιο id, ο αλγοριθμός θεωρεί ότι όλες οι χώρες «ισαπέχουν».

Ο αλγόριθμος **Kprototypes** είναι ουσιαστικά μία βελτίωση των αλγορίθμων kMeans και K-Mode (ο οποίος χρησιμοποιείται για την ομαδοποίηση κατηγορηματικών δεδομένων) που δέχεται τόσο αριθμητικές, όσο και κατηγορηματικές ιδιότητες.

Η είσοδος του αλγορίθμου δημιουργείται με τη μέθοδο getClusteringInput, η οποία διαβάζει το dataset των επεξεργασμένων χρηστών, κρατάει τις χρήσιμες για εκείνη στήλες, δηλαδή το Country\_ID και το Age. Συγκεκριμένα, το age περνάει από τυποποίηση (standardization), μεταφέρονται δηλαδή οι τιμές του στο διάστημα [-1, 1].

Για να αποφασίσουμε τον αριθμό των συστάδων χρησιμοποιήσαμε τη μέθοδο του αγκώνα, την οποία σχεδιάζομε γραφικά με τη μέθοδο plot\_elbow\_curve.

    # Plot elbow curve to help determine optimal number of clusters

    print("\nPlotting elbow curve...")

    clustering.plot\_elbow\_curve(2, 12, 10\_000, processed\_users)

    k = \_input("Choose number of clusters to use: ", int)

main.py

Η μέθοδος τρέχει τον αλγόριθμο σε ένα μικρό υποσύνολο, με μεταβαλλόμενο αριθμό συστάδων k, από 2 μέχρι 12, αποθηκεύοντας το κόστος κάθε τιμή του k. Στη συνέχεια εμφανίζουμε γραφικά τις τιμές αυτές και αποφασίζουμε την τιμή του k για την τελική εκτέλεση του αλγορίθμου.

def plot\_elbow\_curve(start: int, end: int, sample\_size: int,

proc\_users: pd.DataFrame) -> None:

    """Plots elbow curve. Used for optimizing K."""

    data = getClusteringInput(proc\_users)

    if sample\_size > 0 and sample\_size < len(data):

        data = data[:sample\_size]

    categorical\_index = [1]

    no\_of\_clusters = list(range(start, end+1))

    cost\_values = []

    threads = min(PC, cpu\_count())

    print(f"Number of available threads: {cpu\_count()}")

    print(f"Starting testing using {threads} threads...")

    for k in no\_of\_clusters:

        print(f"Testing with {k} clusters...")

        test\_model = KPrototypes(

            n\_clusters=k, init='Huang', n\_init=N\_INIT, n\_jobs=threads

        )

        test\_model.fit\_predict(data, categorical=categorical\_index)

        cost\_values.append(test\_model.cost\_)

    seaborn.set\_theme(style="whitegrid", palette="bright", font\_scale=1.1)

    plt.figure(figsize=(15, 7))

    ax = seaborn.lineplot(

        x=no\_of\_clusters, y=cost\_values, marker="o", dashes=False

    )

    ax.set\_title('Elbow curve', fontsize=18)

    ax.set\_xlabel('No of clusters', fontsize=14)

    ax.set\_ylabel('Cost', fontsize=14)

    ax.set(xlim=(start-0.1, end+0.1))

    plt.plot()

    plt.show()

clustering.py

Chart, line chart

Description automatically generated

Σύμφωνα με το παραπάνω γράφημα, καλές τιμές για το k φαίνεται να είναι οι 3, 5 και ίσως και 9.

Αφού επιλεχθεί από τον χρήστη ο αριθμός των συστάδων, εκτελείται η μέθοδος kPrototypes, μετρώντας το χρόνο εκτέλεσής της.

    # Run and time k-Prototypes

    print("Starting clustering...")

    start\_time = timeit.default\_timer()

    cluster\_assigned\_users = clustering.kPrototypes(k, processed\_users)

    elapsed\_time = timeit.default\_timer() - start\_time

    cluster\_assigned\_users.to\_csv(CLUSTER\_ASSIGNED\_USERS, index=False)

    print(f"Clustering with {k} clusters took {int(elapsed\_time//60)}\

        minutes and {int(round(elapsed\_time % 60, 0))} seconds.")

main.py

Αντίστοιχος κώδικας με το plot\_elbow\_curve περιέχεται και στη βασική μέθοδο του clustering kPrototypes, με τη διαφορά φυσικά ότι αυτή τη φορά τρέχει σε όλο το dataset μας. Ο kPrototypes δεν υποστηρίζει επιτάχυνση κάρτας γραφικών όπως ο κλασικός kMeans της βιβλιοθήκης scikit-learn, παρόλα αυτά έχει οριστεί η παράμετρος n\_jobs=threads, ώστε να παραλληλοποιηθεί όσο γίνεται, έστω κι αν εκτελείται στον επεξεργαστή. Παρόλα αυτά, η εκτέλεσή του διαρκεί σημαντικά περισσότερο από όσο του απλού kMeans και για αυτό αποθηκεύουμε την έξοδό του στο ενδιάμμεσο αρχείο Cluster-Assigned-Users.csv.

def kPrototypes(k: int, proc\_users: pd.DataFrame) -> pd.DataFrame:

    """Runs k-Prototypes algorithm for Users, placing them

    in one of k clusters."""

    categorical\_features\_idx = [1]

    mark\_array = getClusteringInput(proc\_users)

    threads = min(PC, cpu\_count())

    print(f"Number of available threads: {cpu\_count()}")

    print("Starting k-Prototypes using {threads} threads...")

    test\_model = KPrototypes(

        n\_clusters=k, verbose=2, init='Huang', n\_init=N\_INIT, n\_jobs=threads

    )

    clusters = test\_model.fit\_predict(

        mark\_array, categorical=categorical\_features\_idx

    )

    # Add Cluster column to dataframe

    proc\_users['Cluster'] = list(clusters)

    return proc\_users

clustering.py

### Υπολογισμός μέσων βαθμολογιών συστάδων

Αφού χωρίσουμε τους χρήστες σε ομάδες, υπολογίζουμε τη μέση βαθμολογία της κάθε ομάδας για κάθε βιβλίο που έχει αξιολογίσει με τη μέθοδο createAvgClusterRatings.

    # Create a dataframe containing the average

    # rating of every book per cluster

    print("Creating average cluster ratings CSV...")

    avg\_clust\_ratings = functions.createAvgClusterRatings(cluster\_assigned\_users)

    avg\_clust\_ratings.to\_csv(AVG\_CLUSTER\_RATINGS)

main.py

Η μέθοδος αυτή αρχικά συγχωνεύει τις αξιολογήσεις χρηστών μαζί με το επεξεργασμένο dataframe των χρηστών που περιέχει τις συστάδες στις οποίες ανήκουν. Στη συνέχεια, κρατάει μόνο τις απαραίτητες στήλες isbn, Cluster και rating, ομαδοποιεί τις εγγραφές ανά βιβλίο και συστάδα και υπολογίζει το μέσο όρο της κάθε ομαδοποίησης. Τέλος, επιστρέφει το δημιουργημένο dataframe που περιέχει τις μέσες βαθμολογίες συστάδων, το οποίο αποθηκεύεται στο αντίστοιχο ενδιάμεσο αρχείο.

def createAvgClusterRatings(

        cluster\_assignement\_df: pd.DataFrame

    ) -> pd.DataFrame:

    """Function that accepts as input the cluster assignement DataFrame and

    restores User IDs to it. Then, it combines this DF with the Book-Ratings

    CSV, averaging out the book ratings per cluster. The combined DF contains

    the columns isbn, cluster and rating and is finally returned."""

    # Open book ratings CSV in read mode

    books\_ratings\_df = pd.read\_csv(RATINGS)

    # Merge the two DataFrames on UIDs

    result = pd.merge(right=books\_ratings\_df, left=cluster\_assignement\_df,\

        how="left", left\_on="User\_ID", right\_on="uid", validate="one\_to\_many")

    # Drop useless columns

    result.drop(

        ["uid", "User\_ID", "Country", "Country\_ID", "Age"], axis=1, inplace=True)

    # Group ratings by isbn and Cluster and sort resulting DataFrame

    avg\_cluster\_ratings = result.groupby(["isbn", "Cluster"]).mean()

    return avg\_cluster\_ratings

functions.py

### Υπολογισμός συνδυαστικών βαθμολογιών με συμπλήρωση κενών βαθμολογιών από το μέσο όρο συστάδων

Για τη βελτίωση των επιστρεφόμενων βιβλίων, όπου δεν υπάρχει βαθμολογία χρήστη στο 2ο ερώτημα συμπληρώνεται από τη μέση βαθμολογία της συστάδας του χρήστη, όπου αυτή υπάρχει. Αυτό επιτυγχάνεται με την ίδια μέθοδο που χρησιμοποιείται και στο πρώτο ερώτημα calculateCombinedScores, η οποία όμως πλέον καλείται με τα κατάλληλα ορίσματα, όπως φαίνεται παρακάτω:

    # Get the re-calculated document scores by using

    # user's cluster's average ratings to "rate" unrated books

    print("Re-calculating combined scores using user's cluster's average book ratings...")

    combined\_scores\_clusters\_df = functions.calculateCombinedScores(es\_reply, user\_id,\

        use\_cluster\_ratings=True, avg\_clust\_ratings = avg\_clust\_ratings,\

            cluster\_assigned\_users = cluster\_assigned\_users)

    combined\_scores\_clusters\_df.to\_csv(SCORES\_W\_CLUST, index=False)

main.py

# Ερώτημα 3 – Νευρωνικό δίκτυο

Για το 3ο ερώτημα, μας ζητάται να εκπαιδεύσουμε ένα νευρωνικό δίκτυο το οποίο θα προβλέπει τις μέσες βαθμολογίες συστάδων για τα βιβλία που δεν έχουν βαθμολογηθεί από αυτές, χρησιμοποιώντας τις περιλήψεις τους. Πριν όμως περαστούν στο δίκτυό μας, οι περιλήψεις θα πρέπει να υποστούν μία προ-επεξεργασία.

### Word Embeddings

Για τη διανυσματοποίηση των περιλήψεων με τη χρήση Word Embedding, δοκιμάσαμε διάφορες μεθόδους και εν τέλει καταλήξαμε σε 2. Η 1η είναι η χρησιμοποίηση ενός **Embedding Layer** στο νευρωνικό μας δίκτυο και η 2η η εκτέλεση ενός ξεχωριστού **Doc2Vec** μοντέλου, με το οποίο μετατρέψαμε όλες τις περιλήψεις σε πολυδιάστατα διανύσματα και στη συνέχεια τις περάσαμε από ένα ξεχωριστό νευρωνικό δίκτυο για τις προβλέψεις των βαθμολογιών.

### NN with Embedding Layer

Για την υλοποίηση με τον 1ο τρόπο, αρχικά υπολογίζουμε το μέγεθος του λεξικού των περιλήψεων των βιβλίων, καθώς και το μέγεθος της μεγαλύτερης περίληψης σε λέξεις με τη μέθοδο calculateVocab. Στη συνέχεια, εκπαιδεύουμε ένα νευρωνικό δίκτυο πάνω στις βαθμολογίες του cluster στον οποίο ανήκει ο χρήστης, μέσω της μεθόδου trainClusterNetwork.

    # Use a single model with an embedding layer that vectorizes

    # summaries and later predicts missing ratings

    vocab\_size, max\_length = emb\_layer\_networks.calculateVocab(books)

    print(f"Vocab size: {vocab\_size}, Max length: {max\_length}")

    model = emb\_layer\_networks.trainClusterNetwork(users\_cluster,

                                                   avg\_clust\_ratings, books,

                                                   vocab\_size, max\_length)

    combined\_scores\_clusters\_nn, \_ = functions.calculateCombinedScores(

        es\_reply, user\_id, use\_cluster\_ratings=True,

        avg\_clust\_ratings = avg\_clust\_ratings,

        cluster\_assigned\_users = cluster\_assignement, use\_nn=2,

        model=model, books=books, vocab\_size=vocab\_size, max\_length=max\_length)

    combined\_scores\_clusters\_nn.to\_csv(SCORES\_W\_CLUST\_AND\_NN\_EMB, index=False)

main.py

Κατά την εκτέλεσή της, γίνεται σύζευξη των dataframes avg\_clust\_ratings και books ώστε να ανακτήσουμε τις περιλήψεις και τις μέσες βαθμολογίες των βιβλίων που έχουν βαθμολογηθεί από τον cluster του χρήστη. Στη συνέχεια, οι περιλήψεις περνάνε από επεξεργασία ώστε να μετατραπούν σε κατάλληλες εισόδους για το νευρωνικό μας με τη μέθοδο getNetworkInput, από όπου επιλέγουμε και το μοντέλο που θα χρησιμοποιηθει. Τέλος, εκπαιδεύουμε το μοντέλο με τη μέθοδο trainNetwork και εμφανίζουμε γραφήματα σχετικά με την εκπαίδευσή του με τη μέθοδο plotHistory.

def trainClusterNetwork(cluster: int, avg\_clust\_ratings: pd.DataFrame,

                        books: pd.DataFrame, vocab\_size: int, max\_length: int):

    """Train regression NN on user’s cluster’s average ratings."""

    print(f"Preparing network input for cluster {cluster}...")

    # Add book summaries to the avg\_clust\_ratings df

    print(avg\_clust\_ratings.loc[avg\_clust\_ratings["Cluster"]==cluster].head())

    cluster\_ratings = pd.merge(

        right=avg\_clust\_ratings.loc[avg\_clust\_ratings["Cluster"]==cluster],

        left=books, on="isbn", validate="one\_to\_one")

    summaries = cluster\_ratings["summary"].to\_list()

    ratings = cluster\_ratings["rating"].to\_list()

    X, Y, model = getNetworkInput(summaries, ratings, vocab\_size, max\_length)

    print(f"Training network for cluster {cluster}...")

    # Print input and label shapes

    print(X.shape)

    print(Y.shape)

    # Print model summary

    model.summary()

    # Start training

    history = model.fit(X, Y, validation\_split=0.3, epochs=10,

                        batch\_size=10, verbose=1)

    plotHistory(history)

    return model

emb\_layer\_networks.py

Η μέθοδος getNetworkInput προ-επεξεργάζεται τις περιλήψεις με την preProcessSummaryv3, τις κωδικοποιεί με τη χρήση one hot encoding και τέλος προσθέτει μηδενικά (σαν κενές λέξεις ουσιαστικά) στις περιλήψεις που έχουν μικρότερο μέγεθος από τη μεγαλύτερη περίληψη (max\_length). Τέλος, επιλέγεται το μοντέλο που θα χρησιμοποιηθεί στην εκπαίδευση, μετατρέπονται οι βαθμολογίες σε numpy array και επιστρέφεται η έτοιμη είσοδος του νευρωνικού.

def getNetworkInput(summaries: list, ratings: list, vocab\_size, max\_length, classifier = False):

    # Preprocess summaries

    preproc\_summaries = [preProcessSummaryv3(summary) for summary in summaries]

    # One hot encode words of documents

    encoded\_sums = [one\_hot(d,vocab\_size) for d in preproc\_summaries]

    # Add padding

    X = pad\_sequences(encoded\_sums,maxlen=max\_length,padding='post')

    # Normalize ratings

    if classifier:

        Y = np.array([customOHE2(r) for r in ratings])

        return (X, Y, classifier\_model\_1(vocab\_size, max\_length))

    # Define the model

    model = base\_model\_1(vocab\_size, max\_length)

    Y = np.array(ratings)

    return (X, Y, model)

emb\_layer\_networks.py

Η preProcessSummaryv3 μεταφράζει τους ανεπιθύμητους html escape χαρακτήρες (π.χ. &quot;), τα σημεία στίξης και τα περιττά κενά και μετατρέπει όλους οι χαρακτήρες σε πεζοούς. Αυτό επιτυγχάνεται με τη χρήση μιας κανονικής έκφρασης και του html.unescape.

def preProcessSummaryv3(summary) -> str:

    return re.sub('[^A-Za-z0-9]+', ' ', html.unescape(summary))

### Νευρωνικό δίκτυο

Το μοντέλο που επιλέχθηκε είναι ένα regression νευρωνικό δίκτυο. Αυτό σημαίνει ότι, σε αντίθεση με τα classification δίκτυα που κάνουν κατηγοριοποίηση, το μοντέλο μας έχει ως έξοδο μία τιμή. Το μοντέλο μας έχει ένα Embedding layer για την αναγνώριση των περιλήψεων, καθώς και 2 κρυφά layers με μεγέθη 256 και 64. Τέλος, το output layer αποτελείται από 1 νευρώνα, η τιμή του οποίου καθορίζει και την predicted τιμή του δικτύου για τη βαθμολογία ενός βιβλίου. Η συνάρτηση κόστους που επιλέχθηκε είναι το μέσο τετραγωνισμένο σφάλμα, καθώς θέλουμε να τιμωρούμε παραπάνω το δίκτυο για ακραίες τιμές σε σχέση με το μέσο απόλυτο σφάλμα, το οποίο όμως καταγράφουμε, αφού είναι πιο διαισθητική μετρική για να καταλάβουμε πόσο απέχουν κατά μέσο όρο οι προβλεπόμενες βαθμολογίες του δικτύου μας από τις πραγματικές.

def base\_model\_1(vocab\_size, max\_length):

    max\_length = max\_length

    # Create model

    model = Sequential()

    model.add(Embedding(input\_dim=vocab\_size, output\_dim=128,

                        input\_length=max\_length))

    model.add(Flatten())

    model.add(Dense(256, activation='relu'))

    model.add(Dense(64, activation='relu'))

    model.add(Dense(1, activation='relu'))

    model.compile(optimizer='adam',loss="mse",metrics=['mae'])

    return model

emb\_layer\_networks.py

Τέλος, για την πρόβλεψη των τιμών τροποποιήσαμε τη μέθοδο getAvgClusterRating (και συνεπώς και τη calculateCombinedScores ως προς τα ορίσματά της) ώστε να μπορεί να χρησημοποιεί και το δίκτυο για την πρόβλεψη των τιμών που λείπουν, ανάλογα με τα ορίσματά της.

Ουσιαστικά αυτό που αλλάζει είναι ότι, αν δε βρει βαθμολογία ενός βιβλίου ούτε στο avg\_clust\_ratings, χρησιμοποιεί το μοντέλο για να προβλέψει τη βαθμολογία του βιβλίου. Πιο συγκεκριμένα, δέχεται ως όρισμα έναν ακέραιο use\_nn ανάλογα με τη μέθοδο που χρησιμοποιούμε (Embedding layer ή Doc2Vec), καθώς και το μοντέλο, κάνει την απαραίτητη προεπεξεργασία της συνάρτησης και τέλος καλεί το model.predict() ώστε να υπολογίσει τη βαθμολογία.

def getAvgClusterRating(users\_cluster: int, isbn: str, avg\_clust\_ratings:

                        pd.DataFrame, use\_nn: int, model: Sequential = None,

                        vectorized\_books: pd.DataFrame = None,

                        books: pd.DataFrame = None, vocab\_size: int = None,

                        max\_length: int = None) -> tuple:

    """Given a user id and an book's isbn, it returns the

    average rating of user's cluster for the specified book."""

    # Try getting user's cluster's rating of specified book

    try:

        rating = avg\_clust\_ratings.loc[(avg\_clust\_ratings["isbn"] == isbn) &

                                       (avg\_clust\_ratings["Cluster"] ==

                                        users\_cluster)]["rating"].iloc[0]

        return rating

    # If a book hasn't been rated by a cluster and use\_nn = 0,

    # return the median value of 5 stars out of 10

    except:

        # Using vectorized books

        if use\_nn == 1:

            vect\_sum = vectorized\_books["Vectorized\_Summary"].\

                loc[vectorized\_books["isbn"] == isbn].to\_list()[0]\

                    .reshape(1, -1)

            return model.predict(vect\_sum)[0][0]

        # Not using vectorized books

        elif use\_nn == 2:

            summary = books[books["isbn"]==isbn]["summary"].to\_list()[0]

            # Preprocess summary

            preproc\_summary = preProcessSummaryv3(summary)

            # One hot encode words of documents

            encoded\_sum = one\_hot(preproc\_summary, vocab\_size)

            # Add padding

            X = pad\_sequences([encoded\_sum],maxlen=max\_length,padding='post')

            return model.predict(X)[0][0]

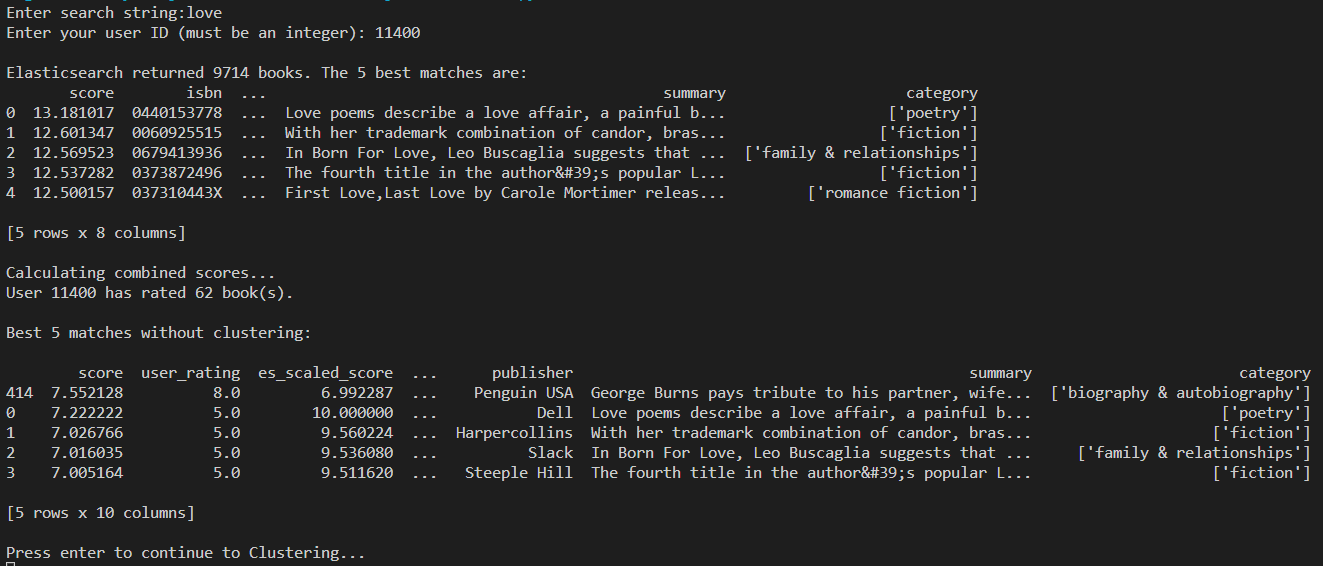
functions.py

Παρατηρήσαμε ότι αυτή η υλοποίηση ήταν ελαφρώς καλύτερη από την 2η μας επιλογή, οπότε είναι και η κύρια που χρησιμοποιούμε. Η 2η μέθοδος δε θα αναλυθεί στην αναφορά αυτή, όμως βρίσκεται στον κώδικα σε σχόλια και δουλεύει κανονικά, αν και φαίνεται να είναι πιο αργή.

# Έξοδος κατά την εκτέλεση

### ElasticSearch

Τρέχουμε το ερώτημα “love” στην elastic ως ο χρήστης 11400 και παρατηρούμε τα 5 καλύτερα βιβλία. Όπως φαίνεται, τα περισσότερα βιβλία έχουν user\_rating 5, που όμως δεν είναι η πραγματική βαθμολογία του χρήστη, έχει προκύψει επειδή την ορίσαμε εμείς ως default rating (αφού ο χρήστης δεν είχε βαθμολογήσει το αντίστοιχο βιβλίο). Παρόλα αυτά, το πρώτο βιβλίο το έχει βαθμολογήσει ο χρήστης με 8 και ενώ έχει σχετικά μικρό σκορ από την elastic (6.99/10), προωθείται πιο ψηλά στην κατάταξη λόγω της υψηλής βαθμολογίας του χρήστη.



### Clustering

Επειδή ο χρήστης δεν υπάρχει στο αρχείο BX-Users.csv, δεν έχει νόημα να ξαναυπολογίσουμε τα σκορ των βιβλίων, αφού δε θα ανήκει σε κάποιον cluster.

Text

Description automatically generated

Ξανατρέχουμε το πρόγραμμα απ την αρχή ως κάποιος χρήστης που αυτήν τη φορά υπάρχει στο BX-Users. Aυτή τη φορά επιλέξαμε το ερώτημα “animal”. Μετά από λίγη ώρα (αρκετή μάλλον λόγω εκτέλεσης σε λάπτοπ), προκύπτει η ομαδοποίηση των χρηστών σε συστάδες, σύμφωνα με την οποία ο χρήστης 12 ανήκει στη συστάδα .

Text

Description automatically generated

Συνυπολογίζοντας πλέον και τις μέσες βαθμολογίες της συστάδας, παρατηρούμε ότι συμπληρώνονται κάποιες τιμές από τις βαθμολογίες των ομάδων.

Graphical user interface, text, application

Description automatically generated

### Νευρωνικό δίκτυο

Κατά την εκτέλεση του νευρωνικού δικτύου, παρατηρούμε ότι ενώ το κόστος αλλά και το μέσο απόλυτο σφάλμα ελαχιστοποιείται κατά την εκτέλεση, κατά την αξιολόγηση παραμένει σταθερό. Αυτό οφείλεται εν μέρει στα δεδομένα μας, καθώς δε φαίνεται να υπάρχει καθαρή συσχέτιση μεταξύ περιλήψεων και μέσων βαθμολογιών ανά ομάδα. Επομένως, αυτό που γίνεται είναι το μοντέλο να «αποστηθίζει» τις βαθμολογίες κατά την εκτέλεση και να μην ξέρει πώς να διαχειριστεί τις καινούριες περιλήψεις που δεν έχει δει. Αυτό το φαινόμενο λέγεται overfitting. Παρόλα αυτά, ακόμα και η μέση απόκλιση 3.5 είναι καλύτερη από τη μέση απόκλιση 5 που έχουμε όταν βάζουμε από μόνοι μας 5/10 στα βιβλία που δεν έχουν βαθμολογία.

Graphical user interface, chart

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated

Όπως παρατηρούμε, πλέον όλα τα βιβλία έχουν κάποια βαθμολογία διαφορετική του 5 στο user\_rating, οι οποίες προκύπτουν κυρίως από τις μέσες βαθμολογίες συστάδων και το νευρωνικό.

A screenshot of a computer

Description automatically generated with medium confidence