ΑΝΑΚΤΗΣΗ ΠΛΗΡΟΦΟΡΙΑΣ

ΥΛΟΠΟΙΗΤΙΚΟ PROJECT 2020

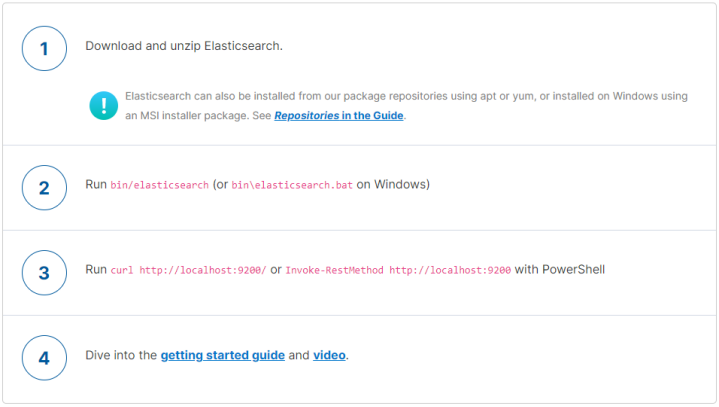
Διδάσκων: Χ. Μακρής

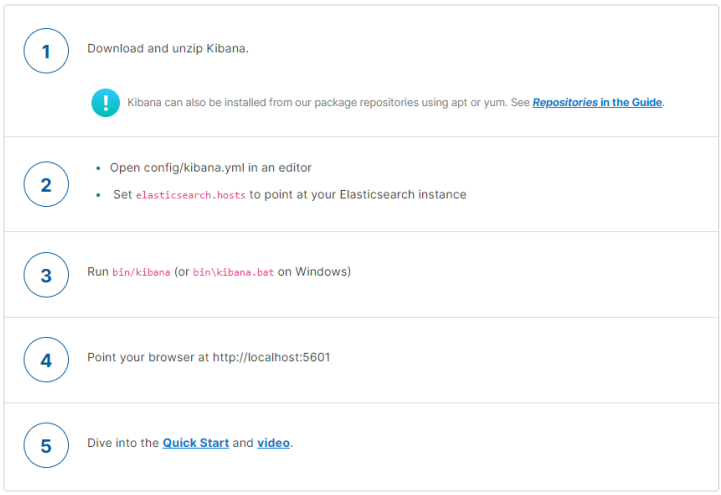
Επικουρικό: Α. Μπομπότας, Γ. Ρόμπολας

ΑΓΓΕΛΟΣ ΡΑΓΚΟΥΣΗΣ 1053566 *up1053566@upnet.gr*
ΣΤΑΥΡΟΣ ΜΠΟΥΡΑΣ 1053565 *up1053565@upnet.gr*

ΕΙΣΑΓΩΓΗ

Στα πλαίσια της παρούσας άσκησης μας ζητείται να υλοποιήσουμε μια μηχανή αναζήτησης κινηματογραφικών ταινιών η οποία θα βασίζεται στην Elasticsearch και θα αποφασίζει την σειρά παρουσίασης των αποτελεσμάτων χρησιμοποιώντας τεχνικές μηχανικής μάθησης. Σαν γλώσσα υλοποίησης χρησιμοποιήσαμε την Python. Η εργασία δημιουργήθηκε σε 2 περιβάλλοντα, το Spyder και το Jupyter. Η εγκατάσταση της Elasticsearch έγινε μέσω της ιστοσελίδας https://www.elastic.co/downloads/elasticsearch όπου είναι η επίσημη διεύθυνση για την εγκατάσταση της μηχανής. Επιπλέον, για πιο εύκολη οπτικοποίηση (visualization) και πλοήγηση (navigation) στα δεδομένα χρησιμοποιήσαμε το Kibana, το οποίο εγκαταστήσαμε μέσω της ιστοσελίδας https://www.elastic.co/downloads/kibana Παρακάτω παρατίθενται οι οδηγίες εγκατάστασης για την Elasticsearch και το Kibana αντίστοιχα.





Οι βασικές βιβλιοθήκες και τα πακέτα της Python που χρησιμοποιήθηκαν για την εργασία είναι τα εξής :

- csv
- elasticsearch
- pandas
- numpy
- keras

ΕΡΩΤΗΜΑ 1

Αρχικά, εγκαταστήσαμε στο σύστημα την Elasticsearch, και στην συνέχεια μέσω ενός προγράμματος διαβάζουμε τις εγγραφές του αρχείου movies.csv και τις εισάγουμε στην Elasticsearch.

```
import csv
from elasticsearch import helpers, Elasticsearch


es = Elasticsearch([{'host': 'localhost', 'port': 9200}])

with open('movies.csv',encoding="utf8") as outfile:
    reader = csv.DictReader(outfile)
    helpers.bulk(es, reader, index="index_name", doc_type="type")
```

Στην συνέχεια ακολουθεί ένα πρόγραμμα το οποίο δέχεται ως είσοδο ένα αλφαριθμητικό και επιστρέφει την λίστα των ταινιών που ταιριάζουν με αυτό διατεταγμένη σε φθίνουσα σειρά σύμφωνα με την μετρική ομοιότητας της Elasticsearch (BM25).

```
from elasticsearch import Elasticsearch

es=Elasticsearch([{'host':'localhost','port':9200}])

x=input("Give a title: ")

res= es.search(index="index_name",body={'query':{'match':{'title':x}}})
all_results = res['hits']['hits']

for  i in range(len(all_results)):
    print(all_results[i]['_source'])

    print("Score: "+str(all_results[i]['_score']))
```

Πράγματι, εάν εισάγουμε ως είσοδο τον τίτλο “Toy Story” θα πάρουμε τα εξής αποτελέσματα:

```
Give a title: Toy Story
{'movieId': '1', 'title': 'Toy Story (1995)', 'genres': 'Adventure|Animation|Children|Comedy|Fantasy'}
Score: 13.762764
{'movieId': '3114', 'title': 'Toy Story 2 (1999)', 'genres': 'Adventure|Animation|Children|Comedy|Fantasy'}
Score: 12.413258
{'movieId': '78499', 'title': 'Toy Story 3 (2010)', 'genres': 'Adventure|Animation|Children|Comedy|Fantasy|IMAX'}
Score: 12.413258
{'movieId': '106022', 'title': 'Toy Story of Terror (2013)', 'genres': 'Animation|Children|Comedy'}
Score: 11.3047695
{'movieId': '4929', 'title': 'Toy, The (1982)', 'genres': 'Comedy'}
Score: 8.264357
{'movieId': '5843', 'title': 'Toy Soldiers (1991)', 'genres': 'Action|Drama'}
Score: 8.264357
{'movieId': '2108', 'title': 'L.A. Story (1991)', 'genres': 'Comedy|Romance'}
Score: 5.4984074
{'movieId': '4296', 'title': 'Love Story (1970)', 'genres': 'Drama|Romance'}
Score: 5.4984074
{'movieId': '127136', 'title': 'True Story (2015)', 'genres': 'Drama|Mystery|Thriller'}
Score: 5.4984074
{'movieId': '898', 'title': 'Philadelphia Story, The (1940)', 'genres': 'Comedy|Drama|Romance'}
Score: 4.9592614
```

ΕΡΩΤΗΜΑ 2

Στο συγκεκριμένο ερώτημα τροποποιούμε το παραπάνω πρόγραμμα έτσι ώστε να δέχεται ως επιπρόσθετη είσοδο έναν ακέραιο αριθμό, το αναγνωριστικό του χρήστη. Επιπρόσθετα, αλλάζουμε τον τρόπο ταξινόμησης της λίστας των αποτελεσμάτων. Τώρα, τα αποτελέσματα θέλουμε να εμφανίζονται σύμφωνα με μία νέα μετρική όπου προσμετράται κατά 30% η βαθμολογία του χρήστη, κατά 10% η μέση βαθμολογία και κατά 60% η μετρική της Elasticsearch.

Αρχικά, δημιουργούμε 3 λίστες:

1) Την id_list η οποία περιέχει τα movieId που επιστράφηκαν από το query.
2) Την score_list η οποία περιέχει μόνο τα scores.
3) Την full_info η οποία περιέχει όλη την πληροφορία.

```python
from elasticsearch import Elasticsearch
import pandas as pd
import numpy as np

es=Elasticsearch([{'host':'localhost','port':9200}])
x=input("Give a title: ")
res= es.search(index="index_name",body={'query':{'match':{'title':x}}})
all_results = res['hits']['hits']

id_list=[]
score_list=[]
full_info=[]

for i in all_results:
    id_list.append(i['_source']['movieId'])   #krataei ta movieid pou epistrafikan
    score_list.append(i['_score'])            #krataei mono ta scores
    full_info.append(i['_source'])            #krataei olh th pliroforia
```

Έπειτα, εισάγουμε τα δεδομένα από το ratings.csv αρχείο και κρατάμε σε 3 διανύσματα ξεχωριστά όλα τα user id των users που βαθμολόγησαν, όλα τα movie id των user που βαθμολόγησαν και τέλος όλες τις βαθμολογίες των users που βαθμολόγησαν. Στη συνέχεια, υπολογίζουμε τους μέσους όρους των ταινιών που ταινιών που επιστράφηκαν από τις βαθμολογίες των users, και τους αποθηκεύουμε στην λίστα average_rating.

```
ratings_data = pd.read_csv("ratings.csv")

#ola ta user id twn users pou vathmologisan
user_id_from_ratings = ratings_data['userId']
user_id_from_ratings = user_id_from_ratings.to_numpy()

#ola ta movies id twn users pou vathmologisan
movie_id_from_ratings = ratings_data['movieId']
movie_id_from_ratings = movie_id_from_ratings.to_numpy()

#oles oi vathmologies twn users pou vathmologisan
rating_from_ratings = ratings_data['rating']
rating_from_ratings = rating_from_ratings.to_numpy()

#oi mesoi oroi twn tainiwn pou epistrafikan apo tis vathmologies twn users
average_rating=[]

for i in range(len(id_list)):
    count =0
    sum_rating=0

    for j in range(len(movie_id_from_ratings)):
        if (int(id_list[i])== int(movie_id_from_ratings[j])):
            sum_rating = sum_rating + rating_from_ratings[j]
            count=count+1
    average_rating.append(sum_rating/count)
```

Μετά, δημιουργούμε έναν πίνακα, τον οποίο θα γεμίσουμε με τις βαθμολογίες του user (που δώσαμε σαν input) για τις ταινίες που μας επέστρεψε η Elasticsearch.

```
input_id = input("Give a user id: " )
ratings_from_user=[]
movies_from_user=[]

for i in range(len(user_id_from_ratings)):
    if(int(input_id)== int(user_id_from_ratings[i])):
        ratings_from_user.append(rating_from_ratings[i])
        movies_from_user.append(movie_id_from_ratings[i])

# oi vathmologies tou user(pou dwsame san input) gia tis tainies pou gurise h elastic search

rated_by_user=np.zeros(len(id_list))

#update
for i in range(len(id_list)):
    for j in range(len(movies_from_user)):
        if(int(id_list[i])==int(movies_from_user[j])):
            rated_by_user[i]=ratings_from_user[j]
```

Τέλος, ορίζουμε την νέα μετρική μας, αποθηκεύουμε τα αποτελέσματα που μας δίνει η μετρική αυτή στον πίνακα new_scores και τα συνδυάζουμε την πληροφορία που περιέχεται στον πίνακα full_info. Με άλλα λόγια με την χρήση της εντολής np.vstack ενώνεται όλη η πληροφορία με τη στήλη των νέων scores της νέας μετρικής.

```
#new metric
new_scores=[]

for i in range(len(id_list)):
    new_score=0.30*rated_by_user[i]+0.10*average_rating[i]+0.60*score_list[i]
    new_scores.append(new_score)

#enwnetai olh h pliroforia me thn stili twn newn score ths neas metrikis
result = np.vstack((full_info,new_scores)).T
print("\n",result)
```

ΕΡΩΤΗΜΑ 3

Παρατηρήθηκε ότι η μετρική του ερωτήματος 2 είναι αναξιόπιστη, διότι η βαθμολογία του χρήστη για την συγκεκριμένη ταινία που αναζητείται έχει υψηλό συντελεστή βαρύτητας (0.3) και συνήθως δεν υπάρχει, δεδομένου ότι κάθε χρήστης έχει βαθμολογήσει μικρό πλήθος ταινιών. Έτσι, ο όρος αυτός μηδενίζεται στο γραμμικό συνδυασμό της μετρικής μας, και άρα το αποτέλεσμα καθορίζεται σχεδόν αποκλειστικά από τη μετρική της Elasticsearch με συντελεστή 0.6. Για την συμπλήρωση των βαθμολογιών όλων των χρηστών για όσες ταινίες δεν έχουν βαθμολογήσει, ακολουθήθηκε η παρακάτω διαδικασία.

Αρχικά, φορτώνουμε πάλι τα Dataframes των ταινιών και των βαθμολογιών των χρηστών (**movies** και **ratings**), όπως φαίνεται στην παρακάτω εικόνα. Δημιουργούμε επίσης ένα τρίτο Dataframe (**merged_csv**) που προκύπτει από την ένωση (merge) των δύο Dataframes και περιέχει όλες τις πληροφορίες μόνο των ταινιών που βαθμολόγησε ο κάθε χρήστης. Στο Dataframe αυτό κάνουμε ομαδοποίηση ανά χρήστη και ανά είδος ταινίας και βρίσκουμε τον μέσο όρο της βαθμολογίας κάθε χρήστη για κάθε είδος ταινίας στο οποίο έχει βαθμολογήσει τουλάχιστον μία ταινία. Αποθηκεύουμε το αποτέλεσμα σε ένα Dataframe (**avg_ratings_per_uId_genres).**

Όμως, στο Dataframe αυτό πρέπει να προστεθούν και οι κατηγορίες ταινιών για τις οποίες ο χρήστης δεν έχει βαθμολογήσει καμία ταινία. Για αυτό, φτιάχνουμε ένα ακόμα Dataframe (**all_genres_per_userId**), στο οποίο, με καρτεσιανό γινόμενο, συσχετίζονται όλοι οι χρήστες με όλες τις κατηγορίες ταινιών, με μηδενική βαθμολογία σε όλες τις κατηγορίες. Ενώνοντας τα δύο τελευταία Dataframes (merge) προσθέτουμε επομένως και τις υπόλοιπες κατηγορίες με μηδενική βαθμολογία. Τέλος, μορφοποιούμε τον τελικό πίνακα σε ένα τελικό Dataframe (**kmeans_input)** το οποίο έχει ως γραμμές όλους τους χρήστες και ως στήλες όλες τις κατηγορίες ταινιών.

Read the csv files

```
ratings = pd.read_csv( '/Users/AngelosRgs/Desktop/ratings.csv' )
movies = pd.read_csv( '/Users/AngelosRgs/Desktop/movies.csv' )
```

Create a dataframe where each row is a user and each column has the average rating of this user for a movie genre

```
all_unique_userIds = ( ratings["userId"].unique() )
all_unique_genres = ( movies["genres"].unique() )
df = np.transpose( [ np.tile(all_unique_userIds, len(all_unique_genres)), np.repeat(all_unique_genres, len
(all_unique_userIds))])
all_genres_per_userId = pd.DataFrame( df, columns = ["userId","genres"] )
```

```
merged_csv = pd.merge( ratings, movies, on = "movieId" )
avg_ratings_per_uId_genres = merged_csv.groupby( ['userId', 'genres'] )["rating"].mean().reset_index()
all_avg_ratings_per_uId_genres = pd.merge( all_genres_per_userId, avg_ratings_per_uId_genres, on= ['userId',
'genres'], how="left" )
all_avg_ratings_per_uId_genres["rating"] = all_avg_ratings_per_uId_genres.rating.fillna(0)
```

```
kmeans_input = pd.pivot_table( all_avg_ratings_per_uId_genres, index='userId', columns='genres',
values='rating' )
```

Στη συνέχεια, εφαρμόζουμε τον αλγόριθμο Kmeans με 6 clusters για το Dataframe που φτιάξαμε, όπως φαίνεται στην παρακάτω εικόνα. Ομαδοποιούμε έτσι τους χρήστες σε 6 ομάδες, με βάση την ομοιότητα των βαθμολογιών τους ως προς όλες τις κατηγορίες ταινιών.

KMeans clustering with 6 clusters

```
kmeans=KMeans( n_clusters=6, init='k-means++',random_state=0 )
y=kmeans.fit_predict( kmeans_input )
centers = pd.DataFrame( kmeans.cluster_centers_ )
centers.columns = kmeans_input.columns.values
```

Στόχος μας τώρα είναι να δημιουργήσουμε το τελικό Dataframe (**final_table**) που περιέχει όλους τους χρήστες (γραμμές) και όλες τις ταινίες (στήλες). Δημιουργούμε αρχικά το Dataframe αυτό συσχετίζοντας κάθε χρήστη με όλες τις ταινίες πάλι μέσω του καρτεσιανού γινομένου πινάκων (μεταξύ των χρηστών και των ταινιών), όπως φαίνεται στην παρακάτω εικόνα. Οι τιμές του πίνακα είναι η πραγματική βαθμολογία κάθε χρήστη για κάθε ταινία.

Create a final dataframe where each row is a user and each column has the rating of the user for a movie

```
all_movieId = (movies["movieId"])
df2 = np.transpose([np.tile(all_unique_userIds, len(all_movieId)), np.repeat(all_movieId, len
(all_unique_userIds))])
all_userIds_movieIds = pd.DataFrame( df2, columns = ["userId","movieId"] )
final_table = pd.merge( all_userIds_movieIds, ratings, on=["userId","movieId"], how="left" )
final_table = final_table.drop( "timestamp", axis = 'columns' )
final_table = final_table.sort_values( by = 'movieId', ascending= False )
final_table = pd.pivot_table( final_table, index='userId', columns='movieId', values='rating', dropna=False )
```

Στο Dataframe τώρα προσθέτουμε, για κάθε χρήστη, τη βαθμολογία που λείπει για κάθε ταινία με τη μέση βαθμολογία για αυτή τη ταινία από τους υπόλοιπους χρήστες του ίδιου cluster, όπως φαίνεται στην παρακάτω εικόνα. Παρατηρούμε όμως ότι κάποιες βαθμολογίες συνεχίζουν να λείπουν, εάν κανένας χρήστης από αυτό το cluster δεν βαθμολόγησε την ταινία. Τότε, συμπληρώνουμε και αυτές τις τιμές με τη μέση βαθμολογία (2.5/5.0).

For each user, fill the missing ratings for every movie with its average rating given by the other users of the same cluster

```
[-]
#del final_table3
final_table3 = pd.DataFrame(columns=final_table.columns)
final_table['Clusters'] = y

for i in range (6):
    k1 = final_table.loc[final_table['Clusters'] == i]
    k2 = k1.apply(lambda x: x.fillna(x.mean()),axis=0)
    final_table3 = final_table3.append(k2)

final_table = final_table3.sort_index()

# fill the movies with no rating with an average rating of 2.5/5.0
final_table = final_table3.fillna(2.5)
final_table = final_table3.drop('Clusters', axis=1)
```

Στη συνέχεια, έχοντας φτιάξει το τελικό Dataframe, πρέπει να συνδυάσουμε τα αποτελέσματα από το προηγούμενο ερώτημα με τις νέες βαθμολογίες. Για αυτό, φορτώνουμε τον πίνακα με τον μέσο όρο των βαθμολογιών των ταινιών του προηγούμενου ερωτήματος, τον πίνακα με τον αριθμό των ταινιών αυτών, και τον πίνακα με τα scores των ταινιών από την Elasticsearch, όπως φαίνεται στην παρακάτω εικόνα.

Load the results from the query of the previous exercise

```
[-]
from numpy import load

# load array
data = load('erotima2_result.npy', allow_pickle = True)
id_list = data[0]
average_rating = data[1]
score_list = data[2]
full_info = data[3]
userId = data[4]
print(data)
```

Ο νέος πίνακας των βαθμολογιών του χρήστη για τις υποψήφιες ταινίες, χωρίς μηδενικές βαθμολογίες, προκύπτει απλά επιλέγοντας από ολόκληρο το Dataframe τη γραμμή που εκφράζει το χρήστη 13 και από τη γραμμή αυτή μόνο τις βαθμολογίες των στηλών που ταιριάζουν στον τίτλο “Toy Story”. Έτσι, κατασκευάζουμε πάλι την ίδια μετρική με το ερώτημα 2 και τυπώνουμε τελικά τις υποψήφιες ταινίες με όλα τα στοιχεία τους και τη νέα τους βαθμολογία.

Save the new ratings of ex.3 for the given user in ex.2 and for the given movie titles in ex.2

```
rated_by_user=np.zeros(len(id_list))

for i in range(len(id_list)):
    k1 = final_table.iloc[int(userId)]
    index = int(id_list[i])
    rated_by_user[i] = k1[index]
print(rated_by_user)
```

Combine the new scores with the existing metrics

```
#new metric
new_scores=[]

for i in range(len(id_list)):
    new_score=0.30*rated_by_user[i]+0.10*average_rating[i]+0.60*score_list[i]
    new_scores.append(new_score)

print("The new scores are:")
print(new_scores)

#enwnetai olh h pliroforia me thn stili twn newn score ths neas metrikis
result = np.vstack((full_info,new_scores)).T
print(result)
```

ΕΡΩΤΗΜΑ 4

Στο ερώτημα αυτό ακολουθείται μια εναλλακτική μέθοδος για την συμπλήρωση των βαθμολογιών που λείπουν. Για κάθε χρήστη, πάνω στις ταινίες για τις οποίες υπάρχουν δεδομένα, γίνεται εκπαίδευση ενός νευρωνικού δικτύου το οποίο χρησιμοποιείται για να προβλεφθεί πώς ο συγκεκριμένος χρήστης θα βαθμολογούσε τις υπόλοιπες ταινίες.

Αρχικά, κατασκευάστηκαν τα embeddings για τους τίτλους των ταινιών που διαθέτουμε, όπως φαίνεται στην παρακάτω εικόνα. Εκτιμήθηκε ότι το πλήθος των διακριτών λέξεων από όλους τους τίτλους των ταινιών είναι περίπου 9000 λέξεις, ενώ ο μεγαλύτερος τίτλος έχει μέγεθος περίπου 30 λέξεις. Έτσι, αρχικά κωδικοποιούμε κάθε λέξη κάθε τίτλου σε ακεραίους στο διάστημα (0:9000), και μετά προσθέτουμε μηδενικά στην αρχή κάθε τίτλου (padding) ώστε όλοι οι τίτλοι να έχουν το ίδιο μήκος (30). Ορίζουμε επίσης την διαστασιμότητα των embeddings σε dim = 100. Κατασκευάζουμε έτσι ένα Embedding() Layer με τη χρήση της βιβλιοθήκης Keras, του οποίου η έξοδος μετά την εκπαίδευση μας δίνει τα επιθυμητά embeddings. Επίσης, με το layer Flatten() ενώνουμε τα 100 embeddings κάθε λέξης σε ένα διάνυσμα για κάθε πρόταση, επομένως όλοι οι τίτλοι μας έχουν πλέον μήκος 100 * 30 = 3000. Όλες τις προτάσεις στο τέλος τις μετασχηματίζουμε σε ένα τελικό Dataframe.

Create the Word Embeddings

[13]

```
titles = movies["title"]
titles_list = [title for title in titles]

# vocabulary size was estimated around 9000 distinct words
vocab_size = 9000
titles_onehot = [one_hot(d, vocab_size) for d in titles_list]

# We set the maximum title length to 30 words and the embeddings dimensions to 100
sentence_length = 30
dim = 100
embedded_titles = pad_sequences(titles_onehot, padding='pre', maxlen = sentence_length)

#create an embedded layer with keras to find the word embeddings
model=Sequential()
model.add(Embedding(vocab_size, dim, input_length=sentence_length))
model.add(Flatten())
model.compile('adam', 'mse')
word_embeddings = model.predict(embedded_titles)

word_embeddings=pd.DataFrame(word_embeddings)
word_embeddings=pd.concat([movies[["movieId"]], word_embeddings], axis=1)
```

Έπειτα, χρησιμοποιούμε το one hot encoding για να κωδικοποιήσουμε τα είδη των ταινιών, και στη συνέχεια τα προσθέσαμε και αυτά στο τελικό Dataframe, όπως φαίνεται στην παρακάτω εικόνα.

Use one-hot encoding for the movie genres and merge them with the embeddings

[14]

```
one_hot = movies['genres'].str.get_dummies()
movies_onehot= movies.merge(one_hot, left_index=True, right_index=True)
movies_onehot.drop(['title','genres'],axis=1,inplace=True)

final_df = pd.merge(word_embeddings, movies_onehot, left_on="movieId", right_on="movieId")
```

Επίσης, για να εκπαιδεύσουμε το τελικό νευρωνικό δίκτυο για κάθε χρήστη, χρησιμοποιήσαμε, όπως φαίνεται στην παρακάτω εικόνα, το one hot encoding και για τα labels του δικτύου, δηλαδή για τις 10 βαθμολογίες από 0 – 5. Για όσες ταινίες ανήκουν σε περισσότερα του ενός είδη (π.χ Action|Drama), τοποθετήσαμε το 1 για όλα τα είδη της ταινίας (Action = 1, Drama = 1), και 0 για τα υπόλοιπα είδη.

Use one-hot encoding for the ratings (10 labels, from 0 to 5.0)

[15]

```
ratings_one_hot=pd.get_dummies(ratings.loc[:, "rating"])
ratings_one_hot=pd.concat([ratings, ratings_one_hot], axis=1)
```

Τέλος, κατασκευάσαμε την συνάρτηση που εκπαιδεύει ένα νευρωνικό δίκτυο για κάθε χρήστη, όπως φαίνεται παρακάτω. Η συνάρτηση δέχεται ως όρισμα τον αριθμό του χρήστη και απομονώνει από τον πίνακα των συνολικών embeddings μόνο τα embeddings που αφορούν τις ταινίες που έχει βαθμολογήσει, αλλά και τα κωδικοποιημένα labels τους. Έπειτα, χωρίζουμε τους τίτλους του χρήστη σε 90% training set και 10% test set, και

εκπαιδεύουμε το δίκτυο με 3 επίπεδα (178 νευρώνες στο πρώτο, 16 στο δεύτερο και 10 στο τρίτο για την κατηγοριοποίηση στις 10 βαθμολογίες).

Create a function to train a neural network for a given user

[38]

```
def trainNeuralNet(ratings_one_hot, user, final_df):

    movies_rated = ratings[ratings["userId"] == user][["movieId", "rating"]]
    df_x = pd.merge(final_df, movies_rated[["movieId"]], left_on="movieId", right_on="movieId").drop(
['movieId'], axis=1).values
    df_y = ratings_one_hot[(ratings_one_hot["userId"]==user)].iloc[:, 4:]

    X_train,X_test,y_train,y_test = train_test_split(df_x,df_y,test_size=0.1, random_state=2)

    model = Sequential()
    model.add(Dense(178, input_dim = df_x.shape[1], activation = 'relu'))
    model.add(Dense(16, activation = 'relu'))
    model.add(Dense(10, activation = 'softmax')) # Softmax for multi-class classification

    model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
    model.fit(X_train, y_train, epochs=25, verbose=0)

    loss, accuracy = model.evaluate(X_test,y_test)
    print("User % d, Test accuracy  % .5f " %(user, accuracy))
    return model, movies_rated
```

Το πρόγραμμα που εκτελεί την συνάρτηση φαίνεται παρακάτω. Τα αποτελέσματα όπως φαίνεται είχαν μεγάλες διακυμάνσεις στο accuracy για κάθε χρήστη και δεν ήταν αξιόπιστα, ανεξαρτήτως ρυθμίσεων των παραμέτρων του νευρωνικού δικτύου, εφόσον κάθε χρήστης έχει βαθμολογήσει ένα πολύ μικρό σύνολο ταινιών και άρα η είσοδος κάθε νευρωνικού δικτύου είναι πολύ μικρή. Για αυτό, εκπαιδεύτηκαν ενδεικτικά 10 νευρωνικά δίκτυα για 10 τυχαίους χρήστες, και τα αποτελέσματα δεν ενσωματώθηκαν στο τελική μετρική, καθώς θεωρήσαμε ότι η μετρική του ερωτήματος 3 είναι πολύ πιο αξιόπιστη.

Main program

[39]

```
from random import randrange

all_unique_userIds = ( ratings["userId"].unique() )

#because of peformance time, test the program for 10 random users
for i in range(10):
    user = randrange( len(all_unique_userIds) )
    model, movies_rated = trainNeuralNet(ratings_one_hot,user,final_df)
```

```
15/15 [==============================] - 1s 47ms/step
User  647, Test accuracy  0.40000
44/44 [==============================] - 0s 6ms/step
User  605, Test accuracy  0.43182
2/2 [==============================] - 0s 123ms/step
User  310, Test accuracy  0.50000
4/4 [==============================] - 0s 62ms/step
User  223, Test accuracy  0.25000
5/5 [==============================] - 0s 51ms/step
User  66, Test accuracy  0.80000
3/3 [==============================] - 0s 83ms/step
User  256, Test accuracy  0.33333
3/3 [==============================] - 0s 81ms/step
User  495, Test accuracy  0.33333
20/20 [==============================] - 1s 32ms/step
User  599, Test accuracy  0.40000
11/11 [==============================] - 0s 22ms/step
User  67, Test accuracy  0.72727
21/21 [==============================] - 0s 11ms/step
User  205, Test accuracy  0.33333
```