ΑΝΑΚΤΗΣΗ ΠΛΗΡΟΦΟΡΙΑΣ
ΕΡΓΑΣΤΗΡΙΑΚΗ ΕΡΓΑΣΙΑ
2021

Φουσέκη Αθηνά, 4ο έτος, st1059623@ceid.upatras.gr , 1059623

Ερώτημα 1

Εγκαθιστούμε την Elasticsearch μέσω αρχείου msi και θέτουμε ως HTTP port τη θύρα 9200 (default). Δημιουργούμε ένα index με το όνομα “cinema”. Περνάμε το περιεχόμενο του αρχείου movies.csv σε ένα dataframe df και προσπελαύνοντας το dataframe γραμμή-γραμμή, περνάμε τις εγγραφές στο index που φτιάξαμε, στην Elasticsearch με την ακόλουθη μορφή:

```
for index, row in df.iterrows():
    temp = {"movieid": row['movieId'], "title": row['title'], "genres": row['genres']}
    es.index(index="cinema",  id=index+1, body=temp)
```

Στη συνέχεια, δημιουργώ ένα κενό dictionary, mov, στο οποίο θα κρατάω τις ταινίες που επιστρέφει η Elasticsearch στο query που θα κάνει ο χρήστης και διάφορες πληροφορίες για αυτές. Επίσης, αρχικοποιώ έναν πίνακα PrettyTable (κάνοντας import τη βιβλιοθήκη prettytable), το οποίο θα χρησιμοποιήσω για την εκτύπωση των ταινιών. Θα αποτελείται από δύο στήλες: μία για τον τίτλο κάθε ταινίας και μία για το BM25 score που επιστρέφει η Elasticsearch.

```
final= prettytable.PrettyTable(["Movies To Watch", "BM25 Score"])
```

Τέλος, δίνω τη δυνατότητα στο πρόγραμμα να να δέχεται ως είσοδο ένα αλφαριθμητικό, το οποίο θα χρησιμοποιήσω για το query μου στην Elasticsearch.

Για παράδειγμα, εάν ο χρήστης δώσει ως είσοδο το αλφαριθμητικό “cinderella” το πρόγραμμα θα επιστρέψει:

```
Give a movie title: cinderella
+-----------------------------------------------------------+------------+
|                      Movies To Watch                      | BM25 Score |
+-----------------------------------------------------------+------------+
|                     Cinderella (1950)                     |  9.089312  |
|                     Cinderella (1997)                     |  9.089312  |
|                     Cinderella (2015)                     |  9.089312  |
|                  Cinderella Man (2005)                    |  8.101168  |
|                Cinderella Story, A (2004)                 | 7.3068085  |
|          Ever After: A Cinderella Story (1998)            | 6.1088104  |
| The Slipper and the Rose: The Story of Cinderella (1976)  | 4.6003113  |
+-----------------------------------------------------------+------------+
```

Κώδικας 1ου Ερωτήματος:

```
from elasticsearch import Elasticsearch
import pandas as pd
import prettytable

# Connect to the elastic cluster
es=Elasticsearch([{'host':'localhost','port':9200}], http_auth=('elastic', ''))

# Delete index
es.indices.delete(index="cinema")

#Create the cinema index
es.indices.create(index="cinema")
print(es.indices.exists(index="cinema")) #returns true


df = pd.read_csv('movies.csv')
df['genres']=df['genres'].str.split('\|') #splitting the genres columns

for index, row in df.iterrows():
    temp = {"movieid": row['movieId'], "title": row['title'], "genres": row['genres']}
    es.index(index="cinema",  id=index+1, body=temp)
    print(row['movieId'], row['title'], row['genres'], end='\t')


mov={} #storing the result from elasticSearch
final= prettytable.PrettyTable(["Movies To Watch", "BM25 Score"])


titl=input("Give a movie title: ") #the user gives the title of the movie they want to search for
res= es.search(index="cinema", body={"query": {"match": {'title':titl}}}, size=20)
for hit in res['hits']['hits']: #loop to pass the result from ElasticSearch to the dictionary
    mov[hit['_id']]=[hit['_source']['title'], hit['_score'], hit['_source']['genres']]

for i in mov.items():
    final.add_row([i[1][0], i[1][1]])

print(final)
```

Ερώτημα 2

Σε αυτό το ερώτημα θέλουμε να προσωπικοποιήσουμε κάπως την αναζήτηση ταινίας, συνεπώς χρειαζόμαστε το id του χρήστη που κάνει το ερώτημα στην Elasticsearch. Ζητάμε πριν τον τίτλο της ταινίας ο χρήστης να μας δώσει το id του:

```
user_log=input("Who are you? :") #asking for the user's ID
```

Για το personalization θα χρειαστούμε επίσης το rating που έχει δώσει ο χρήστης σε κάθε ταινία που επιστρέφει η Elasticsearch, αλλά και τον Μ.Ο των ratings της ταινίας. Στην ιδανική περίπτωση, στα δεδομένα που έχουμε υπάρχει και η βαθμολογία του χρήστη για την ταινία που ψάχνει και γενικά η ταινία να έχει βαθμολογίες. Αλλά, δε ζούμε σε έναν ιδανικό κόσμο, οπότε έχουμε αρκετές ελλιπείς τιμές. Η μετρική μας θα είναι διαφορετική, ανάλογα με το ποιες τιμές έχουμε διαθέσιμες.

```
#my metric
if (df_temp.empty == False) and (df_rt.empty == False):
    my_metric = round(0.35*math.exp(hit['_score']) + 0.2*float(df_temp['rating']) + 0.45*avg_rat, 3)
elif df_temp.empty and (df_rt.empty == False):
    my_metric = round(0.35*math.exp(hit['_score']) + 0.65*avg_rat, 3)
else:
    my_metric = round(float(0.35*math.exp(hit['_score']-1))-(math.log(hit['_score'])), 3)
```

Θέλουμε το score που δίνει η Elasticsearch, όσο μεγαλύτερο είναι, τόσο περισσότερο βάρος να έχει στην μετρική μας, καθώς, όσο πιο υψηλή η BM25, τόσο πιο πολύ ταιριάζει με την αναζήτηση του χρήστη μας, ενώ αν είναι χαμηλή, οι ταινίες που μας επιστρέφει η Elasticsearch είναι αρκετά μη σχετικές. Συνεπώς, θα έχουμε έναν όρο, ο οποίος συναρτήσει της BM25 θα αυξάνεται εκθετικά. Από το συνολικό score της μετρικής μας, το 35% θα προέρχεται από την Elasticsearch, το 30% από το rating του χρήστη για την ταινία και το 35% από τον Μ.Ο. των ratings όλων των χρηστών. Στην περίπτωση που ο χρήστης δεν έχει βαθμολογήσει την ταινία, διατηρούμε τον εκθετικό χαρακτήρα του score της Elasticsearch, αλλά αλλάζουν τα ποσοστά: 35% BM25 και 65% από τον Μ.Ο. των ratings. Τέλος, στην περίπτωση που μία ταινία δεν έχει καθόλου ratings, επειδή δεν γνωρίζουμε καθόλου την άποψη του κοινού για αυτή, θέλουμε να κάνουμε πιο μέτριο το score που έχει. Για αυτό, η μετρική θα είναι BM25_score – log(BM25_score).

*Στην περίπτωση που ο χρήστης δεν έχει βαθμολογήσει την ταινία, κρατάμε “none” ως rating και αντίστοιχα, αν δεν έχει κανένα rating, κρατάμε “no ratings”.

Ερώτημα 3

Από ταινίες, αφαιρούμε όσες δεν έχουν κάποια κατηγορία.
Παρατηρούμε ότι στο σύνολο δεδομένων που έχουμε καταλήξει υπάρχουν ελλιπείς τιμές, καθώς πολλοί χρήστες δεν έχουν βαθμολογήσει καμία ταινία από ορισμένα genres. Έχουμε:

```
10.432190760059612% of users have not rated genre Mystery.
5.663189269746647% of users have not rated genre Sci-Fi.
0.29806259314456035% of users have not rated genre Thriller.
0.5961251862891207% of users have not rated genre Action.
2.235469448584203% of users have not rated genre Adventure.
3.8748137108792844% of users have not rated genre Fantasy.
27.57078986587183% of users have not rated genre Documentary.
0.0% of users have not rated genre Comedy.
0.29806259314456035% of users have not rated genre Crime.
21.460506706408346% of users have not rated genre War.
0.0% of users have not rated genre Drama.
3.8748137108792844% of users have not rated genre Horror.
6.259314456035768% of users have not rated genre Children.
16.24441132637854% of users have not rated genre Animation.
0.44709388971684055% of users have not rated genre Romance.
14.307004470938898% of users have not rated genre Musical.
49.03129657228018% of users have not rated genre IMAX.
44.560357675111774% of users have not rated genre Western.
46.19970193740686% of users have not rated genre Film-Noir.
```

Βλέπουμε ότι έχουμε ελλιπείς τιμές σχεδόν σε όλα τα είδη ταινιών, συνεπώς, για τη συμπλήρωση αυτών δεν μπορούμε να εκπαιδεύσουμε κάποιο μοντέλο, καθώς δεν έχουμε καλό training set. Επίσης, δεν μπορούμε να αφαιρέσουμε ούτε στήλες (όπως τις IMAX, Western και Film-Noir που σχεδόν οι μισοί χρήστες δεν έχουν βαθμολογήσει ταινίες αυτού του είδους), καθώς τις χρειαζόμαστε για επόμενα ερωτήματα. Ακόμα, δεν μπορούμε να αφαιρέσουμε ούτε συγκεκριμένες σειρές, καθώς θα χάσουμε μεγάλο πλήθος δεδομένων. Συνεπώς, θα συμπληρώσουμε τις ελλιπείς τιμές για κάθε κατηγορία με τον Μ.Ο. των υπαρχουσών ratings.

Ως πληροφορία χρειαζόμαστε τα ratings του κάθε χρήστης για τις διάφορες ταινίες και τις κατηγορίες στις οποίες ανήκει η κάθε ταινία. Συνεπώς, κάνουμε ένα join στη στήλη genres από το dataframe df που έχουμε αποθηκεύσει τις ταινίες και στο dataframe df_rat. Φτιάχνουμε ένα νέο dataframe, το df_avg2, το οποίο περιλαμβάνει ως στήλες το userId και τα διάφορα genres. Παίρνουμε όλους τους διαφορετικούς χρήστες που έχουν βαθμολογήσει τις ταινίες με την unique() και για κάθε χρήστη, για κάθε genre υπολογίζουμε τον Μ.Ο.. Για κάθε χρήστης, θα φτιάξουμε ένα νέο row στο df_avg2 και αφού προσθέσουμε το id του χρήστη, με μία for loop θα συμπληρώνουμε κάθε φορά τον Μ.Ο. κάθε genre.

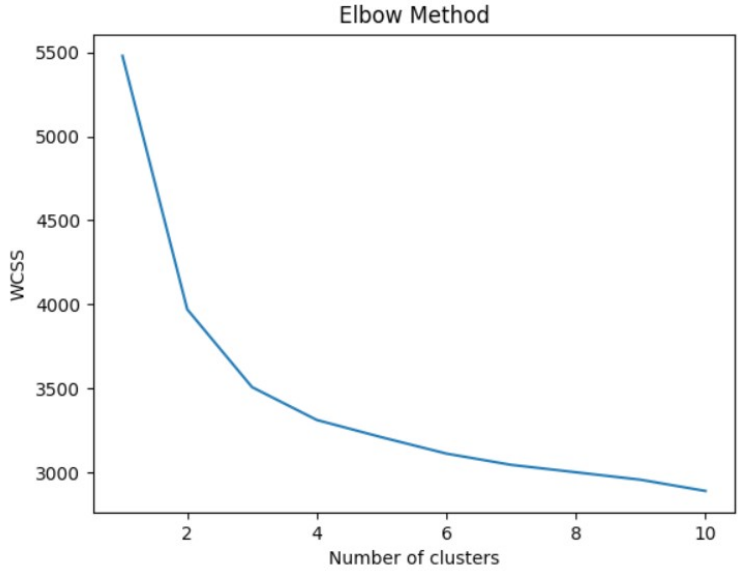
```
i=0
for user in unq_users:

    df_avg2 = df_avg2.append({'userId':str(user)}, ignore_index=True)
    for genre in unq_gens:
        #print("-------------------- GENRE = {} --------------------".format(genre))
        avg= round(df_new[(df_new['genres'] == genre) & (df_new['userId'] == user)]['rating'].mean(axis = 0), 2)
        if math.isnan(avg):
            new_row = {'userId':user, 'genre':genre, 'avg_rating':'no rating'}
            #append row to the dataframe
            df_avg = df_avg.append(new_row, ignore_index=True)
            df_avg2.at[i, genre] = float('nan')
        else:
            new_row = {'userId':user, 'genre':genre, 'avg_rating':avg}
            #append row to the dataframe
            df_avg = df_avg.append(new_row, ignore_index=True)
            df_avg2.at[i, genre] = avg
    i+=1
```

Αν ο Μ.Ο. είναι nan, τότε συμπληρώνουμε με ‘nan’. Για τη συμπλήρωση των nan τιμών χρησιμοποιούμε την fillna() και για τον υπολογισμό του Μ.Ο. την mean().

```
for col in df_avg2.columns:
    print(col)
    if col!= 'userId':
        df_avg2[col].fillna(float(df_avg2[col].mean()), inplace=True)
```

Έχουμε υπολογίσει τον Μ.Ο των βαθμολογιών κάθε κατηγορίας ταινιών για κάθε χρήστη και έχουμε αποθηκεύσει την πληροφορία στο dataframe df_avg2, οπότε μπορούμε πλέον να χρησιμοποιήσουμε την πληροφορία αυτή για το clustering των χρηστών.

Περνάμε στο X την πληροφορία από το df_avg2, πλην της στήλης που περιλαμβάνει το userId, για να εκπαιδεύσουμε το k-Means μοντέλο μας. Για να βρούμε το κατάλληλο k χρησιμοποιούμε το elbow method:



Βλέπουμε ότι το κατάλληλο k είναι το 6.
Δημιουργούμε ένα μοντέλο kmeans, με 6 clusters, αρχικοποίηση kmeans++ για να αποφύγουμε την τυχαία αρχικοποίηση των centroids, μέγιστο αριθμό iterations 300 και n_init=10.

Αφού πάρουμε το clustering, θα ενσωματώσουμε την πληροφορία για κάθε χρήστη, κάνοντας ένα join on index, ανάμεσα στο df_avg2 και το df_pred και περνάμε το αποτέλεσμα στο df_pred_avg.

Στη συνέχεια, θέλουμε να συμπληρώσουμε για κάθε χρήστη τις βαθμολογίες ταινιών που του λείπουν, χρησιμοποιώντας τον μέσο όρο της ταινίας στην συστάδα που ανήκει. Επειδή κάτι τέτοιο θα ήταν αρκετά κοστοβόρο και θα μεγάλωνε σημαντικά το dataset μας, μπορούμε, αντί να έχουμε για κάθε χρήστη βαθμολογία για κάθε ταινία, να έχουμε για κάθε cluster για κάθε ταινία. Όταν ένας χρήστης αναζητήσει μια ταινία την οποία δεν έχει βαθμολογήσει, θα ελέγχουμε σε ποια συστάδα ανήκει ο χρήστης και θα ψάχνουμε σε αυτό το νέο σετ δεδομένων για τον Μ.Ο. της ταινίας στο συγκεκριμένο cluster.

*Στην περίπτωση που η ταινία δεν έχει κανένα rating στο cluster του χρήστη δεν θα υπάρχει ο “M.O. rating συστάδας” ως πληροφορία, συνεπώς,την βαθμολογούμε με τον M.O των ratings των genres (στα οποία ανήκει) στην συστάδα. Ακολουθούμε την ίδια τακτική και στην περίπτωση που η ταινία δεν έχει κανένα rating γενικά.
**Στην περίπτωση που η ταινία δεν έχει κανένα rating και δεν ανήκει σε κάποιο genre (genre = no genres listed), την βαθμολογούμε με 2.5.

```
for movie in df['movieId']:
    for cluster in range(0,clstrs):
        ratings_of_wanted_genres = []
        temp_rt = df_rat[df_rat['movieId'] == movie]

        if temp_rt.empty:
            gens = df.iloc[df[df['movieId']==movie].index.item()]['genres'] #keeping the genres that the movie belongs to in a list -extra to_list probs not need
            temp_avg = df_avg_pred[df_avg_pred['Cluster'] == cluster] #temp dataframe that contains the data for the AVG rating of the movie
            try:
                temp_avg = temp_avg[gens]

                for colu in gens:
                    ratings_of_wanted_genres.append(float(temp_avg[colu].mean())) #list of the average ratings of the wanted genres in a particular cluster
                average_rat = mean(ratings_of_wanted_genres) #calculating the mean of all the ratings the list contains
                movies_no_rat = movies_no_rat.append({'movieId': str(int(movie)),'rating': average_rat, 'cluster': str(int(cluster))},ignore_index=True ) #adding
            except:
                movies_no_rat = movies_no_rat.append({'movieId': str(int(movie)),'rating': 2.5, 'cluster': str(int(cluster))}, ignore_index=True) #adding the dat
        else:

            m = df_rat[(df_rat['movieId'] == movie) & (df_rat['userId'].isin(user_per_cluster_list[cluster]))]['rating'].mean()
            print("m: {}".format(m))
            if math.isnan(m):
                gens = df.iloc[df[df['movieId']==movie].index.item()]['genres'] #keeping the genres that the movie belongs to in a list -extra to_list probs not
                temp_avg = df_avg_pred[df_avg_pred['Cluster'] == cluster] #temp dataframe that contains the data for the AVG rating of the movie

                try:
                    temp_avg = temp_avg[gens]
                    for colu in gens:
                        ratings_of_wanted_genres.append(float(temp_avg[colu].mean())) #list of the average ratings of the wanted genres in a particular cluster

                    average_rat = mean(ratings_of_wanted_genres) #calculating the mean of all the ratings the list contains
                    movies_with_rat = movies_with_rat.append({'movieId': str(int(movie)),'rating': average_rat, 'cluster': str(int(cluster))},ignore_index=True )
                except:
                    movies_with_rat = movies_with_rat.append({'movieId': str(int(movie)),'rating': 2.5, 'cluster': str(int(cluster))}, ignore_index=True) #adding
            else:
                movies_with_rat = movies_with_rat.append({'movieId': str(int(movie)),'rating': m, 'cluster': str(int(cluster))}, ignore_index=True)
```

Επειδή αυτή η διαδικασία παίρνει πολλή ώρα, θα αποθηκεύσουμε τη ζητούμενη πληροφορία σε κάποιο αρχείο csv, ώστε την όταν χρειαστούμε, να την περάσουμε σε ένα dataframe και να είναι εύκολα προσβάσιμη.

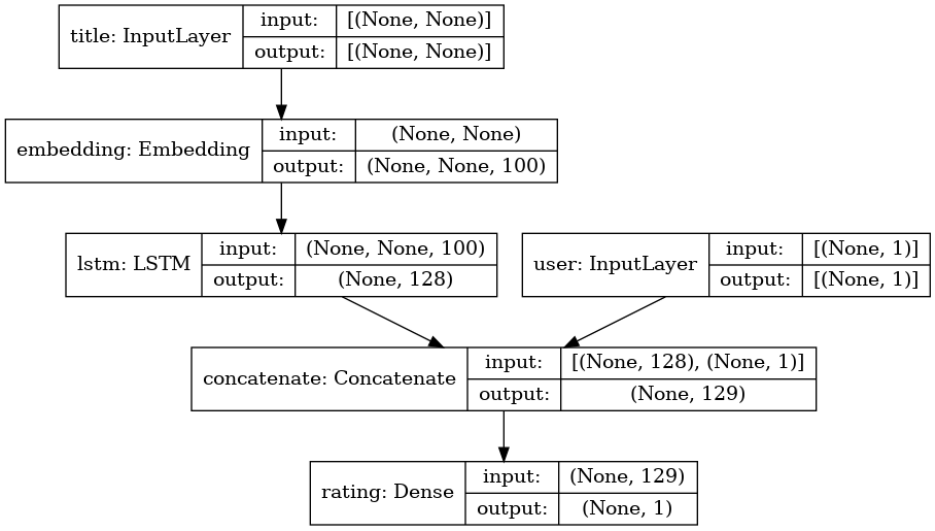
Η personalization μετρική μας θα είναι λίγο διαφορετική πλέον, καθώς έχουμε βαθμολογία της κάθε ταινίας για κάθε χρήστη. Στην περίπτωση λοιπόν που στο dataframe με τα ratings ο χρήστης δεν έχει βαθμολογήσει μια συγκεκριμένη ταινία, θα ψάξουμε στο dataframe movies_with_rat, για να βρούμε τον M.O. της ταινίας στο cluster που ανήκει ο χρήστης.

```
#my metric
if (df_temp.empty == False) and (df_rt.empty == False):
    my_metric2 = round(0.35*math.exp(hit['_score']) + 0.3*float(df_temp['rating']) + 0.35*avg_rat, 3)
elif df_temp.empty and (df_rt.empty == False): #user hasn't given a rating to the movie but the movie has ratings
    users_cluster = df_avg_pred[df_avg_pred['userId'] == user_log ]['Cluster'].values
    print(users_cluster)
    user_rat = movies_with_rat[(movies_with_rat['movieId'] == int(hit['_id'])) & (movies_with_rat['cluster'] == users_cluster[0])]['rating'].values
    user_rat = user_rat[0]
    print('user rating: {}'.format(user_rat))
    my_metric2 = round(0.35*math.exp(hit['_score']) + 0.3*float(user_rat) + 0.35*avg_rat, 3)
else: #the movie has no ratings
    my_metric2 = round(float(0.35*math.exp(hit['_score']-1))-(math.log(hit['_score'])), 3)
```

Ερώτημα 4

Στο ερώτημα αυτό μας ζητείται να βελτιώσουμε την ταξινόμηση των αποτελεσμάτων εκπαιδεύοντας ένα νευρωνικό δίκτυο. Για να το εκπαιδεύσουμε θα χρησιμοποιήσουμε ως πληροφορία τους τίτλους των ταινιών, τα id των χρηστών, και θέλουμε να προβλέψουμε ένα πιθανό rating για την κάθε ταινία. Για να χρησιμοποιήσουμε τους τίτλους των ταινιών θα τους μετατρέψουμε σε word embeddings. Θα φτιάξουμε τα word embeddings χρησιμοποιώντας το GloVe και συγκεκριμένα το αρχείο glove.6B.100d, με τα διανύσματα 100 διαστάσεων.

Φτιάξαμε το ακόλουθο νευρωνικό δίκτυο:



Εφόσον έχουμε να λύσουμε ένα regression problem χρησιμοποιήσαμε ως loss function το mean squared error και ως συνάρτηση ενεργοποίησης του output layer την linear.

```
num_users = 1 # single user every time
title_input = keras.Input(shape=(None,), name="title")  # Variable-length sequence of ints
user_input = keras.Input(shape=(num_users, ), name="user")

title_features = layers.Embedding(num_words, 100, embeddings_initializer=Constant(embedding_matrix),
                                  input_length=max_len, trainable=False )(title_input) #EMBEDDING

# Reduce sequence of embedded words in the title into a single 128-dimensional vector
title_features = layers.LSTM(128, dropout=0.2, activation='sigmoid')(title_features)

# Merge all available features into a single large vector via concatenation
x = layers.concatenate([title_features, user_input])

# Stick a logistic regression for priority prediction on top of the features
rating_pred = layers.Dense(1, activation='linear', name="rating")(x)

# Instantiate an end-to-end model predicting both priority and department
model = keras.Model(
    inputs=[title_input, user_input],
    outputs=[rating_pred]
)


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

Εκπαιδεύσαμε το δίκτυο για 30 epochs, και batch size ίσο με 20 (λόγω περιορισμών στη μνήμη του υπολογιστή). Τέλος ως optimizer χρησιμοποιήσαμε των adam, καθώς είναι καλύτερος από τους Adadelta και RMSprop.

```
model.fit(
    {"title": title_data, "user": user_data},
    {"rating": rating_targets},
    epochs=1,
    batch_size=20,
    verbose = 1
)
```

Τελική μορφή προγράμματος.

Στο πρόγραμμά μας, προηγούνται τα μέρη του κώδικα που αφορούν τη μηχανική μάθηση, ώστε να είναι έτοιμα για τη στιγμή που ο χρήστης θα θελήσει να αναζητήσει κάποια ταινία. Επίσης, του δίνουμε τη δυνατότητα να αναζητήσει πολλές ταινίες, μέχρι να θελήσει να σταματήσει.