# MoviesLearn

## Documentazione progetto Ingegneria della Conoscenza


| ***STUDENTI*** | ***MATRICOLE*** | ***Mail universitarie*** |
|---|---|---|
| Giovanni Pio Malerba | 697916 | g.malerba@studenti.uniba.it |
| Cosimo Rizzo | 755154 | c.rizzo23@studenti.uniba.it |


Link al repository del progetto su github: gmalerba1/progetto_ICON_MoviesLearn: Progetto di ingegneria della conoscenza realizzato da Giovanni Malerba e Cosimo Rizzo (github.com)

## Introduzione

Abbiamo realizzato un programma, chiamato MoviesLearn, che offre le seguenti funzionalità:

1. Predire il voto di un film, a partire da alcune informazioni in input;
2. Interrogare una base di conoscenza per ottenere informazioni utili.

Il programma consente l'interazione con l'utente mediante interfaccia grafica (GUI) o interfaccia a linea di comando (CLI).

## Istruzioni per l'esecuzione e requisiti:

Per l'esecuzione del programma è necessario eseguire uno dei due seguenti script, per usare un'interfaccia grafica oppure un'interfaccia testuale:

- CLI.py
- GUI.py

Il progetto è stato realizzato in Pyhton. L'ambiente di sviluppo è PyCharm. I requisiti per l'esecuzione del programma sono le seguenti librerie:

***Per l'esecuzione di CLI e GUI***

- Sklearn, per gli algoritmi di apprendimento
- TKinter: per l'interfaccia grafica
- Pyswip: (v.0.2.10) per poter usare Prolog all'interno di Python.
    - SWI Prolog: (v 8.4.3 , 64 bit, multithreaded).
      È importante che sia installato in modo corretto, seguire questa guida per altre informazioni:
      pyswip/INSTALL.md at master · yuce/pyswip (github.com)
- Joblib: per caricare e salvare i modelli di apprendimento.
- Pandas: per la manipolazione dei dati all'interno di Python.

***Per l'esecuzione degli altri script:***

- AST: per la lettura delle tabelle innestate dal dataset originale
- ProbFoil: usato per generare delle regole Prolog dopo una fase di apprendimento su una base di conoscenza relazionale
    - Problog: dipendenza necessaria per probfoil.

- Matplotlib e seaborn: per visualizzare i grafici dei modelli di apprendimento

***Note aggiuntive:***

- Il programma è stato testato su Windows 11 e Ubuntu Linux, e risulta funzionante.
  Tuttavia non siamo riusciti ad eseguirlo su MacOS con processore M1 (ARM), per problemi legati all'installazione di SWI Prolog, non funziona dunque la base di conoscenza.
- Potrebbe essere necessario modificare i percorsi dei file letti dal programma, a seconda del sistema operativo utilizzato.

## Organizzazione del dataset

Il dataset è stato prelevato dal sito web Kaggle.
https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset

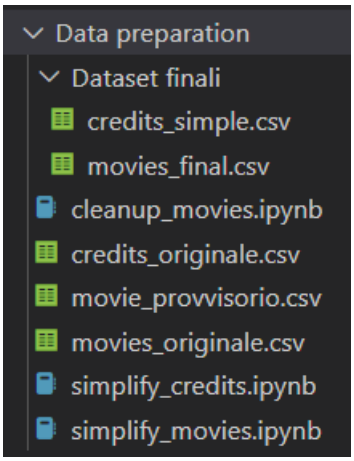
Contiene metadati per tutti i 45.000 film elencati nel database di Full Movie Lens. Tale insieme costituisce i film usciti entro luglio 2017. Questo set di dati consta di un file contenenti le 26 milioni di valutazioni da 270.000 utenti per tutti i 45.000 film. I voti per ogni film sono su una scala da 1 a 10 e sono state ottenuti dalle recensioni degli utenti dal sito ufficiale di GroupLens.
I seguenti file, contenuti nel dataset originale, sono stati opportunatamente manipolati per assolvere al nostro scopo:

- **movies_metadata.csv, i**l file principale contenete informazioni su circa 45.000 film. Include features come budget, genres, id, imdb_id, original_language, original_title, overview, popularity, production_companies, production_countries, release_date, revenue, runtime, tagline, title, vote_average e vote_count.

- **credits.csv**, contiene informazioni sul cast e sulla troupe per tutti i film identificati univocamente da "id".

## Fase di data preparation

***Struttura della cartella***




Nota: il file credits_originale non è presente sul repository github perché troppo grande, è comunque possibile scaricarlo da kaggle (credits.csv)

I file "credits_originale.csv" e "movies_originale.csv" sono i due dataset originali, scaricati da Kaggle.

Entrambi i file fanno uso di tabelle innestate (nested table), scritte in un formato JSON-Like.

In primo luogo, abbiamo provato ad usare la libreria JSON di Python per leggere questi dati, ma senza riuscire a farla funzionare: il formato testuale con cui sono scritte le tabelle non è infatti un JSON vero e proprio, bensì un JSON-Like.

Cercando sul web, abbiamo trovato la soluzione. Per poter avere accesso a queste tabelle facciamo utilizzo della libreria Python AST (Abstract Syntax Tree), che ci consente di accedere ai singoli campi di queste tabelle innestate.
In particolare si è fatto uso del metodo *literal_eval().*

I file sono creati in questo modo:

- simplify_credits(credits_originale) -> credits_simple
- simplify_movies(movies_originale) -> movie_provvisorio
- cleanup_movies(movie_provvisorio) -> movies_final

***Scelte progettuali:***

- Il file "*movies_originale.csv*" è stato semplificato eliminando le seguenti feature:
    - adult
    - belongs_to_collection
    - homepage
    - imdb_id
    - originial_title
    - overview (breve descrizione del film)
    - poster_path
    - production_countries
    - spoken_languages
    - status
    - tagline
    - video

- Inoltre, si è deciso di modificare alcuni attributi, mantenuti parzialmente:

    - genres: si è deciso di semplificare questo attributo, mantenendo solo un genere per film
    - production_companies: si è deciso di semplificare questo attributo, mantenendo una sola production company per film.

- Nel file "*credits_simple.csv*", invece, si è deciso di fare una distinzione:

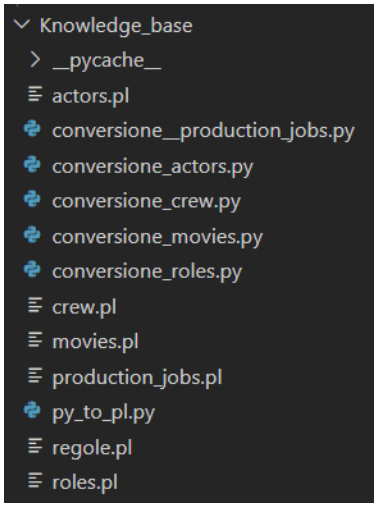
    - Abbiamo creato un file credits_simple: un file semplificato, da usare esclusivamente per le tecniche di apprendimento con SKlearn. Questo file contiene i seguenti campi:
        - ID: id del film
        - Attore1
        - Attore2
        - Attore3
        - Regista
        - Sceneggiatore

      Questi campi sono stati scelti perché ci sembrava che potessero rappresentare bene le feature "actors" e "crew" del file *credits_originale.csv*.
      Degli attori sono stati selezionati i primi tre, che erano, nelle tabelle innestate, disposti in ordine di importanza.
      Lo stesso per regista e sceneggiatore: se multipli, erano disposti sempre in ordine di importanza, e abbiamo deciso di prendere i primi.

    - Il file "*credits_originale.csv*" viene usato per la conversione in base di conoscenza, e da questo vengono mantenute tutte le informazioni.

# Costruzione e struttura della base di conoscenza

***Struttura della cartella***




La base di conoscenza è di tipo relazionale. Sono presenti più file PROLOG contenenti al loro interno fatti e regole.

I file sono generati a partire da:

- Data preparation/Dataset finali/movies_final.csv → movies.pl
- Data preparation/credits_originale.csv → actors.pl, crew.pl, production_jobs.pl, roles.pl
- regole.pl è un file di regole scritte manualmente

Altri file:

- Gli script Python che iniziano con “conversione” sono stati usati per effettuare le rispettive conversioni da file CSV a PROLOG.
  Questi script fanno uso (come nella fase di data preparation) della libreria AST, in particolare della funzione *literal_eval.*
- Il file “py_to_pl.py” usa la libreria pyswip per porre query a PROLOG. Le funzioni definite in questo file vengono chiamate per poter interagire con la base di conoscenza.

***Struttura dei file:***

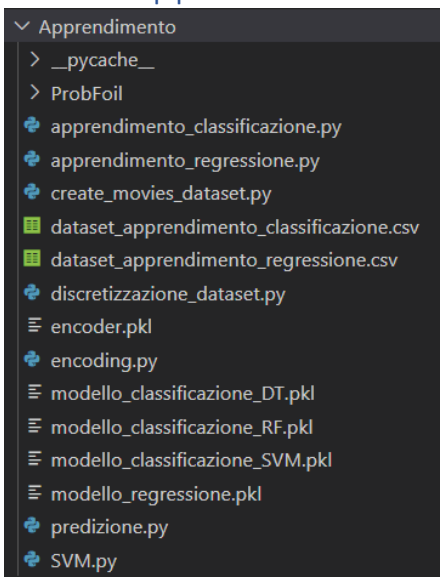
- “actors.pl” contiene un insieme di fatti del tipo ***attore(Nome,ActorId,Gender)***
- “crew.pl” contiene un insieme di fatti del tipo ***membro_crew(CrewID, Nome, Genere)***
- “movies.pl” contiene fatti del tipo ***film(budget, genres, id, original_language, popularity, production_companies, release_date, revenue, runtime, title, vote_average, vote_count)***
- “production_jobs” permette di collegare i membri della crew ad un film, con il loro lavoro all’interno del film con fatti del tipo ***ruolo_crew(MovieID, CrewID ,Job)***
- “roles.pl” permette di collegare gli attori con i film in cui recitano specificando il personaggio che interpretano, con fatti del tipo ***ruolo(MovieID, ActorId ,Role)***

- "regole.pl" contiene un insieme di regole per la base di conoscenza, utili per interrogare la KB. Vengono usate nel file "py_to_pl.py"

***Lista delle operazioni possibili per la KB:***

- Ottenere una lista di film considerati "capolavori"
- Ottenere una lista di film "cult" italiani
- Ottenere una lista dei migliori film di un attore
- Ottenere una lista dei migliori film di un regista
- Verificare se un attore è presente in un film
- Verificare se un regista ha diretto un film
- Verificare se un attore ha lavorato con un regista
- Verificare se un film è stato rilasciato in un certo anno
- Ottenere una lista di film dal voto alto
- Ottenere una lista di film dal budget alto
- Ottenere il cast di un particolare film
- Ottenere la crew di un particolare film
- Ottenere la lista completa di film di una determinata lingua
- Ottenere la lista completa di film di un determinato genere
- Ottenere la lista completa di film di lunga durata
- Ottenere la filmografia completa di un attore
- Ottenere la filmografia completa di un regista
- Ottenere il ruolo di un attore all'interno di un film

## Fase di apprendimento




***Encoding (file encoding.py)***

Questo script Python esegue l'encoding delle feature categoriche, rappresentate come stringhe, del dataset utilizzando la tecnica dell'One-Hot Encoding. Questo passaggio è necessario perché la libreria di machine learning **sklearn** non supporta le le variabili categoriche come input per l'apprendimento, nonostante gli alberi di decisione le supportino a livello teorico.

Importa le librerie necessarie: **joblib** per il salvataggio e il caricamento dei modelli, OneHotEncoder da *sklearn.preprocessing* per l'encoding delle feature, e **pandas** per la manipolazione dei dati.

Carica un dataset dal file CSV "*dataset_apprendimento_classificazione.csv*". Seleziona un insieme di features dal dataset per l'encoding. Crea un oggetto OneHotEncoder e lo addestra sulle features selezionate del dataset. Trasforma le features selezionate del dataset utilizzando l'encoder addestrato, risultando in un insieme di feature codificate. Infine, salva l'encoder addestrato in un file chiamato "*encoder.pkl*" per un utilizzo futuro, garantendo che i dati futuri vengano trasformati nello stesso modo dei dati di addestramento.

***Predizione (file predizione.py)***

Lo script definisce tre funzioni di previsione, ciascuna delle quali utilizza un modello di apprendimento supervisionato diverso per fare previsioni basate su otto caratteristiche di input. Le funzioni sono le seguenti: *predict_random_forest*, *predict_albero_decisione* e *predict_regressione*. Tali funzioni caricano il corrispondente modello, ed encoder da file .pkl, trasformano i dati di input utilizzando l'encoder, e poi, effettuano una predizione utilizzando il modello.

Le due funzioni di classificazione restituiscono due categorie, mentre la funzione di regressione restituisce un voto compreso tra 1 e 10.

- ***dataset_apprendimento_classificazione.csv***
  Al dataset finale è stata applicata una discretizzazione dei giudizi (attraverso il file *discretizzazione_dataset.py*), in quanto valori continui. I voti assegnati sono del tipo "Good" o "Bad", rappresentati numericamente, rispettivamente come 1 o 0.
  Le categorie del voto, originariamente definite con le classi "Good" e "Bad", sono state riportate come valori numerici per evitare di eseguire l'encoding.

- ***dataset_apprendimento_regressione.csv***
  Il suddetto dataset servirà per la regressione lineare. Deriva dall'integrazione tra credits_simple.csv e movies_final.csv, due insiemi di dati opportunatamente bonificati per migliorarne l'usabilità. Fusione che viene implementata dal file *create_movies_dataset.py*

Il **Support Vector Machine (SVM)** è un algoritmo di apprendimento supervisionato utilizzato per la classificazione lineare o non lineare, e sia per la regressione. L'obiettivo principale è trovare l'iperpiano ottimale in uno spazio N-dimensionale che può separare i punti dati in diverse classi nello spazio delle caratteristiche. L'iperpiano cerca di massimizzare il margine tra i punti più vicini di classi diverse. L'algoritmo SVM ottiene la massima efficacia nei problemi di classificazione binaria.

Nel nostro caso, l'algoritmo cerca di trovare l'iperpiano ottimale che separa i film con voti altii da quelli con voti bassi nello spazio delle caratteristiche. La costruzione del modello viene configurata attraverso:

- Definizione dei parametri: **param_grid** è un dizionario che contiene i parametri da ottimizzare (C e gamma). C è un parametro che controlla il compromesso tra ottenere un margine il più largo possibile e minimizzare gli errori di classificazione. Gamma è un parametro del kernel RBF che controlla la forma del confine di decisione.

- Ricerca della griglia: **GridSearchCV** è una funzione che esegue una ricerca esaustiva su specificati valori di parametri per un estimatore. In questo caso, si esegue la ricerca sulla griglia dei parametri C e gamma del classificatore SVM. La ricerca viene eseguita utilizzando la validazione incrociata a 2 fold, ciò significa che i dati vengono suddivisi in due parti e il modello viene addestrato su una parte e testato sull'altra.
- Ottimizzazione del modello: l'obiettivo è trovare la combinazione di parametri che produce il modello SVM più accurato sui dati. Il modello dovrebbe essere in grado di fare buone previsioni sui nuovi dati, senza sovradattamento o sottoadattamento.

La **regressione lineare** è un modello statistico che stima la relazione lineare tra una variabile dipendente e una o più variabili indipendenti. Cerca di adattare una linea retta ai dati in modo da minimizzare la somma dei quadrati delle differenze tra i valori osservati nel dataset e i valori predetti dal modello.
Il Mean Squared Error (*MSE*) è una misura che calcola la media dei quadrati degli errori. Il Mean Absolute Error (*MAE*) è una misura che calcola la media delle differenze assolute tra i valori predetti e i valori reali. L' *R2* score, o coefficiente di determinazione, è una misura statistica che indica quanto bene i risultati del modello di regressione si adattano ai dati reali.

Un **albero decisionale** è un modello di apprendimento supervisionato che può essere utilizzato sia per la classificazione che per la regressione. Ogni nodo interno rappresenta una variabile (una decisione basata su una caratteristica del film), un arco verso un nodo figlio rappresenta un possibile valore per quella proprietà e una foglia indica il valore predetto per la variabile obiettivo (il voto finale), a partire dai valori delle altre proprietà. Nella funzione *DecisionTreeClassifier(max_depth = 20)*, il parametro passato è un criterio di arresto pre-pruning che limita la profondità massima dell'albero e rimuove i rami che si dividono in caratteristiche di bassa importanza (per prevenire l'overfitting).

Il **Random Forest** è un algoritmo di apprendimento supervisionato che può essere utilizzato sia per la classificazione che per la regressione. È un metodo ensemble, ovvero combina l'output di più strutture ad albero decisionali per raggiungere un unico risultato. Ognuno di questi alberi viene addestrato su un sottoinsieme casuale del dataset. Il parametro *n_estimators* specifica il numero di strutture da utilizzare nel modello, il che può migliorare le sue prestazioni e rendere le previsioni più stabili.

Il modello viene addestrato sul dataset_apprendimento_classificazione.csv e cerca di prevedere il giudizio di un film basandosi sui seguenti attributi:

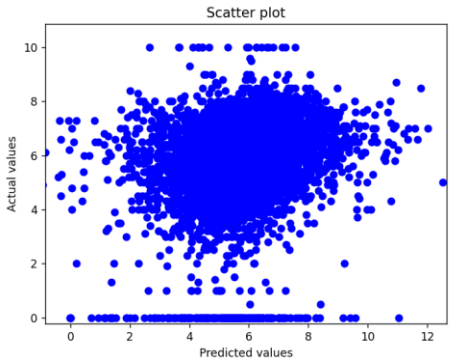
```
features = ['attore1','attore2','attore3','regista','sceneggiatore','genres','original_language','production_companies']
target = ['vote_average']
```

***Valutazione delle prestazioni:***

In seguito all'addestramento, si effettua una valutazione delle prestazioni dei modelli adottati utilizzando diverse metriche come l'accuratezza, la precisione, il recall, e la media armonica F1.

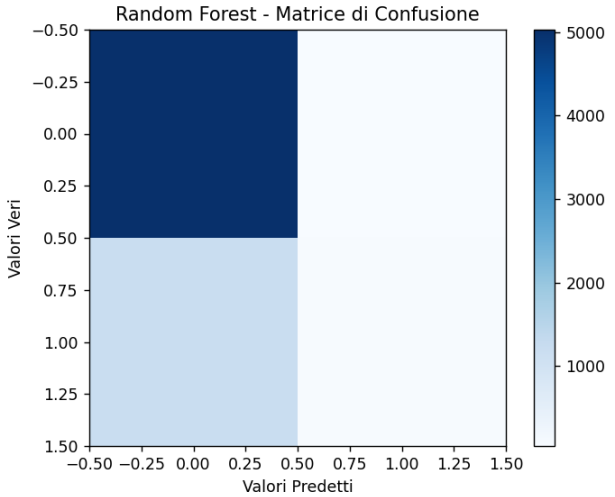
- Regressione lineare

```
----------------------------------------
Modello: Linear Regression
Mean Squared Error:  3.637762803355911
Mean Absolute Error:  1.3517295730251158
R2:  0.9998876463937177
----------------------------------------
```



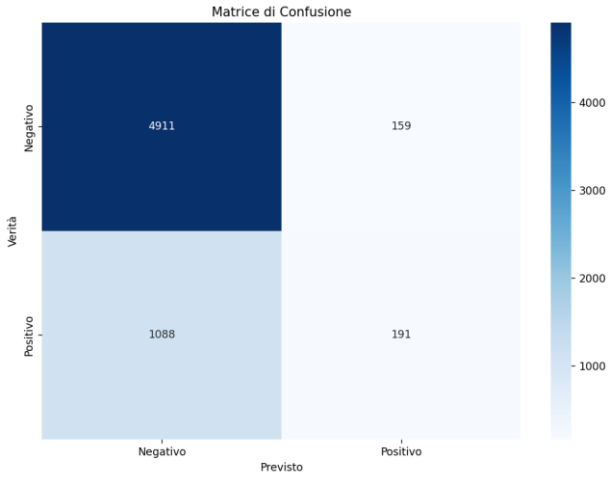

- Random Forest:

```
----------------------------------------
Modello: Random Forest
Accuracy:  0.8035911167112931
Precision:  0.7730899332247123
Recall:  0.8035911167112931
F1:  0.7315804774405515
----------------------------------------
```



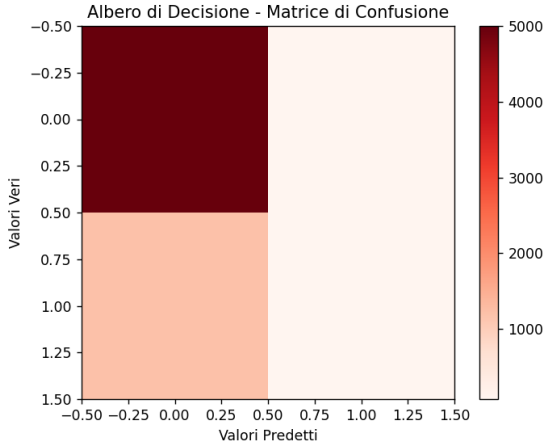

- Support Vector Machine (SVM):

```
Best parameters:
 {'classifier__C': 10, 'classifier__gamma': 0.01}
              precision    recall  f1-score   support

           0       0.82      0.97      0.89      5070
           1       0.55      0.15      0.23      1279

    accuracy                           0.80      6349
   macro avg       0.68      0.56      0.56      6349
weighted avg       0.76      0.80      0.76      6349

------------------------------------------------
Modello: SVM
Accuracy:  0.8035911167112931
Precision:  0.763656531614407
Recall:  0.8035911167112931
F1:  0.7558283619394934
```




- Albero di decisione

```
------------------------------------------
Modello: Albero di decisione
Accuracy:  0.8004410143329658
Precision:  0.7531461569500989
Recall:  0.8004410143329658
F1:  0.7306636126818411
------------------------------------------
```



# Apprendimento su una base di conoscenza relazionale: tentativi con ProbFoil

Durante le nostre ricerche per il progetto, ci siamo imbattuti in questo articolo di medium: Towards Deep Learning for Relational Databases | by Gustav Šír | Towards Data Science, nel quale si discute di metodi di apprendimento su database relazionali.

Ci siamo chiesti se esistesse qualcosa di simile anche per le basi di conoscenza relazionali in Prolog, e cercando, si è trovato questo progetto: RoseSAK/probfoil-defaults: MSc Project: Modification of ProbFOIL2 algorithm to learn categorical defaults (github.com).

Probfoil è uno strumento di machine learning basato su Problog (estensione del linguaggio prolog che permette di definire la probabilità di un fatto), che consente di apprendere delle regole sulla base di conoscenza, dati una serie di esempi positivi e negativi.

***Funzionamento:***

Tramite probfoil vengono presi in input (all'interno di uno o più file) i seguenti:

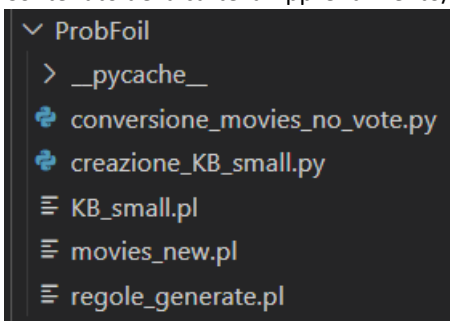
- Data: i fatti che costituiscono la base di conoscenza
- Settings: un file che contiene
    - Target: il predicato per il quale vogliamo apprendere una regola
    - Base: definizione dei tipi di fatti contenuti all'interno della base di conoscenza
    - Mode: contiene le modalità con le quali si può accedere alle variabili.
    - Example mode: open o closed, indica se il programma può creare esempi negativi in automatico o no.
- Esempi: un file che contiene una serie di esempi positivi e/o negativi (a probabilità zero), dal quale poi apprendere una regola.

Il programma viene eseguito da terminale, con il comando *probfoil file.pl* .

Il programma restituisce in output una o più regole per cercare di spiegare gli esempi positivi e negativi inseriti.

***La nostra implementazione:***

Contenuto della cartella Apprendimento/ProbFoil




Abbiamo definito il nostro target per la predizione il prevedere film con un buon voto: abbiamo definito questa proprietà con *high_vote_average(IDFILM)*.

Il nostro procedimento è stato questo:

1. Avevamo bisogno di un file prolog "*movies.pl*" diverso da quello usato per la base di conoscenza, in quanto quest'ultimo conteneva il voto, e sarebbe stato troppo facile per il programma associare un film con proprietà high_vote_average(IDFILM) con un film con un voto alto.
   Dunque, con lo script contenuto in "*conversione_movies_no_vote.py*" abbiamo creato un file "*movies_new.pl*" che fosse uguale a *movies.pl ma* senza il voto.
2. Avevamo bisogno di esempi positivi per la proprietà *high_vote_average(MovieID)* . Per ottenere questi esempi, nel file "*creazione_kb_small.py*" andiamo ad interrogare la base di conoscenza, in modo tale da ottenere gli ID di tutti i film dal voto maggiore o uguale a nove.

3. Lo stesso con gli esempi negativi: abbiamo impostato una serie di fatti a probabilità zero per *high_vote_average(MovieID)*, dopo aver ottenuto gli ID di tutti i film con voto minore di sei.
4. Il file finale che abbiamo ottenuto è "KB_small.pl", che contiene al suo interno target per la predizione, esempi positivi e negativi, definizioni dei tipi di fatti, modalità e i fatti della base di conoscenza (con voto rimosso da "movies.pl")

***I nostri tentativi:***

1. Come prima cosa abbiamo provato ad eseguire l'apprendimento sulla base di conoscenza completa, con circa 140K righe di fatti prolog e almeno 20K esempi (positivi e negativi).
   Dopo aver lasciato il programma in esecuzione per circa 16 ore abbiamo rinunciato.

Ci siamo resi conto di dover ridimensionare il problema, e per fare ciò abbiamo riscritto lo script "*creazione_KB_small.py"* in modo tale da poter impostare un massimo di esempi positivi e negativi da prelevare, e di conseguenza modificare anche il numero di film, attori e membri della crew da prendere in considerazione.
Inoltre, facendo delle prove, abbiamo notato che tra i primi 50 film (dal voto alto) prelevati, più di 20 erano documentari.
Abbiamo pensato che potesse avere a che fare con l'ordine in cui i dati erano stati inseriti all'interno del dataset originale, e abbiamo pensato quindi di prendere gli esempi in modo casuale.

2. Il tentativo successivo è stato con 50 esempi positivi, 50 esempi negativi e circa 3000 righe di fatti prolog, presi in modo casuale.
   Dopo circa 10 ore di esecuzione abbiamo rinunciato.
3. Dopo aver letto la documentazione, abbiamo provato limitando la profondità della ricerca a 3, e inoltre abbiamo ridimensionato ulteriormente il problema, con 25 esempi positivi e 25 negativi, circa 1800 righe di fatti prolog.
   Per limitare la profondità della ricerca abbiamo eseguito il comando *probfoil -l 3 KB_small.pl*.
   Questa volta l'esecuzione è terminata dopo circa mezz'ora, producendo diverse regole prolog.
4. Abbiamo provato con gli stessi fatti ottenuti casualmente del punto 3, aumentando però la profondità della ricerca a 4.
5. L'ultimo tentativo è stato fatto generando un nuovo file, questa volta con 50 esempi positivi, 50 negativi, quindi circa 2800 righe di fatti prolog. Inoltre si è usato probfoil con profondità 4.

Le regole prodotte da questi tentativi possono essere esaminate nel file "*regole_generate.pl",* nella cartella Probfoil.

***Considerazioni***:

Come si può notare dalle regole prodotte al passo 3, sembra che il programma, con una profondità così limitata, non sia riuscito a generalizzare delle regole, e che abbia dato invece solo un grande elenco di regole specifiche per quel training set.

Anche al passo 4 pare che la profondità maggiore non fosse sufficiente per generalizzare abbastanza.

Possiamo notare come nel passo 5 siano state prodotte più regole, dato il numero maggiore di esempi, ma comunque non siano state prodotte delle regole davvero generali.

In ogni caso possiamo trarre delle considerazioni, notando alcune regole in comune tra le varie predizioni:

- Generi più apprezzati:
  - La regola che ci ha sorpreso di più è stata quella sul genere "documentario", presente in tutte le previsioni effettuate. Pare infatti che, anche selezionando esempi in modo casuale, il documentario sia tra i generi più apprezzati e con voto più alto.
  - Altro genere molto apprezzato è il musical (riportato come "music").
- Case di produzione: tra le più apprezzate sono
  - metro-goldwyn-mayer mgm
- Ruoli:

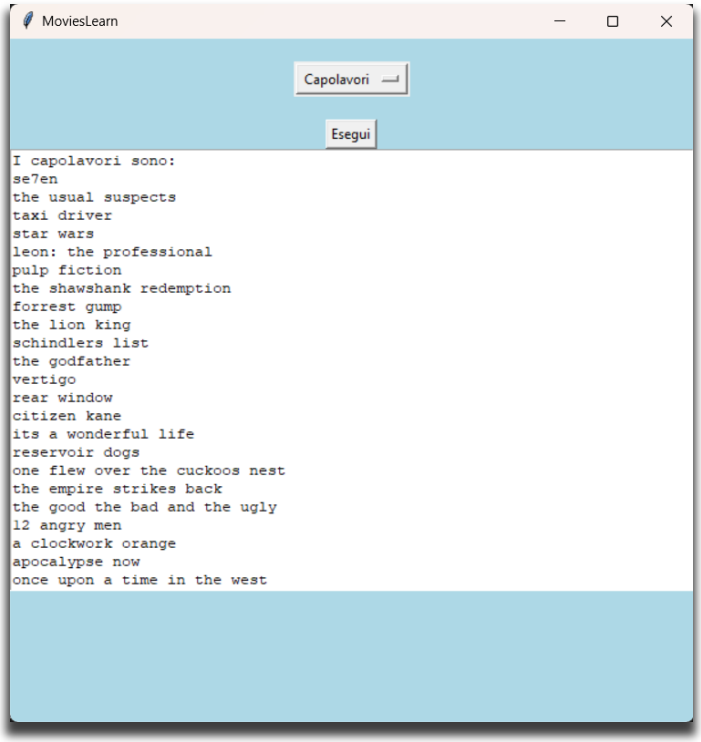
- “himself” ovvero il “cameo” di un attore in un film è risultato il ruolo più apprezzato, probabilmente perché è uno dei ruoli che compaiono più spesso

Vantaggi di questo metodo:

- Rispetto ai metodi di apprendimento supervisionato visti in precedenza con SKlearn, Probfoil ha il vantaggio di produrre delle regole facilmente comprensibili.
- Probfoil ha inoltre il vantaggio di poter usare la rappresentazione relazionale per derivare nuove regole: tra le regole prodotte ce ne sono diverse che coinvolgono più relazioni diverse.

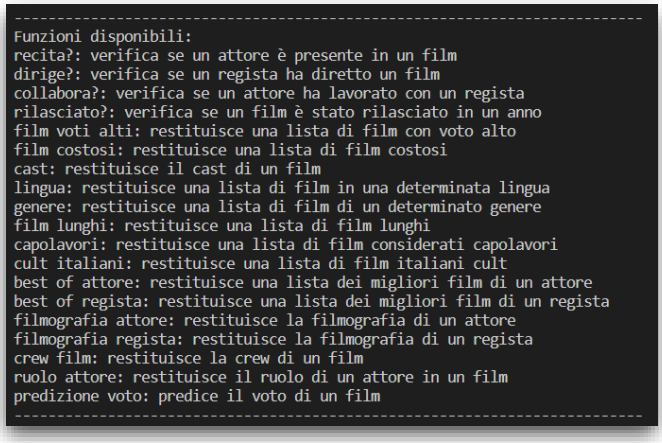
## Interfaccia grafica e interfaccia CLI

L’interfaccia grafica è stata realizzata con la libreria TKinter di python e si presenta in questo modo:




È possibile scorrere con il mouse sul widget di testo per leggere la lista completa di film restituiti, se la lunghezza supera quella del widget.

Per utilizzare l’interfaccia a linea di comando, digitare “help” per ottenere i comandi disponibili:



# Esempi di input e output

Presentiamo qualche esempio per le funzioni più complesse della base di conoscenza:

- Capolavori:
  se7en, the usual suspects, taxi driver, star wars, leon: the professional, pulp fiction, the shawshank redemption, forrest gump, the lion king, schindlers list, the godfather, vertigo, rear window, citizen kane, its a wonderful life, reservoir dogs, one flew over the cuckoos nes, the empire strikes back, the good the bad and the ugly, 12 angry men, a clockwork orange, apocalypse now, once upon a time in the west, goodfellas, psycho, the godfather: part ii , once upon a time in America, dead poets society, the shining, back to the future, life is beautiful, american history x, fight club, princess Mononoke, the green mile, memento, the lord of the rings: the fellowship of the ring, spirited away, the lord of the rings: the two towers, my neighbor totoro, the pianist, city of god, the lord of the rings: the return of the king, howls moving castle, the prestige, the dark knight, inception, the intouchables, the grand budapest hotel, interstellar, whiplash, the imitation game, room, your name, lion
- Cult italiani:
  cinema paradiso, the good the bad and the ugly, once upon a time in the west, life is beautiful, bicycle thieves
- Best of attore (Robert De Niro):
  heat, casino, taxi driver, goodfellas, the godfather: part ii, once upon a time in America, raging bull, the deer hunter, jackie brown, the untouchables, meet the parents, shark tale, meet the fockers, stardust, machete, little fockers, limitless, silver linings playbook, lenny bruce: swear to tell the truth, the family, american hustle, the intern, joy, dirty grandpa
- Best of regista (Chistopher Nolan):
  memento, insomnia, batman begins, the prestige, the dark knight, inception, the dark knight rises, interstellar, dunkirk

Esempi di predizione del voto, per film potenzialmente con voto alto e basso:

- Predizione voto (Robert De Niro, Al pacino, Leonardo Dicaprio, Martin Scorsese, Quentin Tarantino, Crime, en, Universal Pictures):
  - Random Forest: good movie (voto > 7)
  - Albero di decisione: good movie
  - Regressione: 10

- Predizione voto (Christian De Sica, Massimo Boldi, Enzo Salvi, Neri Parenti, Neri Parenti, Comedy, it, filmauro):
  - Random Forest: bad movie (voto < 7)
  - Albero di decisione: bad movie
  - Regressione: 3.35

# Conclusioni

I metodi di apprendimento utilizzati producono risultati simili tra loro, in particolare, sono facilmente comparabili i metodi di classificazione (random forest, albero di decisione e SVM), le cui predizioni sono quasi sempre concordanti. Tra i modelli di classificazione adoperati, quello leggermente migliore degli altri sembra essere il modello SVM, misurando le metriche di accuratezza, precisione e recall.

Anche la regressione lineare sembra funzionare bene, a volte però prevedendo voti troppo alti. Possiamo inoltre notare dal valore R^2 che il modello di regressione si sia adattato bene al training set.

Per quanto riguarda, invece, l'apprendimento sulla base di conoscenza relazionale con Probfoil, nonostante i nostri tentativi e restringendo progressivamente l'insieme degli esempi, positivi e negativi, e i fatti della KB, sembra che il sistema non sia stato in grado di generare delle regole davvero generali capaci di selezionare un insieme di film con voti alti.
Un altro problema sono stati gli attributi di tipo numerico, non sfruttati dal programma. Una soluzione possibile potrebbe essere una discretizzazione preventiva per questi ultimi, prima di applicare probfoil.
Inoltre probabilmente si sarebbero potuti ottenere risultati migliori con un numero di esempi e fatti più grande e con un limite maggiore alla profondità per l'algoritmo, avendo più tempo a disposizione.