Movie Industry Insider

Documentazione progetto ICon

Gruppo di lavoro

- **Giovanni Stea**, 758470, g.stea8@studenti.uniba.it
- **Raffaele Nacci**, 760456, r.nacci8@studenti.uniba.it

Repo: https://github.com/giosteh/ProgettoICon-MovieIndustryInsider

A.A. 2023-24

Introduzione.

Il progetto, realizzato in Python, mira a studiare e indagare, tramite gli strumenti propri dell'ingegnere della conoscenza, gli aspetti produttivi e commerciali dell'**industria cinematografica** degli ultimi quarant'anni.

Allo scopo, sono stati utilizzati due dataset reperiti dal web: il primo relativo ai *film* usciti dal 1980 al 2020, e il secondo invece relativo agli *artisti* del cinema (registi, attori, ecc.).

Sommario.

I dati sono stati opportunamente pre-processati nella fase inziale, per essere poi integrati all'interno di una *knowldege base*, definita in Prolog, che modellasse e rendesse esplicita la struttura di individui e relazioni presente nei dati. Sono state quindi definite e ingegnerizzate diverse *clausole* che permettessero, tramite la KB e il ragionamento automatico, di derivare nuove informazioni con le quali arricchire i dati già a nostra disposizione.

Successivamente ci si è concentrati su problemi relativi alla predizione del voto IMDb e dell'efficienza finanziaria di un film. I due problemi, rispettivamente di regressione e classificazione, sono stati affrontati utilizzando *metodi di machine-learning e deep-learning*.

L'impiego di metodi di *feature selection* ci ha permesso poi, tramite i modelli addestrati, di studiare e individuare quali siano le caratteristiche che più determinano la qualità e il successo commerciale di un film.

Infine, tramite strumenti di *modellazione bayesiana*, si sono studiate le relazioni di dipendenza (e indipendenza) tra le variabili del dominio, e la misura dell'impatto che queste ultime hanno sulle performance di un prodotto dell'industria hollywoodiana e non solo.

Elenco argomenti di interesse.

- **Rappresentazione e ragionamento relazionale**. Si è utilizzato il linguaggio Prolog per costruire una KB e ingegnerizzare nuove features altrettanto informative tramite ragionamento automatico.
- **Apprendimento supervisionato e Reti neurali**. Sono stati addestrati diversi modelli di machine learning (alberi decisionali, random forest, ecc.) e deep learning (reti neurali feed-forward) sul nuovo dataset derivato.
- **Ragionamento e apprendimento con incertezza**. Si è sia costruita che appresa una rete bayesiana a partire dai dati, allo scopo di indagare e analizzare le relazioni causali tra le variabili del dominio.

Descrizione dei dati e preprocessing.

A partire dalla comune passione per il *cinema*, si è deciso di incentrare il progetto sull'analisi del dominio relativo all'industria cinematografica e delle sue dinamiche produttive.

A questo scopo si sono cercati su Kaggle dei dataset che potessero fare al caso nostro: la scelta è ricaduta su due dataset distinti, uno contenente un discreto numero (più di 7500) di film usciti dal 1980 al 2020 e le relative caratteristiche, e un'altro, scelto in modo da *correlare i dati* con il primo, contenente invece dati riguardanti attori, registi, sceneggiatori, ecc. per un totale di più di 500 mila istanze.

Si è anche tentato di arricchire i dati ricercando ulteriori dataset, in particolare relativi alle compagnie di produzione, non trovando però nulla di utile per il caso di studio. Per sopperire alla mancanza di altri dati, è stata fondamentale la fase successiva di studio del dominio e *ingegnerizzazione della base di conoscenza*, tramite la quale è stato possibile derivare ulteriori informazioni, utilizzate poi nella fase di apprendimento.

Ovviamente si è coscienti che non si tratta di dati esaustivi che comprendano tutti i film effettivamente prodotti e distribuiti tra il 1980 e il 2000, né tantomento tutti i registi, sceneggiatori e attori che abbiano lavorato ad un film, ma, nonostante ciò, si dispone di *campione discretamente grande* che conta diverse migliaia di esempi.

Descrizione dei dati.

Ogni *film* nel dataset rispettivo è descritto dalle seguenti features:

- `` `name` ``. Il titolo completo del film.
- `` `rating` ``. Il grado di adeguatezza alla visione del film per certe fasce di pubblico.
- `` `genre` ``. Il genere del film.
- `` `year` ``. L'anno di uscita del film.
- `` `released` ``. La data di uscita del film.
- `` `score` ``. Il voto ricevuto dal film su IMDb.
- `` `votes` ``. Il numero di voti ricevuti dal film su IMDb.
- `` `director` ``, `` `writer` ``, `` `star` ``. Regista, sceneggiatore e attore principale del film.
- `` `country` ``. Il paese di produzione del film.
- `` `budget` ``. Il budget (in dollari) speso per il film.

- `gross`. I ricavi (in dollari) ottenuti dal film.
- `company`. Il nome della compagnia di produzione del film.
- `runtime`. La durata in minuti del film.

Mentre ogni *artista*, nel secondo dataset, è descritto dalle seguenti features:

- `primaryName`. Nome e cognome dell'artista.
- `birthYear`, `deathYear`. Anno di nascita e morte (se presente) dell'artista.
- `primaryProfession`. Le professioni principali dell'artista.
- `knownForTitle`. Il titolo del film per cui l'artista è conosciuto.

I link ai dataset sono indicati nella sezione relativa ai riferimenti.

Preprocessing dei dati.

Entrambi i dataset sono stati poi sottoposti a una fase di *preprocessing iniziale* (nel notebook `initial_preprocessing.ipynb`) al fine di pulire i dati e prepararli alla successiva fase di costruzione della KB. Sono stati utilizzati i tools della libreria `pandas`.

In particolare, i passaggi di preprocessing eseguiti sui due dataset sono stati i seguenti:

- sul dataset dei film (`movies.csv`)
    1. è stata aggiunta una nuova colonna `id`, in modo da avere un attributo che identificasse univocamente ciascun film;
    2. sono state rimosse le colonne `released` e `writer`, poiché ritenute non utili agli scopi preposti per i task;
    3. sono state rimosse le righe con `budget` o `gross` nulli, poiché non si aveva modo di ottenere valori certi riguardo l'ammontare di budget e ricavi di ciascun film per cui questi mancavano;
    4. sono stati sostituiti manualmente i valori nulli delle colonne `runtime`, `rating`, `country` e `company` (tramite ricerca e verifica sul web).
- sul dataset degli artisti (`combined.csv`)
    1. la colonna `primaryProfession` (stringa che elenca da una a tre professioni dell'artista) è stata prima suddivisa in tre attributi distinti, per poi selezionare

solo le prime due professioni principali, in quanto la terza presentava per la maggiorparte valori nulli;

2. le colonne `deathYear` e `birthYear` sono state portate nel formato intero e i valori mancanti per `deathYear` (per gli artisti ancora in vita) sono stati impostati a 0.

A questo punto si evidenziano *due relazioni uno-a-molti tra film e artisti*: un'artista può aver diretto uno o più film (nel caso di un regista) o recitato come star principale in uno o più film (nel caso di un attore), mentre ciascun film ha un solo regista e una sola star.

Per questo motivo sono stati selezionati, dal dataset processato degli artisti, solo quelli che occorressero in almeno uno dei film del dataset processato `movies_adj.csv` tramite *left-join*, e sono stati salvati nel nuovo dataset `artists.csv`.

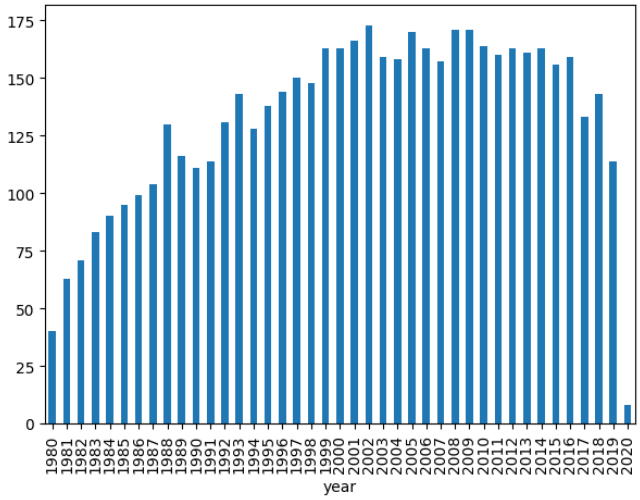
Di seguito sono riportate le cardinalità dei dataset impiegati, prima e dopo la fase di preprocessing appena descritta.

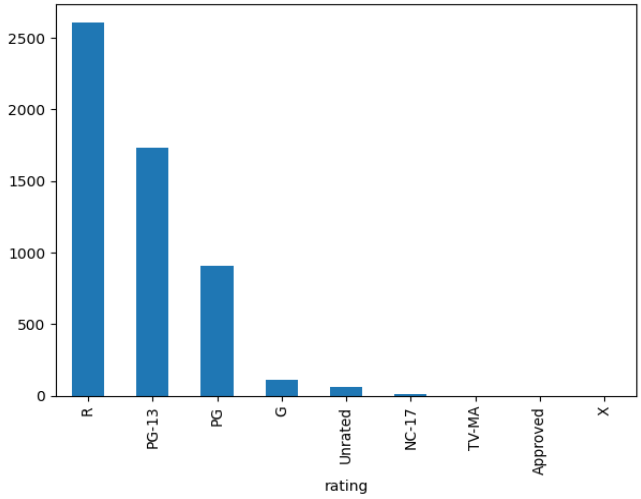
	Prima	*Dopo*	*Usciti nelle sale negli ultimi 30 anni*
FILM	7668	**5436**	**4046**
ARTISTI	546421	**3811**	-

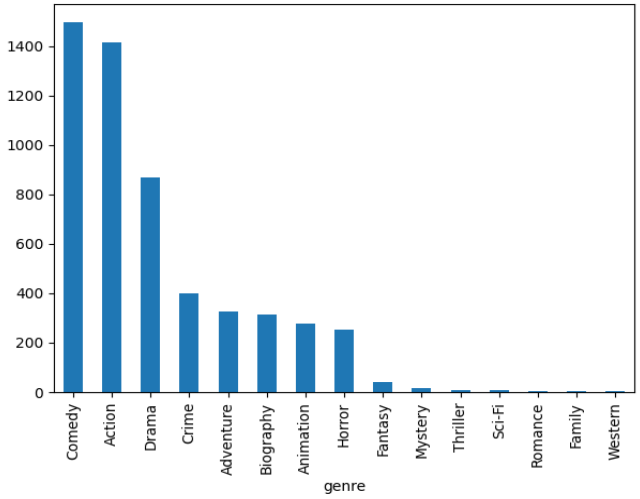
Prima di continuare con il capitolo successivo relativo alla definizione e costruzione della KB a partire dai dati, alla pagina successiva sono riportati alcuni grafici che mostrano la distribuzione dei dati relativamente alle feature più significative per i film.

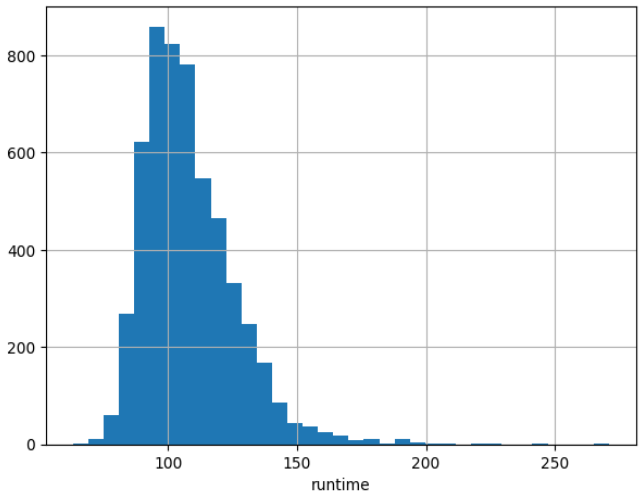
Si fanno dunque le seguenti considerazioni:

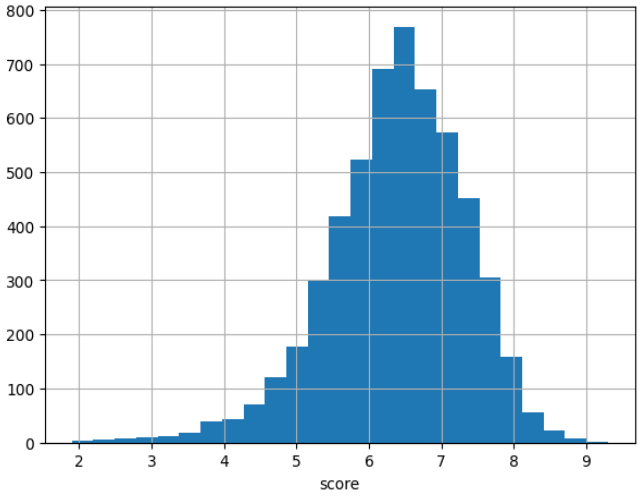
- il numero di film presenti per ogni anno segue una distribuzione pressoché *uniforme* (con fisiologiche oscillazioni), soprattutto dal 1990 circa al 2020;
- le feature relative a rating e genere presentano alcune categorie molto minoritarie;
- la durata e il voto IMDb dei film seguono, come prevedibile, una *distribuzione normale (o gaussiana)*. In particolare vediamo come la media di durata di un film sia intorno ai 100 minuti e il voto medio circa 6.5;
- il budget dei film, così come l'incasso (non mostrato in figura), segue invece quella che sembra una *distribuzione esponenziale*: vediamo infatti i film prodotti decrescano esponenzialmente all'aumentare del budget.

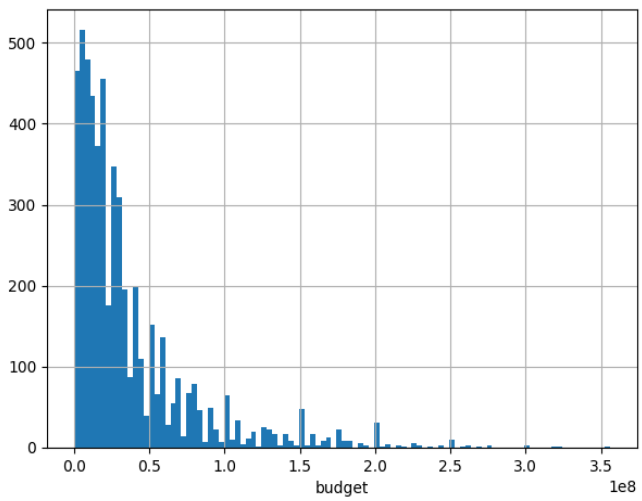

year


R
PG-13
PG
G
Unrated
NC-17
TV-MA
Approved
X
rating


Comedy
Action
Drama
Crime
Adventure
Biography
Animation
Horror
Fantasy
Mystery
Thriller
Sci-Fi
Romance
Family
Western
genre


runtime


score


budget
1e8

Modellazione della KB e feature engineering.

Una volta preprocessati i dati di film e artisti come descritto, si è ritenuto opportuno sfruttare la struttura nella forma di *individui e relazioni* inerentemente presente nei dati presi in esame: si è allora definita una *knowledge base (KB)* utilizzando il linguaggio logico `Prolog`, integrato con il resto del codice attraverso la libreria `pyswip`. La KB è stata prima di tutto *popolata di fatti* utilizzando i dataset processati; sono state poi definite *clausole* in modo da inferire nuova conoscenza (e quindi *nuove feature*) a partire dai fatti che potesse essere utilizzata nei successivi task.

Individui e relazioni.

É immediato notare che dai dati raccolti relativi a film e artisti emerga una struttura che può essere modellata utilizzando *individui e relazioni tra individui*. Si sono quindi definite, utilizzando per ognuna un simbolo di funzione distinto, due *classi di individui*.

- `movie(M)` che denota il singolo film ("movie") con `` `id`: M ``.
- `professional(P)` che denota il singolo artista con `` `primaryName`: P `` (nome completo), qui indicato con il nome "professional".

Tra gli individui sopra specificati esistono *due relazioni uno-a-molti*, in quanto per ogni film è stato scelto di mantenere sia il regista che la star principale. Le relazioni definite sono quindi le seguenti:

- `directed_by(M, D)` specifica che il film `movie(M)` è stato diretto dal regista `professional(D)`.
- `starring(M, S)` specifica invece che nel film `movie(M)` l'attore `professional(S)` ha recitato come star principale.

Come già detto, si tratta di due relazioni uno-a-molti poiché ogni film vede un solo regista e una sola star principale, mentre un regista può aver naturalmente diretto più film e un attore può aver recitato come star in più film.

Fatti.

A questo punto sono stati definiti, per entrambe le classi di individui, una serie di fatti che, nella KB, specificassero le *proprietà associate a ciascun individuo*. Le proprietà specificate nei fatti corrispondono ai valori delle feature dei dataset processati.

I fatti definiti per il generico individuo `movie(M)` sono:

```
title(M, Tt)
rating(M, Rt)
genre(M, Gn)
year(M, Yr)
score(M, Sc)
votes(M, Vt)
runtime(M, Rn)
budget(M, Bd)
gross(M, Gr)
```

I fatti definiti invece per il generico individuo `professional(P)` sono:

```
birth_year(P, Br)
death_year(P, Dt)
known_for(P, Kn)
primary_profession(P, Pr)
secondary_profession(P, Sc)
```

I fatti sono salvati nel file `facts.pl`.

Clausole.

Sono state poi definite (nel file `clauses.pl`) una serie di clausole `Prolog` in modo da poter *fare ragionamento sulla KB*: in particolare inferire nuova conoscenza a partire dai fatti e quindi *ingegnerizzare nuove feature*. Le clausole sono presentate in parte riportando codice completo e spiegazione del funzionamento in linguaggio naturale e in parte riportando solo nome del predicato e funzionamento in breve.

Sono presentate prima le clausole che riguardano il singolo film.

`age(M, Age)`: calcola l'età del film in riferimento al 2024.

`age_category(M, Cat)`: assegna al film una categoria rispetto all'anno di uscita ("new", "old", "very-old").

`runtime_category(M, Cat)`: assegna al film una categoria rispetto alla sua durata in minuti ("short", "medium", "long").

Il binning effettuato nelle clausole è stato pensato sulla base dello studio delle distribuzioni, a meno che non sia specificato diversamente.

```
budget_efficiency(M, Eff) :-
      budget(M, B),
      gross(M, G),
      B \= 0,
      Eff is G / B.
```

`budget_efficiency(M, Eff)`: calcola il rapporto tra gli incassi lordi del film e il suo budget (in dollari).

Il valore di efficienza di budget rappresenta un indicatore molto importante rispetto alla performance commerciale/economica di un film, infatti, poiché un film si consideri un successo commerciale questo rapporto dovrebbe arrivare addirittura a 3, in quanto dal budget sono esclusi i costi relativi alla promozione e alla distribuzione nelle sale.

`budget_efficiency_category(M, Cat)`: assegna al film una categoria rispetto alla sua efficienza commerciale ("low", "mid", "high").

Il binning di budget efficiency è stato pensato proprio sulla base delle considerazioni fatte sopra riguardo gli obiettivi commerciali di una produzione cinematografica.

`budget_category(M, Cat)`: assegna al film una categoria rispetto al suo budget in dollari ("low", "mid", "high", "very-high").

`popularity_category(M, Cat)`: assegna al film una categoria rispetto al numero di votazioni ricevute su IMDb ("low", "mid", "high", "very-high").

```
score_mean(Mean) :-
      findall(S, score(_, S), Scores),
      sum_list(Scores, Total),
      length(Scores, N),
      N > 0,
      Mean is Total / N.

score_std(Std) :-
      score_mean(M),
      findall(S, score(_, S), Scores),
      sum_of_squares(Scores, M, SumSq),
      length(Scores, N),
      N > 1,
      Std is sqrt(SumSq / (N - 1)).
```

`score_mean(Mean), score_std(Std)`: calcolano la media e la deviazione standard del voto IMDb dei film nella base di conoscenza.

Per lo scopo è stato anche definito il predicato `sum_of_squares/3`.

```
sum_of_squares([], _, 0).

sum_of_squares([V|Rest], Avg, SumSq) :-
      sum_of_squares(Rest, Avg, RestSumSq),
      SumSq is RestSumSq + (V - Avg)^2.
```

`is_acclaimed(M)`: è vero se il film ha un voto IMDb maggiore di media più deviazione standard di score.

`is_panned(M)`: è vero se il film ha un voto IMDb inferiore a media meno deviazione standard di score.

`score_category(M, Cat)`: assegna al film una categoria rispetto al voto IMDb utilizzando una suddivisione basata su media e deviazione standard della distribuzione normale dei voti ("very-low", "low", "high", "very-high").

Ne viene mostrato un esempio per la categoria "high"; le altre categorie sono derivate similmente dividendo lo spazio dei voti in quattro bins usando media e deviazione standard.

```
score_category(M, "high") :-
      score(M, X),
      score_mean(Mean),
      score_std(Std),
      X >= Mean, X < Mean + Std.
```

`genre_regrouped(M, Cat)`: rimappa le categorie di genere minoritarie a "other".

`rating_regrouped(M, Cat)`: rimappa le categorie di rating minoritarie a "other".

Ora si elencano invece le clausole relative agli artisti (per brevità si riportano solo le clausole relative ai registi in quanto per gli attori le clausole sono identiche a eccezione dell'utilizzo della relazione `starring(M, S)` in luogo di `directed_by(M, D)`).

```
age_in_movie(P, U, Age) :-
      year(U, UY),
      birth_year(P, B),
      Age is UY - B.
```

`age_in_movie(P, U, Age)`: calcola l'età dell'artista in un certo film.

`age_in_movie_category(P, U, Cat)`: assegna all'artista una categoria rispetto all'età che ha in un certo film.

Per evitare di inferire conoscenza su film futuri (che in un contesto reale non si potrebbe avere) si sono progettate le clausole sopra riportate e le successive in modo che considerassero, nell'aggregare i dati, solo conoscenza passata sui film di un certo artista.

```
director_experience(D, U, N) :-
      year(U, UY),
      findall(M,
      (directed_by(M, D), year(M, Y), Y < UY),
      Movies),
      length(Movies, N).
```

`director_experience(D, U, N)`: calcola il numero di film realizzati dal regista fino all'anno in cui ha realizzato il film `U`.

`director_experience_category(D, U, Cat)`: assegna al regista una categoria rispetto al numero di film realizzati fino all'anno in cui ha realizzato il film `U`.

```
director_is_acclaimed(D, U) :-
      year(U, UY),
      findall(M,
      (directed_by(M, D), year(M, Y), Y < UY, is_acclaimed(M)),
      Movies),
      length(Movies, N),
      N > 0.
```

`director_is_acclaimed(D, U)`: è vero se il regista ha realizzato almeno un film acclamato prima dell'anno in cui ha realizzato il film `U`.

```
director_is_panned(D, U) :-
      year(U, UY),
      findall(M,
      (directed_by(M, D), year(M, Y), Y < UY, is_panned(M)),
      Movies),
      length(Movies, N),
      N > 0.
```

`director_is_panned(D, U)`: è vero se il regista ha realizzato almeno un film stroncato prima dell'anno in cui ha realizzato il film `U`.

Queste ultime due clausole sono state pensate appunto per non essere mutuamente esclusive, ma piuttosto per fornire conoscenza sullo storico dei film dell'artista in questione: in particolare relativamente alla qualità (molto positiva e/o molto negativa) dei suoi film.

```
director_budget_efficiency(D, U, Avg) :-
      year(U, UY),
      findall(E,
      (directed_by(M, D), year(M, Y), Y < UY, budget_efficiency(M, E)),
      Effs),
      length(Effs, N),
      sum_list(Effs, Total),
      (N > 0 -> Avg is Total / N ; Avg is 0).
```

`director_budget_efficiency(D, U, Avg)`: calcola la media dell'efficienza di budget del regista, considerando i suoi film fino all'anno in cui ha realizzato il film `U`.

`director_budget_efficiency_category(D, U, Cat)`: assegna al regista una categoria rispetto alla sua media di budget efficiency, considerando solo i film realizzati fino all'anno in cui ha realizzato il film `U`.

Query e inferenza.

Una volta definite le clausole sopra riportate, è stata eseguita la *fase di inferenza sulla KB* tramite query `Prolog` (eseguite in Python con `pyswip`): in particolare si è eseguita una query per ogni nuova feature fino a comporre tutte le colonne del nuovo dataset.

Nello specifico i dataset derivati sono tre:

- i primi due sono identici a eccezione della feature relativa alla efficienza di budget:
  - nel primo (`movies_features_reg.csv`), pensato per il task di regressione sulla feature `` `score` ``, la variabile `` `budget_efficiency` `` è derivata come feature numerica da usare per la predizione;
  - nel secondo (`movies_features_cls.csv`), pensato per il task di classificazione proprio sulla budget efficiency, la feature è derivata come categorica con il nome di `` `budget_efficiency_cat` ``.
- il terzo (`movies_features_nb.csv`) è invece derivato pensando all'utilizzo con Naive Bayes, quindi tutte le feature sono state derivate come categoriche.

Nella derivazione sono state escluse le feature ``.

Per tutti e tre i dataset sono state poi selezionate solo le righe relative ai *film usciti negli ultimi 30 anni*: ciò è stato fatto principalmente per far sì che le feature derivate per registi e attori fossero efficaci nell'aggregare conoscenza sui film precedenti. Se non si fosse proceduto in questo modo, le righe relative ai film dei primi anni considerati dal dataset avrebbero presentato valori non utili nelle feature derivate dalla KB.

Apprendimento (regressione e classificazione).

Approntati i nuovi dataset contenenti sia feature originali che nuove feature ingegnerizzate a partire dalla conoscenza della KB, si sono individuati due task di apprendimento la cui risoluzione avrebbe potuto *fornire "insights" sull'industria cinematografica, nello specifico sulla buona riuscita della produzione di un film*:

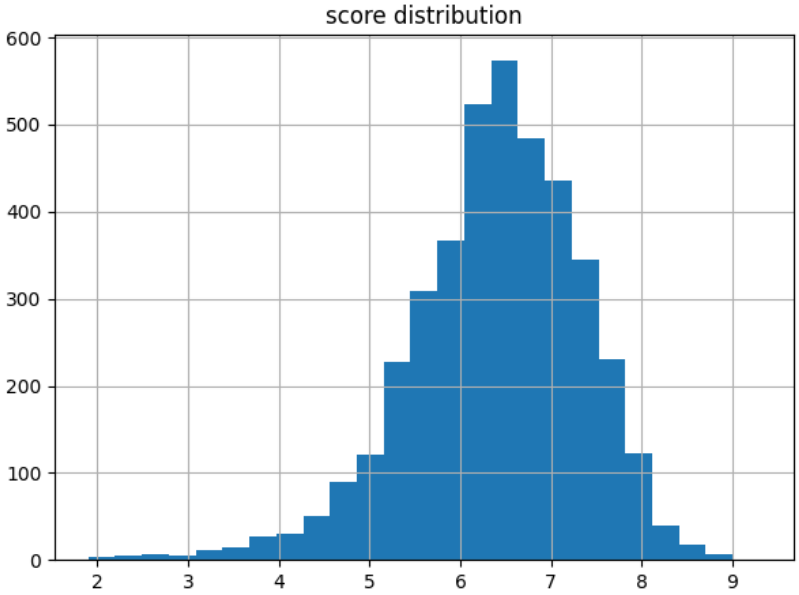
- un primo *task di regressione* sulla feature `` `score` ``, una variabile numerica che varia tra 0 e 10 indicando il voto IMDb di un certo film e che, considerando il gran numero di utenti della piattaforma, è un buon indicatore della qualità artistica di un film
- un secondo *task di classificazione* sulla feature `` `budget_efficiency_cat` ``, una variabile categorica che invece classifica un film sulla base della sua efficienza rispetto al budget stanziato dalla produzione. Può assumere tre valori:
  - `"low"` se la budget efficiency del film è inferiore a 1, ovvero il film non ha nemmeno recuperato il budget investito, è stato cioè un "flop" commerciale
  - `"mid"` se la budget efficiency del film è compresa tra 1 e 3, ovvero il film ha recuperato almeno quanto è stato investito per la produzione effettiva del film, ma non tanto da superare anche i costi di promozione e distribuzione
  - `"high"` se la budget efficiency del film è superiore a 3, ovvero il film è riuscito ad incassare in modo da coprire i costi di produzione, promozione e distribuzione e ottenere anche un discreto margine di profitto effettivo.

Il secondo dei due task presenta sicuramente una sfida più ostica, ma è allo stesso tempo quello che più sarebbe utile risolvere nel contesto dell'industria cinematografica; il primo è invece più interessante dal punto di vista artistico.

Per entrambi i task sono stati eseguiti una serie di esperimenti impiegando diversi modelli di apprendimento automatico (messi a disposizione da `scikit-learn`) dalla complessità crescente, allo scopo non solo di ottenere buone performance di regressione/classificazione, ma anche e soprattutto per comprendere, per quanto possibile, quali siano le *variabili che più determinano la qualità artistica e la riuscita commerciale di un film.*

Descrizione delle feature target.

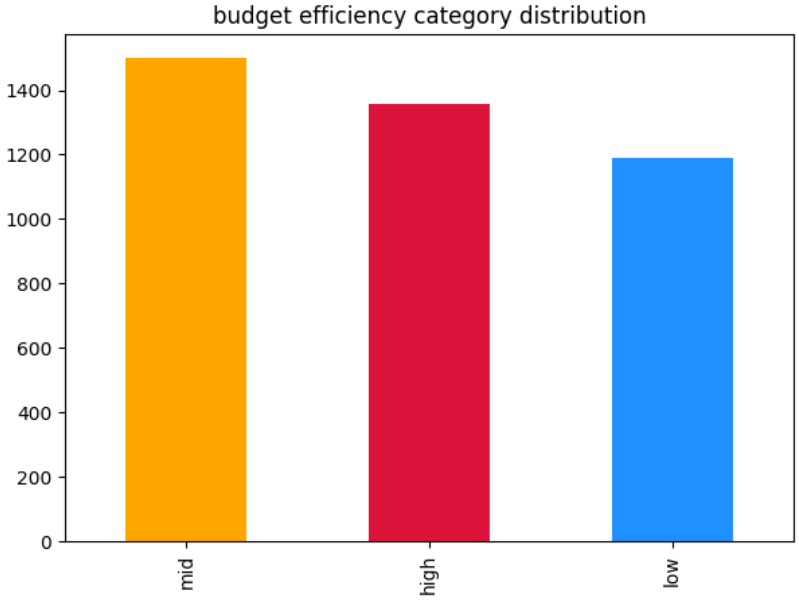
La feature target per il task di regressione, come detto, è lo `` `score` `` IMDb dei film, il quale presenta una distribuzione gaussiana, come mostrato già in precedenza e come è possibile vedere dal seguente grafico.



La target `score` non ha subito un preprocessing particolare, in quanto, trattandosi di un task di regressione, l'obiettivo era proprio quello di predire al meglio possibile il valore numerico della feature, in altre parole la sua distribuzione, la quale poteva essere alterata nel caso si fosse proceduto con una normalizzazione.

Per la feature target `budget_efficiency_cat`, che reca la *classe di efficienza economica* del film, è invece stata impiegata una codifica con `LabelEncoding` (0, 1, 2) come previsto dai modelli di classificazione di `scikit-learn`. Mostriamo di seguito la sua distribuzione nel dataset di riferimento.

Dal grafico vediamo come le classi non siano particolarmente sbilanciate, nonostante ciò, per il task di classificazione, si è comunque scelto di condurre due serie di esperimenti, una addestrando i modelli sul dataset non bilanciato e un'altra bilanciando le classi.



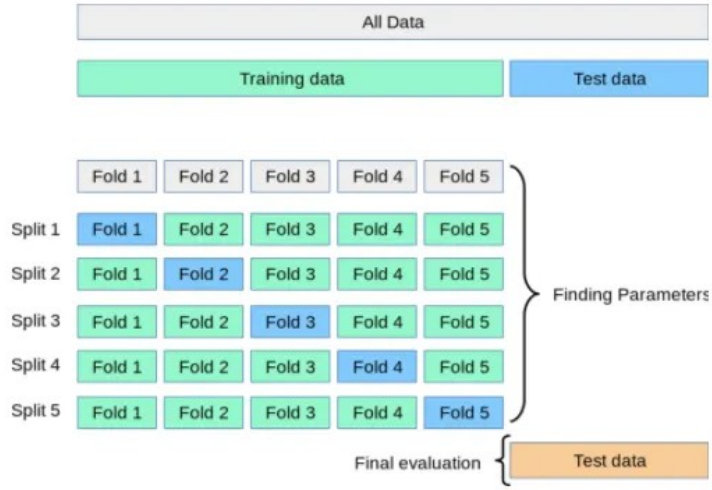
Approccio e modelli utilizzati.

L'approccio seguito per i task di regressione e classificazione è stato speculare: in particolare, per ognuno dei due task si sono scelti e addestrati quattro *modelli dalla complessità crescente*. Più un modello è complesso infatti, più questo sarà in grado di riconoscere pattern complessi nei dati, di contro però, sarà più prono all'*overfitting*, ovvero al sovra-adattamento ai dati di training con conseguente perdita della *capacità di generalizzare su nuovi dati.* Nell'addestramento dei modelli si è stati attenti soprattutto a questo aspetto tramite tecniche di validazione incrociata.

Le implementazioni scelte per i modelli sono quelle della libreria `scikit-learn` e `xgboost`. Nello specifico, i modelli impiegati sono, nell'ordine:

1. uno *lineare* (`RidgeRegressor, LogisticRegression`)
2. uno *ad albero* (`DecisionTreeRegressor, DecisionTreeClassifier`)
3. un *ensemble basato su alberi* (`RandomForestRegressor, RandomForestClassifier`)
4. uno *ad albero che usa il boosting* (`XGBoostRegressor, XGBoostClassifier`)

Si è preferito utilizzare modelli basati su alberi per due motivi: questi eseguono inerentemente *feature selection* valutando la capacità di ciascuna feature di discernere gli esempi nel training set, e assegnano alle feature uno score di importanza, il che li rende quindi dei *modelli spiegabili.*



L'addestramento di ciascun modello ha previsto tre fasi:

1. *tuning degli iper-parametri* eseguito per ottenere i migliori parametri dell'algoritmo che esegue il training del modello. Per fare ciò si è utilizzata la *k-fold cross validation* con k pari a 5 (come mostrato in figura) misurando la media delle metriche ottenute sui 5 fold usati a turno come *validation set.*

Questa fase è stata a sua volta suddivisa in due sottofasi:

1. una prima ricerca dei migliori iper-parametri, tramite il metodo `GridSearch` di `scikit-learn`, il quale esplora un insieme predefinito di combinazioni, valutate usando la cross validation;
2. una seconda fase (eseguita solo per i modelli basati su alberi) che esegue una 5-*fold cross validation manuale*, su un range di valori più ampio, per gli iper-parametri relativi all'altezza del/degli alberi e al numero di stimatori (il range è ottenuto a partire dai valori trovati con `GridSearch`).

2. *training del modello* (con gli iper-parametri trovati) su tutto il training set, escludendo ovviamente i dati di test.
3. *testing (valutazione) del modello* sul test set, che comprende gli esempi non utilizzati per validazione e training.

Risultati per il task di regressione.

Mostriamo di seguito i risultati dei modelli per il task di regressione su `score`. Per la predizione sono state utilizzate tutte le feature derivate e riottenute dalla KB, a eccezione di `id` e `title`, le quali non hanno alcuna importanza al fine della predizione.
Aggiungiamo che le feature categoriche su rating e genere del film sono state codificate con one-hot encoding, le feature `runtime`, `director_age` e `star_age` sono state scalate con uno standard scaler e le feature `popularity`, `budget`, `director_experience` e `star_experience` sono state scalate tra 0 e 1 con il metodo min-max.

Le performance dei modelli sono state misurate utilizzando le metriche classiche *MAE* (Mean Absolute Error) e *MSE* (Mean Squared Error).

Prima di riportare i risultati per ogni modello sono indicati in una tabella i valori degli iper-parametri testati con `GridSearch`. Prima di ciascun modello verranno indicati i valori scelti.

La tabella è riportata alla pagina seguente.

Modello	*Iper-parametri*
`RidgeRegressor`	`Alpha: [.05, .1, .5, 1, 2, 5]`

`DecisionTreeRegressor`	`Criterion: ["squared_error", "friedman_mse"]`
	`Max depth: [5, 10, 15, 20]`
	`Min samples split: [2, 5, 10, 15]`
	`Min samples leaf: [2, 4, 8, 10, 12]`

`RandomForestRegressor`	`Criterion: ["squared_error", "friedman_mse"]`
	`Num estimators: [100, 200, 300]`
	`Max depth: [5, 7, 10, 15]`
	`Min samples split: [2, 5, 10]`
	`Min samples leaf: [2, 4, 8, 10]`

`XGBoostRegressor`	`Num estimators: [100, 200, 300]`
	`Max depth: [3, 5, 7, 10]`
	`Learning rate: [.01, .1, .2]`
	`Min child weight: [1, 3, 5]`
	`Subsample: [.6, .8, 1]`

Di seguito sono elencati risultati dei modelli ed eventuali considerazioni.

Ridge regressor.

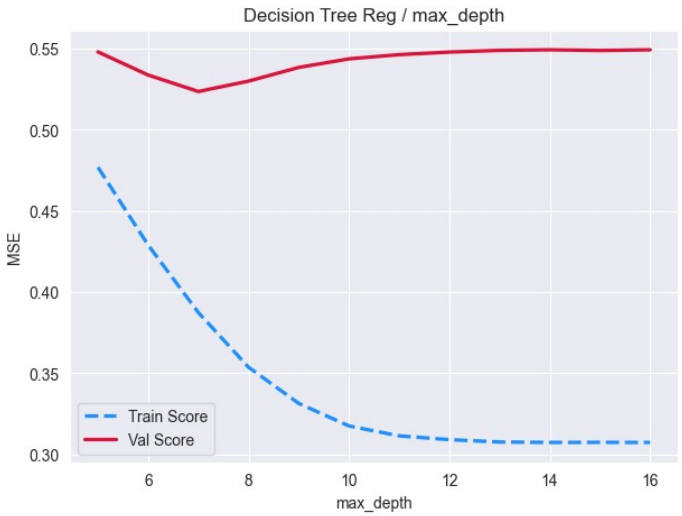
É stato scelto `Alpha: 0.5;` ottenendo i seguenti risultati.

`MAE: 0.5354`
`MSE: 0.5116`

Decision Tree regressor.

Sono stati scelti `Criterion: "squared_error", Max depth: 7, Min samples leaf: 12, Min samples split: 2;` ottenendo i seguenti risultati.

Notiamo come all'aumentare dell'altezza dell'albero (quindi della sua complessità) vi sia, come prevedibile, overfitting sul training set. Ciò è riscontrabile soprattutto con l'aumento dell'errore sul validation una volta superata un'altezza di 7.



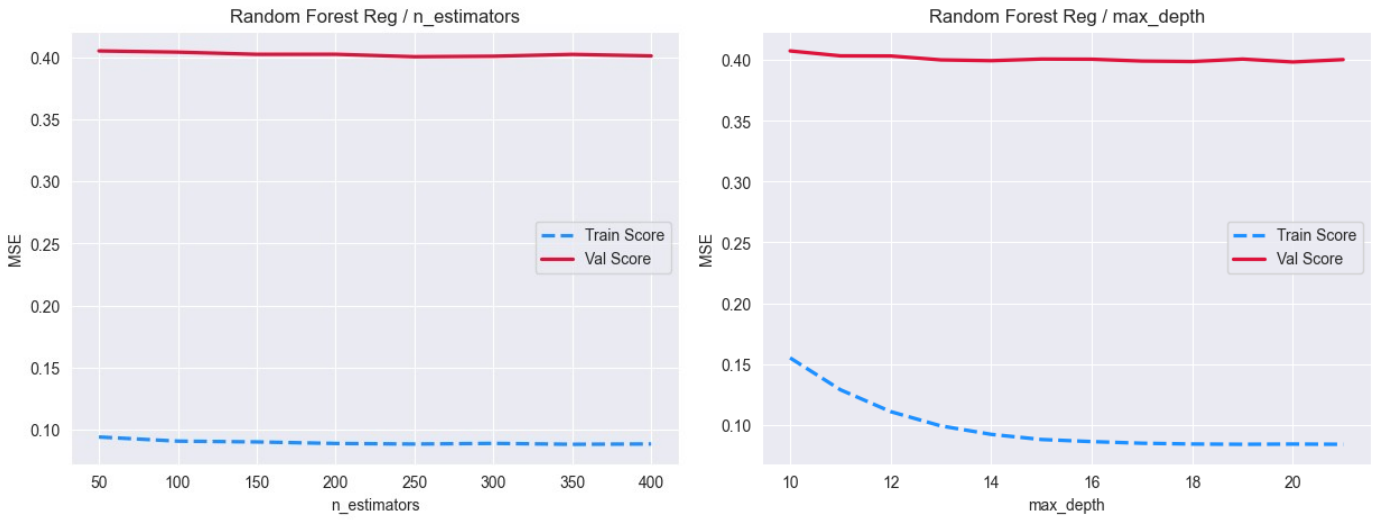
MAE: 0.5788
MSE: 0.5801

Il singolo decision tree esibisce una performance peggiore nell'errore commesso rispetto alla classica regressione lineare con regolarizzazione Ridge, che comunque sbaglia di solo 0.5 circa nel predire un voto da 0 a 10.

Random Forest regressor.

Sono stati scelti `Criterion: "friedman_mse", Max depth: 20, Min samples leaf: 2, Min samples split: 2, Num estimators: 250;` ottenendo i seguenti risultati.

Notiamo come, in questo caso, non vi sia particolare overfitting riscontrabile, in quanto le performance sul validation rimangono pressoché stabili all'aumentare della complessità.



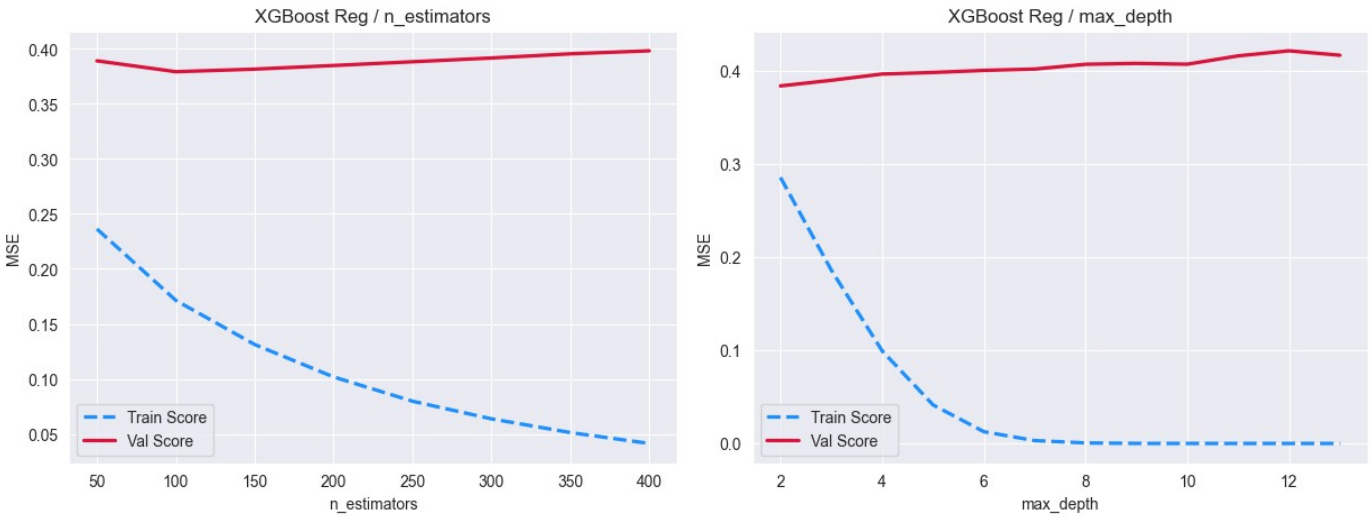
MAE: 0.4849
MSE: 0.4368

La Random Forest ottiene delle migliori performance scendendo ad un errore di 0.43.

XGBoost regressor.

Sono stati scelti `Learning rate: 0.1, Max depth: 2, Min child weight: 5, Num estimators: 100, Subsample: 1;` ottenendo i seguenti risultati.

Notiamo, nel caso del boosting, un leggero overfitting sia all'aumentare del numero di stimatori (dopo i 100) che all'aumentare della profondità massima degli alberi.



MAE: 0.4912
MSE: 0.4523

Il modello di regressione che ha meglio performato è stato la Random Forest, il quale è stato tra l'altro anche il modello che ha esibito il *minore overfitting* e quindi la migliore capacità di *generalizzazione sul test set.*

Risultati per il task di classificazione.

Mostriamo di seguito i risultati esibiti dai modelli per il task di classificazione su `` `budget_efficiency_cat` ``. Per la predizione sono state utilizzate tutte le feature derivate e riottenute dalla KB, a eccezione di `` `id` ``, `` `title` `` e `` `popularity` ``: le prime due sono state escluse in quanto irrilevanti per la predizione, mentre `` `popularity` `` è stata esclusa per simulare ancora meglio un contesto reale in cui si vorrebbe conoscere l'esito commerciale di un film prima della sua uscita.

Aggiungiamo che le feature categoriche su rating e genere sono state codificate con one-hot encoding, le feature `runtime`, `director_age` e `star_age` sono state scalate con uno standard scaler e le feature `budget`, `director_experience` e `star_experience` sono state scalate tra 0 e 1 con il metodo min-max.

Le performance dei modelli sono state misurate utilizzando le metriche classiche di *Accuracy* (numero di predizioni corrette / numero di predizioni totali) e usando la matrice di confusione per visualizzare *precision, recall e F1-score*.

Prima di riportare i risultati per ogni modello sono indicati in una tabella i valori degli iper-parametri testati con `GridSearch`. Prima di ciascun modello verranno indicati i valori scelti.

Modello	*Iper-parametri*
`LogisticRegression`	`Max iter: [10000, 20000]` `Penalty: ["l1", "l2"]`

`DecisionTreeClassifier`	`Criterion: ["gini", "entropy"]`
	`Max depth: [5, 10, 15, 20]`
	`Min samples split: [2, 5, 10, 15]`
	`Min samples leaf: [2, 4, 8, 10, 12]`

`RandomForestClassifier`	`Criterion: ["gini", "entropy"]`
	`Num estimators: [100, 200, 300]`
	`Max depth: [5, 7, 10, 15]`
	`Min samples split: [2, 5, 10]`
	`Min samples leaf: [2, 4, 8, 10]`

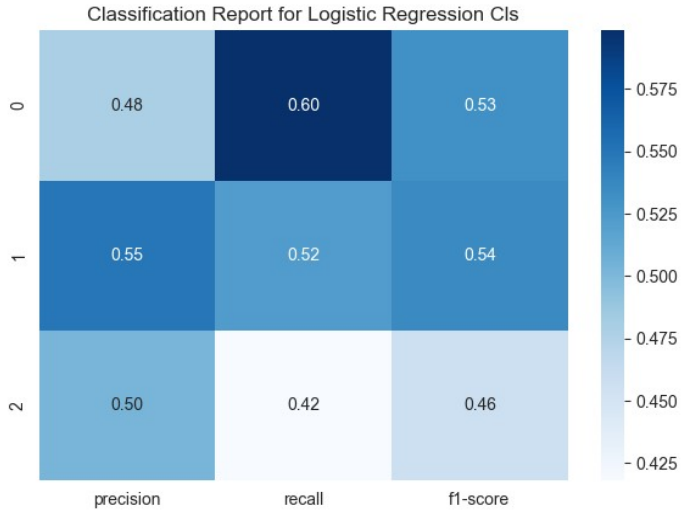
`XGBoostClassifier`	`Num estimators: [100, 200, 300]`
	`Max depth: [3, 5, 7, 10]`
	`Learning rate: [.01, .1, .2]`
	`Min child weight: [1, 3, 5]`
	`Subsample: [.6, .8, 1]`

Di seguito sono riportati i risultati dei modelli ed eventuali considerazioni. Verranno riportati solo i risultati dei modelli in seguito al resampling (effettuato con la tecnica `SMOTE`): in generale le performance sono state migliori ma similari a quelle ottenute senza resampling (difatti lo scarto tra il numero di istanze per classe non era molto elevato).

Logistic Regression (multinomial).

Sono stati scelti `Max iter: 10000, Penalty: "l1";` ottenendo i seguenti risultati.

`Accuracy: 50.62%`

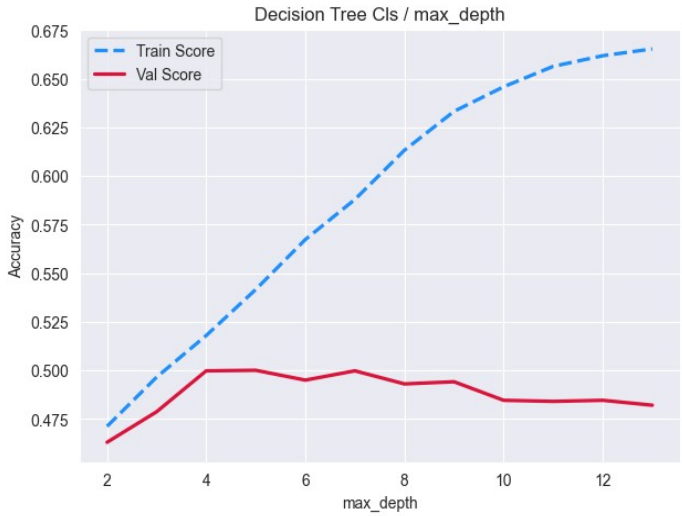


La regressione logistica ottiene già un ottimo risultato in termini di accuracy (più del 50%), considerando che si tratta di un problema di classificazione a tre classi, nel quale la baseline di riferimento è un classificatore random agnostico, che otterrebbe un'accuracy del 33.3%.

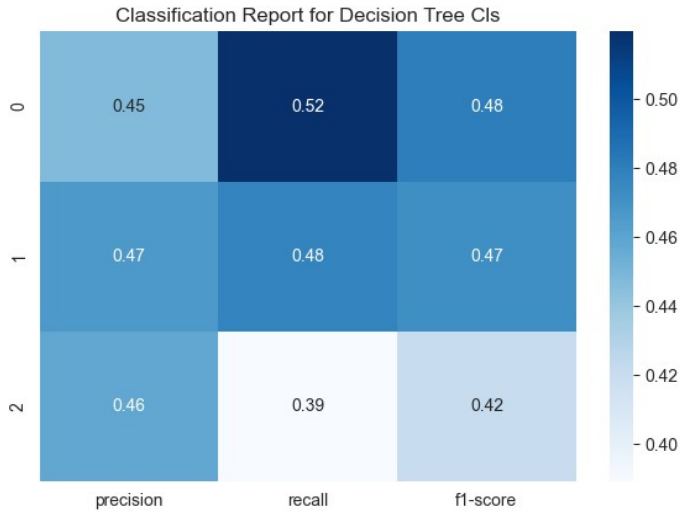
Decision Tree classifier.

Sono stati scelti `Criterion: "gini", Max depth: 5, Min samples leaf: 12, Min samples split: 2;` ottenendo i seguenti risultati.

Notiamo come all'aumentare dell'altezza dell'albero (quindi della sua complessità) vi sia un prevedibile overfitting sul training set. Ciò è riscontrabile soprattutto con l'aumento dell'errore sul validation una volta superata un'altezza di 5.



`Accuracy: 45.68%`

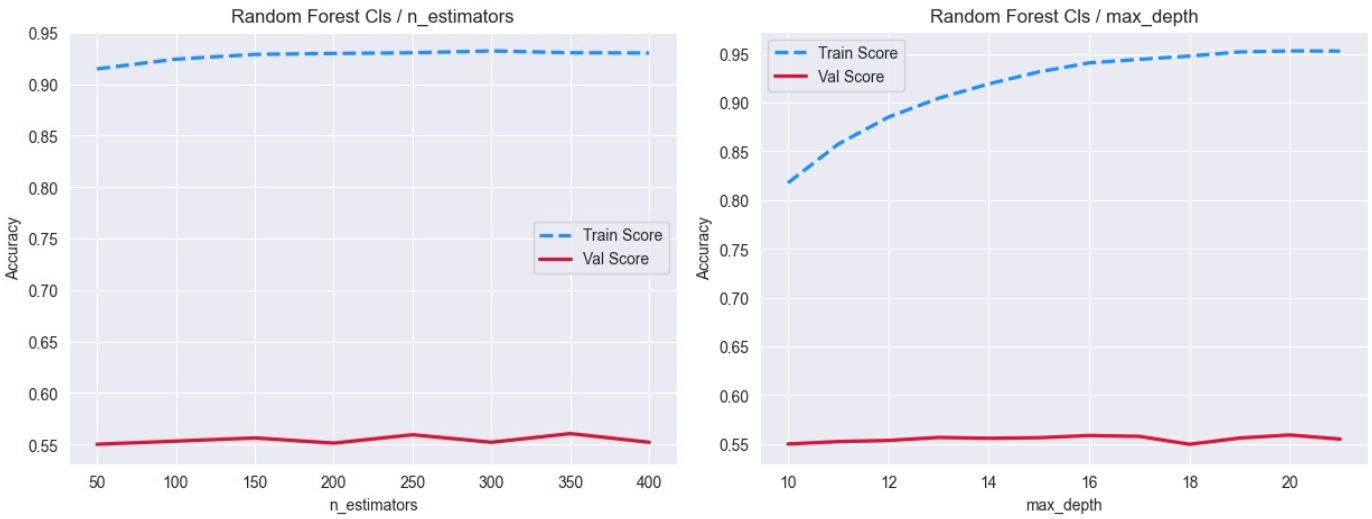


L'albero decisionale esibisce, rispetto alla regressione logistica, un peggioramento rispetto alla regressione logistica (similmente a quanto si è visto per la regressione) riscontrabile sia nell'accuracy che in precision e recall.

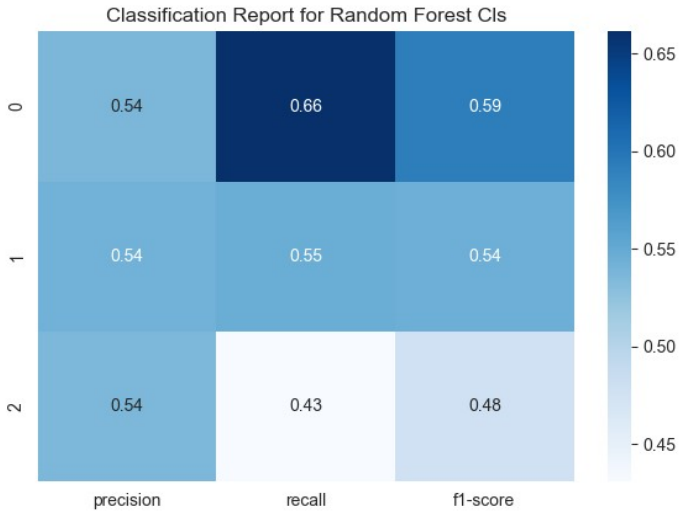
Random Forest classifier.

Sono stati scelti `Criterion: "entropy", Max depth: 20, Min samples leaf: 2, Min samples split: 10, Num estimators: 350;` ottenendo i seguenti risultati.

Notiamo come, in questo caso, non vi sia particolare overfitting riscontrabile, in quanto le performance sul validation rimangono pressoché stabili al crescere della complessità.



`Accuracy: 53.83%`

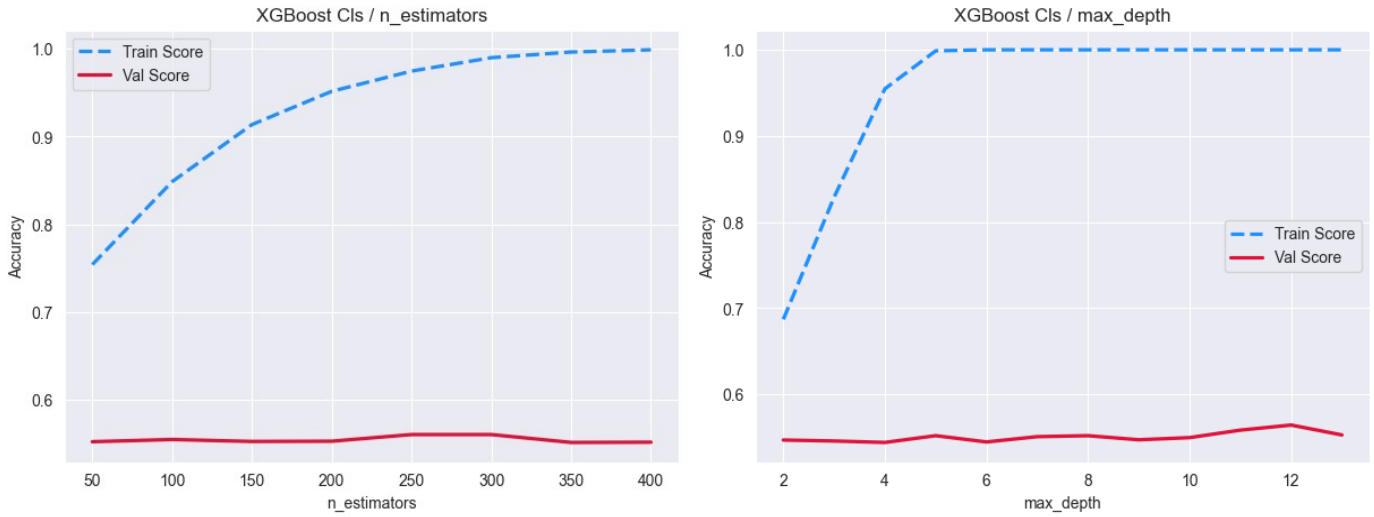


Con la Random Forest otteniamo un'accuracy che supera anche il 50% arrivando a quasi il 54% di istanze classificate correttamente (+20 sulla baseline). Vediamo inoltre come il modello si comporti bene in termini di precision (54%), un po' meno sul richiamo.

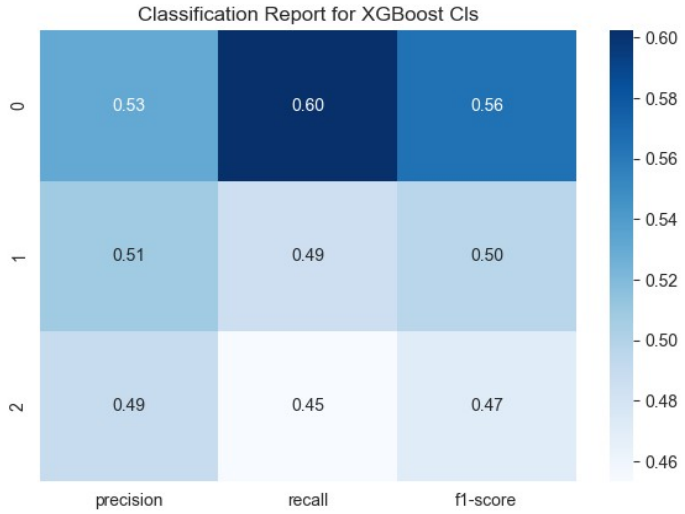
XGBoost classifier.

Sono stati scelti `Learning rate: 0.1, Max depth: 12, Min child weight: 1, Num estimators: 250, Subsample: 0.6;` ottenendo i seguenti risultati.

Notiamo che anche nel caso del boosting non si ha un overfitting evidente: le performance sul validation set rimangono infatti stabili con il crescere della complessità del modello.



`Accuracy: 50.99%`



Similmente a quanto riscontrato per la regressione, il modello basato sul boosting ha performance peggiori della Random Forest, e simili al modello lineare.

Concludiamo constatando che, anche in questo caso, il modello che meglio si è adattato ai dati e ha mostrato le migliori *capacità di generalizzazione* (e minimo overfitting) si è rivelato essere la Random Forest (ensemble di alberi decisionali).

Nel capitolo successivo le proprietà di *spiegabilità* della Random Forest (derivante dall'essere basata su alberi) saranno sfruttate per comprendere quali siano state le *feature con più alto potere predittivo* sia per il task di regressione che per quello di classificazione e per fare quindi *feature selection.*

Feature selection e sfida con modelli neurali.

Dopo l'addestramento dei modelli per entrambi i task presentati si è voluto meglio indagare quali fossero state le variabili del dominio considerato che più hanno influenzato le predizioni, per quanto riguarda sia il voto IMDb del film che i suoi risultati finanziari.

L'obiettivo di questa fase è stato individuare le k feature più rilevanti per la predizione con modelli "shallow" ed utilizzarle per mettere alla prova un modello "deep": nello specifico *addestrare una rete neurale MLP (Multi Layer Perceptron)* e osservare se ed eventualmente di quanto le performance di quest'ultima sono migliori rispetto al modello "shallow". Per lo scopo è stato scelto $k = 15$, ovvero proprio la metà delle feature, considerando le variabili categoriche codificate con one-hot encoding.

Questa fase è articolata come segue:

- una prima fase di studio della rilevanza delle 30 feature di partenza in due modi:
  - con la visualizzazione in un grafico della *mutual information* calcolata tra ciascuna variabile e la feature target del task
  - con la visualizzazione, sempre tramite un grafico, della *feature importance* che il miglior modello "shallow" ha assegnato alle feature.
- una seconda fase che, in continuità con la precedente, si pone di *quantificare l'efficacia delle feature* nel migliorare le predizioni del modello. Nello specifico viene eseguita una *forward selection* sulle feature: si è iterato, fino a scegliere k feature, scegliendo ogni volta la variabile che portasse il maggiore miglioramento secondo una certa metrica (diversa a seconda del task).
  Per ogni iterazione si è ottenuta quindi la feature che, aggiunta alle feature già selezionate, più migliorasse le predizioni del modello. Per ogni nuovo sottoinsieme di feature il modello è stato inoltre riaddestrato e valutato con 5-fold cross-validation.
- una terza e ultima fase in cui, dopo l'analisi del risultato delle fasi precedenti, si è deciso quali delle k feature impiegare per l'addestramento del modello MLP. A questo punto, dopo il training della rete neurale si è osservato se e quanto questa fosse riuscita a migliorare i risultati ottenuti precedentemente.

Per quanto riguarda l'*architettura della rete neurale* si è optato per un'architettura a 3 hidden layer attivati con ReLU, ciascuno seguito da un layer di dropout (al 20%) per regolarizzare meglio i pesi. La dimensione degli hidden layer è stata scelta secondo l'euristica di raddoppiare il numero $k = 15$ di feature in input e poi decrescere scegliendo sempre una potenza di 2: le dimensioni scelte sono state quindi 32, 16, 8.

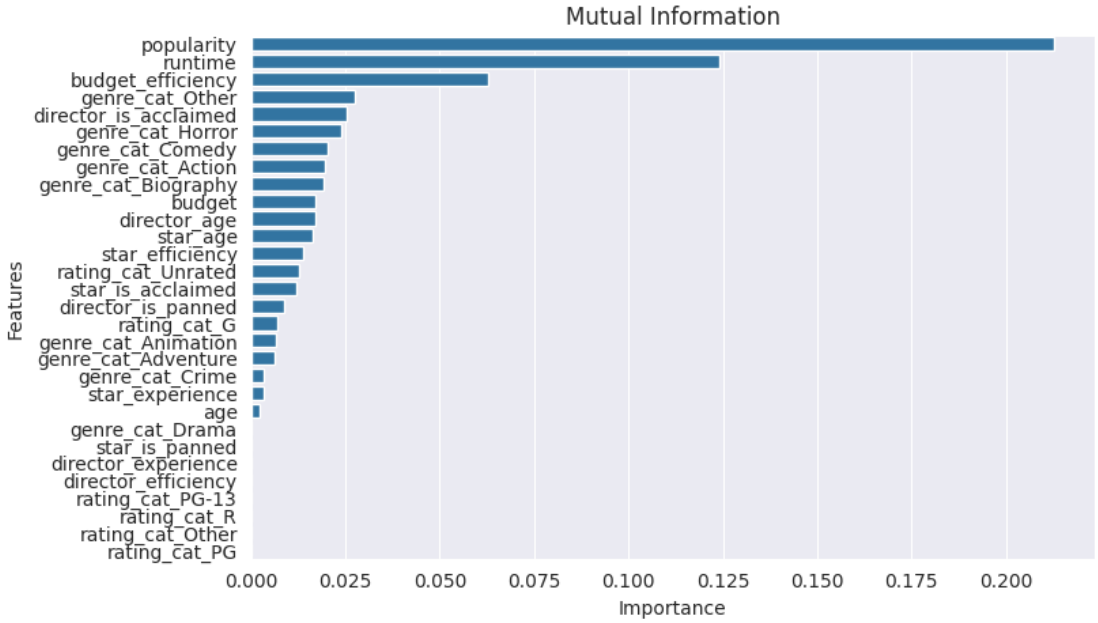
L'output layer è stato ovviamente pensato diversamente distinguendo i due task: per il task di regressione l'output layer è dato da un solo neurone non attivato, mentre per il task di classificazione si sono invece previsti tanti neuroni quante sono le classi, attivati con la funzione Softmax per ottenere una distribuzione di probabilità sulle classi.
In entrambi i casi si è utilizzato un ottimizzatore classico basato su SGD (Stochastic Gradient Descent) con learning rate di 0.01 e weight decay, combinato con una loss MSE e cross-entropy rispettivamente per il task di regressione e classificazione.

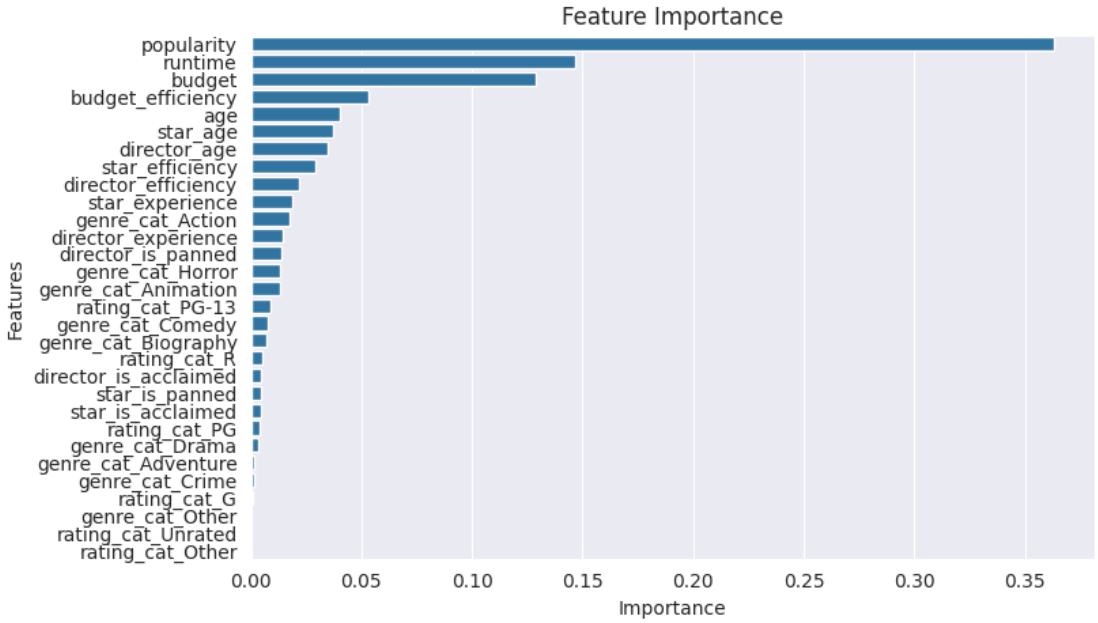
Fare tuning degli iper-parametri per un MLP (numero e dimensione degli hidden layer, numero di epoche di training), anche se relativamente piccolo, può essere molto oneroso in termini computazionali; si è scelto perciò di non eseguire un vero e proprio tuning, in quanto per l'architettura sono state seguite le best practice consigliate nel caso di dataset tabulari (2-3 layer con dimensioni non esagerate) e per il numero di epoche si è invece utilizzata la tecnica dell'early stopping, in modo da far sì che il training si fermasse automaticamente quando non ci fosse più stato miglioramento delle performance.

Di seguito sono riportati i risultati ottenuti ed eventuali considerazioni.

Feature selection e MLP training per il task di regressione.

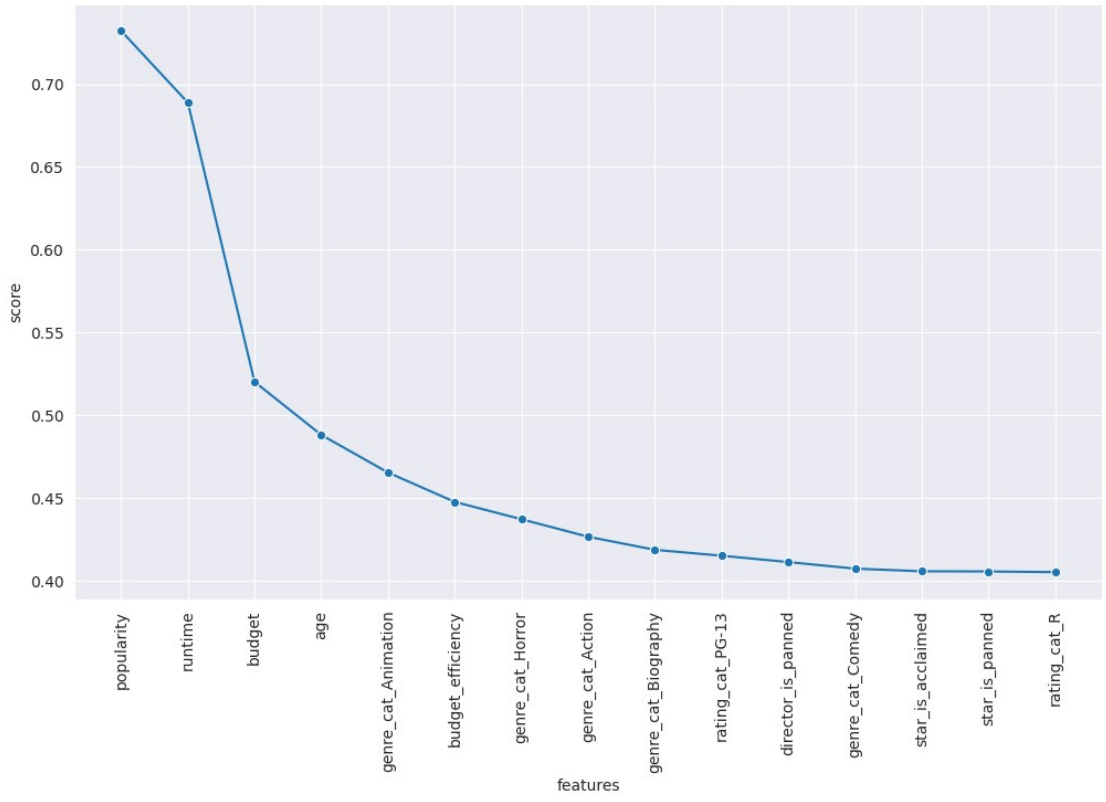
Mostriamo di seguito i grafici che visualizzano mutual information e feature importance.





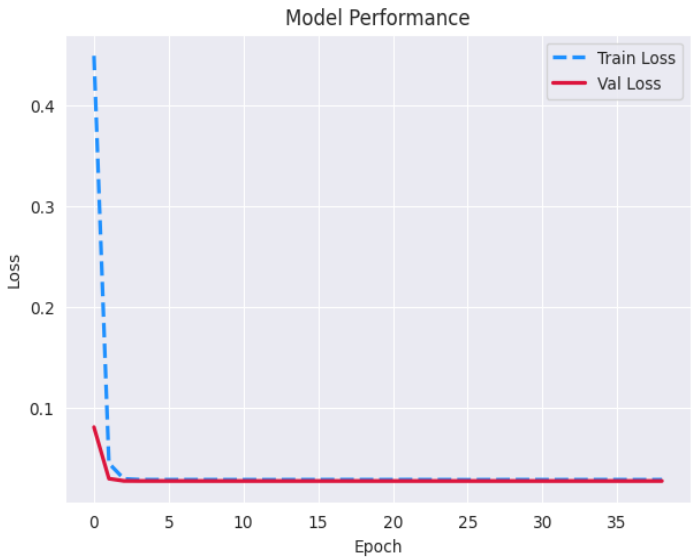
Notiamo come la feature più importante in entrambi i casi è stata quella relativa alla popolarità del film, seguita dal runtime, evidenziando quindi un possibile legame tra popolarità e durata del film e il voto del pubblico. Tra le prime posizioni troviamo anche alcune feature derivate: la budget efficiency e diverse tra le feature che aggregano lo storico di regista e star principale del film, dimostrando quindi il potere predittivo di queste ultime.

Di seguito invece è mostrato il grafico che riassume l'andamento della forward selection.



Abbiamo conferma dei risultati precedenti: notiamo infatti come solo con la feature sulla popolarità il modello raggiunga un errore inferiore a 1, di circa 0.75, e poi decresca fino a 0.52 aggiungendo solo runtime e budget al sottoinsieme. Vediamo inoltre come alcuni generi e rating, insieme alle feature che indicano se un regista/attore è acclamato e/o stroncato, abbiano comunque una loro importanza predittiva, che conduce ad un errore di circa 0.4.

Vediamo in ultimo i risultati di training della rete neurale sul sottoinsieme di feature scelto. Le feature scelte sono state:
`[`popularity`,`runtime`,`budget`,`age`,`budget_efficiency`,`genre_cat_Horror`, `genre_cat_Animation`,`genre_cat_Action`,`rating_cat_PG-13`,`director_age`, `star_age`,`director_experience`,`star_experience`,`director_is_panned`, `star_is_panned`]`.

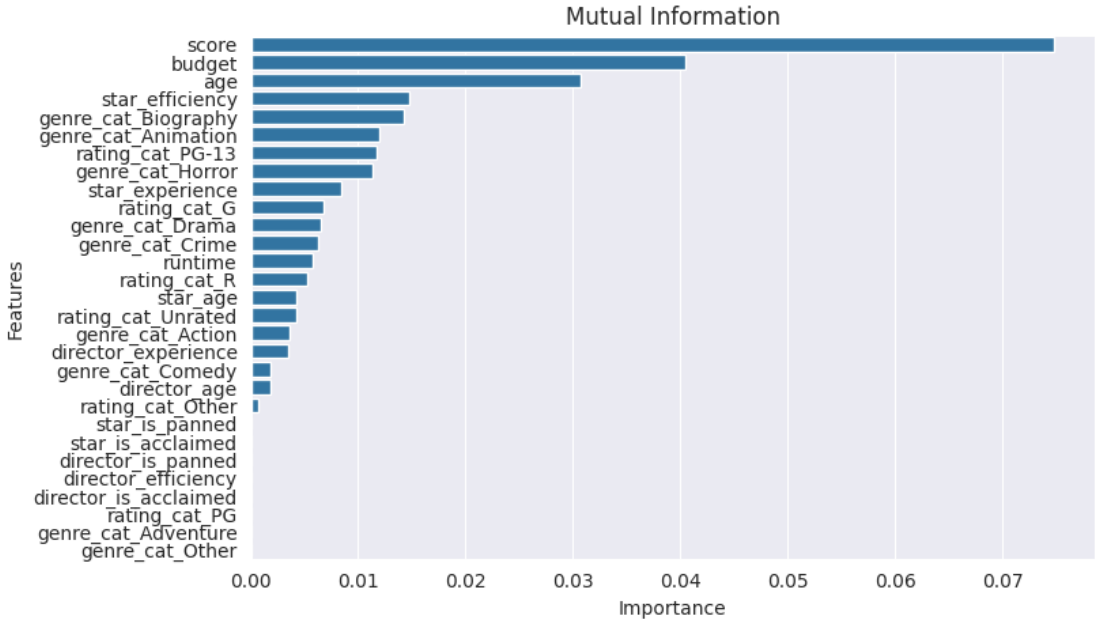


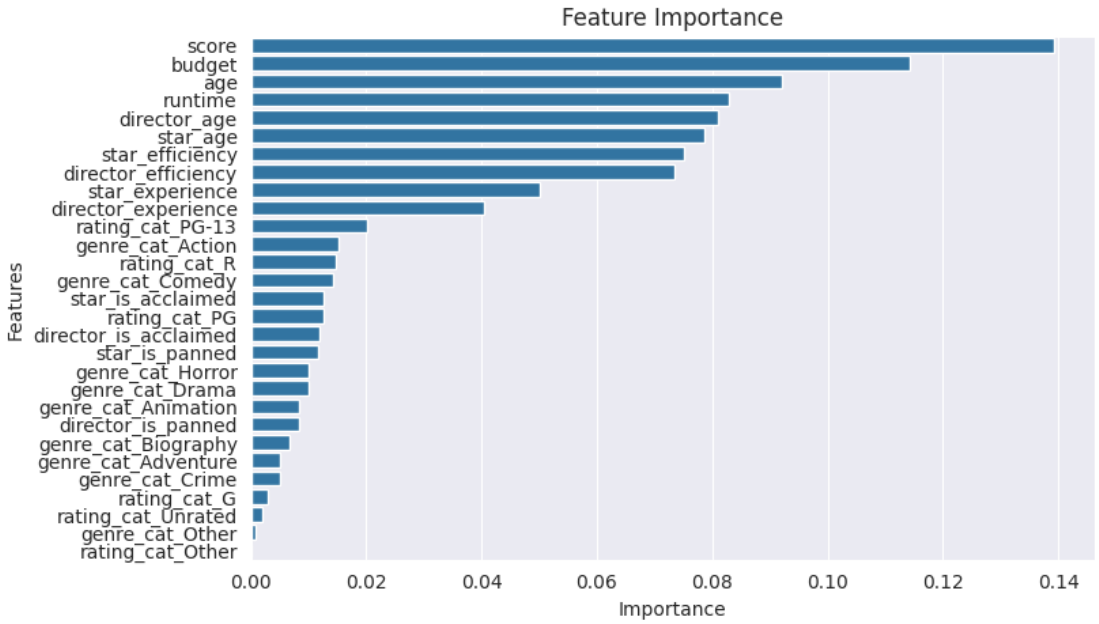
MSE: 0.0333

La rete ha subito raggiunto la convergenza dopo 3-4 epoche ottenendo un errore inferiore a 0.1, pari a circa 0.03, battendo di misura la Random Forest che aveva ottenuto 0.43.

Feature selection e MLP training per il task di classificazione.

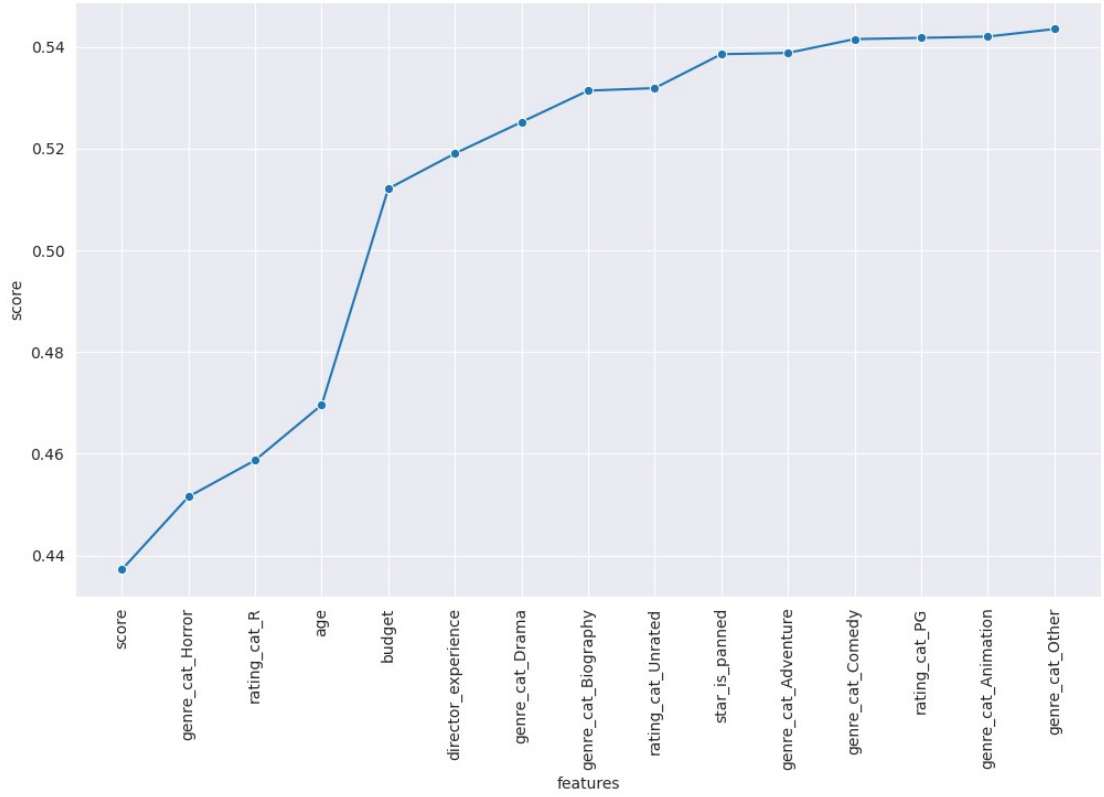
Mostriamo di seguito i grafici che visualizzano mutual information e feature importance.





Notiamo come la feature più importante in entrambi i casi è stata quella relativa al voto IMDb del film, seguita da budget e age, evidenziando come dalla qualità del film possa quindi dipendere il suo successo commerciale. Tra le prime posizioni troviamo anche alcune feature derivate: in particolare quelle relative allo storico di efficiency di regista e star principale.

Di seguito invece è mostrato il grafico che riassume l'andamento della forward selection.



Abbiamo conferma dei risultati precedenti: notiamo infatti come solo con la feature sullo score IMDb il modello raggiunga un'accuracy di quasi il 45%, e poi cresca fino al 51% aggiungendo le feature di genere Horror, rating R e budget al sottoinsieme. Nella top 15 vediamo anche diversi altri generi e alcune feature derivate.

Vediamo in ultimo i risultati di training della rete neurale sul sottoinsieme di feature scelto. Le feature scelte sono state:
`[`score`,`runtime`,`budget`,`age`,`director_efficiency`,`star_efficiency`, `director_experience`,`star_experience`,`director_age`,`star_age`, `genre_cat_Comedy`,`genre_cat_Drama`,`genre_cat_Horror`,`rating_cat_PG-13`, `rating_cat_R`]`.

Accuracy: 47.78%

In questo caso la rete non ha propriamente raggiunto la convergenza, mostrando un comportamento meno stabile durante il training; ha infine raggiunto un'accuracy del 47%, inferiore però al 54% ottenuto dalla Random Forest.

In conclusione possiamo dire di aver scoperto qualcosa in più su quali siano le variabili del dominio cinematografico che hanno maggiore rilevanza, quindi maggiore *potere predittivo*, nel prevedere l'esito di una produzione, sia in termini artistici che finanziari.
Si è inoltre messo alla prova un modello più complesso e potente, quale è il Multi Layer Perceptron, osservando come, con meno feature in input sia riuscito a ottenere dei risultati di regressione migliori rispetto ad un modello che ha avuto a disposizione tutte le feature ed è stato sottoposto a tuning specifico.

Nel successivo e ultimo capitolo si analizzeranno le variabili del dominio con un approccio diverso dai precedenti: si adotterà infatti un *approccio bayesiano*.

Analisi (e apprendimento) bayesiani.

Come ultima fase del progetto si è optato per esplorare ed analizzare il dominio di studio utilizzando un *approccio bayesiano* al problema. Nello specifico si sono utilizzati il classificatore Naive Bayes (`MultinomialNB` di `scikit-learn`) per tentare di ottenere migliori risultati nel task di classificazione, e si sono poi impiegate le Bayesian Net (tramite la libreria python `pgmpy`) per studiare la *dipendenza causale tra le variabili* e comprendere ancora meglio come queste influenzino in particolare gli incassi al botteghino di un film, nella forma dell'efficienza di budget.

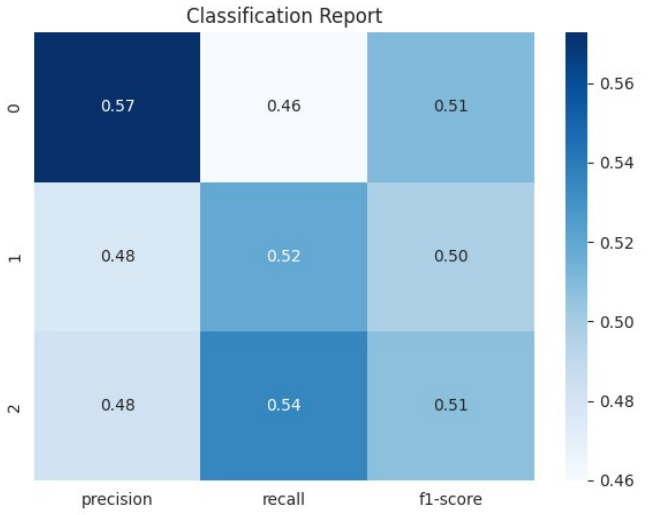
Risultati del Naive Bayes.

Per l'addestramento del Naive Bayes, che lavora meglio con variabili discrete, è stato previsto un dataset specifico: `movies_features_nb.csv`, il quale contiene le stesse feature di `movies_features_cls.csv` con la differenza che sono state tutte derivate come feature categoriche. L'apprendimento delle probabilità a posteriori del modello è effettivamente ottenuto sul dataset contenente i film usciti negli ultimi 30 anni per gli stessi motivi citati nei precedenti capitoli.

Si mostrano di seguito i risultati di accuracy del Naive Bayes, con e senza resampling, insieme alla matrice di confusione.

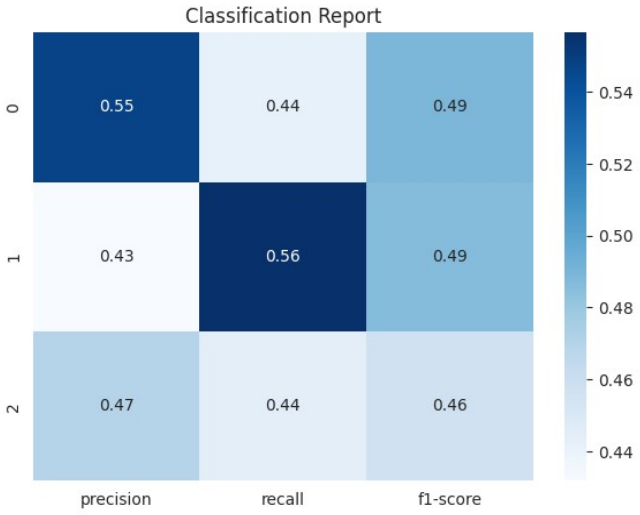
Senza resampling

`Accuracy: 50.49%`



Con resampling

`Accuracy: 47.65%`

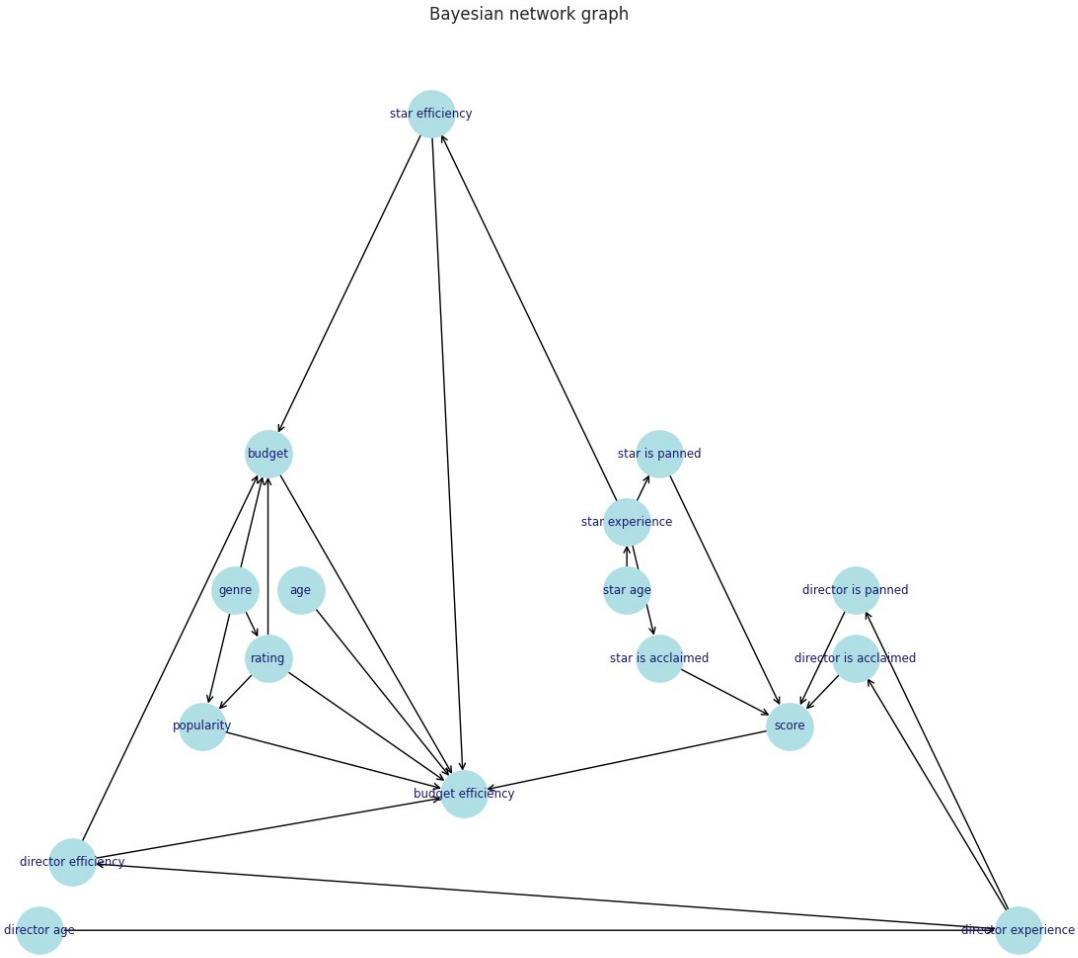


Vediamo come i risultati, nel caso del Naive Bayes, mostrino una migliore performance da parte del modello addestrato sul dataset senza resampling delle classi minoritarie, e, in generale però, notiamo come questo non raggiunga l'accuracy ottenuta dalla Random Forest, fermandosi a poco più del 50%, con risultati abbastanza omogenei in termini di precisione, richiamo e F1-score (al contrario del modello addestrato sul dataset con resampling).

Costruzione e apprendimento delle Bayesian Net.

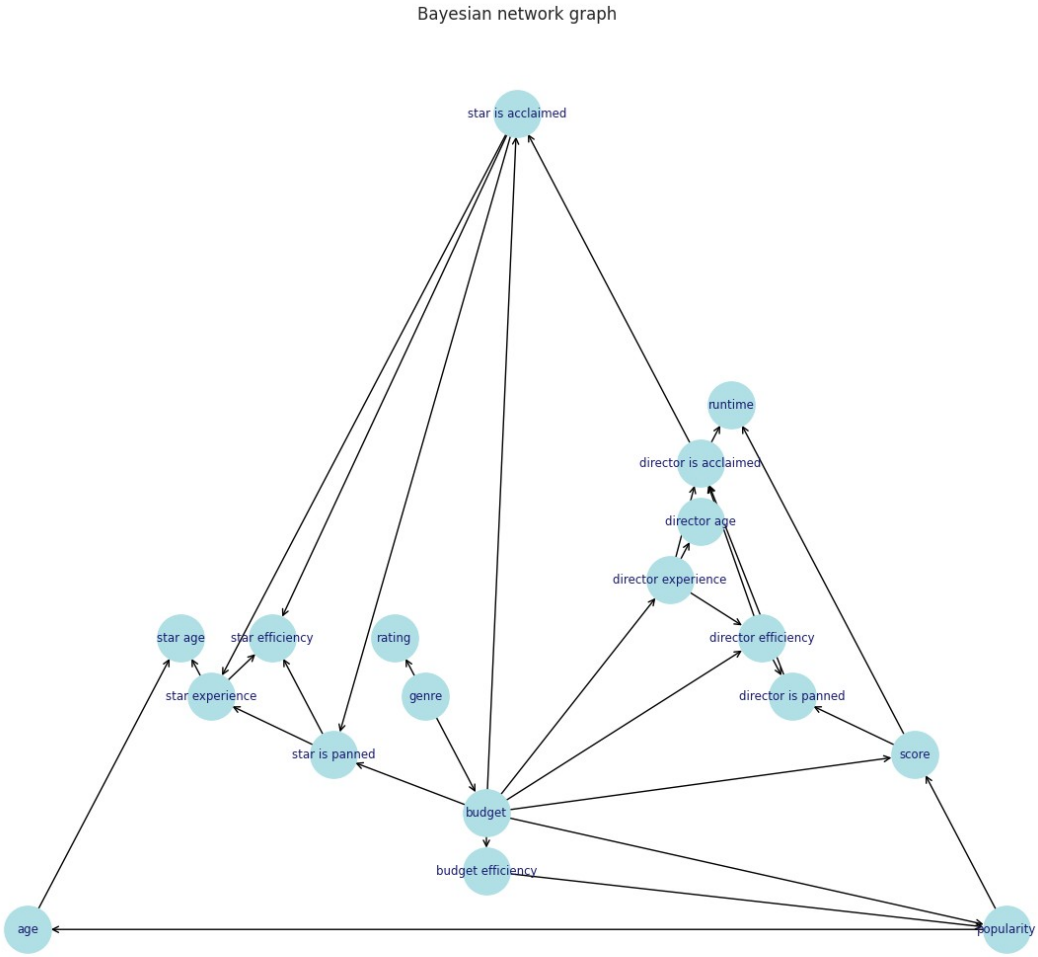
Infine si è voluto indagare meglio la relazione di causalità tra le variabili del dominio e come da queste dipendesse il successo commerciale del film attraverso le *Bayesian Net*. Allo scopo è stato impiegato lo stesso dataset con feature discrete menzionato in precedenza.

Nello specifico si è proceduto in prima istanza costruendo una rete bayesiana specificandone la struttura tramite ipotesi sulla dipendenza tra le variabili in esame. La rete risultante è presentata di seguito come grafo.



Nel grafo è possibile osservare le relazioni di dipendenza supposte tra le variabili.

Di seguito si è voluto invece anche provare ad utilizzare i dati per guidare l'apprendimento della *struttura della rete*, ovvero delle relazione di dipendenza causale tra le variabili, così da confrontarla con la struttura ipotizzata nella fase precedente.
L'apprendimento della rete è stato eseguito tramite `HillClimbing` (con massimizzazione del *BIC score*), mentre le CPT (le tabelle contenenti le distribuzioni di probabilità condizionata della rete) sono state apprese usando uno stimatore di massima verosimiglianza. La struttura della rete appresa dai dati è mostrata di seguito.



Notiamo come alcune relazioni di causalità ipotizzate sono state inferite anche dai dati: la dipendenza del rating dal genere, la dipendenza della budget efficiency dal budget, e la dipendenza dell'efficienza di budget di un regista/attore dalla sua esperienza.

Vediamo però anche come dai dati siano state inferite *differenti relazioni tra le variabili* rispetto a quelle derivanti dalle ipotesi precedenti: ad esempio la dipendenza causale tra popolarità (e budget) di un film e il suo voto IMDb, quella tra score e durata del film, o ancora quella tra budget e popolarità raggiunta dal film.

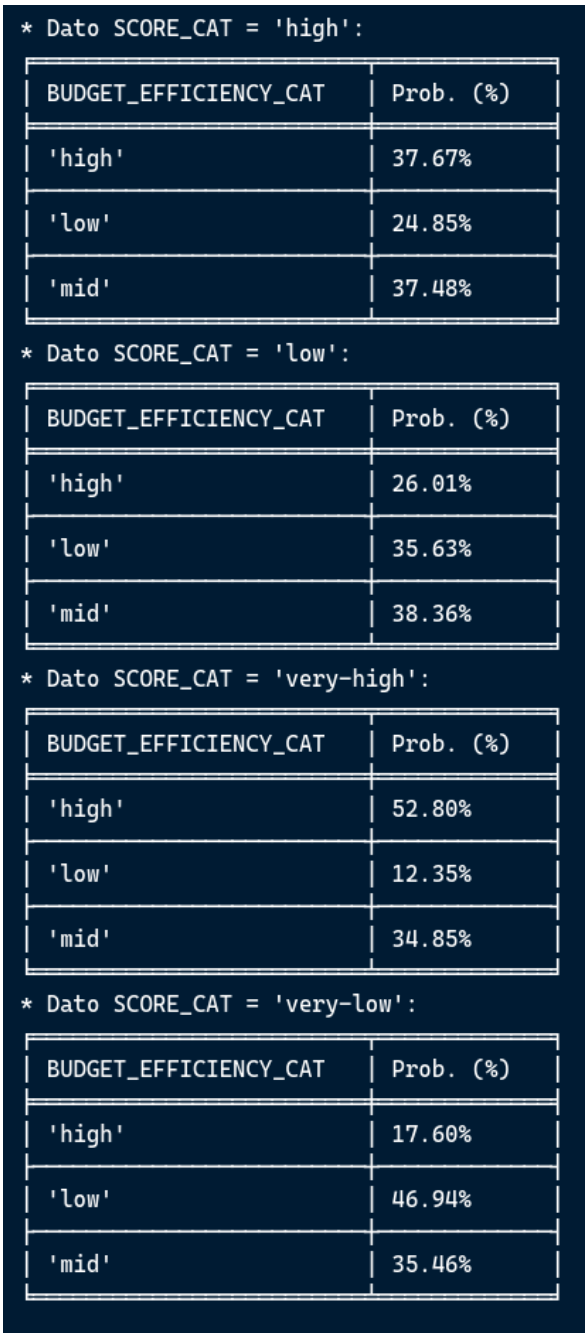
In conclusione possiamo vedere come l'apprendimento della struttura della rete a partire dati ha permesso di scoprire altre relazioni di causalità tra le feature prima non ipotizzate, ma assolutamente *coerenti con il dominio di studio*, ed ha anche permesso di confermare diverse ipotesi fatte in prima istanza.

Query sulla Bayesian Net.

Si è anche sfruttata la rete bayesiana interamente appresa dai dati per eseguire alcune *query per calcolare la probabilità condizionata della variabile target* `budget_efficiency_cat` dati i valori delle distribuzioni discrete di alcune delle feature: sono state calcolate quindi le probabilità a posteriori di `budget_efficiency_cat` stanti evidenze delle variabili `genre_cat`, `rating_cat`, `runtime_cat`, `score_cat`.

Sono riportate di seguito solo le i risultati rispetto alle ultime due, per brevità.

```
* Dato RUNTIME_CAT = 'long':
╒═══════════════════════╤═══════════╕
│ BUDGET_EFFICIENCY_CAT │ Prob. (%) │
╞═══════════════════════╪═══════════╡
│ 'high'                │ 39.43%    │
├───────────────────────┼───────────┤
│ 'low'                 │ 23.02%    │
├───────────────────────┼───────────┤
│ 'mid'                 │ 37.54%    │
╘═══════════════════════╧═══════════╛
* Dato RUNTIME_CAT = 'medium':
╒═══════════════════════╤═══════════╕
│ BUDGET_EFFICIENCY_CAT │ Prob. (%) │
╞═══════════════════════╪═══════════╡
│ 'high'                │ 32.32%    │
├───────────────────────┼───────────┤
│ 'low'                 │ 30.68%    │
├───────────────────────┼───────────┤
│ 'mid'                 │ 36.99%    │
╘═══════════════════════╧═══════════╛
* Dato RUNTIME_CAT = 'short':
╒═══════════════════════╤═══════════╕
│ BUDGET_EFFICIENCY_CAT │ Prob. (%) │
╞═══════════════════════╪═══════════╡
│ 'high'                │ 28.65%    │
├───────────────────────┼───────────┤
│ 'low'                 │ 35.06%    │
├───────────────────────┼───────────┤
│ 'mid'                 │ 36.29%    │
╘═══════════════════════╧═══════════╛
```



Vediamo come le distribuzioni a posteriori dell'efficienza di budget mostrino come la probabilità di avere un'efficienza finanziaria medio-alta aumentino per film di lunga durata e siano invece più basse per film più brevi.

Constatiamo inoltre in maniere abbastanza chiara che, come intuito nelle fasi precedenti, la qualità di un film (espressa dal suo score) incida in maniera importante sul suo successo al botteghino rispetto al budget investito: si nota infatti come la probabilità di avere un'alta efficienza, dato un voto molto alto, supera addirittura il 50% (52.8%), e di contro, la probabilità di avere un'efficienza tale da non coprire nemmeno il budget investito, a fronte di un voto molto basso, è pari a quasi il 47%.

Conclusioni

In conclusione possiamo dire di aver raggiunto una discreta comprensione del dominio dell'industria cinematografica, utilizzando diversi strumenti a disposizione dell'ingegnere della conoscenza: costruzione di una KB, ragionamento automatico, modelli di machine-learning e deep-learning, e inferenza bayesiana.

Il lavoro svolto potrebbe essere però migliorato in diversi modi: raccogliendo ad esempio nuovi dati in maggiore quantità (non solo sui film, ma anche sulle case di produzione), ed estendendo (modificandola eventualmente) la knowledge-base.
Potrebbe essere utile quindi sperimentare anche altri modelli per migliorare le performance di predizione, in special modo sul task di classificazione (risultato più ostico della regressione), e in ultimo, sfruttare le reti bayesiane per indagare l'entità della dipendenza tra altre variabili del dominio, magari integrando gli eventuali dati raccolti sulle compagnie di produzione.

Riferimenti Bibliografici

[1] *Artificial Intelligence: Foundation of Computational Agents*, Poole & Mackworth, 2023