RELAZIONE PROGETTO DATA AND WEB MINING 2021

TMDB BOX OFFICE PREDICTION

Can you predict a movie's worldwide box office revenue?

RELAZIONE DI

Ferrari Simone

Trolese Giulio

UNIVERSITÀ CA' FOSCARI, VENEZIA

SOMMARIO

INTRODUZIONE 3
Obiettivo del progetto 3
FASE PRELIMINARE 3
Raccolta funzioni utili 3
PRIMA ANALISI DEL DATAFRAME 3
Head, Info e Describe del dataset 3
MANIPOLAZIONE DEL DATAFRAME 3
Rimozione delle features poco utili 3
Manipolazione delle features strutturate 4
Analisi dei valori anomali pre-splitting 4
Split in Train e Test 8
Analisi dei valori anomali post-splitting 8
Preparazione al One-Hot Encoding 8
ESECUZIONE DEGLI ALGORITMI 8
Baseline 8
Decision Tree Regressor 9
Ensemble Methods su Tree Regressor 9
Bagging 9
Boosting 10
Random Forest 10
CONCLUSIONI FINALI 11
Valutazione dei risultati 11

INTRODUZIONE

Obiettivo del progetto

Il progetto di quest'anno si basa su una **competizione Kaggle**, in particolare la TMDB Box Office Prediction.
Ci vengono forniti dati su oltre **3000 film** del passato presi dal The Movie Database con lo scopo di **predirre** il loro **incasso**.

FASE PRELIMINARE

Raccolta funzioni utili

Tutte le **funzioni utilizzate** nel nostro progetto sono **raccolte** e opportunamente **commentate** in questa sezione del notebook: in questo modo risulta semplice ritrovarle nel documento.

PRIMA ANALISI DEL DATAFRAME

Head, Info e Describe del dataset

Come prima cosa **analizziamo** il **dataframe**, valutando quali **features** possono essere **utili** per il nostro scopo e quali, invece, possono essere **rimosse**.

Da questa analisi, **osserviamo che**:

- Il dataframe comprende **23 colonne** (features), di cui alcune sembrano poco utili al nostro scopo;
- Alcune features sono **dati strutturati**, comprendendo più informazioni all'interno della stessa colonna. Esempi sono `belongs_to_collection`, `genres`, ...
- Alcuni **valori** sono **sospetti**, ad esempio:
  - Alcune features hanno valori **NaN**
  - Il minimo `budget` è pari a **0**... ma un film può avere budget 0? Poco probabile.
  - Il minimo di `runtime` è pari, nuovamente, a **0**... anche in questo caso, è poco probabile, in quanto un film sicuramente dura più di 0 minuti!
  - Ha senso una `revenue` pari ad **1**? Non troppo...
  - Osserviamo che `status` ha solo due possibili valori
  - Si osserva invece che `id` e `imdb_id` hanno valori **univoci**

MANIPOLAZIONE DEL DATAFRAME

Rimozione delle features poco utili

ID e IMDB_ID

`id` e `imdb_id` hanno **valori univoci**, per cui possono essere rimossi: non hanno rilevanza nella predizione.

Titoli dei film

Gli `original_title` e `title`, nella maggior parte dei casi, sono **valori univoci** (`2975/3000` e `2969/3000`), mentre quei pochi **titoli duplicati** sono molto probabilmente **incorrelati** tra di loro.
Possiamo quindi rimuoverli dal nostro dataframe.

Homepage, PosterPath e Status

Le `homepage` e i `poster_path` risultano essere **features** da cui è **difficile estrapolare informazioni utili**.
Possiamo quindi rimuoverle.

Lingua originale

L'`original_language` la riteniamo **poco utile** ai fini di predirre la revenue, in quanto sono molto più **d'interesse le lingue** in cui un film è stato **tradotto**: contenute in `spoken_languages`.

Overview e Tagline

Le features `overview` e `tagline` sono di tipo **testuale** e sono **complesse** da **analizzare**.
Le rimuoviamo, in quanto, per i nostri scopi, l'insieme delle `keywords` ci può essere di **sufficiente aiuto**.

Manipolazione delle features strutturate

Come accennato qui sopra, nel dataframe sono presenti alcune **features strutturate**.
In particolare, esse sono: `belongs_to_collection`, `genres`, `production_companies`, `production_countries`, `spoken_languages`, `keywords`, `cast` e `crew`.
Di seguito **spieghiamo le modifiche apportate**.

Come sono strutturate tali features?

Prima di tutto **analizziamo tali features**, e da tale analisi emerge che:

- Tutti i dati strutturati sono **composti da più valori**, ma tutti hanno `name` in comune.
  Decidiamo quindi di **tenere solamente il campo** `name` in quanto è sufficiente a discriminare i valori.

Modifica dei dati strutturati

Modifichiamo i **dati strutturati** in **liste** mantenendo solo il campo `name`, in questo modo è molto più **semplice gestirle**.

Modifica del cast

Analizzando il `cast` si nota che ha un **ordine** (campo `order`).
Ci siamo chiesti: **quest'ordine significa qualcosa?** Gli attori con ordine più basso sono gli **attori principali?**
Prendendo alcuni **campioni di film** a noi **noti**, abbiamo visto che **questa nostra teoria sembra corretta**.
Riteniamo che il **nome** dell'attore sia **sufficiente** per aiutarci nel nostro scopo.
Modifichiamo quindi la features `cast` estrapolando **solo il nome** degli attori e mantenendo i **5 attori più rilevanti** per ogni film, in quanto attori di ordine alto probabilmente hanno poca rilevanza nel film.

Modifica della crew

Dall'analisi della `crew` notiamo che ogni membro della crew ha un **dipartimento** (`department`).
Fra le varie tipologie, **notiamo** sicuramente **il regista** (`director`).

Riteniamo che il **nome del regista sia significativo**, per cui lo manteniamo, mentre **il resto della crew** la codifichiamo in un **numero decimale** indicante il numero di componenti della crew (`crew_count`), in quanto risulterebbe **troppo espansivo** mantenere una colonna per ogni dipartimento.

Analisi dei valori anomali pre-splitting

Arrivati a questo punto, il nostro dataframe è molto più maneggevole, ma spesso non contiene dati corretti.
Di seguito analizziamo le singole features rimaste, visualizzando i dati ed eventualmente sistemandoli includendo dati presi dall'esterno o applicando operazioni costati.

Revenue

Analizzando la `revenue` abbiamo **osservato che**:

- **Nessun** film ha valore di `revenue` pari a `NaN`
- `57` film hanno valore di `revenue` veramente **basso**.
  Esplorando i dati e **cercando** informazioni **sul web**, abbiamo notato che alcuni film hanno una `revenue` bassa in quanto sono scritti in **unità diverse** (migliaia, milioni...) mentre altri hanno `revenue` realmente basse.
  Ad esempio, alcuni film con `revenue` pari a `500` stanno ad indicare `500.000`, altri film con `revenue` ad esempio pari a `200` invece indicano una effettiva `revenue` di `200`!!
  C'è un chiaro problema di unità di misura, ma, in generale, abbiamo notato un numero maggiore di valori bassi che realmente sarebbero alti, e solitamente i valori espressi in al più **3 cifre** sono considerati in **milioni**: per cui abbiamo deciso che i **valori sotto** la soglia di `999` debbano essere **moltiplicati** per `1.000.000`.

Belongs to collection

Analizzando il `belongs_to_collection` abbiamo **osservato che**:

- **Nessun** film ha valore di `belongs_to_collection` pari a `NaN`
- `2396` film non hanno **alcun valore** di `belongs_to_collection`.

Non potendo sapere se la mancanza di valori è dovuta ad errori o se realmente i film non appartengono ad alcuna collezione, siamo costretti a mantenere invariati i dati.

Dato **l'elevato numero di empty**, riteniamo più utile conoscere il **numero di collections** (solitamente una sola) di cui il film fa parte rispetto a sapere in quali collections: questo per vari motivi, il principale è la difficoltà di gestione dei dati, ad esempio, applicando **one hot encoding**, si verrebbero a creare **molte colonne, spesso sparse**.

Budget

Analizzando il `budget` abbiamo **osservato che**:

- **Nessun** film ha valore di `budget` pari a `NaN`
- `830` film hanno **valore** di `budget` veramente **basso**.
  Come per le `revenue`, abbiamo indagato su tali valori, notando che a volte il valore segnato è espresso in migliaia, altre volte invece non abbiamo trovato informazioni.
  Non potendo ottenere maggiori informazioni, abbiamo deciso di applicare la **mediana** ai valori inferiori alla soglia di `999`. Tale operazione **verrà svolta successivamente allo split in train e test**.

Genres

Analizzando i `genres` abbiamo **osservato che**:

- **Nessun** film ha valore di `genres` pari a `NaN`
- `7` film non hanno **alcun valore** di `genres`.

Si può notare una forte popolarità di alcuni generi: il che è poco significativa.
Si potrebbe pensare di mantenere solo i **generi meno frequenti**, ma ciò porterebbe, eseguendo **one hot encoding**, ad avere **colonne** molto **sparse**: ciò non ci piace.
Decidiamo quindi di **eliminare la features** in quanto il genere di un film probabilmente non influisce sulla revenue.

Popularity

Analizzando la `popularity` abbiamo **osservato che**:

- **Nessun** film ha valore di `popularity` pari a `NaN`
- `20` film hanno **valori** di `popularity` estremamente **alte** rispetto la media.
  Analizzando i film con tali valori, notiamo che molti di questi film appartengono a collezioni di grande notorietà, ad esempio Pirati dei Caraibi, The Avengers, The Maze Runner... quindi riteniamo che una popolarità così elevata sia un **segno distintivo e corretto**: ne segue che non apportiamo modifiche.

Production Companies

Analizzando le `production_companies` abbiamo **osservato che**:

- **Nessun** film ha valore di `production_companies` pari a `NaN`
- `156` film hanno **valori** di `production_companies` **mancanti**.
  Non abbiamo potuto applicare miglioramenti.

Analizzando il numero di `production_companies` ci accorgiamo che sono assolutamente **troppe**...
Non potendo applicare **one hot encoding** (verrebbero troppe **colonne sparse**), dobbiamo ridurre di molto!
Abbiamo deciso di mantenere le **most-significative** fra tutti le compagnie, in modo da ottenere una lista delle compagnie più frequenti nel dataframe (che, secondo noi, sono anche le più famose, tranne alcune eccezioni)
Per ogni film, si mantiene il numero di compagnie **most-significative** che hanno prodotto il film.

Production Countries

Analizzando le `production_countries` abbiamo **osservato che**:

- **Nessun** film ha valore di `production_countries` pari a `NaN`
- `55` film hanno **valori** di `production_countries` **mancanti**.

Analizzando il numero di **valori univoci**, notiamo che le frequenze di `production_countries` indicano una forte prevalenza di stati famosi nel mondo del cinema.

Sono presenti, poi, **ridotte frequenze per stati più "rari"**, che assumiamo essere un indicatore più significativo. Riteniamo plausibile che i film prodotti in uno stato "poco presente" nel mondo del cinema possano avere caratteristiche più simili rispetto a molti film prodotti, ad esempio, in america.
Per cui abbiamo deciso di **scartare** tutte le `production_countries` **poco significative**, ovvero con una **occorrenza maggiore** di `50`.

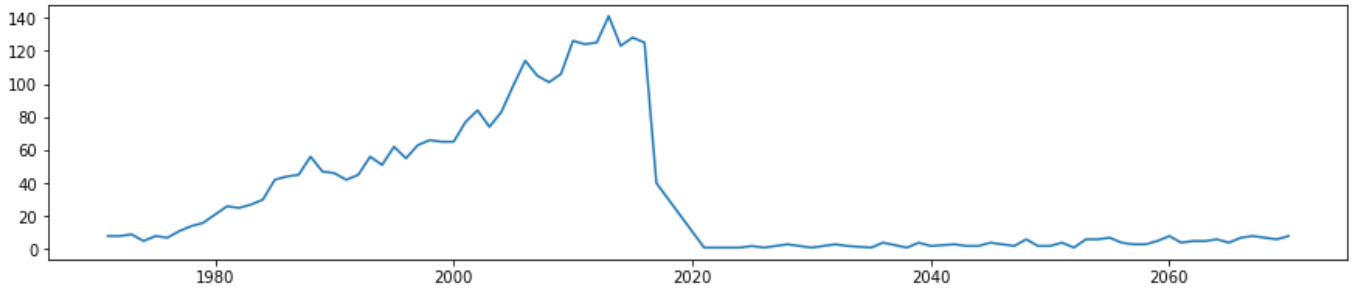
Dopo di che, una volta ottenuta la lista delle `production_countries` più significative, abbiamo **inserito una colonna** `production_countries_most_significative` contenente un **valore binario** indicante se la `production_countries,` del relativo film, **appartiene o meno** a **tale lista**.
In questo modo **si tiene traccia se il film è stato prodotto in una nazione "significativa"** oppure no.

Questa scelta è stata presa per non far esplodere il dataframe eseguendo one hot encoding delle `production_countries`.

Release Date

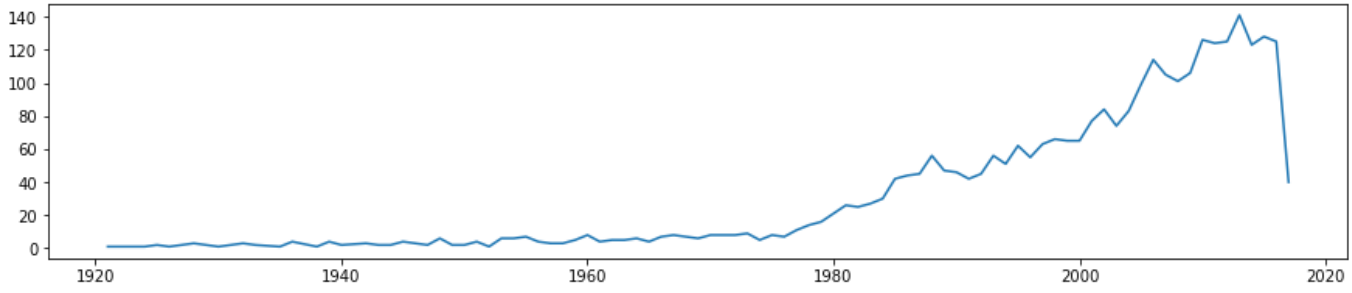
La data di rilascio di un film, nel nostro dataframe, è rappresentato nella forma "`gg/mm/aa`": non è molto utile.
Abbiamo quindi deciso di **suddividere** la data in **diverse colonne**: `year_release`, `month_release`, `week_of_year_release`, `day_of_week_release`.

Dal grafico dell'`year_release` notiamo che sono presenti film con **anno** di **rilascio superiore all'anno attuale**: indaghiamo.



Cercando gli `imdb_id` di un numero sufficiente di film, con data maggiore del 2018, si può notare che le date sono **riferite** al **1900** e **non** al **2000**.
Ad esempio, il primo film con `imdb_id=tt0059418` è stato rilasciato nel 1965 e non nel 2065.
(https://www.imdb.com/title/tt0059418/)

Non potendo controllare tutti i film manualmente, assumiamo che tutti i film con data superiore al 2018 sono in realtà riferiti a 100 anni prima, per cui **modifichiamo tali dati sottraendo 100 anni.**



Runtime

Analizzando le `runtime` abbiamo **osservato che**:

- `2` film ha valore di `runtime` pari a `NaN`
  Li abbiamo sistemati a mano.
- `12` film hanno valori di `runtime` pari a 0.
  La sistemazione dei valori sostituendo gli 0 con la **mediana** viene fatta **successivamente lo spitting in train e test.**

Spoken languages

Analizzando le `spoken_languages` abbiamo osservato che:

- `0` film ha valore di `spoken_languages` pari a `NaN`
- `20` film hanno valori mancanti di `spoken_languages`.

Il numero di **valori univoci** è pari a `56`: non troppo elevato, ma riteniamo **più utile** conoscere il **numero di lingue** in cui il film è tradotto rispetto a sapere in quali lingue.
Quindi convertiamo le `spoken_languages` in **valore numerico rappresentante il numero** di `spoken_languages`.

Keywords

Analizzando le `keywords` abbiamo **osservato che**:

- `0` film ha valore di `keywords` pari a `NaN`
- `276` film hanno valori mancanti di `keywords`.

Non applichiamo alcuna manipolazione, per il momento.

Cast

Dato **l'elevatissimo numero** di attori contenuto nel cast, siamo obbligati ad applicare qualche modifica.
Idea:

- Di ogni film abbiamo già **tenuto solo i primi 5** (i più importanti, come già analizzato in precedenza)
- Si mantengono i "**most-significative**" fra tutti gli attori selezionati, in modo da ottenere una lista degli attori più frequenti nel dataframe (che, secondo noi, sono anche i più famosi, tranne alcune eccezioni)
- Per ogni film, si mantiene il **numero di attori** `cast_most_significative` che hanno **recitato** in quel film

Director

Come per gli attori, vogliamo **creare una lista di registi più frequenti** (quelli con almeno 5 film, secondo noi, sono anche i più famosi, tranne alcune eccezioni) e tenere, per ogni film, una **variabile binaria** indicante se il `director` del film è **tra quelli più importanti** oppure no.

Idea:

- Di ogni film abbiamo già tenuto il `director`
- Si mantengono i "**most-significative**" fra tutti i registi selezionati, in modo da ottenere una lista dei registi più frequenti nel dataframe
- Per ogni film, si **mantiene** il il **valore binario** `director_most_significative` che **indica** se il **regista** è tra quelli **importanti**.

Riepilogo delle features

Dopo aver effettuato le **manipolazioni** qui sopra descritte, nel nostro dataframe **sono presenti le seguenti features**: `belongs_to_collection`, `budget`, `popularity`, `runtime`, `spoken_languages`, `keywords`, `revenue`, `year_release`, `month_release`, `week_of_year_release`, `day_of_week_release`, `cast_count`, `crew_count`.

Inoltre, sono state **aggiunte** le seguenti **features**: `production_companies_most_significative`, `production_countries_most_significative`, `cast_most_significative`, `director_most_significative`

Split in Train e Test

Arrivati a questo punto, **splittiamo** il **dataframe** in **train** e test.
Abbiamo quindi splittato, al momento **senza validation**, in `67/33`

Analisi dei valori anomali post-splitting

Le seguenti **modifiche** devono essere **applicate dopo lo splitting** in quanto altrimenti si andrebbero a **modificare** i dati del dataset di training e test in **modo non indipendente** uno dall'altro, andando a compromettere le valutazioni.

Budget

Come anticipato in precedenza, andiamo a **sistemare** i valori di `budget` anomali applicando la **mediana** a tutti i valori inferiori a `999`.
Tale operazione deve essere svolta sia sul dataset train che su quello di test.

Runtime

In precedenza abbiano notato dei valori di `runtime` pari a 0: andiamo a **sistemare** i valori di `runtime` pari a `0` applicando la **mediana** a tutti i valori inferiori a `999`.

Preparazione al One-Hot Encoding

Attualmente **l'unica colonna non numerica** è `keywords`: risulta essere un problema per l'esecuzione degli algoritmi di predizione.
Un modo per risolvere tale problema è eseguire **One Hot Encoding**, ma, il risultato finale sarebbe di troppe colonne: dobbiamo capire come gestirle.

Keywords

Dato l'elevato numero di `keywords` (`7400` in totale), siamo costretti a trovare un modo per ridurle.
L'idea è quindi di **mantenere** solo le `keywords` **meno presenti**, ovvero le più **significative**, ma notiamo che anche mantenendo le `keywords` con solo `2` occorrenze, il **numero rimane elevato**.
Siamo quindi obbligati a **scartare la feature** `keywords`, in quanto eseguire one hot encoding, oltre ad introdurre **molte colonne**, le renderebbe anche **molto sparse**: solo 2 entry!

Un'alternativa sarebbe mantenere le "**most-common**", ma più una `keyword` è frequente e meno significativa è, per cui non ha senso.
Si potrebbe provare un miglioramento raccogliendo le `keywords` in gruppi per **semantica**, ma è un compito troppo dispendioso.

ESECUZIONE DEGLI ALGORITMI

Baseline

Eseguiamo una **prima predizione** senza l'applicazione di alcun modello per **impostare** una **baseline**.

Idea: basandoci su delle nostre **ipotesi**, la revenue potrebbe essere espressa dalla **seguente formula**:

$$Revenue = Budget * 1,5 * popularity$$

In quanto ci aspettiamo che **un film**, come minimo, **guadagni** almeno il **50% del budget** e che la **popolarità** del film **incida** sulla revenue finale.

Eseguendo tale predizione, otteniamo un RMSLE di circa `2.97`

Decision Tree Regressor

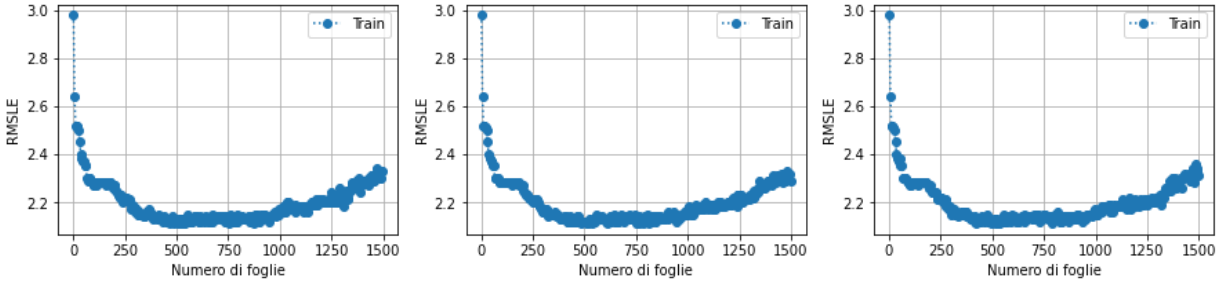
Siamo finalmente pronti per provare un **primo algoritmo**!
Andiamo quindi a **costruire** un modello `DecisionTreeRegressor` **senza specificare alcun parametro** ed eseguiamolo.
Come **metodo di giudizio** dell'errore abbiamo **utilizzato l'**`RMSLE` in quanto **meno sensibile agli outlier**.
Il risultato non ci soddisfa pienamente.

Al momento abbiamo eseguito un primo algoritmo basato sugli **alberi di decisione**.
Sugli alberi è possibile fare **fine tuning** di alcuni parametri, ad esempio sul **numero massimo di foglie** specificate.

Questa decisione, però, non può essere presa sul dataset di test.
Effettuiamo quindi un **ulteriore splitting**: **train**, **validation** e **test**.

Fine tuning del numero massimo di foglie

Passiamo quindi alla fase di **fine tuning** dei parametri: splittiamo il dataset utilizzando **k-fold cross validation**, per ottenere una separazione più imparziale.
Rieseguiamo lo splitting in train e test aggiungendo anche il **validation** cambiando le percentuali di splitting, in modo da avere un **60/20/20**.



Notiamo che, eseguendo diverse volte k fold cross validation su un numero di foglie che va da `2` a `1500`, otteniamo l'**RMSLE minimo** quando il numero **massimo di foglie** è attorno al `500` (ovviamente questo numero varia data la casualità del sample di divisione in train e validation).
Possiamo quindi **scegliere la media tra i 3 valori**, in quanto vediamo che i grafici sono molto piatti nella zona d'interesse, quindi la varianza tra `400` e `900` è abbastanza costante.

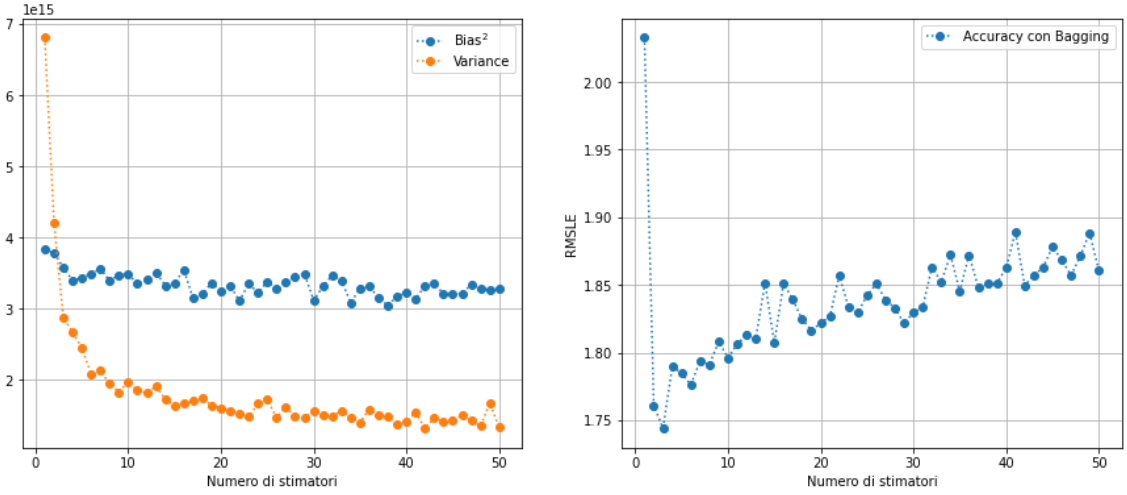
Considerazione post fine tuning

Notiamo che **eseguire il decision tree con il numero di foglie scelto** ci da una **miglior performance** rispetto all'algoritmo senza nessun numero di foglie specificato. Possiamo ritenerci soddisfatti.

Ensemble Methods su Tree Regressor

Bagging

Proviamo a usare l'algoritmo di **bagging** per vedere se riusciamo a **diminuire** il **RMSLE**.
Eseguiamo inoltre fine tuning sul numero di stimatori.



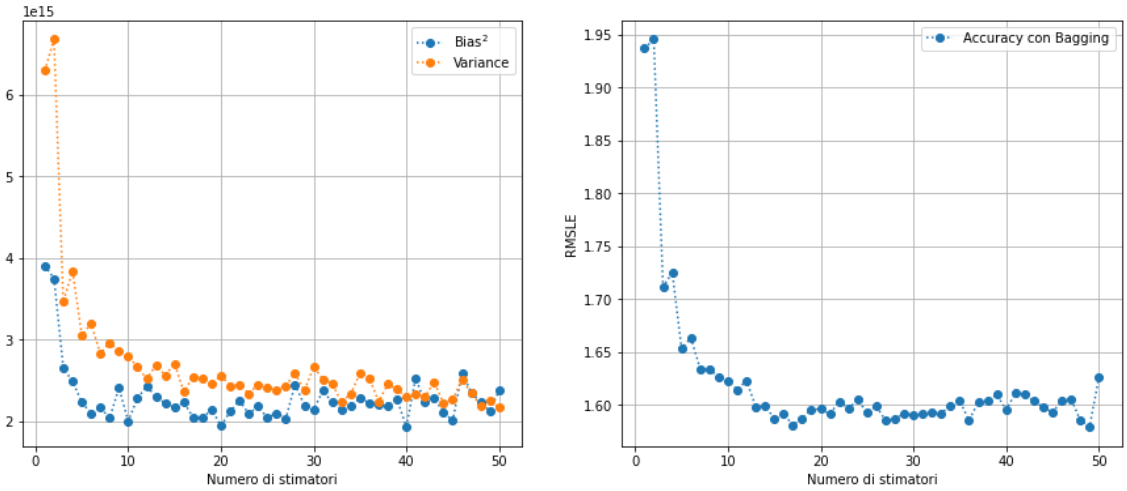
Dal **primo plot** vediamo che il **bagging** effettivamente **abbassa** la **varianza**, come ci aspettavamo.

Dal **secondo plot** notiamo che **l'aumento** degli **stimatori** comporta un **miglioramento dell'accuratezza**, ma solo per valori bassi.
Infatti, inserendo un numero maggiore di 5 stimatori l'RMSLE aumenta!
Usiamo quindi 5 stimatori.

Possiamo notare che, nel nostro caso, **il bagging non porta notevoli miglioramenti:** il valore di RMSLE, rispetto al tree regressor, è diminuito...
Apprezziamo comunque che la **varianza viene ridotta**, per cui i risultati dovrebbero essere più stabili.

Boosting

Proviamo a usare l'algoritmo di **boosting** per vedere se riusciamo a **diminuire** il **RMSLE**.



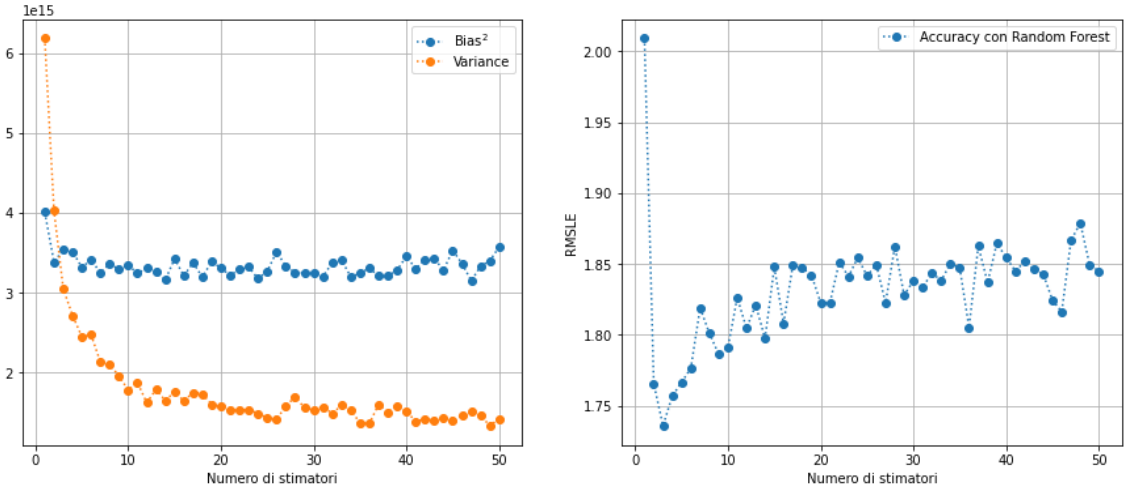
Dal **primo plot** vediamo che il **boosting** effettivamente **abbassa il bias**, come ci aspettavamo.
Inoltre, anche la varianza ne trae beneficio.

Notiamo dal **secondo plot** che **l'aumento** degli **stimatori** comporta un **miglioramento dell'accuratezza**.
Si può inoltre notare che inserire un numero maggiore di 20 stimatori non porta grandi miglioramenti, anzi, aumenta l'RMSLE.
Usiamo quindi 15 stimatori.

A **differenza del bagging**, con il boosting **abbiamo un miglioramento**: l'errore si è abbassato in modo significativo.
Inoltre, con il boosting, otteniamo un algoritmo più stabile in quanto vengono ridotti sia Bias che Variance.

Random Forest

Proviamo a usare l'algoritmo di **random forest** per vedere se riusciamo a **diminuire** il **RMSLE**.



Dal **primo plot** vediamo che **random forest** effettivamente **abbassa notevolmente** la **varianza** ed ha un **leggero effetto** sul **bias**.

Notiamo dal **secondo plot** che **l'aumento** degli **stimatori** comporta un **miglioramento dell'accuratezza**.

Si può inoltre notare che inserire un numero maggiore di circa 5 stimatori non porta grandi miglioramenti, anzi, aumenta l'RMSLE.
Usiamo quindi 4 stimatori.

A **differenza** del **bagging** , con il **random forest** abbiamo, in media, un **miglioramento**: l'errore si è abbassato in modo leggermente significativo.
A **differenza** del **boosting**, invece, con il random forest abbiamo un **peggioramento**: l'errore si è alzato in modo significativo

CONCLUSIONI FINALI

Valutazione dei risultati

Arrivati a questo punto, **abbiamo eseguito 4 algoritmi**:

Tree Regressor
Tree Regressor con **Bagging**
Tree Regressor con **Boosting**
Random Forest Regressor

Andiamo ora a **valutare quale** algoritmo si comporta in modo **migliore** con i nostri dati.

```
Algoritmo: Decision Tree        RMSLE: 2.1564
Algoritmo: Bagging              RMSLE: 2.1209
Algoritmo: AdaBoost             RMSLE: 1.9983
Algoritmo: Random Forest        RMSLE: 2.1434
```

Osservazioni:
Possiamo quindi concludere che, per il nostro dataframe, l'algoritmo **AdaBoost** sembra essere il **migliore**.
I **restanti algoritmi**, invece, sembrano spesso fornire **risultati simili**, anche se spesso il **random** forest fornisce **risultati migliori** rispetto il **Decision Tree Regressor** standard **e** quello con **Bagging**.
Infine, il **bagging** sembra **peggiorare** la nostra **predizione** rispetto al modello più semplice.