Dipartimento di Informatica

Corso di laurea in Informatica

Caso di Studio per l'esame di Ingegneria della Conoscenza

Movie Income Predictor

Progetto di:
Giuseppe Tedone (758476) g.tedone14@studenti.uniba.it

Link Repository:
https://github.com/giuseppetedone02/MovieIncomePredictor

Anno Accademico **2023-2024**

Indice

1 Introduzione

Con la digitalizzazione e la nascita e sviluppo dei social media, l'industria cinematografica è stata protagonista di una notevole crescita, con oltre 1000 film prodotti ogni anno. Tuttavia, solo una piccola percentuale di questi film viene considerata un successo: sono, infatti, numerosi i fattori che é necessario considerare per poter consentire al proprio film di ottenere il successo sperato, spaziando da fattori *interni*, ovvero relativi al film prodotto o che si intende produrre, a fattori *esterni*, ovvero relativi alle caratteristiche dei possibili film competitor (ad es., periodo di rilascio del film, strategie di marketing adottate, ...). É possibile, quindi, comprendere la capacitá di prevedere in modo affidabile il possibile incasso di un film risulti un compito cruciale, poiché utile per ridurre i rischi e massimizzare i rendimenti.

Il caso di studio in questione si pone l'obiettivo di realizzare un modello che vada a prevedere l'incasso di un film integrando fattori classici, come anno di rilascio, rating, cast e regia, con fattori sociali, come le reazioni condivise dal pubblico online. Per fare ció, si fará uso di tecniche di **apprendimento supervisionato, apprendimento non supervisionato, apprendimento probabilistico** e **programmazione logica**, che rappresentano strumenti utili per ottimizzare le previsioni, in modo tale da aiutare gli stakeholder del settore a prendere decisioni piú attente riguardo alla distribuzione e al marketing, aumentando cosí la probabilitá di successo di un film.

1.1 Elenco argomenti di interesse

1. **Programmazione Logica**: É stato utilizzato il linguaggio **Prolog** per la definizione della Knowledge Base del modello sottoforma di assiomi (regole e fatti). I paragrafi del libro di riferimento[4] sono i seguenti: Rappresentazione e ragionamento proposizionale (Capitolo 5 - Proposizioni ed inferenze) e relazionale (Capitolo 15 - Individui e Relazioni).

2. **Apprendimento supervisionato**: Si é fatto uso di alcuni dei modelli base dell'apprendimento supervisionato al fine di addestrarli e valutarli sugli esempi , in modo tale da individuare il modello maggiormente performante. I paragrafi del libro di riferimento[4] sono i seguenti: Fondamenti di Apprendimento Supervisionato, Valutare le Predizioni, Modelli Base per l'Apprendimento Supervisionato (Alberi di Decisione), Overiftting, Cross Validation, Modelli Compositi (Bagging con Random Forest, Boosting con Gradient-Boosted Trees) (Capitolo 7 - Apprendimento supervisionato)

3. **Apprendimento non supervisionato**: Sono state utilizzate tecniche di **hard clustering** per la suddivisione degli esempi in vari cluster. I paragrafi del libro di riferimento[4] sono i seguenti: Apprendimento non Supervisionato, k-Means (Capitolo 10 - Apprendimento e Incertezza)

4. **Apprendimento probabilistico**: É stata costruita ed appresa una Rete Bayesiana sulla base del dataset di riferimento, utilizzandola in seguito per la generazione di esempi sintetici e l'inferenza probabilistica. I paragrafi del libro di riferimento[4] sono i seguenti: Ragionamento sotto Incertezza (Capitolo 9), Apprendimento sotto Incertezza (Capitolo 10)

5. **Tecniche di Over/Under-sampling**: Sono state applicate tecniche di Over-sampling e Under-sampling per equilibrare e bilanciare la classe target, generando o rimuovendo esempi al fine di migliorare le performance dei modelli utilizzati. I paragrafi del libro di riferimento[3] sono i seguenti: Over-sampling (Naive random over-sampling, From random over-sampling to SMOTE and ADASYN). I paragrafi della documentazione di riferimento [7] sono i seguenti: Over-sampling Techniques (Random Over-sampling, SMOTE), Under-sampling Techniques (Random Under-sampling)

2 Creazione del Dataset

Prima di poter eseguire i vari esperimenti di apprendimento automatico, é necessario costruire un dataset che sia adatto per il nostro caso di studio. Fortunatamente, per il nostro scopo sono presenti numerosi dataset, contenenti informazioni ben strutturate sui vari film rilasciati negli ultimi decenni. Questi dataset sono facilmente reperibili consultando piattaforme online, tra cui é possibile citare *Kaggle*, oppure consultando siti ufficiali, quali *IMDb* o *Box Office Mojo*, al cui interno vi sono delle sezioni apposite che consentono agli utenti di scaricare file contenenti le principali informazioni sui film da loro memorizzati. Una volta ottenuti, tali dataset dovranno essere analizzati, al fine di ripulirli da dati inutilizzabili e consentire cosí alla costruzione di un dataset solido e affidabile.

2.1 Dataset utilizzati

Nonostante la semplicitá con cui é possibile reperire i dataset riguardanti il problema in esame, la maggior parte di essi sono stati scartati per una delle seguenti motivazioni:

- **I dati contenuti sono riguardanti serie TV**: tali dataset sono stati scartati in quanto, oltre a non possedere un numero adeguato di righe, troppo inferiore rispetto a quanto sperato, risultavano in minor quantità rispetto ai dataset riguardanti i film. Ció rendeva questi dataset difficilmente integrabili con i dataset richiesti per il caso di sudio.

- **Dati mancanti**: alcuni dataset possedevano dati mancanti per molte delle colonne utilizzate nel dataset finale. Si tratta quindi di dataset dapprima considerati per il caso di studio, ma che solo successivamente si é deciso di scartare per tale motivazione.

- **Dati ridondanti**: alcuni dei dataset scartati contenevano dati giá presenti in dataset precedentemente considerati per la costruzione del dataset finale.

In seguito all'analisi dei dataset proposti dalle varie piattaforme, per la costruzione del dataset finale si é deciso di utilizzare i seguenti dataset:

- **boxofficemojo dataset**: dataset disponibile su Kaggle contenente dati su 3243 film ottenuti consultando il sito web di **Box Office Mojo**, rilasciati in un periodo di vent'anni che va dal 2000 all'Aprile del 2020. Le colonne presenti descrivono ogni film sulla base di caratteristiche quali titolo, anno di rilascio, regista, attori e informazioni sul budget e sull'incasso, quest'ultimo suddiviso in *domestic*, *international* e *worldwide*.

- **movie-stats**: dataset preso da un progetto di GitHub, ma comunque disponibile su Kaggle, il cui obiettivo é quello di analizare dati provenienti da film rilasciati

nel periodo 1986-2016, al fine di comprendere l'andamento generale dell'ambiente cinematografico. Il dataset in questione contiene dati su 6820 film recuperati consultando il sito web di **IMDb** ed ognuno di essi é descritto da caratteristiche simili a quanto descritto nel dataset precedente, con l'aggiunta dei **voti** e degli **score** forniti dagli utenti del sito.

Il dataset finale, denominato `Movie_dataset.csv` all'interno del progetto, é stato quindi costruito unendo i dati ottenuti dai due dataset precedentemente descritti, ottenendo un totale di **8121 esempi** utili per l'apprendimento da parte dei vari modelli.

2.2 Pre-processing dei dati

Una prima fase di pre-processing é stata eseguita rimuovendo quelle colonne che non risultavano funzionali in quanto non aggiungevano informazione significativa all'esempio durante il task di apprendimento. Le feature rimosse dal dataset costruito sono:

- **Movie ID**: il codice identificativo di un film. É subito possibile comprendere come tale dato non aggiunga alcuna informazione utile per l'apprendimento;
- **Plot**: la feature non risultava utile ai fini dell'apprendimento da parte dei modelli;
- **Producer, Cinematographer, Composer**: nonostante la loro significatività per il problema in esame, si é deciso di eliminare queste feature a causa della mancanza di troppe corrispondenze all'interno del dataset;
- **Released**: rappresentata in formato `yyyy-mm-dd`, é una feature rimossa in favore della feature **Year**. É stata rimossa anche per ridurre la complessità del dataset finale.

In seguito alla selezione delle feature utili per il caso di studio, sono state apportate ulteriori modifiche al dataset per risolvere il problema della **mancanza** delle informazioni per alcuni film. Nello specifico, sono stati rimossi gli esempi che

- Non possiedono informazioni sul budget: il budget é una feature fondamentale per il problema esaminato, pertanto la sua assenza comporta la rimozione dell'esempio interessato;
- Non possiedono informazioni sul rating MPAA: come per il budget, anche il rating MPAA é significativo per i task presentati nel progetto. Per questo motivo, é stato necessario rimuovere tutti gli esempi che non avessero tale informazione;
- Non possiedono informazioni sull'incasso: essendo la nostra feature target, é necessario che l'incasso sia disponibile per ogni esempio.

Al termine di questo filtraggio, il numero di film disponibili sono scesi da **8121 a 6042 caricati correttamente**. Si tratta di una perdita significativa (2079 esempi), ma necessaria per il progetto, poichè senza le informazioni sull'incasso non sarebbe stato possibile procedere con un addestramento corretto da parte dei modelli; inoltre, risulta una perdita comunque accettabile dato l'elevato numero di esempi ancora utilizzabili.

Infine, durante la fase di pre-processing sono sorti ulteriori problemi relativi alla **ridondanza** ed **ambiguità** dei dati. In particolare, erano presenti alcuni film che non sono stati correttamente uniti durante l'operazione di merge dei dataset per via dei loro nomi: erano, infatti, presenti film il cui titolo era interamente scritto in minuscolo, mentre altri film il cui titolo era interamente scritto in maiuscolo. La correzione di tale errore ha comportato la rimozione di ulteriori righe dal dataset, arrivano ad un totale di **5830 esempi**. Considerando il pre-processing eseguito in precedenza per ovviare alla mancanza di informazioni, e soprattutto considerando il numero elevato di righe rimosse come risultato di tale operazione, il numero di esempi rimossi in seguito a questo filtraggio (212 esempi) é nettamente inferiore, quindi più che tollerabile.

Un'ultima considerazione da fare in merito al dataset finale é legata alle trasformazioni e raffinamenti effettuati sulle sue colonne. Per poter ottenere le features del film, elencate di seguito, é stato necessario rinunciare ad alcune colonne ed eseguire delle trasformazioni per meglio uniformare i dati tra le varie righe. Di seguito sono elencate le scelte effettuate:

1. **Eliminazione più attori principali o generi**: originariamente il dataset prevedeva delle colonne aggiuntive per indicare la presenza di più attori principali del film (rappresentati dalle colonne `main_actor_2, main_actor_3, main_actor_4`) e di più generi (indicati dalle colonne `genre_2, genre_3, genre_4`). Si é deciso di rimuovere tali colonne a causa della mancanza di molte occorreneze tra le varie righe del dataset. La rimozione delle colonne relative ai generi é stata necessaria anche in vista della fase di **feature engineering**, nella quale é stata prevista una determinata rappresentazione per questo dato;

2. **Rappresentazione MPA (o MPAA)**: si é scelta la rappresentazione MPAA per il rating dei film in quanto la maggioranza dei produttori richiede questa valutazione da parte dell'associazione competente, pertanto é riconosciuta come metrica generale di valutazione dei film. L'MPAA, inoltre, rientra anche nel nostro obiettivo: questa valutazione viene utilizzata principalmente dalle famiglie per decidere quali film vedere con i figli o far vedere loro. Pertanto, suddividendo i film in base a questa valutazione, possiamo aiutare il modello ad apprendere regolarità e pattern riguardanti i film di determinate categorie (ad es. i film NC-17, ovvero under 17, avranno un audience minore rispetto ai film per tutte le etá, pertanto l'incasso potrebbe essere inferiore).

   Sulla base di questa scelta, durante l'analisi dei dati raccolti é stato necessario adattarli a tale metrica di valutazione: ció significa tradurre tutte le valutazioni non conformi a questo standard.Un esempio é dato dalla presenza di valutazioni

che rispettavano lo standard ***TV Parental Guidelines system***, sistema di valutazione applicato sempre negli Stati Uniti, che vedeva la rappresentazione dei rating in formati quali *TV-14* o *TV-MA*, corrispondenti rispettivamente a *PG-13* e *R*.

É stato cosí ottenuto il dataset finale, visibile nel progetto nella cartella `dataset` con il nome `Movie_dataset_cleaned.csv`

2.3 Features di un film

Una volta rimosse le feature meno significative ed eseguite tutte le considerazioni necessario, il dataset é in grado di descrivere i vari film utilizzando le seguenti caratteristiche o features (in rosso la feature target):

- **Title**: *stringa* che rappresenta il titolo del film
- **Year**: *intero* rappresentante l'anno di rilascio del film
- **MPAA**: corrisponde al rating attribuito al film, sulla base del ***Motion Picture Association film rating system***, utilizzando negli Stati Uniti per valutare l'idoneità di un film per un determinato pubblico in base al suo contenuto. Corrisponde ad una feature categorica con dominio [*G, PG, PG-13, R, NC-17, Not Rated, Unrated, X*]
- **Runtime**: *intero* che rappresenta la durata totale del film (in minuti)
- **Company**: *stringa* che rappresenta la casa di produzione del film
- **Director**: *stringa* che indica il nome del regista del film
- **Writer**: *stringa* che riporta il nome dello sceneggiatore del film
- **Main Actor**: *stringa* rappresentante l'attore principale del film
- **Budget**: *intero* che rappresenta il budget necessario per produrre il film
- **Worlwide Gross**: *intero* che si riferisce all'incasso totale generato dal film a livello globale, ovvero la somma di tutti i guadagni ottenuti nelle sale cinematografiche di tutto il mondo
- **Genre**: corrisponde al genere principale del film. É una feature categorica con dominio [*Action, Adventure, Animation, Comedy, Crime, Drama, Fantasy, Horror, Mystery, Romance, Sci-Fi, Thriller, Western*]
- **Score**: *intero* che rappresenta il rating, da 1 a 100, attribuito dagli utenti del sito di IMDb per il film in questione
- **Votes**: *intero* rappresentante il numero totale di voti effettuati dagli utenti del sito di IMDb per il film in questione

2.4 Features Engineering dei dati

Sono state sperimentate diverse rappresentazioni delle features, in modo tale da individuare quella più adatta e che meglio rappresentasse le informazioni. La rappresentazione finale prevede la suddivisione del dataset in 81 colonne, corrispondenti alle colonne descritte nel paragrafo precedente, ma con alcune modifiche per preparare i dati ai task di apprendimento supervisionato e non supervisionato. Di seguito le modifiche e trasformazioni effettuate:

1. In aggiunta alla colonna `Year` sono state aggiunte le colonne `Decade` e `Recent`, rispettivamente un *intero* che rappresenta il decennio in cui é avvenuto il rilascio del film e una feature *booleana* che indica se il film é stato rilasciato nell'ultimo decennio (dal 2010 in poi). L'aggiunta di queste due colonne risulta utile sia per ovviare ad eventuali problemi legati alla cardinalitá alta della feature `Year`, maggiore rispetto alle due colonne appena citate, ma anche perché con queste colonne é possibile catturare pattern e cambiamenti che avvengono su periodi di tempo differenti: la feature `Decade`, ad esempio, permette di catturare cambiamenti che avvengono su periodi di tempo più lunghi, permettendo ai modelli di comprendere come cambiano gli incassi dei film da un decennio ad un altro; la feature `Recent`, invece, può essere utile per identificare se ci sono differenze significative tra i film più recenti e quelli più vecchi.

2. La colonna `Genre` é stata suddivisa dedicando una colonna ad ogni genere. Di conseguenza, é stata sfruttata una tecnica di **one-hot encoding** per convertire la feature categorica in più feature booleane utilizzando variabili indicatrici. Lo stesso ragionamento é stato eseguendo per codificare le feature `MPAA` e `Country`;

3. Per la colonna `Runtime` si é fatto uso del **binning** per suddividere la durata del film in 3 bins (o soglie): *<90, 90-120, >120*. In questo modo, oltre a ridurre la cardinalità della feature, é stato possibile suddividere in modo equilibrato i film nelle soglie in base alla loro durata. Queste soglie sono state poi codificate in interi da 0 a 2 in modo tale da consentire ai modelli di apprendimento l'utilizzo del dato;

4. L'ultima trasformazione é stata eseguita sulle feature `Director, Writer, Main Actor` e `Company`. Non essendo possibile applicare il one-hot encoding per rendere feature booleana ogni possibile valore, si é deciso di optare per una rappresentazione differente. In particolare, per ogni occorrenza il valore di queste colonne é stato sostituito con il numero di film in cui il regista, scrittore, attore o compagnia é coinvolto/a. Queste colonne possono aiutare a identificare quali registi, sceneggiatori, attori e compagnie hanno un impatto significativo sul successo di un film, facilitando cosí l'analisi dei dati.

Oltre alle trasformazioni sopra elencate, le feature rimanenti sono state standardizzate/normalizzate principalmente per ridurre la loro cardinalità (come successo per

le feature `Budget` e `Worldwide Gross`) e perché più adatte per la fase di apprendimento. Infine, per avere un'idea generale sulla distribuzione delle feature categoriche nel dataset, si riportano le distribuzioni dei generi, rating e runtime. Com'é possibile

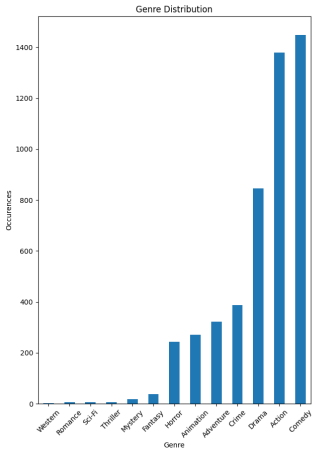


Figure 2.1: *Distribuzione dei generi*

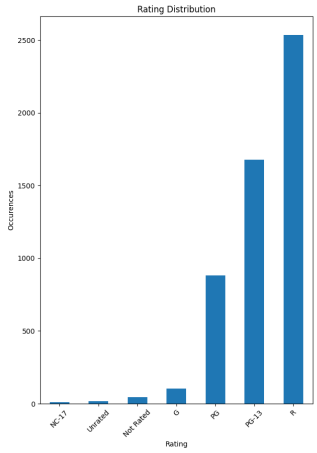


Figure 2.2: *Distribuzione dei ratings*

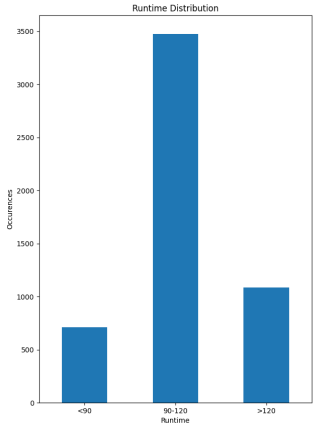


Figure 2.3: *Distribuzione dei runtime*

osservare dalle figure, il dataset é fortemente sbilanciato a causa di una maggiore presenza di film facenti parte dei generi e dei rating piú popolari; per risolvere questa problematica, saranno quindi proposte delle soluzioni atte a bilanciare maggiormente gli esempi rientranti nella categoria minoritaria. In generale, quindi, possiamo confermare come questo sia un problema di regressione con classe target sbilanciata.

3 Ragionamento logico e Prolog

Il **ragionamento logico** consiste in una forma di ragionamento basata sul *ragionamento deduttivo*, ovvero incentrato sull'utilizzo di regole logiche per dedurre nuovi fatti a partire da quelli esistenti. Queste regole logiche sono definite con l'aiuto della logica matematica che consente la definizione di una *Knowledge Base* contenente vari *assiomi*, ovvero informazioni considerate sempre vere ed inerenti al dominio di interesse che si intende approfondire. Nel nostro caso di studio é stata definita una knowledge base contenente le principali informazioni inerenti i film presenti all'interno del dataset esaminato. Nello specifico, é stato nostro interesse quello di rappresentare nella knowledge base (accessibile consultando il file `./src/kb.pl` del progetto) una serie di assomi, suddivisi in **fatti** e **regole** logiche, che consentissero al nostro agente di inferire ulteriori informazioni sui film memorizzati. Per fare ció si é scelto di utilizzare **Prolog**, linguaggio di programmazione dichiarativo basato sul ragionamento logico, usufruendo della libreria `pyswip` per l'integrazione tra il linguaggio Python e Prolog.

All'interno della nostra knowledge base sono stati innanzitutto definiti tutti i generi ed i rating ritenuti validi, mostrati di seguito:

Algorithm 1: Generi e rating della KB

```
genre('Action').
genre('Adventure').
genre('Animation').
genre('Comedy').
genre('Crime').
genre('Drama').
genre('Fantasy').
genre('Historical').
genre('Horror').
genre('Musical').
genre('Mystery').
genre('Romance').
genre('Sci-Fi').
genre('Thriller').
genre('War').
genre('Western').

rating('G').
rating('PG').
rating('PG-13').
rating('R').
rating('NC-17').
```

Successivamente, sono stati definiti fatti utili per specificare le informazioni relative ai film presenti nel dataset iniziale. Esempi di fatti aggiunti nella knowledge base sono mostrati di seguito:

Algorithm 2: Film della KB

```
movie('The White Balloon', 1995, 'Unrated', 85, 'Farabi Cinema Foundation', 'Iran', 'Jafar Panahi', 'Abbas Kiarostami', 'Aida Mohammadkhani', 150000, 9249400, 'Drama', 77, 690000).
movie('L.I.E.', 2001, 'Unrated', 97, 'Alter Ego Entertainment', 'United States', 'Michael Cuesta', 'Stephen M. Ryder', 'Brian Cox', 700000, 18460590, 'Crime', 71, 960000).
movie('Old Joy', 2006, 'Unrated', 76, 'Film Science', 'United States', 'Kelly Reichardt', 'Jonathan Raymond', 'Daniel London', 300000, 3010470, 'Drama', 68, 700000).
movie('The Texas Chainsaw Massacre 2', 1986, 'Unrated', 101, 'Cannon Films', 'United States', 'Tobe Hooper', 'L.M. Kit Carson', 'Dennis Hopper', 4700000, 80258720, 'Comedy', 56, 2900000).
. . .
```

Infine, sono state definite due regole, `roi(Movie, ROI)` e `success(Movie)`, utili rispettivamente per calcolare il **Return on Investment** (**ROI**), metrica utilizzata come indicatore del successo finanziario di un film, e verificare che il film fornito in input sia un successo, sulla base del calcolo del ROI corrispondente. Le regole definite nella knowledge base sono indicate di seguito:

Algorithm 3: Regole della KB

```
roi(Movie, ROI) :-
    movie(Movie, _, _, _, _, _, _, _, _, Budget, Gross, _, _, _),
    ROI is ((Gross - Budget) / Budget) * 100.

success(Movie) :-
    roi(Movie, ROI),
    ROI > 300.
```

All'interno del file `./src/movie_prolog.py` é stato definito il codice necessario per definire la knowledge base. Nello specifico, il codice si occupa di scrivere gli assomi alla base della knowledge base ed eseguire le query necessarie per ricavare le informazioni sul ROI e sul successo di ogni film, sulla base delle regole precedentemente definite. Queste due nuove informazioni sono state integrate al database di partenza aggiungendo due nuove colonne, `Success` e `ROI`, e salvando il nuovo dataset in un apposito file (accessibile al percorso `./resources/dataset/Movie_dataset_with_prolog_results.csv` con l'obiettivo di utilizzare queste nuove informazioni nel task di apprendimento supervisionato per migliorare le performance dei modelli esaminati. Ovviamente, é stato eseguito feature engineering per permettere ai modelli di sfruttare appieno i nuovi dati ottenuti in seguito all'utilizzo di Prolog.

4 Apprendimento Non Supervisionato

L'**apprendimento non supervisionato** é una tecnica di apprendimento automatico in cui un modello viene addestrato senza che ad esso siano fornite informazioni sulle feature target. Un metodo generale per l'apprendimento non supervisionato è il **clustering**, usato con lo scopo di raggruppare dati in insiemi (definiti appunto **cluster**) che condividono caratteristiche simili, senza avere informazioni predefinite su categorie o etichette di alcun tipo. Tale metodo viene applicato con l'obiettivo di scoprire strutture e pattern nascosti nei dati, organizzandoli in cluster in modo che gli oggetti all'interno di ciascun cluster siano più simili tra loro rispetto a quelli di altri cluster.

In questo caso di studio abbiamo eseguendo un task di apprendimento non supervisionato a partire dal nostro dataset pre-processato con l'obiettivo di eseguire **hard clustering**, ovvero di inserire ogni esempio all'interno di un cluster specifico. Tale informazione sará poi integrata nel dataset di partenza, al fine di utilizzare questo nuovo dato per il task di apprendimento supervisionato, descritto nel capitolo successivo. Per raggiungere questo scopo, abbiamo deciso di utilizzare l'algoritmo di hard clustering **KMeans**. Questo algoritmo si occupa di suddividere gli esempi in un numero prefissato di cluster (indicato con `k`) e modificare la loro distribuzione fino a quando non si riesce a minimizzare la variabilitá all'interno dei vari cluster.

4.1 Individuazione del numero di cluster

Prima di applicare il KMeans sul dataset é necessario individuare il numero corretto di cluster che l'algoritmo dovrá considerare. Per determinare il numero ottimale di cluster sulla base dei dati in input é stata applicata una strategia nota come *regola del gomito*. La regola del gomito si basa sulla creazione di un grafico che riporta i valori di **inertia**, ovvero la somma dei quadrati delle distanze tra i punti e i centroidi del cluster, rispetto al numero di cluster `k`; l'idea alla base dell'algoritmo é di trovare un punto in cui l'aggiunta di ulteriori cluster non migliora significativamente la qualità del clustering. Questo rappresenterá il nostro gomito. Di seguito si mostra il codice utilizzato per eseguire la regola del gomito:

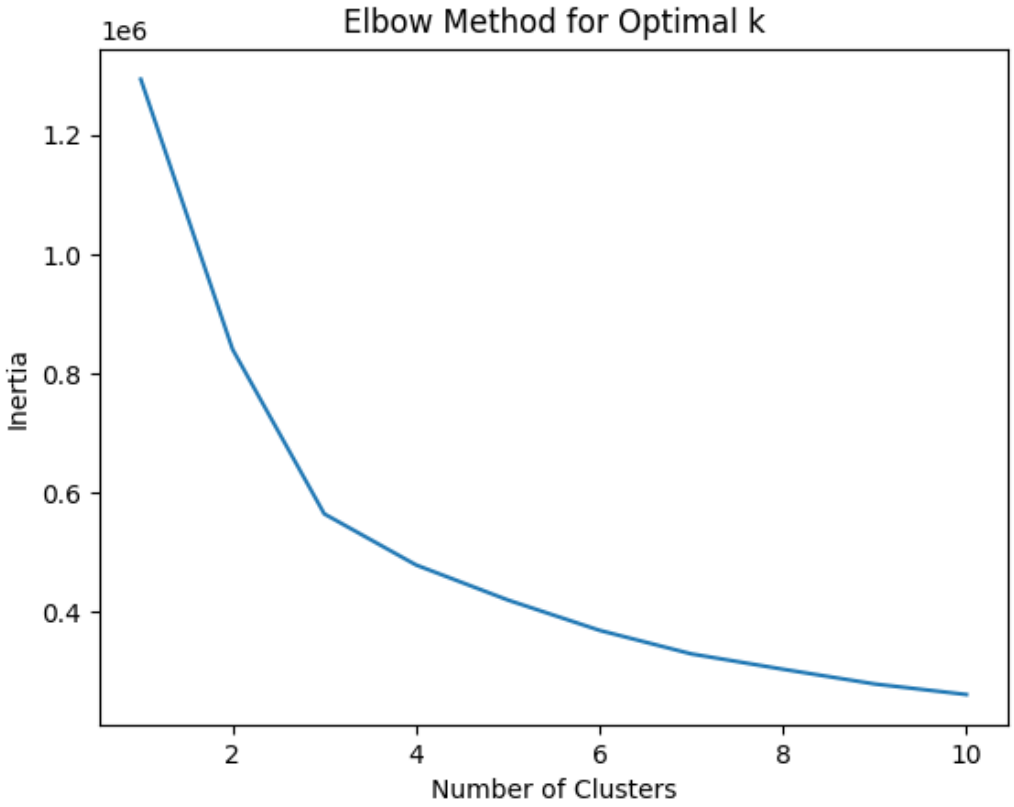
Algorithm 5: Regola del gomito per determinare il numero di cluster

```
def elbow_method(y):
    # Create a list to store the inertia values
    inertia_values = []
    k_values = range(1, 11)

    # Iterate over the k values
    for k in k_values:
        # Create a KMeans model with k clusters
        model = KMeans(n_clusters=k, n_init=10, init='random')
```

```
10
11          # Fit the model to the Log_Worldwide_Gross column
12          model.fit(y)
13
14          # Append the inertia value to the list
15          inertia_values.append(model.inertia_)
16
17      # Create a KneeLocator object to find the optimal k value
18      kneeLocator = KneeLocator(
19          k_values,
20          inertia_values,
21          curve='convex',
22          direction='decreasing'
23      )
24
25      # Plot the inertia values
26      plt.plot(k_values, inertia_values)
27      plt.xlabel('Number of Clusters')
28      plt.ylabel('Inertia')
29      plt.title('Elbow Method for Optimal k')
30      plt.savefig('../resources/plots/unsupervised_learning/
            elbow_method.png')
31      plt.show()
32
33      # Return the number of clusters
34      return kneeLocator.elbow
```

Come é possibile vedere dal codice sopra riportato, si é deciso di eseguire questa regola impostando un numero di cluster che varia da 1 a 10, successivamente si applica per ogni `k` il KMeans utilizzando 10 inizializzazioni (con il parametro `n_init=10`, che indica il numero di *Random Restart*) e un'inizializzazione casuale dei centroidi (con il parametro `init='random'`). Infine, si utilizza un oggetto `KneeLocator`, offerto dalla libreria `kneed`, per identificare il punto del "gomito" nel grafico dell'inertia. Questo é il risultato ottenuto:

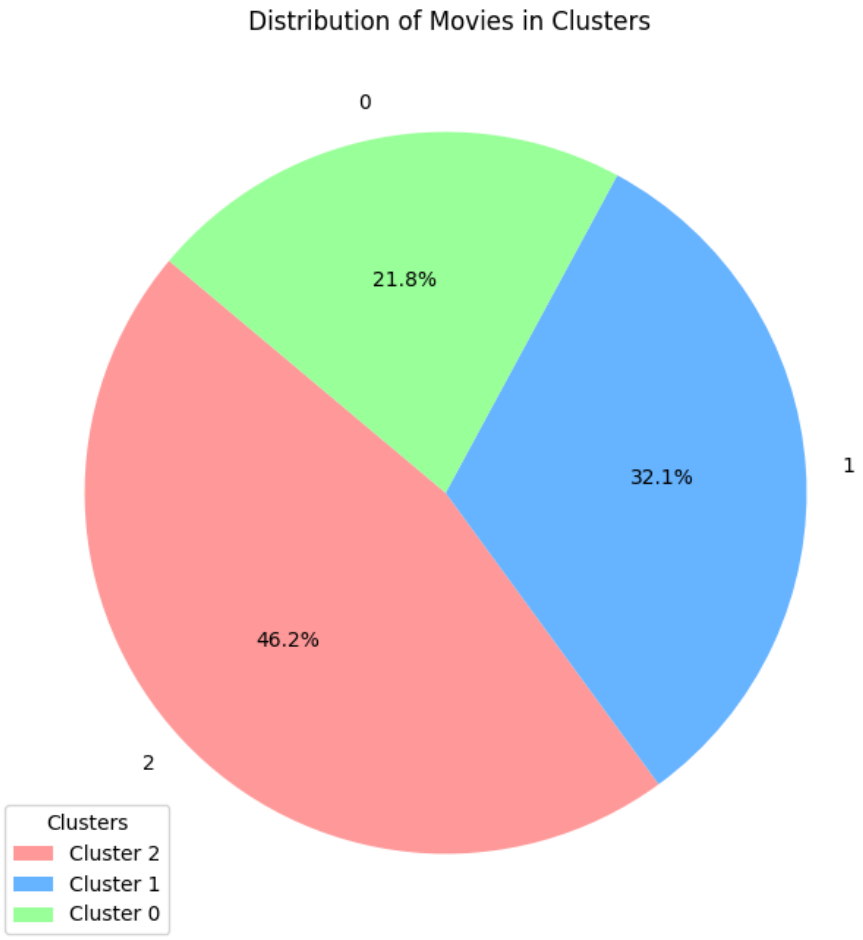


Regola del gomito per trovare il k ottimale

Come si puó osservare dal grafico sovrastante, la regola del gomito ha definito 3 come numero ottimale di cluster.

4.2 Hard clustering con algoritmo KMeans

Dopo aver definito il numero ottimale di cluster, é possibile eseguire il KMeans per eseguire l'hard clustering degli esempi del dataset. Nello specifico, l'algoritmo é stato applicato con l'obiettivo di suddividere gli esempi sulla base di specifiche feature del dataset, ritenute rilevanti per il problema in esame: sono state selezionate, infatti, solo le feature che potrebbero influenzare maggiormente il clustering, in quanto maggiormente correlate con l'incasso di un film. Ció permette di ridurre la probabilitá di introdurre rumore nei dati che potrebbe ridurre la qualitá dei cluster. Di seguito si rappresenta il risultato ottenuto in seguito all'applicazione del KMeans:



Come possiamo vedere la distribuzione dei film non é perfetta, ma comunque é accettabile in quanto i cluster ottenuti sono quasi perfettamente bilanciati. Oltre alla visualizzazione dei cluster ottenuti, nel file `./resources/logs/unsupervised_learning/cluster_stats.txt` é possibile osservare il **Silhouette Score**, che misura quanto bene i cluster sono separati l'uno dall'altro, ed i valori di media e mediana per ciascuna feature utilizzata per l'apprendimento non supervisionato. Nel nostro caso il silhouette score ottenuto é circa 0.40, indicando una separazione moderata tra i cluster.

In seguito al clustering effettuato, il risultato ottenuto, che rappresenta l'indice del cluster in cui ogni film é stato inserito, é stato memorizzato in un nuovo dataset, accessibile al percorso `./resources/dataset/Movie_dataset_clusters.csv` del progetto.

5 Apprendimento Supervisionato

Il task di apprendimento supervisionato proposto é descrivibile come un task di **regressione**: a partire dalle caratteristiche principali di un film, quali budget, rating o genere, si cerca di predire quale sarà il possibile incasso globale lordo di un film. Si tratta di un task di regressione poiché la feature target, `Worldwide Gross`, presenta un dominio continuo, essendo questo il logaritmo del guadagno globale di un film (logaritmo definito in seguito alla fase di feature engineering).

5.1 Modelli e metodologie utilizzate

Metodologia utilizzata: Per poter addestrare ed individuare il modello più adatto per il problema esaminato sono state rispettate varie fasi, elencate di seguito:

1. Scegliere, per ogni modello, i valori migliori degli iper-parametri
2. Addestrare ogni modello sulla base degli iper-parametri selezionati
3. Valutare ogni modello con i valori migliori per i suoi iper-parametri

Modelli adottati e iperparametri selezionati: I modelli considerati per questo tipo di task sono descritti di seguito, indicando per ognuno di essi gli iper-parametri considerati ed i valori corrispondenti:

1. **Alberi di Decisione** (Decision Trees), implementati attraverso la libreria `scikit-learn`
   - **criterion**: Funzione che misura la qualità di uno split. Serve per definire il criterio di suddivisione dei nodi dell'albero.
     I valori sperimentati sono:
     - **squared_error**: Misura la qualità della divisione utilizzando il Mean Squared Error (MSE). Ottimizza la divisione minimizzando la *L2 loss*, cioè usando la media dei valori di ciascun nodo terminale;
     - **friedman_mse**: Una variante del MSE che include il punteggio di miglioramento di Friedman per le potenziali divisioni. Cerca di migliorare la qualità delle divisioni tenendo conto non solo dell'errore quadratico, ma anche della complessità delle suddivisioni;
     - **absolute_error**: Misura la qualità della divisione utilizzando il Mean Absolute Error (MAE). Ottimizza la divisione minimizzando la *L1 loss*, utilizzando la mediana dei valori di ciascun nodo terminale;
     - **poisson**: Utilizza la riduzione della devianza di Poisson come criterio per determinare la qualità delle divisioni;
   - **max_depth**: Massima profondità dell'albero.
     I valori sperimentati sono: [5, 10, 15, 20, 40];

- **min_samples_split**: Numero minimo di campioni per suddividere un nodo.
  I valori sperimentati sono: [2, 5, 10, 15, 20, 30];
- **min_samples_leaf**: Numero minimo di campioni per essere foglia.
  I valori sperimentati sono: [1, 2, 5, 10, 15, 20, 30, 50, 100];
- **random_state**: Controlla la casualità dello stimatore.
  Il valore scelto per il *seed* é 42 per garantire la riproducibilità dei risultati.

2. **Random Forest**, implementata attraverso la libreria `scikit-learn`.

   - **n_estimators**: Il numero di stimatori (alberi) nella foresta.
     Il valore utilizzato é 100 in quanto ottimale per il nostro caso di studio. Con un valore superiore, infatti, il modello richiedeva un tempo elevato per poter eseguire l'apprendimento, non permettendoci cosí di ottenere risultati in tempi ragionevoli;
   - **criterion**: Funzione che misura la qualità di uno split. Serve per definire il criterio di suddivisione dei nodi dell'albero.
     I valori sperimentati sono: ["squared_error", "absolute_error", "friedman_mse", "poisson"];
   - **max_depth**: Massima profondità degli alberi.
     I valori sperimentati sono: [None, 10, 20];
   - **n_jobs**: Numero di processori da utilizzare per l'addestramento del modello.
     Il valore utilizzato é -1 in modo tale da utilizzare tutti i processori disponibili;
   - **random_state**: Controlla la casualità dello stimatore.
     Il valore scelto per il *seed* é 42 per garantire la riproducibilità dei risultati.

3. **Boosting di Alberi di Decisione** (Gradient Boosted Trees), implementata attraverso la libreria `lightgbm`. Nello specifico, come modello é stato utilizzato `LGBMRegressor`, presente nel framework di ensemble learning **LightGBM**[6].

   - **n_estimators**: Numero di stimatori (alberi) utilizzati.
     I valori sperimentati sono: [50, 100, 200, 500];
   - **learning_rate**: Velocità con cui il modello impara dai dati durante il processo di addestramento. Un learning rate più basso richiede più iterazioni per raggiungere la convergenza, ma può portare a una migliore generalizzazione e a modelli più stabili. D'altra parte, un learning rate più alto potrebbe accelerare il processo di addestramento.
     I valori sperimentati sono: [0.01, 0.02, 0.05, 0.1, 1.0];
   - **max_depth**: Massima profondità degli alberi appresi.
     I valori sperimentati sono: [10, 20, 40];
   - **class_weight**: I pesi associati a ciascuna classe. La modalità "bilanciata" utilizza i valori della target feature per regolare automaticamente i pesi in

modo inversamente proporzionale alle frequenze delle classi nei dati di input.
I valori sperimentati sono: [None, "balanced"];

- **verbose**: Controlla il livello di dettaglio delle informazioni di log che vengono stampate durante l'addestramento del modello.
Il valore scelto é -1 per disabilitare completamente i messaggi di log;
- **random_state**: Controlla la casualità dello stimatore.
Il valore scelto per il *seed* é 42 per garantire la riproducibilità dei risultati.

4. **Boosting di Alberi di Decisione** (Extreme Gradient Boosting), implementata attraverso la libreria `xgboost`. Nello specifico, come modello é stato utilizzato `XGBRegressor`, presente in **XGBoost**[1], framework di gradient boosting che combina diverse versioni di alberi decisionali deboli per creare un modello complessivo più potente.

   - **n_estimators**: Numero di stimatori (alberi) utilizzati.
   I valori sperimentati sono: [20, 50, 100];
   - **learning_rate**: Velocità con cui il modello impara dai dati durante il processo di addestramento. Un learning rate più basso richiede più iterazioni per raggiungere la convergenza, ma può portare a una migliore generalizzazione e a modelli più stabili. D'altra parte, un learning rate più alto potrebbe accelerare il processo di addestramento.
   I valori sperimentati sono: [0.01, 0.05, 0.1];
   - **max_depth**: Massima profondità degli alberi appresi.
   I valori sperimentati sono: [10, 20, 40];
   - **lambda**: Parametro di regolarizzazione L2. Un valore più alto di lambda aumenta la forza della regolarizzazione, riducendo così il rischio di overfitting.
   I valori sperimentati sono: [0.01, 0.1, 0.5];
   - **verbosity**: Controlla il livello di dettaglio delle informazioni di log che vengono stampate durante l'addestramento del modello.
   Il valore scelto é 0 per disabilitare completamente i messaggi di log;
   - **random_state**: Controlla la casualità dello stimatore.
   Il valore scelto per il *seed* é 42 per garantire la riproducibilità dei risultati.

La differenza principale tra i framework LightGBM e XGBoost sta nella strategia di crescita degli alberi: LightGBM, infatti, non fa crescere un albero livello-per-livello, come succede in XGBoost, ma esegue una divisione foglia-per-foglia.

Addestramento e ricerca dei migliori iper-parametri: Prima di procedere con l'addestramento dei modelli, é necessario ricercare i migliori iper-parameteri per ognuno di essi. Per fare ciò é stata utilizzata una **Repeated K-Fold Cross-Validation**,

prevista dalla libreria `scikit-learn`. La *Repeated K-Fold Cross-Validation* é una tecnica di cross-validation usata per valutare le performance di un modello in modo più robusto rispetto alla semplice *K-Fold Cross Validation*: con questa procedura, infatti, il dataset viene suddiviso in k fold ed il modello viene addestrato k volte (come nella procedura tradizionale), ma in aggiunta l'intero processo viene ripetuto per n volte. Ogni ripetizione comporta una nuova suddivisione dei dati in k fold, e per ognuno di essi si calcola la metrica di valutazione presa in considerazione; al termine del procedimento le performance del modello vengono aggregate (calcolando, ad esempio, la media o deviazione standard) per ottenere in questo modo una stima più affidabile delle performance del modello.

In aggiunta a questa tecnica, per la ricerca degli iper-parametri é stata impiegata la classe `GridSearchCV` predisposta da `scikit-learn` per implementare una **ricerca esaustiva** sulla griglia specificata di iper-parametri del modello in questione. La grid search é stata impostata con l'obiettivo di trovare i migliori iper-parameter in grado di minimizzare la metrica `neg_mean_squared_error`, corrispondente all'**errore quadratico medio negativo**. La motivazione per cui si é scelta l'errore quadratico medio negativo e non positivo sta nella convenzione rispettata dalle funzione di ricerca di `scikit-learn`[5]: la funzione `GridSearchCV`, cosí come altre funzioni presenti nella libreria, cercando di massimizzare la metrica di scoring definita, pertanto se utilizzassimo il `mean_squared_error` come metrica (che dev'essere minimizzata) la funzione cercherebbe di massimizzare l'errore, invece di ridurlo.

Valutazione dei modelli: Per quanto riguarda la fase di valutazione, si é deciso anche in questo caso di utilizzare una `Repeated K-Fold Cross-Validation`, passata come parametro del metodo `cross_validate` offerto da `scikit-learn`, al fine di valutare le performance dei modelli utilizzando gli iper-parametri individuati nella fase precedente. Le metriche su cui ogni modello viene valutato, disponibili nella libreria `scikit-learn`[5], sono descritte di seguito:

- **Mean Squared Error (MSE)**: La funzione `mean_squared_error` calcola l'**errore quadratico medio**, una metrica di rischio corrispondente al valore atteso dell'errore quadratico (*squared error o loss*).
- **Mean Absolute Error (MAE)**: La funzione `mean_absolute_error` calcola l'**errore assoluto medio**, una metrica di rischio corrispondente al valore atteso della perdita di errore assoluto (*absolute error loss*) o *L1-norm loss*.
- **Mean Squared Logarithmic Error (MSLE)**: La funzione `mean_squared_log_error` calcola una metrica di rischio corrispondente al valore atteso dell'**errore logaritmico al quadrato** (*squared logarithmic error o loss*).
- R^2: La funzione `r2_score` calcola il **coefficiente di determinazione**, solitamente indicato come R^2. Rappresenta la proporzione di varianza (ossia la dispersione dei dati) del target (la variabile dipendente `y`) che è stata spiegata dalle

variabili indipendenti nel modello (le feature). Fornisce un'indicazione della bontà di adattamento e quindi una misura di quanto bene i campioni non visti siano probabilmente previsti dal modello, attraverso la *proporzione di varianza spiegata*. Un R^2 più alto indica che il modello ha catturato bene i pattern nei dati, mentre un R^2 basso indica che c'è ancora molta varianza non spiegata. Il valore massimo é 1.0, ma puó essere anche negativo (perché il modello può essere arbitrariamente peggiore).

Per ciascuna metrica sono state calcolate anche varianza e deviazione standard relative al modello esaminato.

5.2 Risultati dell'addestramento e della valutazione dei modelli

In un primo esperimento si é deciso di addestrare i modelli descritti in precedenza sugli esempi del dataset, senza che questi ultimi siano stati soggetti ad alcuna manipolazione. L'obiettivo é stato quello di valutare il comportamento dei modelli proposti ai dati originali, in modo tale da apportare le dovute correzioni solo in seguito. Di seguito sono riportati i migliori iper-parametri, trovati mediante la procedura di `GridSearchCV`, per ciascun modello:

Hyperparameter	**Value**
DecisionTree__criterion	friedman_mse
DecisionTree__max_depth	15
DecisionTree__min_samples_leaf	1
DecisionTree__min_samples_split	10
DecisionTree__random_state	42

Decision Tree Parameters

Hyperparameter	**Value**
LGBMRegressor__class_weight	None
LGBMRegressor__learning_rate	0.05
LGBMRegressor__max_depth	40
LGBMRegressor__n_estimators	500
LGBMRegressor__random_state	42
LGBMRegressor__verbose	-1

LGBM Regressor Parameters

Hyperparameter	**Value**
RandomForest__criterion	squared_error
RandomForest__max_depth	20
RandomForest__n_estimators	100
RandomForest__n_jobs	-1
RandomForest__random_state	42

Random Forest Parameters

Hyperparameter	**Value**
XGBRegressor__lambda	0.5
XGBRegressor__learning_rate	0.1
XGBRegressor__max_depth	10
XGBRegressor__n_estimators	100
XGBRegressor__random_state	42
XGBRegressor__verbosity	0

XGB Regressor Parameters

Di seguito si riportano i risultati ottenuti per ogni metrica e modello:

Metric	Score Mean	Score Variance	Score Std
MSE	0.038295044826338796	0.00024050387440733325	0.015508187334673684
MAE	0.0973535289234981	2.724989873696076e-05	0.0052201435552061935
MSLE	0.0001174788120184274	3.6145109728259606e-09	6.0120803161850395e-05
R2	0.9857383350963825	3.3338802569004546e-05	0.005773976322172143

Risultati delle metriche di valutazione per il Decision Tree

Metric	Score Mean	Score Variance	Score Std
MSE	0.020032093456273447	0.0001919345928714926	0.013854046083057923
MAE	0.04489426018367012	2.766394694945505e-05	0.005259652740386484
MSLE	6.820831711305719e-05	3.3278237716198246e-09	5.7687292982248913e-05
R2	0.9925381673982067	2.635606901068303e-05	0.005133816222916733

Risultati delle metriche di valutazione per il Random Forest

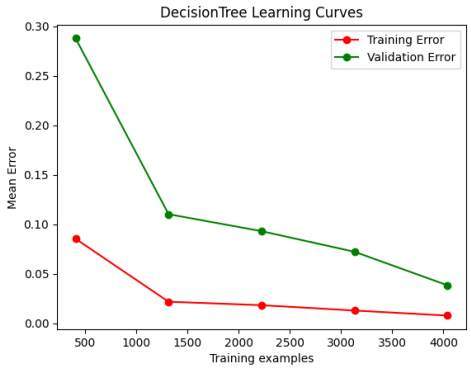
Metric	Score Mean	Score Variance	Score Std
MSE	0.01916568304787417	0.00016645519024947666	0.012901751441160098
MAE	0.04846803829317163	2.1023530361698977e-05	0.004585142349120578
MSLE	6.511082796555166e-05	2.9813476896795255e-09	5.4601718742907037e-05
R2	0.9928655578611224	2.2849639288593055e-05	0.004780129630940258 5

Risultati delle metriche di valutazione per l'LGBMRegressor

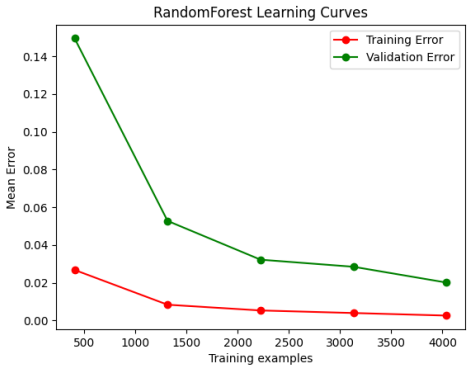
Metric	Score Mean	Score Variance	Score Std
MSE	0.02416730303247582	0.0003197821960609331	0.01788245497857979
MAE	0.04774458877150145	3.6691220057303514e-05	0.0060573277984028165
MSLE	7.957584483674735e-05	4.370720594407332e-09	6.611142559654369e-05
R2	0.9910021167206523	4.390530604827512e-05	0.006626107911004402

Risultati delle metriche di valutazione per l'XGBRegressor

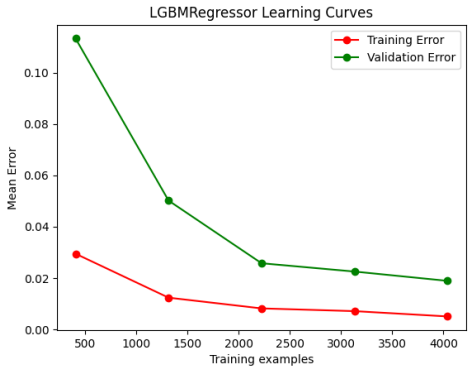
Valutazione: Di seguito si riportano le curve di apprendimento di ciascun modello:



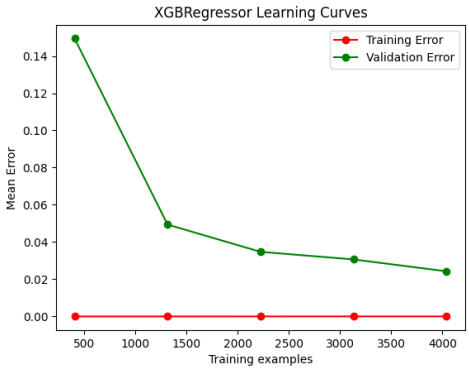
Decision Tree Learning Curves



Random Forest Learning Curves



LGBMRegressor Learning Curves



XGBRegressor Learning Curves

Analizziamo nel dettaglio i grafici sopra illustrati:

- La *Figura 3.1* rappresenta le curve di apprendimento per il modello di **Decision Tree**. Da questo grafico é possibile osservare come le prestazioni del modello siano generalmente buone, con un errore di training e di validazione che diminuisce all'aumentare degli esempi forniti, con valori minimi rispettivamente di 0.0077 e 0.07. Osservando gli errori ottenuti, non sembrano esserci segnali di overfitting, poiché l'errore di validazione non é molto piú alto dell'errore di addestramento e diminuisce man mano che aumentano i dati, ed inoltre non sono visibili segnali di underfitting. Il modello, quindi, risulta abbastanza complesso per catturare tutte le caratteristiche dei dati e adattarsi quindi ai dati di test, dimostrando di essere stabile ed in grado di generalizzare bene con i dati forniti.

- La *Figura 3.2* e la *Figura 3.4* rappresentano le curve di apprendimento rispettivamente per il modello **Random Forest** e **Gradient Boosted Tree di XGBoost**. Sono stati raggruppati in un'unica analisi in quanto é possibile osservare

che questi presentano comportamenti simili tra loro: osservando gli errori ottenuti, in entrambi i casi é possibile osservare come le prestazioni siano migliori rispetto al modello precedentemente analizzato. Anche in questo caso i problemi di **overfitting** e di **underfitting** non sono presenti, data la differenza minima presente tra errore di training e di validazione e l'assenza di una stabilizzazione delle curve. Anche le metrice riportate nella pagina precedente indicano un buon comportamento da parte dei modelli, con errori molto bassi che indicano la capacitá dei modelli di generalizzare bene nonostante lo sbilanciamento delle classi target. Unica differenza é relativa all'errore di training, decisamente inferiore nel modello XGBRegressor.

- La *Figura 3.3*, infine, rappresenta le curve di apprendimento del modello **LGBMRegressor**. Gli errori relativamente elevati su entrambi i set di dati potrebbero suggerire che il modello non é in grado di catturare completamente la complessità dei dati, indicando quindi la presenza di **underfitting**, ma se si considera che i valori dell'errore di training sono comunque relativamente bassi si potrebbe ipotizzare che il modello sta cercando di generalizzare meglio i dati man mano che aumenta la loro quantitá, rendendo l'underfitting un problema meno preoccupante. Simile é la questione dell'overfitting: la differenza significativa tra errori di training e test indica che l'overfitting é piú probabile rispetto all'underfitting, suggerendo che il modello potrebbe aver imparato del "rumore" dal training set invece di individuare pattern utili per i dati. Tuttavia, osservando i valori ottenuti per le varie metriche, che migliorano con l'aumentare degli esempi forniti, é possibile dedurre che questo non sia un problema assai rischioso.

Complessivamente, osservando i grafici ed i valori di media, varianza e deviazione standard di ogni modello si puó affermare che il modello che si é comportato meglio é il **Gradient Boosted Tree di LightGBM**: la metrica R^2, ad esempio, riporta un valore di 0.9928, indicando come il modello sia in grado di spiegare circa il 99,28% della varianza dei dati; questo rappresenta un valore eccellente, in quanto indica la grande capacitá del modello di adattarsi ai dati. Anche i valori delle altre metriche suggeriscono una buona adattabilitá rispetto agli altri modelli esaminati: la deviazione standard dell'errore sui dati di test, ad esempio, è relativamente bassa, il che suggerisce che il modello ha prestazioni relativamente consistenti attraverso i diversi campioni di test.

Nota: le osservazioni riportate durante la valutazione non tengono conto della trasformazione logaritmica applicata alla feature target, ma vogliono fornire un'analisi generale dei modelli impiegati per comprenderne il comportamento e l'adattabilitá dei modelli nel caso in cui si rispetti la scala logaritmica. La trasformazione (applicata mediante il metodo `log1p` della libreria `numpy`) trasforma il valore decimale dell'incasso nel sua controparte logaritmica, utilizzando la formula `log(1 + x)`. Tale trasformazione é stata applicata per ridurre l'impatto degli *outliers*, ovvero valori estremamente alti o bassi degli incassi dei film, al fine di rendere i modelli piú stabili e meno sensibili a

questi valori. I valori ottenuti nei grafici sopra riportati possono sembrare abbastanza bassi, portandoci quindi a considerare i modelli come accettabili, ma se convertissimo i valori della feature target ai suoi valori originali potremmo renderci conto di come i modelli non siano perfetti, raggiungendo errori alti per ogni modello. Le possibili cause di questi errori possono essere legate ad eventuali problemi con i dati (ad es., presenza di **outliers** o **non linearitá** del fenomeno in base alle input features predisposte) oppure a problemi causati da standardizzazioni e normalizzazioni eseguite sui valori di alcune feature (quindi errori causati dal feature engineering effettuato). Per comprendere le possibili ragioni dietro questi errori, si é pensato di applicare alcune tecniche di **Over-sampling**, utili per generare esempi per i valori estremi al fine di bilanciare maggiormente il dataset. Da questa prima valutazione si puó prevedere che l'applicazione di tecniche di **Under-sampling** potrebbe portare a risultati peggiori rispetto a quanto osservato prima, tuttavia anche queste tecniche sono state applicate per verificare il comportamento dei modelli.

6 Tecniche di Over-sampling e Under-sampling

Per **Oversampling** si intende una tecnica di pre-elaborazione dei dati utilizzata per affrontare problemi di squilibrio tra classi in un dataset. Puó succedere, infatti, che quando una o piú classi risultano sottorappresentate rispetto alle altre, un modello addestrato su questo tipo di dataset tende ad adattarsi bene per i dati facenti parte della classe maggioritaria, trascurando per il piú delle volte le classe minoritaria. Il compito dell'over-sampling, quindi, é quello di riequilibrare il dataset generando esempi artificiali per la classe minoritaria, al fine di migliorare la sua rappresentatività.

Nel nostro caso di studio, la tecnica di over-sampling viene utilizzata per ovviare alle problematiche riscontrate durante il task di apprendimento supervisionato: com'é stato possibile osservare dalle tabelle illustrate nel paragrafo precedente, i quattro modelli impiegati nell'apprendimento supervisionato hanno prodotto buoni risultati, ma presentano comunque un margine di miglioramento considerando gli errori ottenuti ed i valori delle metriche. A questo problema si aggiunge anche quello relativo alle curve di apprendimento: queste curve sembrano simboleggiare un buon comportamento da parte dei modelli, ma bisogna ricordare che gli errori raffigurati nei grafici sono ottenuti in seguito ad una conversione dei valori della feature target in scala logaritmica. Considerando la scala originale dei dati, tali prestazioni non risultano piú ottimali, in quanto gli errori devono essere convertiti alla loro scala effettiva per poterci rendere conto dell'effettivo valore predetto dai modelli.

Si é deciso, inoltre, di esplorare anche la tecnica di **under-sampling**, tecnica il cui obiettivo é quello di rispettare un comportamento inverso rispetto a quello dell'over-sampling: se quest'ultimo si occupa di generare esempi sintetici per la classe minoritaria, l'under-sampling mira a ridurre il numero di istanze nella classe maggioritaria, sempre con lo scopo di bilanciare la distribuzione delle classi e migliorare le prestazioni del modello. Si é consapevoli che l'utilizzo di questa tecnica non possa portare ad un miglioramento dei modelli, date le loro performance attuali, ma si é comunque scelto di provare questa tecnica per confrontare il comportamento dei modelli di apprendimento automatico con le due tipologie di tecniche.

6.1 RandomOverSampler e SMOTE

Un primo approccio é stato quello di eseguire l'over-sampling sul training set ricorrendo alle classi `RandomOverSampler` e `SMOTE` della libreria `imblearn`[2]. Il ragionamento alla base di queste classi é il seguente:

- Il compito del **RandomOverSampler** é quello di generare nuovi esempi sintetici duplicando in modo casuale gli esempi che rientrano nella classe target minoritaria. Sebbene inizialmente il Random Oversampling possa sembrare una tecnica

poco affidabile, é stato possibile osservare come in certi casi e le prestazioni dei modelli hanno subito un miglioramento, aumentando la capacitá di questi ultimi di adattarsi ai dati.

- **SMOTE** (acronimo di *Synthetic Minority Oversampling Technique*) é una tecnica molto diffusa per generare nuovi esempi. A differenza del Random Oversampling, SMOTE crea nuovi esempi per la classe minoritaria piuttosto che limitarsi a duplicare esempi esistenti, utilizzando la seguente formula:

  `New Sample = Original Sample + Random * (Neighbor - Original Sample)`

  dove *Original Sample* è l'esempio della classe minoritaria, *Neighbor* è uno dei vicini più prossimi dell'esempio e *Random* è un valore casuale compreso tra 0 e 1.

L'approccio basato su queste due tecniche si é rivelato essere fallimentare. Nonostante la loro affidabilitá e capacitá, queste tecniche possono essere utilizzate solo se si é in presenza di un problema di **classificazione**. Queste tecniche, infatti, non sono progettate e sviluppate per affrontare problemi di regressione sbilanciata, perché RandomOverSampler e SMOTE funzionano proprio grazie alla presenza di dati categorizzati.

Nonostante la problematica incontrata, si é provato comunque ad utilizzare le due classi per fare apprendimento supervisionato, adattandole per farle funzionare sul problema di regressione. Un esempio di adattamento é disponibile consultando il file `oversampling.py`, presente nella cartella `src` del progetto.

Algorithm 1: RandomOverSamplerTransformer

```
class RandomOverSamplerTransformer():
    def __init__(self, threshold=25):
        self.threshold = threshold

    def fit_resample(self, X, y):
        y = np.array(y).reshape(-1, 1)

        # Discretize the target variable into different bins
        discretizer = KBinsDiscretizer(n_bins=20, encode='ordinal',
            strategy='uniform')
        y_discretized = discretizer.fit_transform(y).flatten()

        # Count the number of examples per class before oversampling
        occurrences = {item: list(y_discretized).count(item) for item
             in y_discretized}

        # Apply RandomOverSampler
        self.sampling_strategy = {cls: self.threshold for cls, count
            in occurrences.items() if count < self.threshold}
        print("Sampling strategy:", self.sampling_strategy)
```

```
18
19          self.random_oversampler = RandomOverSampler(sampling_strategy
               =self.sampling_strategy, random_state=42)
20
21          return self.random_oversampler.fit_resample(X, y_discretized)
```

In questo file si puó osservare che, per utilizzare la tecnica nei problemi di regressione, é stato necessario creare una nuova classe (nominata `RandomOverSamplerTransformer`), che riscrivesse il metodo `fit_resample` per adattarlo al problema in esame. Ben presto si é deciso di abbandonare questo approccio, data la sua infattibilitá: in seguito a vari tentativi effettuati per implementare questa classe nel modo corretto, ci si é resi conto che l'unico modo per far funzionare la classe era di discretizzare la classe target, `Worldwide Gross`, in un numero prefissato di bins. Questo metodo, quindi, rende il nostro caso di studio un problema di classificazione, e non é il percorso che si desidera seguire.

Tuttavia, si é voluto comunque proseguire per dimostrare l'impossibilitá dell'approccio. La classe appena descritta é stata utilizzata in combinazione con SMOTE e grazie a questa unione é stato possibile addestrare solo per tre dei quattro modelli definiti. In seguito ad una prima fase di addestramento, peró, é stato possibile osservare immediatamente come questa tecnica non sia adatta per i problemi di regressione. Di seguito si riportano i risultati ottenuti con questo tipo di addestramento:

Metric	Score Mean	Score Variance	Score Std
MSE	18.312238744967527	14.33470315096477	3.786119801454356
MAE	3.786047062825933	0.15442387662448256	0.3929680351179757
MSLE	0.09776057703318314	0.0009153756806086289	0.030255176096143102
R2	-4.139586502842493	1.285675398384797	1.1338762711975223

Risultati delle metriche di valutazione per il Decision Tree

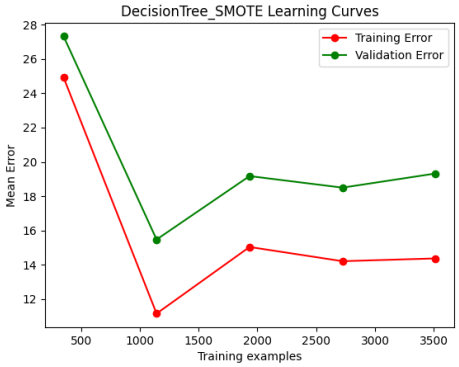
Metric	Score Mean	Score Variance	Score Std
MSE	15.849421095131099	9.779217164147145	3.127173990066294
MAE	3.7774353251578905	0.1416609208782517	0.37637869344352065
MSLE	0.06692415342505283	0.0002781149780941821	0.016676779608011318
R2	-3.4492604628749497	0.8893865987696289	0.9430729551681719

Risultati delle metriche di valutazione per il Random Forest

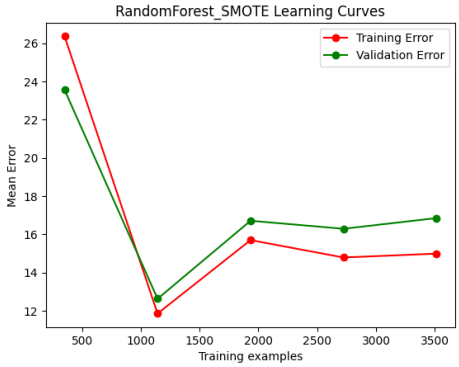
Metric	Score Mean	Score Variance	Score Std
MSE	15.162022643056787	8.944723032265262	2.990772982401918
MAE	3.646898559835037	0.12976192305257714	0.3602248229267066
MSLE	0.06644194733402091	0.0002765537658219121	0.01662990576707854
R2	-3.2559751488405397	0.8100868275280785	0.9000482362229696

Risultati delle metriche di valutazione per l'LGBMRegressor

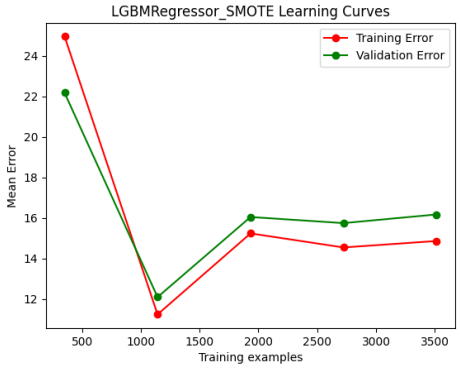
Di seguito sono rappresentate le curve di apprendimento ottenute in seguito all'addestramento dei tre modelli:



DecisionTree_SMOTE Learning Curves



RandomForest_SMOTE Learning Curves



LGBMRegressor_SMOTE Learning Curves

Com'é possibile osservare, gli errori ottenuti sono nettamente superiori rispetto a quanto ottenuto senza over-sampling.

6.2 SMOGN

Non essendo possibile utilizzare RandomOverSampler e SMOTE, si é optato per alternative che fossero adatte per i problemi di regressione. Un esempio é dato da SMOGN. **SMOGN** (acronimo di *Synthetic Minority Over-sampling Technique for Regression with Gaussian Noise*) é una tecnica che rappresenta un'estensione del tradizionale SMOTE, adattandolo ai problemi di regressione. Come SMOTE e RandomOverSampler, il suo compito é quello di bilanciare la distribuzione delle feature target continue, identificando le zone del dataset che rientrano nella minoranza e generando esempi sintetici. A differenza di SMOTE, SMOGN utilizza un'approccio piú avanzato, combi-

nando alla formula usata dal primo metodo un **rumore gaussiano**, utile per evitare di creare esempi troppo simili a quegli esistenti.

6.2.1 Primo approccio: metodo custom di resample e gestione dei valori mancanti

Una prima prova di utilizzo di questa tecnica é visibile nel file `oversampling.py` della cartella `src` del progetto. Si é deciso di creare un metodo apposito, dal nome `smogn_resample_data`, che consente di applicare la tecnica SMOGN ad un dataset ottenuto unendo i dati relativi alle feature di input (senza feature target) e alla feature target.

Algorithm 2: SMOGN Resample Data

```
def smogn_resample_data(X_train, y_train, targetColumn):
    # Convert columns to numeric, if they aren't already
    X_train = pd.DataFrame(X_train)
    X_train = X_train.apply(pd.to_numeric, errors='coerce')
    y_train = pd.to_numeric(y_train, errors='coerce')

    train_data = pd.concat([pd.DataFrame(X_train), pd.Series(y_train,
        name=targetColumn)], axis=1)

    # Preliminary data check
    if train_data.isnull().values.any():
        print("I dati contengono valori nulli. Pulizia dei dati in
            corso...")
        train_data = handle_missing_values(train_data)

    try:
        train_data_resampled = smogn.smoter(
            data=train_data,
            y=targetColumn,
            samp_method='extreme'
        )
    except ValueError as e:
        if "synthetic data contains missing values" in str(e):
            print("Gestione dei valori mancanti nei dati sintetici...
                ")
            train_data = handle_missing_values(train_data)
            train_data_resampled = smogn.smoter(data=train_data, y=
                targetColumn)
        else:
            raise

    X_train = train_data_resampled.drop(columns=[targetColumn]).
        to_numpy()
    y_train = train_data_resampled[targetColumn].to_numpy()

    return X_train, y_train
```

Come é possibile osservare dal codice sopra riportato, in seguito all'ottenimento del dataset venivano effettuati alcuni controlli per verificare la validitá e la correttezza dei dati. Nel caso in cui i dati in input o i dati sintetici creati da SMOGN presentano valori mancanti, viene invocato il metodo `handle_missing_values`, illustrato di seguito, che correggeva il problema imputando i valori mancanti nei dati per le feature numeriche e categoriche. In particolare, la prima correzione viene utilizzata per sostituire i valori mancanti con la **mediana** dei valori presenti nella caratteristica numerica corrispondente, la seconda correzione, invece, sostituisce i valori mancanti con la **moda** dei valori presenti nella caratteristica categorica corrispondente.

Algorithm 3: Trattamento dei valori mancanti

```
def handle_missing_values(data):
    # Separate numeric and categorical features
    numeric_features = data.select_dtypes(include=['float64', 'int64'
        ]).columns
    categorical_features = data.select_dtypes(include=['object']).
        columns

    # Impute median for numeric features
    data[numeric_features] = data[numeric_features].fillna(data[
        numeric_features].median())

    # Impute mode for categorical features
    for feature in categorical_features:
        data[feature] = data[feature].fillna(data[feature].mode()[0])

    return data
```

In seguito ad una prima fase di apprendimento, si é potuto osservare come questo metodo di applicazione della tecnica SMOGN non sia ottimale; sebbene l'apprendimento abbia prodotto risultati accettabili, si tratta comunque di risultati ottimizzabili e migliorabili. Probabilmente la causa dei miglioramenti non ottimali ottenuti é da attribuire ai **valori mancanti** e al metodo con cui sono stati gestiti: il metodo `smogn_resample_data` é stato piú volte raffinato in quanto i dati passati ed elaborati da quest'ultimo risultavano avere valori nulli, comportamento insolito considerando l'assenza di valori nulli nel dataset; ció comportava anche ad una generazione da parte di SMOGN di esempi sintetici contenenti valori mancanti. Nonostante alla fine si sia arrivati ad un'implementazione "stabile", é subito possibile comprendere come questi comportamenti anomali portino i modelli ad avere difficoltá a generalizzare bene con questi dati. Si é scelto quindi di optare per un metodo classico, descritto di seguito.

6.2.2 Secondo approccio: applicazione di SMOGN nella pipeline di trasformazioni

Non potendo fare affidamento su questo metodo, si é scelto di applicare la tecnica di over-sampling al dataset usato per l'addestramento e la valutazione dei modelli. L'approccio si limita a chiamare il metodo `smogn.smoter` per applicare l'over-sampling ai dati di training. I parametri utilizzati nel metodo specificano, oltre ai dati di input e alla colonna target, anche il **metodo di campionamento**, impostato su `extreme` per far focalizzare l'over-sampling sui valori estremi della distribuzione target.

Al termine dell'operazione, viene prodotto un nuovo DataFrame contenente il dataset originale con l'aggiunta dei dati sintetici. Le tabelle ed i grafici rappresentati di seguito mostrano rispettivamente i risultati dell'applicazione di SMOGN e le curve di apprendimento dei modelli con l'applicazione di SMOGN.

Metric	Score Mean	Score Variance	Score Std
MSE	0.09465739372547287	7.340482199869348e-05	0.0085676614077993
MAE	0.17684130649705518	3.888797871880082e-05	0.006236022668239815
MSLE	0.00031608590127124716	7.640992872452106e-10	2.764234590705374e-05
R2	0.9759848684236865	5.2440736674940385e-06	0.0022899942505373323

Risultati delle metriche di valutazione per il Decision Tree con SMOGN

Metric	Score Mean	Score Variance	Score Std
MSE	0.05619186795900688	0.00010260446831397833	0.010129386374009945
MAE	0.11913531449902603	2.370530947126882e-05	0.004868809861893235
MSLE	0.00018295032613225123	1.013278819368953 6e-09	3.1832040766638786e-05
R2	0.9856622928676414	7.080150978905181e-06	0.0026608553096523647

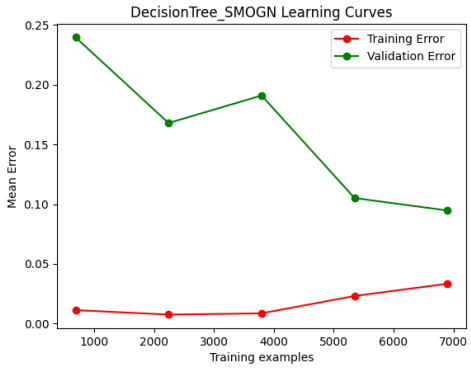
Risultati delle metriche di valutazione per il Random Forest con SMOGN

Metric	Score Mean	Score Variance	Score Std
MSE	0.05077844881065098	2.5853703280766202e-05	0.005084653703131237
MAE	0.12932807575267988	1.029163917381 3822e-05	0.0032080584741886833
MSLE	0.00017133241192861134	4.664055915433583e-10	2.159642543439442e-05
R2	0.987114351544568	1.9710339027243764e-06	0.0014039351490451317

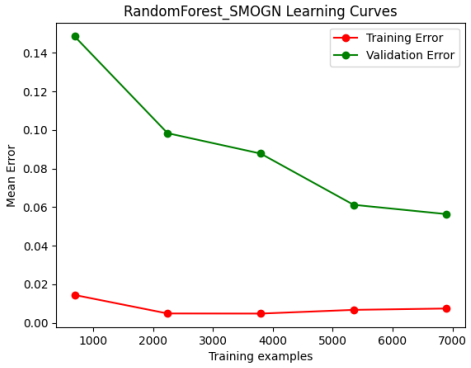
Risultati delle metriche di valutazione per l'LGBMRegressor con SMOGN

Metric	Score Mean	Score Variance	Score Std
MSE	0.057496766445957706	0.00018452295661570877	0.013583922725623433
MAE	0.11796611964504966	2.8008108132949294e-05	0.005292268713222080 5
MSLE	0.0001850930266745437	1.7191265267846192e-09	4.146235071464978e-05
R2	0.9853303505505447	1.2296679764209862e-05	0.003506662197048621

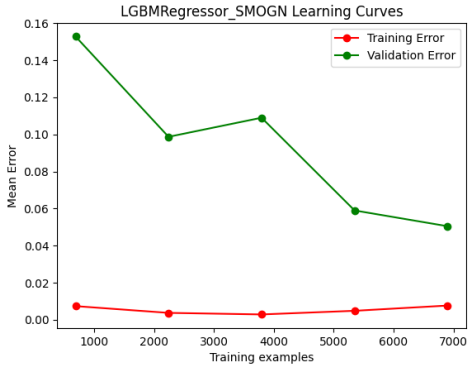
Risultati delle metriche di valutazione per l'XGBRegressor con SMOGN



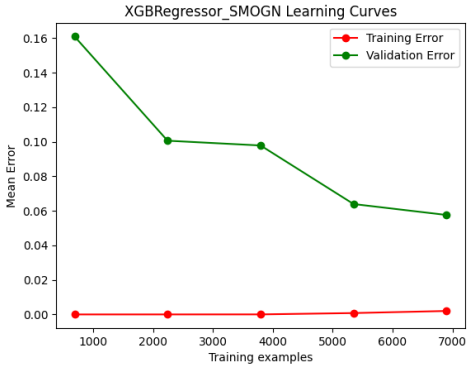
DecisionTree_SMOGN Learning Curves



RandomForest_SMOGN Learning Curves



LGBMRegressor_SMOGN Learning Curves



XGBRegressor_SMOGN Learning Curves

Valutazione: É possibile osservare come l'applicazione della tecnica SMOGN abbia portato ad un peggioramento molto importante. Il **Decision Tree** ed il **LGBMRegressor**, ad esempio, non hanno beneficiato di questo over-sampling, ottenendo un leggero peggioramento che é evidenziato dalle metriche (adesso la metrica R^2 del Decision Tree riesce a spiegare circa il 97,5% della varianza dei dati) ma anche dalle

curve di apprendimento: da un errore minimo di test di 0.03 si é arrivati ad un errore di 0.1. In merito alle curve di apprendimento, é possibile notare inoltre come il loro andamento abbia subito un importante variazione: analizzando la curva del Decision Tree, ad esempio, si puó osservare che a partire dai 4000 esempi la curva di training inizia ad incrementare il suo valore, mentre la curva di test presenta un insolito picco, facendo ritornare l'errore a 0.20, per poi scendere nuovamente dai 5000 esempi in poi. Questo potrebbe indicare un leggero **underfitting**, suggerendo quindi che il modello non risulta abbastanza complesso per individuare pattern o generalizzare correttamente sui dati presentati.

Tale peggioramento vale anche per le metriche e le curve di apprendimento dell'**XGBRegressor**, in cui é possibile osservare un aumento della distanza tra le due curve, oltre ad un aumento degli errori; anche le metriche indicano un leggero peggioramento, con un aumento degli errori piú incisiva rispetto al primo modello di cui abbiamo discusso.

Lo stesso vale per il **Random Forest**. Sebbene i valori di varianza e deviazione standard delle metriche riportano alcuni miglioramenti, le medie non suggeriscono un miglioramento dei modelli con questo over-sampling; anche le curve di apprendimento lo comunicano, in quanto riportano un aumento degli errori di test ed una curva che tende a stabilizzarsi dopo i 5000 esempi. Complessivamente, SMOGN non si é rivelato utile in quanto ha permesso di aumentare la distanza presente tra le curve, introducendo un leggero underfitting per alcuni modelli, ed un aumento degli errori delle varie metriche considerate.

Bisogna sempre ricordare che tali valori sono stati registrati considerando una feature target in scala logaritmica, pertanto i peggioramenti registrati per Decision Tree e XGBRegressor, seppur a prima vista sembrino di piccola intensitá, sono comunque significativi se interpretati su scala originale. Complessivamente, quindi, possiamo affermare che la tecnica SMOGN di over-sampling non si é rivelata utile per bilanciare il dataset squilibrato sulla classe target.

6.3 ImbalancedLearningRegression

Un terzo ed ultimo tentativo é stato eseguito considerando metodi differenti da quelli finora utilizzati. Le tecniche di over-sampling usate sono offerte dalla libreria `Imbalanced Learning Regression`[7], un'implementazione Python di tecniche di campionamento per la regressione. Mediante questa libreria é possibile usufruire sia di tecniche di over-sampling (ad es., Random Over-sampling e SMOTE) che tecniche di under-sampling (ad es., Random Under-sampling). Nel nostro caso, abbiamo voluto sperimentare l'utilizzo delle tecniche **Random Over-sampling**, il cui ragionamento e funzionamento é stato giá discusso in precedenza, e **Random Under-sampling**, tecnica utilizzata per bilanciare le classi in un dataset rimuovendo esempi dalla classe maggioritaria. Quest'ultima tecnica cerca di ridurre la dimensione della classe maggioritaria selezionando alcuni campioni da essa in modo casuale fino a raggiungere un numero comparabile con quello della classe minoritaria.

Cosí come per SMOGN, anche in questi due casi é stato sufficiente richiamare due metodi, rispettivamente `iblr.ro` e `iblr.random_under` (con *iblr* alias di *Imbalanced Learning Regression*), per applicare le due tecniche sul dataset iniziale al fine di ottenere due nuovi DataFrame su cui eseguire addestramento e valutazione. Il metodo di campionamento rispettato é stato fissato anche in questo caso ad `extreme` per confrontare le performance dei vari metodi di over-sampling e under-sampling sotto le stesse condizioni. Di seguito si condividono i risultati ottenuti dapprima con il Random Over-sampling poi con il Random Under-sampling.

Metric	**Score Mean**	**Score Variance**	**Score Std**
MSE	0.011210104564316226	3.0687320532784755e-05	0.005539613753032313
MAE	0.039934724788834475	4.460098656080689e-06	0.0021118945655691926
MSLE	2.682181889317359e-05	1.9714795014452947e-10	1.4040938364102646e-05
R2	0.9972799503177306	1.7125050828108375e-06	0.0013086271748709933

Risultati delle metriche di valutazione per il Decision Tree con Random Over-sampling

Metric	**Score Mean**	**Score Variance**	**Score Std**
MSE	0.005537815505488601	1.6695115942505385e-05	0.004085965729482491
MAE	0.021564217844700546	2.7883398225515937e-06	0.0016698322737782958
MSLE	1.4171757821002504e-05	1.0573730801972428e-10	1.0282864776886074e-05
R2	0.9986242708225748	1.0375155322522219e-06	0.001018585063827377

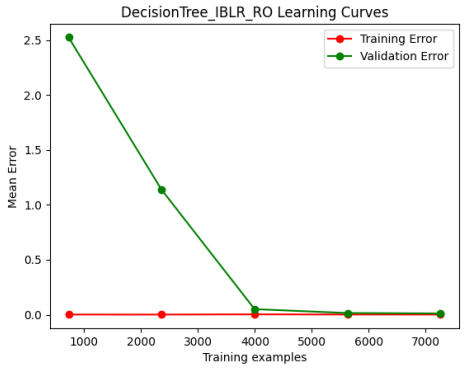
Risultati delle metriche di valutazione per il Random Forest con Random Over-sampling

Metric	**Score Mean**	**Score Variance**	**Score Std**
MSE	0.003739491480682542	5.6336625968755705e-06	0.0023735337783304393
MAE	0.02361215394560008	1.2142157171179912e-06	0.0011019145688836278
MSLE	8.600373317787864e-06	3.0923291972754256e-11	5.560871511980317e-06
R2	0.9990986247720073	2.9826485888366883e-07	0.0005461363006463394

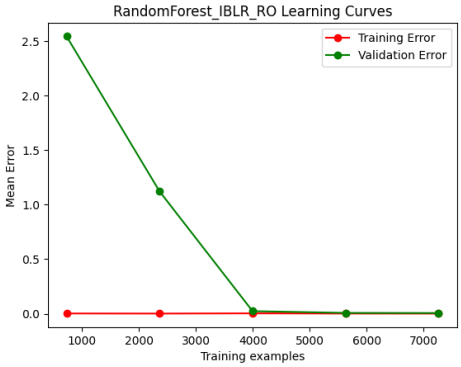
Risultati delle metriche di valutazione per l'LGBMRegressor con Random Over-sampling

Metric	Score Mean	Score Variance	Score Std
MSE	0.005250940738626626	1.2571033832742301e-05	0.003545565375612513
MAE	0.02161971011770492	3.0274049132685687e-06	0.0017399439396913248
MSLE	1.2669942641720654e-05	8.284848564018674e-11	9.102114349984114e-06
R2	0.9987013873161613	7.458360038643618e-07	0.0008636179733333263

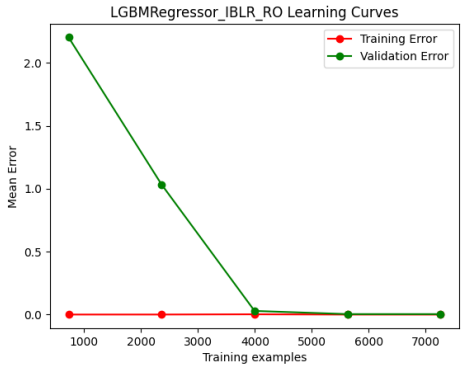
Risultati delle metriche di valutazione per l'XGBRegressor con Random Over-sampling



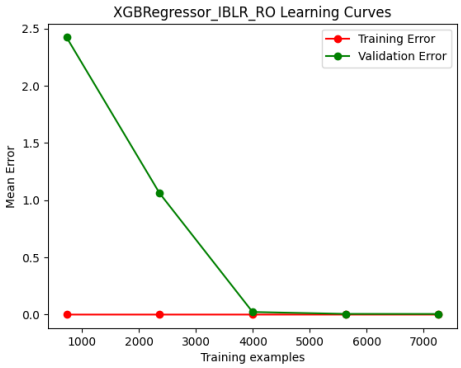
DecisionTree_IBLR_RO Learning Curves



RandomForest_IBLR_RO Learning Curves



LGBMRegressor_IBLR_RO Learning Curves



XGBRegressor_IBLR_RO Learning Curves

Valutazione: Si puó subito notare come la tecnica di Random Over-sampling abbia portato ad un notevole miglioramento delle performance di tutti i modelli esaminati. Osservando le metriche, é immediata la differenza presente tra i valori di media, varianza e deviazione standard originali ed i valori ottenuti dopo aver applicato questa

tecnica. Anche le curve di apprendimento registrano un netto miglioramento: il modello **LGBRegressor**, definito come il miglior modello con i dati non trasformati, si conferma anche in questo caso come modello migliore, con una netta riduzione degli errori di training e di test, che ora definiscono curve nettamente piú vicine tra loro, ed un annullamento dei segnali di underfitting di cui soffriva nel caso della tecnica SMOGN. Anche gli altri modelli non sono da meno: i modelli **Random Forest** e **XGBRegressor** anche con questa tecnica hanno ottenuto un miglioramento, con errori di test iniziali molto alti, ma che si attenuano rapidamente all'aumentare dei dati. Se si considera che tali valori sono relativi ad una scala logaritmica, é possibile comprendere che convertendo questi errori in scala originale si é in grado di ottenere dei risultati decisamente migliori rispetto a quanto osservato inizialmente. Complessivamente, il Random Over-sampling si mostra come la tecnica migliore da applicare al set di dati utilizzato per il nostro caso di studio.

Di seguito si condividono i risultati ottenuti nel caso del Random Under-sampling:

Metric	Score Mean	Score Variance	Score Std
MSE	0.05509551721101078	0.0003703346378838533	0.019244080593363074
MAE	0.13759482234231352	9.150188158262175e-05	0.00956566158624806
MSLE	0.00016690726842190134	4.653204054589428e-09	6.821439770744464e-05
R2	0.982992930462839	4.025140269511098e-05	0.006344399317123015

Risultati delle metriche di valutazione per il Decision Tree con Random Under-sampling

Metric	Score Mean	Score Variance	Score Std
MSE	0.0332668279011767	0.0008620794221705687	0.029361189045584796
MAE	0.06433862802739176	9.540529872928774e-05	0.00976756360252073
MSLE	0.00011170448041594952	9.673635751851468e-09	9.835464275697141e-05
R2	0.9900142208961549	7.13657172579559e-05	0.00844782322601248

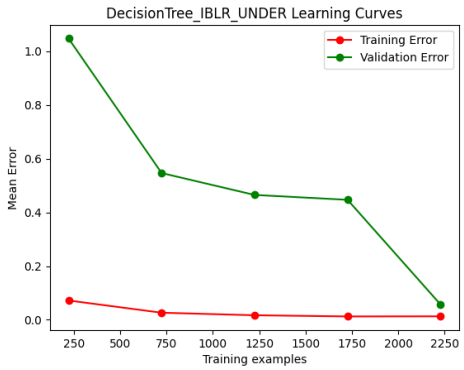
Risultati delle metriche di valutazione per il Random Forest con Random Under-sampling

Metric	Score Mean	Score Variance	Score Std
MSE	0.026142671186032123	0.00013653956249720593	0.011685014441463302
MAE	0.06907662025083368	2.269014889357047e-05	0.004763417774410562
MSLE	9.482694705059408e-05	4.378331226975916e-09	6.616895969392232e-05
R2	0.9919664975072923	1.2768065100729136e-05	0.003573242938946236

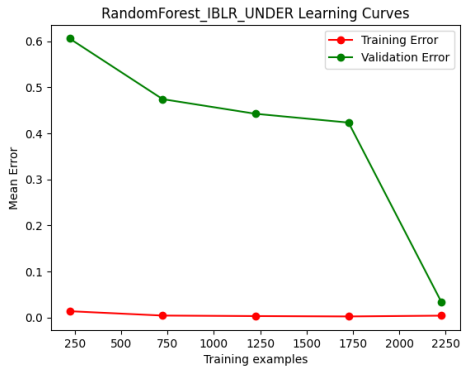
Risultati delle metriche di valutazione per l'LGBMRegressor con Random Under-sampling

Metric	Score Mean	Score Variance	Score Std
MSE	0.037323494432950334	0.00097684646936768	0.031254543179635184
MAE	0.06852954496777014	8.637988918168086e-05	0.009294078178156286
MSLE	0.00012058250019799292	8.717170471493327e-09	9.336578854962521e-05
R2	0.9887732005975242	7.888154449418267e-05	0.008881528274693644

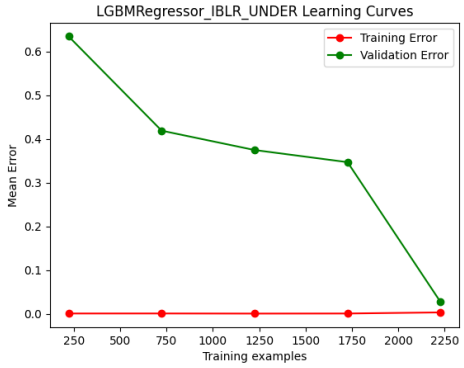
Risultati delle metriche di valutazione per l'XGBRegressor con Random Under-sampling



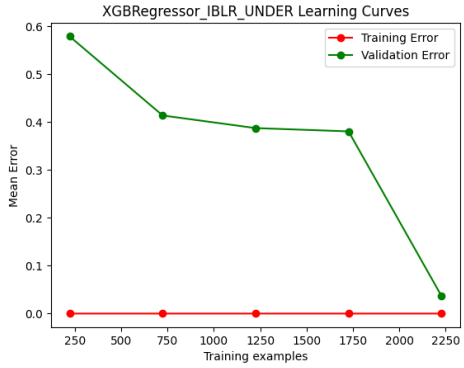
DecisionTree_IBLR_UNDER Learning Curves



RandomForest_IBLR_UNDER Learning Curves



LGBMRegressor_IBLR_UNDER Learning Curves



XGBRegressor_IBLR_UNDER Learning Curves

Valutazione: A differenza del Random Over-sampling, con l'applicazione del Random Under-sampling si registra un peggioramento dei modelli sia in termini di metriche che di curve di apprendimento. É subito possibile notare come ogni modello dopo i 750 esempi inizi a soffrire fortemente di **overfitting**, con errori di test che sono piú

alti rispetto a quelli ottenuti inizialmente ed un errori di training quasi nulli. É quindi possibile confermare come per il nostro caso di studio la tecnica di under-sampling non risulti efficace, o in generale come ridurre il numero di esempi nel nostro caso non porti ad un vantaggio in termini di performance dei modelli.

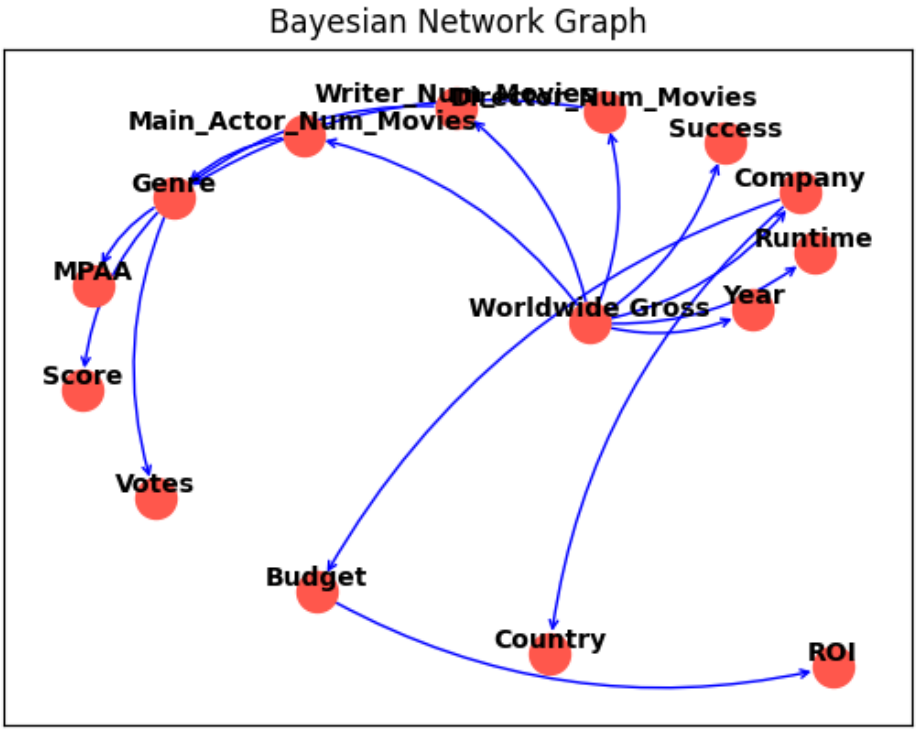
7 Apprendimento Probabilistico

In questa sezione si é deciso di esplorare un tipo di apprendimento differente da quanto provato nelle sezioni precedenti. Nello specifico, si é deciso di provare ad applicare tecniche di **apprendimento probabilistico**, forma di ragionamento che sfrutta la teoria della probabilità (soprattutto dipendenza e indipendenza tra variabili e regola di Bayes) per apprendere una **Rete bayesiana** (o **Belief Network**) al fine di estrarre dal dataset informazioni sulla distribuzione dei campioni in esso e generare nuovi esempi utili per aggiornare le credenze dei modelli riguardano ad un determinato fenomeno.

7.1 Apprendimento della Rete Bayesiana

Un primo passo da compiere per poter implementare l'apprendimento probabilistico é quello di creare ed apprendere la nostra rete bayesiana. Per poter creare la nostra rete bayesiana, é necessario che venga definita la struttura della rete, da esprimere sotto forma di grafo orientato aciclico (DAG). Un primo tentativo é stato quello di apprendere la struttura della rete utilizzando una tecnica nota come `HillClimbSearch`: si tratta di un algoritmo di ricerca locale che cerca di massimizzare una funzione obiettivo (nel nostro caso la massima verosimiliglianza) in relazione al dataset fornito. Si tratta di un algoritmo relativamente semplice ed efficiente, ma non applicabile nel nostro caso: per poter consentire a questo metodo di apprendere una rete bayesiana che simula in modo affidabile la distribuzione dei campioni nel dataset, pur specificando un numero di interazione abbastanza basso (ad es., sono stati fatti tentativi riducendo il numero massimo di interazioni a 4), é richiesta una capacitá di spazio di memoria in RAM molto elevata (con un numero massimo di iterazioni pari a 4 é necessario occupare 66.1 GiB di memoria).

Questa impossibilitá ci ha portato a cercare dei compromessi che, seppur limitando il numero di dipendenze tra variabili, ci permettono di apprendere una rete bayesiana che sia abbastanza robusta per il task di apprendimento; sono state quindi individuate tutte le dipendenze piú importanti e coerenti con il nostro caso di studio, e sono state indicate ed utilizzate per la creazione della rete bayesiana. Di seguito é rappresenta la rete bayesiana ottenuta in seguito alla definizione delle dipendenze tra variabili:



Struttura della rete bayesiana

In seguito alla definizione e all'apprendimento della rete bayesiana, sono state memorizzate le *Conditional Probability Distributions* (o *CPD*) delle variabili in relazione alle dipendenze definite nella fase precedente. Questi valori sono consultabili all'interno del file disponibile al percorso `./resources/logs/bayesian_network/cpd.txt` del progetto. Di seguito si illustrano alcuni esempi ottenuti in seguito all'apprendimento della rete:

Worldwide Gross	Probability
0.0	0.9048
1.0	0.0637
2.0	0.0178
3.0	0.0091
4.0	0.0028
5.0	0.0008
7.0	0.0006
9.0	0.0004

Distribuzione di Probabilità di Worldwide Gross

Runtime	WG(0.0)	WG(1.0)	WG(2.0)
0.0	0.0318	0.0179	0.0
1.0	0.4800	0.3006	0.2234
2.0	0.3646	0.3690	0.3404
3.0	0.1012	0.25	0.3511
4.0	0.0147	0.0417	0.0638
5.0	0.0040	0.0149	0.0213

Conditional Probability Distribution di Worldwide Gross dato Runtime (Parte 1)

Runtime	WG(3.0)	WG(4.0)	WG(5.0)
0.0	0.0	0.0	0.0
1.0	0.25	0.1333	0.25
2.0	0.125	0.1333	0.25
3.0	0.3125	0.4667	0.5
4.0	0.2292	0.2	0.0
5.0	0.0833	0.0	0.0
6.0	0.0	0.0667	0.0
7.0	0.0	0.0	0.0
9.0	0.0	0.0	0.0

Conditional Probability Distribution di Worldwide Gross dato Runtime (Parte 2)

Si puó osservare come `Worldwide Gross` sia una variabile indipendente, di conseguenza il suo valore non dipende dalle altre variabili. É necessario precisare che questi valori sono stati ottenuti in seguito ad una fase di pre-processing delle colonne del dataset. Il procedimento di pre-processing risulta simile a quanto fatto per i task di apprendimento supervisionato e non supervisionato, ma in questo caso si é deciso di discretizzare i valori relativi alle colonne `Budget, Runtime` e `Worldwide Gross`, data l'elevata varianza dei valori.

7.2 Generazione di sample e inferenza probabilistica

Mediante la rete bayesiana appresa é possibile generare degli esempi sintetici, ovvero nuove configurazioni di variabili che possono essere utili per l'addestramento di un modello o per fare inferenza su nuovi dati. Di segutio si vuole presentare un esempio generato dalla rete sfruttando le probabilità apprese:

Runtime	**Year**	**Main_Actor_Num_Movies**	**Country**
2.0	2006	26	United States

Writer_Num_Movies	**Score**	**Company**	**Votes**
6	67	Paramount Pictures	1500000

MPAA	**Director_Num_Movies**	**Budget**	**Genre**
PG-13	6	0.0	Action

Esempio sintetico generato

Generato questo esempio, abbiamo provato a predire l'incasso di questo nuovo film ed i risultati sono mostrati di seguito:

Worldwide Gross	**phi(Worldwide Gross)**
Worldwide Gross(0.0)	0.4155
Worldwide Gross(1.0)	0.4999
Worldwide Gross(2.0)	0.0846
Worldwide Gross(3.0)	0.0000
Worldwide Gross(4.0)	0.0000
Worldwide Gross(5.0)	0.0000
Worldwide Gross(7.0)	0.0000
Worldwide Gross(9.0)	0.0000

Distribuzione di Probabilità Condizionata di Worldwide Gross

Da questi risultati possiamo dedurre che la rete bayesiana appresa ha delle buone performance e rappresenta una buona base per generare esempi credibili ed affidabili.

Un esempio di applicazione della rete bayesiana puó essere quello di generare esempi in grado di supportare i task di apprendimento supervisionato per risolvere i problemi di sbilanciamento e mancanza dei dati e migliorare cosí le performance dei modelli. Ció é avvalorato anche dal fatto che mediante la rete bayesiana appresa é possibile generare esempi che per alcune variabili presentano valori scelti in modo arbitrario: in questo modo, si possono creare nuovi esempi per le classi minoritarie del dataset e aggiungerli alla conoscenza dei modelli di apprendimento. Di seguito si riporta un esempio di query che calcola la probabilitá dell'incasso di un film avente rating MPAA pari ad *NC-17* (utilizzato per delinare contenuto per adulti):

Query: CPD di Worldwide Gross per film con MPAA rating di NC-17

```
inference.query(variables=['Worldwide Gross'], evidence={'MPAA': 'NC-17'})
```

Risultato:

Worldwide Gross	phi(Worldwide Gross)
Worldwide Gross(0.0)	0.9141
Worldwide Gross(1.0)	0.0576
Worldwide Gross(2.0)	0.0159
Worldwide Gross(3.0)	0.0082
Worldwide Gross(4.0)	0.0028
Worldwide Gross(5.0)	0.0007
Worldwide Gross(7.0)	0.0004
Worldwide Gross(9.0)	0.0002

8 Conclusioni

Nel caso di studio presentato abbiamo avuto modo di utilizzare ed approfondire le principali tecniche di apprendimento supervisionato per uno specifico task di regressione: predire il potenziale incasso di un film sulla base delle sue principali caratteristiche. Per lo scopo sono stati impiegati modelli basati su alberi di decisione, che si sono rivelati piú o meno adatti, raggiungendo buone performance per il problema in questione. In particolare, il modello di Gradient Boosted Tree di LightGBM si é rivelato il piú efficace tra quelli esaminati, ma nonostante ció presenta comunque alti margini di miglioramento. In merito al problema appena citato, abbiamo anche discusso dell'efficacia delle tecniche di over-sampling nel nostro dominio di interesse, in particolare dimostrando la buona riuscita dei modelli in seguito all'applicazione del Random Over-sampling sul dataset di partenza. Sono state esplorate anche tecniche differenti di apprendimento, quali l'apprendimento non supervisionato, al fine di suddividere i film in cluster differenti a seconda di caratteristiche simili tra loro; l'applicazione di questa tecnica di hard clustering ha portato a dei risultati accettabili, ma che necessitano ancora di maggiori approfondimenti per comprendere potenziali miglioramenti dell'algoritmo impiegato. Infine, abbiamo creato e appreso una rete bayesiana per consentire la creazione di nuovi esempi e per fare inferenza su questi nuovi dati, in modo tale da comprendere l'efficacia della struttura per generare esempi robusti, utili per fornire un eventuale supporto ai task di apprendimento supervisionato.

8.1 Sviluppi futuri

Di seguito si elencano possibili sviluppi futuri che possono estendere o migliorare ulteriormente il progetto:

1. **Esplorazione delle tecniche di over-sampling per i task di apprendimento non supervisionato**: Le tecniche di over-sampling sono state impiegate solamente nel task di apprendimento supervisionato per risolvere il problema di sbilanciamento dei dati, ma lo steso procedimento non é stato eseguito sul task di apprendimento non supervisionato. L'applicazione delle tecniche utilizzate nel progetto, o ulteriori tecniche e configurazioni che potrebbero risultare migliori rispetto a quelle utilizzate, potrebbe rivelarsi utile per comprendere l'impatto che questi algoritmi potrebbero avere sulla distribuzione dei film nei vari cluster.

2. **Applicazione di tecniche di riduzione della dimensionalitá dei dati**: Sebbene nel task di apprendimento supervisionato si siano esplorati vari modi per ridurre o aumentare il numero di esempi al fine di ridurre lo sbilanciamento della classe target nel dataset utilizzato, non é stato applicato alcun accorgimento per ridurre la dimensionalitá dei dati. Ad esempio, sarebbe possibile ampliare ulteriormente il progetto considerando tecniche quali il *Principal Component Analysis* (*PCA*) per comprendere l'importanza delle feature utilizzate nei vari task

ed eliminare le piú ridondanti e rumorose (*feature selection*) che potrebbero non contribuire significativamente alla predizione del target.

3. **Esplorare modi alternativi per fare feature engineering**: Nel corso di questo caso di studio é stato piú volte evidenziata la differenza sostanziale tra la scala logaritmica impiegata e la scala originale dei valori del dataset, in termini di rappresentazione e calcolo degli errori di training e test dei vari modelli di apprndimento. Potrebbe essere funzionale esaminare modi differenti per fare feature engineering, al fine di esplorare metodi alternativi di rappresentazione dei dati, che possano risultare piú significativi ed efficaci rispetto a quanto provato.

4. **Implementazione di un'interfaccia per l'utilizzo della soluzione**: Dal punto di vista funzionale, sarebbe utile sviluppare un'interfaccia per consentire ad altri utenti di sfruttare queste tecniche di apprendimento e provare a predire l'incasso di un film sulla base di caratteristiche fornite in input.

Riferimenti Bibliografici

[1] Tianqi Chen and Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System*. New York, NY, USA, 2016. URL https://xgboost.readthedocs.io/en/stable/.

[2] imbalanced learn. Over-sampling. Online, 2014-2024. URL https://imbalanced-learn.org/stable/over_sampling.html.

[3] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. The imbalanced-learn developers. *Journal of Machine Learning Research*, Online, 2014-2024. URL https://imbalanced-learn.org/stable/index.html.

[4] David L. Poole and Alan K. Mackworth. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, Cambridge, UK, 3rd edition, 2023. URL https://artint.info/3e/html/ArtInt3e.html.

[5] scikit learn. Metrics and scoring: quantifying the quality of predictions. Online, 2024. URL https://scikit-learn.org/stable/modules/model_evaluation.html.

[6] Yu Shi, Guolin Ke, Damien Soukhavong, James Lamb, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu, Nikita Titov, and David Cortes. *lightgbm: Light Gradient Boosting Machine*, 2024. URL https://github.com/Microsoft/LightGBM. R package version 4.5.0.99.

[7] Wenglei Wu, Nicholas Kunz, and Paula Branco. Imbalancedlearningregression-a python package to tackle the imbalanced regression problem. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 645–648. Springer, 2022. URL https://imbalancedlearningregression.readthedocs.io/en/latest/#.