Ομάδα

Δημήτριος Κυριακίδης el17077

Χαράλαμπος Δάλπης el17067

Τεχνητή Νοημοσύνη: Εργαστηριακή Άσκηση 2

Ο στόχος της εργασίας είναι η κατασκευή ενός συστήματος προτάσεων (Recommendation System) για ταινίες. Οι προτάσεις αυτές θα πηγάζουν τόσο από τα χαρακτηριστικά της ταινίας όσο και από ορισμένες αξιολογήσεις του κάθε χρήστη.

Στα δεδομένα της άσκησης περιλαμβάνονται ένα αρχείο με το όνομα movies_metadata.csv το όποιο περιέχει τα χαρακτηριστικά κάθε ταινίας όπως θέμα, σκηνοθέτης ηθοποιοί, λέξεις κλειδιά κ.α. από το imdb καθώς και τα αρχεία ratings.csv τα όποια περιέχουν πραγματικές αξιολογήσεις χρηστών, χωρισμένες σε train και σε test.

Εκφώνηση

Στην παρούσα εργασία σας ζητείται να μελετήσετε και να υλοποιήσετε τα παρακάτω:

Μέρος 1

Στο μέρος 1 και 2 θα εργαστείτε μόνο με το αρχείο movies_metadata.csv, ενώ στο μέρος τρία θα δουλέψετε και με τα αρχεία των αξιολογήσεων.

Ερώτημα 1α

Αρχικά θα πρέπει, αφού μελετήσετε τη δομή και τα χαρακτηριστικά του movies_metadata.csv, να κατασκευάσετε μια βάση γνώσης για την Prolog η όποια ουσιαστικά θα αποτελεί τον κόσμο με τον όποιο θα εργαστείτε στην συνέχεια. Τα κατηγορήματα που θα δημιουργηθούν θα σας βοηθήσουν και στην κατασκευή του recommender και θα είναι της μορφής:

```
director(Movie, Director).
genre(Movie, Genre).
```

Ερώτημα 1β

Αφού δημιουργήσετε τον κόσμο του προβλήματος, στη συνέχεια καλείστε να δημιουργήσετε, σε Prolog, απλούς κανόνες οι οποίοι θα βρίσκουν όλες τις ταινίες με:

1. Κοινό θέμα (κάποιες λέξεις σχετικά με το genre κοινές)
2. Αρκετά κοινό θέμα (κάποιες λιγότερες λέξεις σχετικά με το genre κοινές π.χ. 3)
3. Σχετικά κοινό θέμα (λίγες λέξεις σχετικές με το genre κοινές π.χ. 1)
4. Κοινός σκηνοθέτης
5. Ακριβώς ίδια πλοκή (κάποιες λέξεις κλειδιά της πλοκής κοινές)
6. Σχετικά ίδια πλοκή (κάποιες λιγότερες λέξεις κλειδιά κοινές)
7. Ίδιους τους βασικούς ηθοποιούς (και τους 3)
8. Αρκετά ίδιους βασικούς ηθοποιούς (ορισμένους βασικούς ηθοποιούς κοινούς π.χ. 2)
9. Σχετικά ίδιους ηθοποιούς (π.χ. 1 από τους 3)
10. Ίδια γλώσσα
11. Είναι έγχρωμες ή ασπρόμαυρες
12. Κοινό studio παραγωγής
13. Κοινή χώρα παραγωγής
14. Ίδια δεκαετία

Αξίζει να σημειωθεί ότι στα παραπάνω μπορείτε να προσθέσετε περισσότερα ερωτήματα ή να αλλάξετε την κλιμακωσιμότητα των queries (πέρα από το ίδιο, αρκετά ίδιο, σχετικά ίδιο) που θα κατασκευάσετε(π.χ. μια κλίμακα από 1 στα 5 όπου αυτό είναι δυνατόν), μιας και αυτά στην συνέχεια θα χρησιμοποιηθούν για την λειτουργία του recommender. Έτσι μπορείτε να προσθέσετε queries με τα όποια μπορεί να παράγονται καλύτερες συστάσεις (το αρχείο movie_metadata.csv περιέχει πολλές πληροφορίες ακόμα για κάθε ταινία όπως έτος κυκλοφορίας, βαθμολογία στο imdb, facebook_likes κ.α.). Το συγκεκριμένο μέρος εργασίας είναι προπαρασκευαστικό οπότε όσο καλύτερα και πλουσιότερα τα ερωτήματα που θα φτιάξετε σε αυτό το μέρος τόσο καλύτερη θα είναι η απόδοσή των συστημάτων συστάσεων των επόμενων ερωτημάτων.

Περισσότερες πληροφορίες για το dataset μπορείτε να διαβάσετε σε αυτό το Link (https://www.kaggle.com/georgefila/movies-metadata):

Μέρος 2: Recommendation System

Στο σημείο αυτό καλείστε με βάση αυτά που κάνατε στο μέρος 1 να κατασκευάσετε queries τα όποια θα σας επιστρέφουν παρόμοιες (σε χαρακτηριστικά) ταινίες. Τα ερωτήματα αυτά θα είναι κλιμακούμενα, δηλαδή θα υπάρχουν ερωτήματα που επιστρέφουν αρκετά κοινές ταινίες αλλά και που επιστρέφουν λιγότερο και λιγότερο κοινές (σε μια κλίμακα π.χ. από 1 σε 5). Για παράδειγμα:

```
find_simmilar_movies_5("Pirates Of The Caribbean", M).
M = "Pirates Of The Caribbean: On Stranger Tides"
M = "The Chronicles Of Narnia"
M = "Prince Of Persia: The Sands Of Time"
...
```

Για παράδειγμα, το παραπάνω ερώτημα θα επιστρέφει αρκετά κοινές σε περιεχόμενο ταινίες με την ταινια "Pirates Of The Caribbean". Θα υπάρχουν και αντίστοιχα ερωτήματα που θα βρίσκουν λιγότερο όμοιες ταινίες. Ο δείκτης ομοιότητας των ταινιών είναι αυθαίρετος και μπορείτε να τον ορίσετε εσείς όπως θέλετε, αρκεί να υπάρχει κάποια λογική σύνδεση με τα δεδομένα που περιέχονται στο αρχείο movies_metadata.csv.

Συνεπώς η συνάρτηση που θα κάνει τις προτάσεις (recommendation) με είσοδο μια ταινία πρέπει να επιστρέφει-εκτυπώνει μια λίστα με τις προτεινόμενες ταινίες κατά φθίνουσα σειρά ομοιότητας.

3ο Μέρος: Recommendation System Με βάση τις προτιμήσεις - Αξιολογήσεις του χρήστη

Σε αυτό το σημείο θα εργαστείτε με τα αρχεία ratings τα όποια περιέχουν αξιολογήσεις (από 1 μέχρι 5) για τις παραπάνω ταινίες. Το προηγούμενο σύστημα συστάσεων προτείνει στον χρήστη ταινίες αποκλειστικά με βάση την ομοιότητά τους. Σε αυτό το σημείο θα γίνει μια αναβάθμιση του συστήματος έτσι ώστε να παράγονται καλύτερες συστάσεις οι όποιες θα λαμβάνουν υπόψιν και τις προτιμήσεις του χρήστη, οι όποιες θα εξάγονται από τις αξιολογήσεις που έχει κάνει μέχρι στιγμής.

Η εκπαίδευση του recommender θα γίνεται ως εξής:

Για κάθε ταινία θα υπάρχει ένα score το όποιο αρχικά θε είναι ίσο με 0 και θα διαμορφώνεται από τις αξιολογήσεις κάθε user. Έτσι για έναν χρήστη με βάση τις αξιολογήσεις που υπάρχουν στο αρχείο train_ratings θα πρέπει:

1. Για κάθε ταινία που έχει βαθμολογήσει να βρίσκονται οι κοινές ταινίες ανά κλίμακα και στο μέχρι τώρα σκορ κάθε παρόμοιας ταινίας θα προστίθεται ένα βάρος το όποιο θα μπορούσε να είναι το ποσοστό ομοιότητας της ταινίας (δηλαδή ένα βάρος για κάθε κλίμακα, αν δύο ταινίες μοιάζουν ενισχύουμε το βάρος που προσθέτουμε από το να μοιάζουν λιγότερο) επί τον βαθμό που έχει βάλει ο χρήστης για την αρχική ταινία (διαφορετικό είναι ο χρήστης να έχει βάλει 5/5 ή 1/5 σε μια ταινία από 3/5).
2. Στη συνέχεια, ανάλογα με το σκορ που έχει σχηματιστεί για κάθε ταινία, θα επιλέγεται αν αυτή θα μπορούσε να είναι προτεινόμενη για τον χρήστη ή όχι και θα μετράμε το πόσο καλά τα πήγε το σύστήμά μας με βάση ορισμένες μετρικές.

Η λογική πίσω από την παραπάνω διαδικασία είναι ότι παρόμοιες ταινίες θα έχουν ανάλογο βαθμό. Για παράδειγμα αν ένας χρήστης έχει αξιολογήσει αρκετές ταινίες οι όποιες είναι sci-fiction με 5/5 τότε μια ταίνια sci-fiction την όποια δεν έχει δεί λογικά θα του αρέσει και θα έπρεπε να την προτείνουμε.

Μετά την εκπαίδευση του συστήματος σας καλείστε να δοκιμάσετε τον recommender που κατασκευάσατε στην πράξη. Για τον σκοπό αυτό θα φορτώσετε το αρχείο test_ratings.csv όπου περιέχονται οι αξιολογήσεις του ίδιου χρήστη για άλλες ταινίες. Το σύστημά σας πρέπει να προβλέπει αν μια ταινία θα πρέπει να προταθεί στον χρήστη. Μια ταίνια θα έπρεπε να έχει προταθεί στον χρήστη, αν έχει βαθμό μεγαλύτερο του 3. Συνεπώς για την επίβλεψη του συστήματός σας θα πρέπει για κάθε μια από τις ταινίες του αρχείου test_ratings.csv να επιστρέφεται 1 ή 0 αν η ταινία θα έπαιρνε βαθμό μεγαλύτερο του 3 ή όχι, δηλαδή αν θα έπρεπε να την είχαμε προτείνει ή όχι.

Στην συνέχεια, σε συνδιασμό με τις πραγματικές απαντήσεις του χρήστη θα αξιολογήσετε το σύστημά σας χρησιμοποιόντας τις μετρικές: precision, recall, f1 οι οποίες είναι οι πλέον γνωστές μετρικές και ευρέως χρησιμοποιούμενες τεχνικές για την επίβλεψη-μέτρηση απόδοσης ανάλογων συστημάτων.

1. Precision: Δείχνει πόσο ακριβές είναι το σύστημα. Υπολογίζει πόσα από τα στιγμιότυπα τα όποια προβλέψαμε ότι ανήκουν σε μια κλάση όντως ανήκουν σε αυτή. Η μετρική αυτή μας δίνει μια εικόνα σχετικά με τον αριθμό των ταινίων που προβλέψαμε ως προτεινόμενες ενώ δεν θα έπρεπε.
2. Recall: Υπολογίζει πόσα από τα στιγμιότυπα που ανήκουν σε μια κλάση (π.χ. προτεινόμενες ταινίες) προβλέφθηκαν σωστά.
3. F1: Είναι ένας μέσος μεταξύ των παραπάνω δυο μετρικών, έτσι ώστε να διατηρείται μια ισορροπία μεταξύ τους. Υπολογίζεται από την παρακάτω σχέση:

$$F_1 = 2\frac{Precision \times Recall}{Precision + Recall}$$

Οι παραπάνω συναρτήσεις παρέχονται από την βιβλιοθήκη scikit-learn.

Τέλος για την καλύτερη επίβλεψη του συστήματος σας μπορείτε να εκπαιδεύσετε τον recommender σας με ένα υποσύνολο ταινίων από ελάχιστες, λίγες μέχρι και πολλές (π.χ. 3, 5, 10, 50, ...) για να μελετήσετε κατά πόσο σας βοηθούν οι επιπλέον αξιολογήσεις κάθε φορά (δηλαδή κατά πόσο βελτιώνονται οι παραπάνω μετρικές στο test set).

Έτσι π.χ. μπορείτε να εντοπίσετε περιπτώσεις όπως για παράδειγμα ότι με έναν recommender μπορεί να μην επιτυγχάνετε πολύ υψηλό σκορ όσο με άλλους, αλλά το βέλτιστο σκόρ σας επιτυγχάνεται πολύ γρηγόρα π.χ. με μόνο 10 ταινίες αντί 100. Έτσι για παράδειγμα αν ο αλγόριθμος τα πηγαίνει πολύ καλά για τρεις ταινίες και στη συνέχεια το σκορ βελτιώνεται ελάχιστα τότε αυτός ενδείκνυται για ένα σύστημα συστάσεων για νέους χρήστες όπου δεδομένου λίγων ταινιών ο αλγόριθμος είναι σε θέση να το να προτείνει καλές συστάσεις. Ενώ αν η καλύτερη απόδοση του αλγορίθμου σας είναι βέλτιστη με περισσότερες ταινίες π.χ. 50 τότε αυτός ο αλγόριθμος συστάσεων ενδείκνυται για παλιούς χρήστες με πολλές αξιολόγησεις.

Κατασκευή Περιβάλλοντος Εργασίας

Διάβασμα Αρχείων στο Colab (Μόνο για το Colab)

Αν η υλοποίηση γίνει στο google colab τότε μπορεί να χρησιμοποιηθεί το google drive ως file system. Για να γίνει Mount το google drive τρέχουμε τον παρακάτω κώδικα και κλικάρουμε στο link που θα μας εμφανιστεί.

```
from google.colab import drive
drive.mount('/content/drive')
import os
os.listdir('/content/drive/My Drive')
```

Έπειτα στην σελίδα που άνοιξε επιλέγουμε το mail μας και στο επόμενο παράθυρο που θα μας ανοίξει πατάμε Να επιτρέπεται. Στην συνέχεια αντιγράφουμε τον κωδικό που θα μας βγάλει και τον κάνουμε paste στο Input που έχει ανοίξει στο colab. Έτσι πλέον αν έχουμε ανεβάσει ένα αρχείο στο google drive μπορούμε να το βρούμε στην θέση:

```
movies_filename = '/content/drive/My Drive/' + movies_metadata.csv
```

Μπορούμε πλέον κανονικά να δουλέψουμε φτιάχνοντας φακέλους ή αρχεία και γενικότερα κάνοντας οτιδήποτε θα κάναμε αν ήμασταν τοπικά.

Prolog μέσω Python

Το πακέτο που θα χρησιμοποιηθεί για την επικοινωνία Python και Prolog είναι το pyswip (https://pypi.org/project/pyswip/ (https://pypi.org/project/pyswip/)). Για να δουλέψει το Pyswip χρειάζεται να υπάρχει το Swi-Prolog το όποιο αν δουλεύουμε τοπικά πρέπει να το εγκαταστήσουμε, ακολουθώντας αντίστοιχες οδηγίες στην σελίδα του εργαλείου. Για να γίνει του Swi-Prolog η εγκατάσταση στο Google Colab πρέπει να τρέξουμε τον παρακάτω κώδικα:

```
!sudo apt-get install software-properties-common
!sudo apt-add-repository ppa:swi-prolog/stable
!sudo apt-get update
!sudo apt-get install swi-prolog
```

Σε κάποιο σημείο της εκτέλεσης εμφανίζεται ένα μήνυμα ότι πρέπει να πατήσουμε enter σε ένα input για να συνεχίσει η διαδικασία. Έπειτα από αυτό η εκτέλεση θα συνεχίσει χωρίς κάποιο πρόβλημα.

Τέλος πρέπει να εγκαταστήσουμε το pyswip (**όπου και να δουλεύουμε**) όπως παρακάτω:

```
!pip install pyswip
```

Κώδικας για κατασκευή περιβάλλοντος εργασίας

Μόνο για Google Colab

Κώδικας για να γίνει Mount to Google Drive

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
%%capture
#install swi-prolog
!sudo apt-get install software-properties-common
!sudo apt-add-repository ppa:swi-prolog
!sudo apt-get update
!sudo apt-get install swi-prolog
#install pyswip
!pip install pyswip
```

Μέρος 1: Μελέτη των Metadata, Δημιουργία κόσμου και των βασικών queries.

In [ ]:
```
## Η βιβλιοθήκη pandas είναι χρήσιμη για την εργασία με τέτοια δεδομένα
import pandas as pd
## Διάβασμα του αρχείου 'movie_metadata.csv'
## Τοποθετήθηκε στο ίδιο directory με το παρόν notebook
data = pd.read_csv("./movies_metadata.csv")
## Στο csv υπαρχούν κελία με nan τιμές
## Στις θέσεις αυτές βάζουμε 'UNK' πράγμα που  κάνουμε με την παρακάτω συνάρτηση
data.fillna("UNK", inplace = True)
## Preview the first 5 lines of the loaded data
data.head()
```

Out[ ]:

	Unnamed: 0	budget	genres	homepage	id	plot_keywords	lang
0	0	237000000	Action|Adventure|Fantasy|Science Fiction	http://www.avatarmovie.com/	19995	culture clash|future|space war|space colony|so...	E
1	1	300000000	Adventure|Fantasy|Action	http://disney.go.com/disneypictures/pirates/	285	ocean|drug abuse|exotic island|east india trad...	E
2	2	245000000	Action|Adventure|Crime	http://www.sonypictures.com/movies/spectre/	206647	spy|based on novel|secret agent|sequel|mi6|bri...	Fra
3	3	250000000	Action|Crime|Drama|Thriller	http://www.thedarkknightrises.com/	49026	dc comics|crime fighter|terrorist|secret ident...	E
4	4	260000000	Action|Adventure|Science Fiction	http://movies.disney.com/john-carter	49529	based on novel|mars|medallion|space travel|pri...	E

5 rows × 27 columns

In [ ]:
```
def cleanText(text):
    text = text.replace(u'\xa0', u'')
    text = text.replace(u"'", u'')
    text = text.lstrip('\n')
    text = text.lstrip('"')
    text = text.rstrip('\n')
    text = text.rstrip(';')
    text = text.rstrip('"')
    text = text.replace('""', '"')
    text = text.replace(';"', '')
    text = text.replace('";', '')
    text = text.replace('&amp"', '&')
    text = text.replace('"', '')
    return text
```

In [ ]:
```
from pyswip import Prolog

## create World
## Ορίζουμε τον κόσμο μας
prolog = Prolog()
```

In [ ]:
```
## Για κάθε row του πίνακα φτιάχνουμε τα κατηγορήματα που θέλουμε να αποθηκέυσουμε
## αρχικά σε μια λίστα με το όνομα literals
literals = []
for row in data.itertuples(index = True, name = 'Pandas'):
    movieTitle = cleanText(getattr(row, 'movie_title'))

    ## print(list(eval(getattr(row, 'production_companies'))))

    for genre in getattr(row, 'genres').split("|"):
        literals.append(f"genre('{movieTitle}','{genre}')")

    for plotKeyword in getattr(row, 'plot_keywords').split("|"):
        plotKeyword = plotKeyword.replace("'", '')
        literals.append(f"plot_keyword('{movieTitle}','{plotKeyword}')")

    literals.append(f"language('{movieTitle}','{getattr(row, 'language')}')")

    for productionCompany in list(eval(getattr(row, 'production_companies'))):
        pcname = productionCompany["name"].replace("'", "")
        literals.append(f"production_company('{movieTitle}','{pcname}')")

    ## some names include apostrophes, others include nicknames in double quotes
    name = cleanText(getattr(row, 'director_name'))
    literals.append(f"director('{movieTitle}','{name}')")

    name = cleanText(getattr(row, 'actor_1_name'))
    literals.append(f"actor('{movieTitle}','{name}')")

    name = cleanText(getattr(row, 'actor_2_name'))
    literals.append(f"actor('{movieTitle}','{name}')")

    name = cleanText(getattr(row, 'actor_3_name'))
    literals.append(f"actor('{movieTitle}','{name}')")

    if(getattr(row, 'title_year') != "UNK"):
        year = int(getattr(row, 'title_year'))
        decade = str(year - year % 10)
        literals.append(f"decade('{movieTitle}','{decade}')")

## Η Prolog θέλει τα κατηγορήματά της με την σειρά
literals.sort()
with open("literals.txt", 'w', encoding = "utf-8") as f:
    for l in literals: f.write(l + '\n')

for literal in literals:
    prolog.assertz(literal)
```

Η παραπάνω μέθοδος διαβάζει τα αρχικά δεδομένα, και κατασκευάζει και αποθηκεύει όλα τα κατηγορήματα της μορφής director(movieName, directorName) κτλ στο αρχείο literals.txt, ούτως ώστε σε επόμενες εκτελέσεις του notebook να διαβάζουμε τα χρήσιμα δεδομένα απ' ευθείας από εκεί.

In [ ]:
```
## FOR SUBSEQUENT EXECUTIONS
with open("/content/drive/My Drive/Colab Notebooks/AI/literals.txt", 'r', encoding = "utf-8") as f:
    for line in f: prolog.assertz(line)
```

In [ ]:
```
## Κάνουμε consult ένα έτοιμο αρχείο στον κόσμο που περιέχει τα ζητούμενα κατηγορήματα
prolog.consult("/content/drive/My Drive/Colab Notebooks/AI/predicates.pl")
```

In [ ]:
```
q = prolog.query("sameDirector('Iron Man', M)")
for sol in q: print(sol)
q.close()
```

```
{'M': 'Elf'}
{'M': 'Iron Man 2'}
{'M': 'Made'}
{'M': 'The Jungle Book'}
{'M': 'Zathura: A Space Adventure'}
```

In [ ]:
```
q = prolog.query("atLeastOneCommonActor('Iron Man 2', M)")
for sol in q: print(sol)
q.close()
```

```
{'M': 'Brooklyns Finest'}
{'M': 'Crash'}
{'M': 'Flight'}
{'M': 'Hotel for Dogs'}
{'M': 'Iron Man 3'}
{'M': 'Mission to Mars'}
{'M': 'Out of Sight'}
{'M': 'Reign Over Me'}
{'M': 'Swordfish'}
{'M': 'The Family Man'}
{'M': 'The Guard'}
{'M': 'Volcano'}
{'M': 'Contagion'}
{'M': 'Glee: The Concert Movie'}
{'M': 'Infamous'}
{'M': 'Iron Man 3'}
{'M': 'Jefferson in Paris'}
{'M': 'Mortdecai'}
{'M': 'Se7en'}
{'M': 'Shakespeare in Love'}
{'M': 'Sky Captain and the World of Tomorrow'}
{'M': 'The Good Night'}
{'M': 'The Pallbearer'}
{'M': 'The Royal Tenenbaums'}
{'M': 'The Talented Mr. Ripley'}
{'M': 'Two Lovers'}
{'M': 'Captain America: Civil War'}
{'M': 'Captain America: The Winter Soldier'}
{'M': 'Don Jon'}
{'M': 'Eight Legged Freaks'}
{'M': 'Ghost World'}
{'M': 'Her'}
{'M': 'In Good Company'}
{'M': 'Lost in Translation'}
{'M': 'Match Point'}
{'M': 'The Black Dahlia'}
{'M': 'The Horse Whisperer'}
{'M': 'The Island'}
{'M': 'The Other Boleyn Girl'}
{'M': 'The Prestige'}
{'M': 'The Spirit'}
{'M': 'We Bought a Zoo'}
```

In [ ]:
```
q = prolog.query("atLeastThreeCommonGenres('Iron Man', M)")
s = set()
for sol in q: s.add(sol['M'])
q.close()
print(s)
```

```
{'Star Trek Beyond', 'Beastmaster 2: Through the Portal of Time', 'Timeline', 'Anacondas: The Hunt for the Bl
ood Orchid', 'Iron Man 2', 'U.F.O.', 'Captain America: The Winter Soldier', 'Teenage Mutant Ninja Turtles II
I', 'Race to Witch Mountain', 'Starship Troopers', 'The Hunger Games: Catching Fire', 'Thunderbirds', 'Star T
rek: Insurrection', 'Star Wars', 'A Sound of Thunder', 'Ant-Man', 'The Fifth Element', 'TRON: Legacy', 'Mad M
ax: Fury Road', 'Small Soldiers', 'Suicide Squad', 'The Host', 'Jurassic Park III', 'The Avengers', 'Green La
ntern', 'Deep Rising', 'The Day After Tomorrow', 'Blade: Trinity', 'The Empire Strikes Back', 'Battleship',
'Avatar', 'Resident Evil: Afterlife', 'Teenage Mutant Ninja Turtles II: The Secret of the Ooze', 'X-Men', 'Ob
livion', 'The Core', 'I Am Number Four', 'Underworld: Rise of the Lycans', 'Star Trek II: The Wrath of Khan',
'Star Trek III: The Search for Spock', 'Teenage Mutant Ninja Turtles', 'Terminator Genisys', 'Mystery Men',
'Serenity', 'Jurassic World', 'Enders Game', 'RoboCop 3', 'Sheena', 'The Shadow', 'X2', 'Divergent', 'Star Wa
rs: Clone Wars: Volume 1', 'Mad Max 2: The Road Warrior', 'The Saint', 'Transformers', 'AVP: Alien vs. Predat
or', 'Man of Steel', 'Congo', 'Six-String Samurai', 'Dragonball Evolution', 'Superman', 'Planet of the Apes',
'Armageddon', 'Dune', 'Knowing', 'Paycheck', 'Star Trek: Generations', 'Sky Captain and the World of Tomorro
w', 'The Wolverine', 'After Earth', 'The Covenant', 'Megaforce', 'X-Men Origins: Wolverine', 'Street Fighter:
The Legend of Chun-Li', 'Jupiter Ascending', 'The Abyss', 'Jimmy Neutron: Boy Genius', 'Shin Godzilla', 'The
Time Machine', 'Spy Kids 3-D: Game Over', 'Guardians of the Galaxy', 'X-Men: Days of Future Past', 'Spawn',
'Wild Wild West', 'Star Trek: First Contact', 'Superman III', 'Predator', 'Damnation Alley', 'Mad Max', 'Pred
ators', 'The Hunger Games: Mockingjay - Part 2', 'Firefox', 'Captain America: The First Avenger', 'Star Trek
Into Darkness', 'Southland Tales', '2012', 'Captain America: Civil War', 'The Island', 'Avengers: Age of Ultr
on', 'The Black Hole', 'Escape from L.A.', 'Journey to the Center of the Earth', 'Superman Returns', 'Pacific
Rim', 'Superman IV: The Quest for Peace', 'Return of the Jedi', 'Logans Run', 'Final Fantasy: The Spirits Wit
hin', 'Iron Man 3', 'G.I. Joe: The Rise of Cobra', 'X-Men: First Class', 'Star Wars: Episode III - Revenge of
the Sith', 'Independence Day: Resurgence', '9', 'The Matrix Reloaded', 'Steel', 'Journey to Saturn', 'Journey
2: The Mysterious Island', 'Men in Black', 'Star Trek', 'Transformers: Age of Extinction', 'X-Men: The Last S
tand', 'Star Trek V: The Final Frontier', 'Transformers: Dark of the Moon', 'Men in Black II', 'Batman', 'Bab
ylon A.D.', 'Superman II', 'Moonraker', 'The Lost World: Jurassic Park', 'Star Trek: Nemesis', 'Star Trek VI:
The Undiscovered Country', 'Star Wars: Episode I - The Phantom Menace', 'Independence Day', 'Total Recall',
'Fantastic Four', 'Star Wars: Episode II - Attack of the Clones', 'Godzilla 2000', 'The Incredible Hulk', 'Tr
ansformers: Revenge of the Fallen', 'The Matrix Revolutions', 'The Lovers', 'The Blood of Heroes', 'Mad Max B
eyond Thunderdome', 'G.I. Joe: Retaliation', 'John Carter', 'Titan A.E.', 'Inception'}
```

In [ ]:
```
q = prolog.query("sameDirector('Home Alone', M)")
s = set()
for sol in q: s.add(sol['M'])
q.close()
print(s)
```

```
{'Harry Potter and the Chamber of Secrets', 'Home Alone 2: Lost in New York', 'Percy Jackson & the Olympians:
The Lightning Thief', 'Stepmom', 'Bicentennial Man', 'Pixels', 'Mrs. Doubtfire', 'I Love You, Beth Cooper',
'Rent', 'Harry Potter and the Philosophers Stone'}
```

In [ ]:
```
q = prolog.query("atLeastTwoCommonActors('Home Alone', M)")
s = set()
for sol in q: s.add(sol['M'])
q.close()
print(s)
```

```
{'Home Alone 2: Lost in New York'}
```

Στο αρχείο predicates.pl έχουν συμπεριληφθεί τα κατηγορήματα για όλα τα ζητούμενα του πρώτου μέρους (ορισμένα ενδεικτικά παραπάνω), όπως και τα κατηγορήματα find_sim για το σύστημα συστάσεων. Να σημειωθεί ότι, παρ' όλο που έχουμε γράψει 9 κατηγορήματα find_sim, κλιμακούμενης εξειδίκευσης, για την υλοποίηση της μεθόδου simpleRecommender, στην οποία βασίζονται τα μέρη 2 και 3, έχουν χρησιμοποιηθεί μόνο τα 1 έως και 4. Αυτό επειδή η βιβλιοθήκη pyswip δεν διαχειρίζεται ικανοποιητικά μεγάλο όγκο δεδομένων και κατηγορημάτων, με αποτέλεσμα, όταν προσπαθούμε να τρέξουμε όλα τα κατηγορήματα σε μη ευτελές ποσοστό των δεδομένων εκπαίδευσης, η ροή του notebook να διακόπτεται με σφάλμα (crash).

2ο Μέρος: Recommendation System με βάση μόνο τα χαρακτηριστικά των ταινιών.

Στο σημείο αυτό με βάση τους κανόνες που κατασκευάστηκαν στο Μέρος 1 θα κατασκευαστούν κατηγορήματα για την ομοιότητα ταινιών. Παρακάτω δίνεται ένα μικρό παράδειγμα ενός κανόνα και πώς αυτός θα μπορούσε να γραφτεί μέσω του Pyswip. Επίσης όπως αναφέρεται και σε σχόλιο παραπάνω θα μπορεί να γραφτεί και μια βάση δεδομένων με τους κανόνες και να γίνει απευθείας consult.

Στο παρακάτω παράδειγμα το 5 και το 4 εκφράζουν την ομοιότητα των ταινιών π.χ. οι ταινίες που παράγονται μέσω του find_similar_5 είναι πιο όμοιες από αυτές που παράγονται μέσω του find_similar_4.

In [ ]:
```
def simpleRecommender(movie):
    s = set()
    for i in range(1, 5):
        q = prolog.query("find_sim_" + str(i) + "('" + movie +"', M)")
        for sol in q:
            m = sol['M']
            s.add(sol['M'])
        q.close()
    return s
```

In [ ]:
```
q = prolog.query("find_sim_3('Iron Man', M)")
s = set()
for sol in q:
    s.add(sol['M'])
q.close()
print(s)
```

```
{'X-Men Origins: Wolverine', 'The Amazing Spider-Man 2', 'Teenage Mutant Ninja Turtles', 'Spider-Man 3', 'Iro
n Man 3', 'Mystery Men', 'Iron Man 2', 'Guardians of the Galaxy', 'X-Men: Days of Future Past', 'Captain Amer
ica: The Winter Soldier', 'Deadpool', 'The Shadow', 'Superman III', 'X2', 'Man of Steel', 'X-Men: The Last St
and', 'Ant-Man', 'Thor: The Dark World', 'Superman II', 'Batman v Superman: Dawn of Justice', 'Spider-Man 2',
'Kick-Ass 2', 'Fantastic Four', 'The Avengers', 'Thor', 'Avengers: Age of Ultron', 'Superman', 'Hulk', 'Red S
onja', 'Superman Returns', 'Fantastic 4: Rise of the Silver Surfer', 'X-Men', 'The Wolverine'}
```

In [ ]:
```
list(simpleRecommender("Iron Man"))
```

3ο Μέρος: Recommendation System Με βάση τις προτιμήσεις-Αξιολογήσεις του χρήστη-Εκπαίδευση και Πρόβλεψη

Αρχικά μελετάμε τις αξιολογήσεις κάθε χρήστη για να καταλάβουμε την δομή και τις πληροφορίες κάθε αρχείου.

In [ ]:

```
from tqdm.notebook import tqdm
from sklearn.metrics import precision_score, recall_score, f1_score
import numpy as np
import random

ratingWeights = {0: -1, 1: -0.5, 2:0, 3:0, 4:0.5, 5:1}
scoreWeights = {i:i + 1 for i in range(1)} ## ανάλογα με τα επίπεδα ομοιότητας που έχουν οριστεί στην simple_recommender

def trainRecommender(ratings, ratingWeights, scoreWeights, numberOfMovies = 10):
    """
    Στην συνάρτηση αυτή μπορούμε να ορίζουμε ποιο υποσύνολο των αξιολογήσεων θα χρησιμοποιήσουμε για το train μαζί με τα βάρη ομοιότητας και σκορ
    Σε συνδυασμό με τον αριθμό των ταινιών που θέλουμε να χρησιμοποιήσουμε σαν σύνολο δεδομένων π.χ. 10 από τις 100 ή 3 από τις 100 κ.ο.κ
    Αν θέλουμε να χρησιμοποιήσουμε όλες τις ταινίες σαν training set τότε ορίζουμε το number_of_movies = - 1
    """

    if(numberOfMovies > len(ratings)): numberOfMovies = len(ratings)

    if(numberOfMovies != -1):
        indices = random.sample(range(len(ratings)), numberOfMovies)
        ratings = ratings.iloc[indices]

    movieScore = {}
    for row in tqdm(ratings.itertuples(index = True, name = 'Pandas'), total = len(ratings)):
        movie = cleanText(getattr(row, 'movie_title'))
        rating = getattr(row, 'rating')

        similarMovies = simpleRecommender(movie)

        for sm in similarMovies:
            if sm not in movieScore:
                movieScore[sm] = ratingWeights[int(rating)] * scoreWeights[0]
            else:
                movieScore[sm] += ratingWeights[int(rating)] * scoreWeights[0]
                ## το weight θα το ορίσετε ανα επίπεδο ομοιότητας οι πολύ όμοιες ταινίες θα έχουν μεγαλύτερο βάρος
    return movieScore


## αυτό είναι ένα παράδειγμα για το πως θα μπορούσε να υλοποιήθει η predict
## έχουμε ορίσει ότι μια ταινία θα έπρεπε να είναι προτεινόμενη αν είχε σκορ > 0
def predictExample(ratings, movieScore):
    real, pred = [], []
    for i, row in enumerate(ratings.itertuples(index = True, name = 'Pandas')):
        movie = cleanText(getattr(row, 'movie_title'))
        rating = getattr(row, 'rating')

        if movie in movieScore: ## αν έχουμε σχηματίσει βαθμολογία για την ταινία αυτή
            pred.append(int(movieScore[movie] > 0)) ## heuristic για το αν μια ταινία είναι προτεινόμενη
            real.append(int(rating > 3))## έτσι ορίζουμε ότι μια ταινία θα έπρεπε να είναι προτεινόμενη
            ## η συνθήκη αυτή δεν μπορεί να αλλάξει
        else: ## δεν μπορούμε να προτείνουμε κάτι για το όποιο δεν έχουμε σχηματίσει εικόνα
            pred.append(0)
            real.append(int(rating > 3))

    return real, pred


def getMetrics(real, pred):
    metrics = {}
    metrics["precision"] = precision_score(real, pred, zero_division = 0) ## do not warn
    metrics["recall"] = recall_score(real, pred)
    metrics["f1"] = f1_score(real, pred)
    return metrics
```

Οι παραπάνω εκπαιδεύουν, ελέγχουν και μετρούν την απόδοσή του συστήματος συστάσεων μας. Για την εκπαίδευση του συστήματος μπορούμε κάθε φορά να χρησιμοποιήσουμε ένα τυχαίο υποσύνολο του training set. Όμως, είναι πιθανό το υποσύνολο των ταινιών αυτών να επηρεάζει τα αποτελέσματα στο training set. Για παράδειγμα, από 3 ταινίες στις 10 μπορούν για ένα πείραμα τα αποτελέσματα μας να είναι ίδια, και αυτό να μην οφείλεται στο γεγονός ότι ο ταξινομητής μας τα πηγαίνει καλά στις 3 ταινίες, αλλά στο γεγονός ότι οι υπόλοιπες 7 είναι τέτοιες ταινίες που δεν μας βοηθούν καθόλου, και αν είχαμε επιλέξει διαφορετικές 3 ταινίες να τα πηγαίναμε άσχημα. Οπότε προτείνουμε να τρέξετε κάθε φορά έναν αριθμό πειραμάτων για κάθε υποσύνολο ταινιών (π.χ. 10 πειράματα με 3 ταινίες, 10 πειράματα για 20 ταινίες κ.ο.κ.) και να κρατήσετε σαν τελικό σκορ το μέσο όρο όλων των πειραμάτων.

In [ ]:
```
import pandas as pd

trainRatings = pd.read_csv("/content/drive/My Drive/Colab Notebooks/AI/train_ratings.csv")
testRatings = pd.read_csv("/content/drive/My Drive/Colab Notebooks/AI/test_ratings.csv")
```

Training Size 10

In [ ]:
```
metrics = []
for i in range (10):
    movieScore = trainRecommender(trainRatings, ratingWeights, scoreWeights, 10)
    real, pred = predictExample(testRatings, movieScore)
    metrics.append(getMetrics(real, pred))

for metric in metrics[0].keys():
    print (f"average {metric}: {np.mean([m[metric] for m in metrics])}")
```

```
average precision: 0.49887205723738226
average recall: 0.3111111111111111
average f1: 0.3630509982477902
```

Training Size 30

In [ ]:
```
metrics = []
for i in range (10):
    movieScore = trainRecommender(trainRatings, ratingWeights, scoreWeights, 30)
    real, pred = predictExample(testRatings, movieScore)
    metrics.append(getMetrics(real, pred))

for metric in metrics[0].keys():
    print (f"average {metric}: {np.mean([m[metric] for m in metrics])}")
```

```
average precision: 0.49767221203325035
average recall: 0.5680555555555555
average f1: 0.5277637833758737
```

Entire Training Set

In [ ]:
```
metrics = []
for i in range (10):
    movieScore = trainRecommender(trainRatings, ratingWeights, scoreWeights, -1)
    real, pred = predictExample(testRatings, movieScore)
    metrics.append(getMetrics(real, pred))

for metric in metrics[0].keys():
    print (f"average {metric}: {np.mean([m[metric] for m in metrics])}")
```

```
average precision: 0.5140186915887849
average recall: 0.763888888888889
average f1: 0.6145251396648044
```

Παρατηρούμε ότι αυξάνοντας το ποσοστό του train set που αξιοποιούμε, βελτιώνονται οι μετρικές ταξινόμησης. Σημαντικότερη βελτίωση παρατηρείται στο recall $\left(\frac{TP}{TP+FN}\right)$, ενώ το precision $\left(\frac{TP}{TP+FP}\right)$ σημειώνει μικρή βελτίωση. Αυτό σημαίνει ότι σταδιακά πετυχαίνουμε περισσότερες από τις ταινίες που ήθελε ο χρήστης, παράλληλα όμως αυξάνονται και οι μη σχετικές προτάσεις.