Alma Mater Studiorum - Università di Bologna
LM Informatica

Report Data Anlytics

Francesco Saverio Beccafichi (1052139)

A.A. 2022/2023

Indice

Capitolo 1

Introduzione

1.1 Introduzione

Il seguente progetto è stato sviluppato per l'esame "Data Analytics" A.A. 2022/2023, della Laurea Magistrale in Informatica.

L'obiettivo del progetto è quello di realizzare uno studio di data analytics che preveda, l'implementazione di tutte le fasi della pipeline analitica osservata durante il corso:

- Data Acquisition
- Data Visualization
- Data Preprocessing
- Modeling
- Evaluation.

Lo scopo del progetto è di predire il voto medio di un film a partire dalle sue caratteristiche. Il dataset utilizzato è MovieLens [1], un reccomendation system con oltre 60000 film, con rating e tag annessi. I dati sono stati raccolti da circa 150000 utenti tra il 1995 e il 2019. Ogni film, inoltre, ha associato un genoma che identifica una specifica caratteristica del film (ad esempio "Zombies") e la sua rilevanza, che può variare in un range tra 0 ed 1, dove 1 indica che la caratteristica è molto rilavante per il film e 0 il contrario.

Per raggiungere questo obiettivo sono state utilizzate diverse tecniche di Machine Learning supervisionato, tra cui quelle tradizionali, quelle basate su reti neurali ed, infine, Tabnet, un modello deep per dati tabulari.

Capitolo 2

Data-Acquisition

In questa fase l'attenzione del progetto è stata rivolta alla raccolta dei dati che verranno analizzati. L'aquisizione può avvenire in diversi modi, per lo sviluppo del progetto è stata scelta l'aquisizione statitica. Il dataset **MovieLens 25M** è strutturato in sei differenti file:

- ***genome-score.csv***: ogni riga del file rappresenta la rilevanza di ogni film. Contiene l'ID del film, l'ID del tag e la rilevanza di ogni tag.
- ***genome-tag.csv***: contiene i tag utilizzati per descrivere il film nel file "genome-score.csv". Ogni riga del file rappresenta un tag e contiene l'ID del tag con la sua descrizione.
- ***links.csv***: contiene i link tra gli ID del dataset Movielens e gli ID dello stesso film su altri siti web. Ogni riga del file rappresenta un film e contiene l' ID del film su MovieLens, l'ID del film su MDb e l'ID del film su TMDb.
- ***movies.csv***: contiene tutte le informazioni che riguardano i film, tra cui titolo, anno di uscita e genere. Ogni riga del file rappresenta un film e contiene l'ID del film, titolo con il relativo anno di uscita e genere.
- ***ratings.csv***: contiene le valutazioni degli utenti per i film. Ogni riga del file rappresenta una valutazione, contiene l'ID dell'utente, ID del film, la valutazione effettuata dall'utente e la data della valutazione.
- ***tags.csv***: contiene le etichette aggiunte dagli utenti ai film. Ogni riga rappresenta un'etichetta e contiene l'ID del film, ID dell'utente e l'etichetta.

Per l'analisi effettuata si sono tenuti in considerazione solo i file *movies.csv*, *genome-tag.csv*, *genome-score.csv* e *ratings.csv*. I dati sono stati unificati in un singolo file, in modo da ottenere un file dove ad ogni film sia associato la rilevanza di ogni tag ad esso associato. Per ogni film è stata calcolata la valutazione media delle recensioni degli utenti, ottenendo dei valori continui, per questo motivo si è deciso di affrontare questo problema come uno di regressione.

Capitolo 3

Data-Visualization

In questa sezione si mira ad avere grafici e rappresentazioni visive in modo da comunicare i dati in maniera chiara ed efficace. Questa parte del lavoro è utile per capire i dati in maniera efficiente e per provare a scoprire alcune relazioni. Siccome il dataset riporta un alto numero di features si è deciso di rappresentare visivamente solamente alcune informazioni chiave, rendendo più agevole la comprensione e l'analisi dei dati.

3.0.1 Distribuzione voti film

Per ognuno dei 13817 film il voto medio è rappresentato da un numero decimale che varia tra 0 e 5.

Di seguito si può osservare un grafico(Figura 3.1) delle densità che rappresenta la distribuzione dei voti all'interno del dataset. Il grafico serve ad evidenziare aree di maggiore e minore densità di dati, permettendo di individuare eventuali zone di concentrazione dei dati.

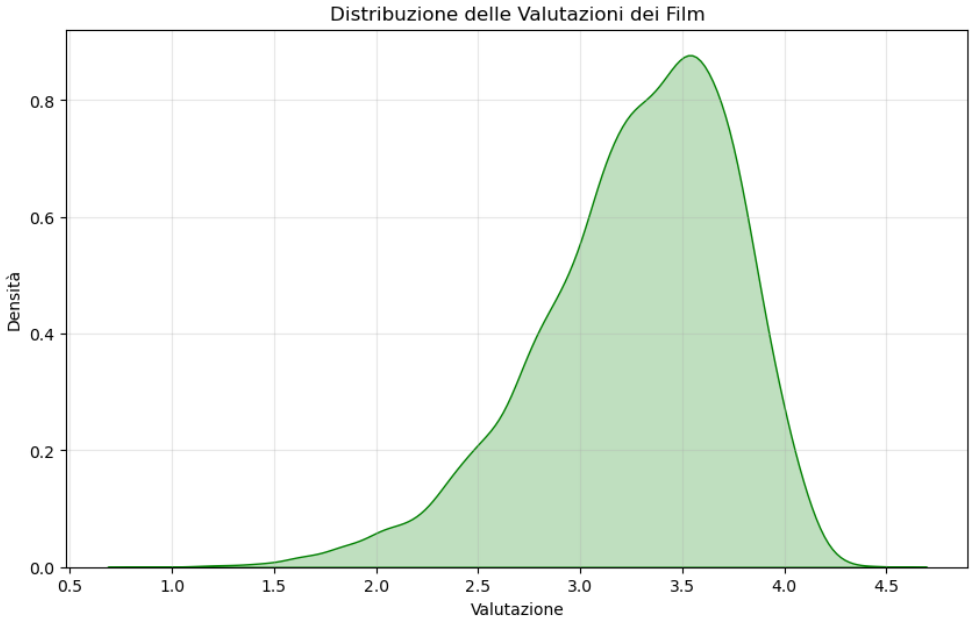


Figura 3.1: Densità delle valutazioni dei film

Quello che emerge dal grafico è che la maggior densità ottenuta nella distribuzione dei voti si presenta, in corrispondenza circa del valore 3,5.

3.0.2 Correlazione genere voto

E' stata calcolata la correlazione tra genere e voto dei film, e i risultati sono riportati nella Figura 3.2, dove è possibile notare come alcuni tag hanno una maggiore correlazione con il voto rispetto ad altri.

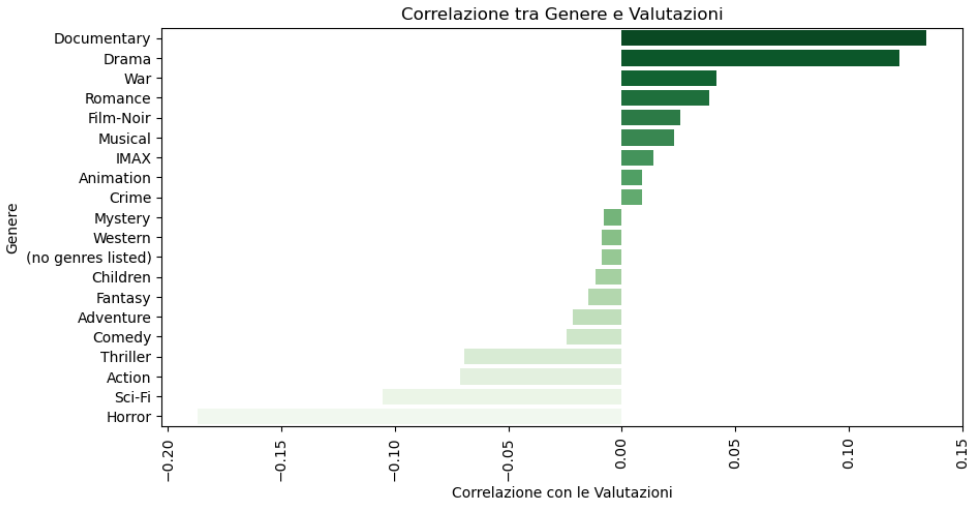


Figura 3.2: Correlazione tra generi e valutazioni

Si può notare che i generi Documentary e Drama mostrano una forte correlazione positiva, al contrario i generi Horror e Sci-Fi hanno una correlazione fortemente negativa. Questo ci indica che il genere del film può influenzare il voto medio atteso.

Nella Figura 3.3, si può vedere il voto medio per ogni genere di film.

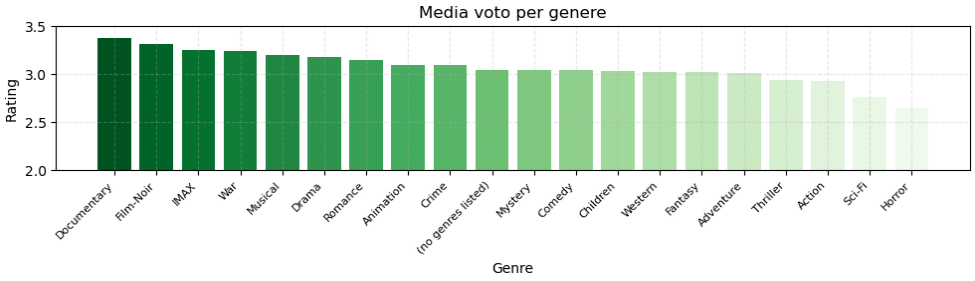


Figura 3.3: Correlazione tra generi e valutazioni

3.0.3 Correlazione tag voto

Successivamente è stata calcolata anche la correlazione tra tag e voto del film. I risultati sono visibili nella Figura 3.4, queste due matrici evidenziano quali tag dei film possono maggiormente influenzare la valutazione di un film da parte di un utente.

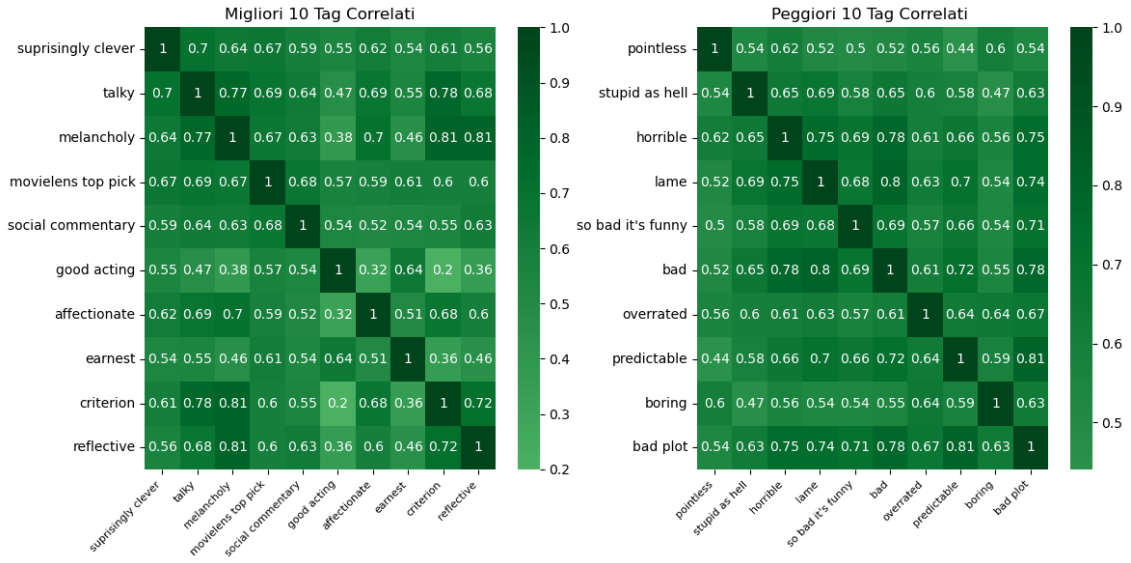


Figura 3.4: Correlazione tra generi e valutazioni

Inoltre è stato creato anche un grafico, Figura 3.5, che rappresenta la distribuzione della correlazione tra tag e voto, evidenziando zone che hanno una maggiore correlazione rispetto ad altri.

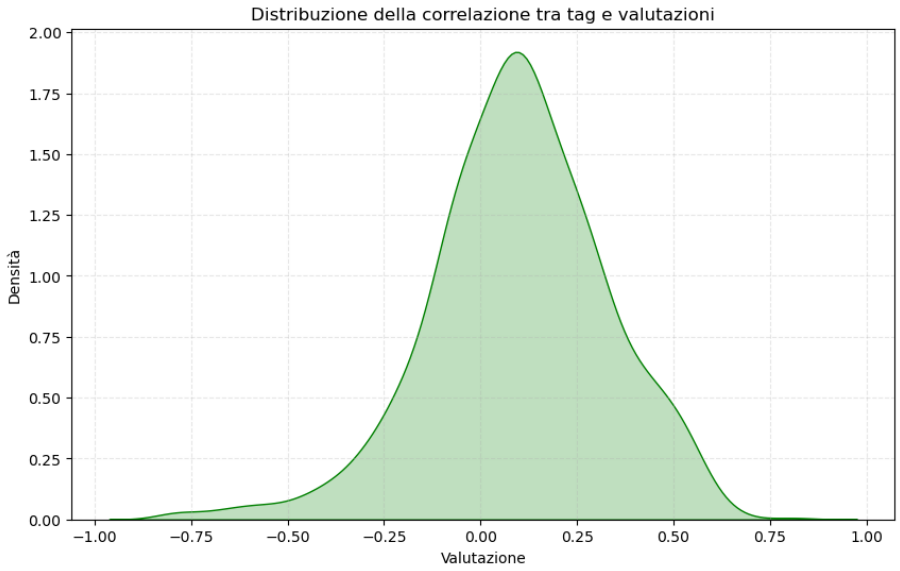


Figura 3.5: Correlazione tra generi e valutazioni

3.0.4 Tag più importanti

E' stata fatta un'analisi per valutare l'importanza dei vari tag nel processo di previsione del voto medio. Per rendere possibile questa analisi si è utilizzato l'apprendimento automatico Random Forest, perchè questo tipo di alberi decisionali sono in grado di fornire una spiegazione dell'importanza delle features utilizzate dal modello.

Va comunque tenuto presente che il valore di importanza che si ottiene non può essere considerato rappresentativo per tutti i modelli ma ci aiuta a capire quali sono i tag più importanti che hanno un'influenza maggiore sul risultato.

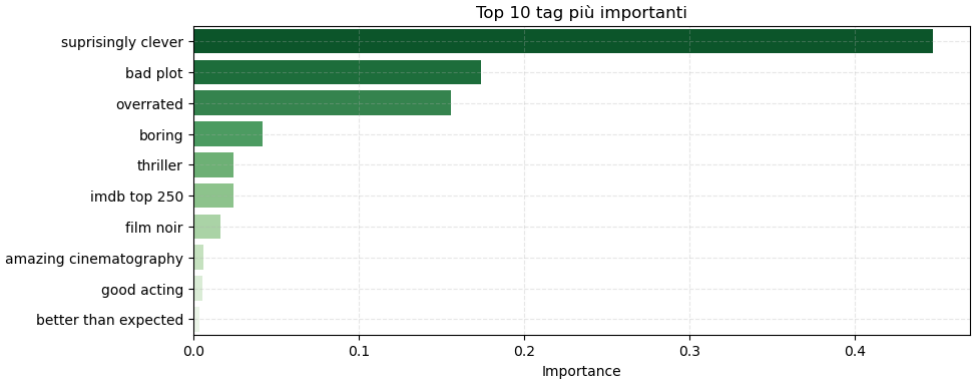


Figura 3.6: Top 10 tag più importanti

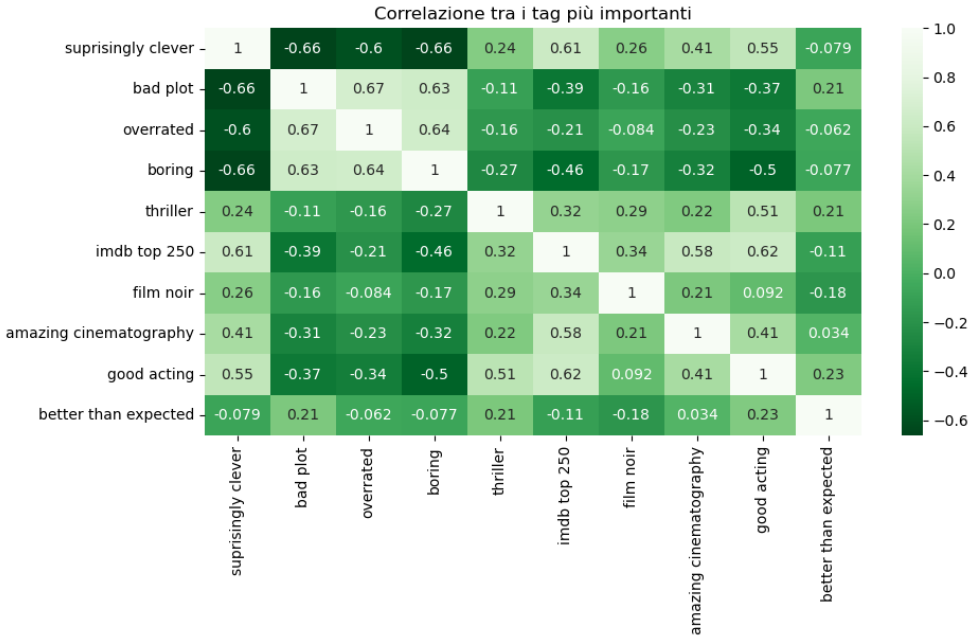


Figura 3.7: Correlazione tra i 10 tag più importanti

3.0.5 Visualizzazione 2D dei dati

Per rappresentare i film sulla base del rating in uno spazio bidimensionale è stato utilizzato l'algoritmo t-Distributed Stochastic Neighbor Embedding(t-SNE) che è in grado di ridurre la dimensionalità. Questo algoritmo riesce a trovare una raprresentazione a bassa dimensionalità dei dati in cui, i punti che presentano una certa similarità nel data sono anche vicini nello spazio utilizzato dall'algoritmo. Questo processo, consente di visualizzare i dati in uno spazio bidimensionale o tridimensionale conservando le relazioni inziali dei dati.

L'algoritmo per il corretto funzionamento ha bisogno di due passaggi principali. Il primo, prevede il calcolo di una matrice di somiglianza tra i punti originali del dataset, questo è possibile utilizzando una funzione di distanza come la distanza euclidea o la distanza cosenica. Successivamente l matrice viene convertita, attraverso una funzione di similarità, in una distribuzione di proprietà. Nel secondo passaggio, l'algoritmo cerca di rappresentare i punti del dataset in uno spazio a bassa dimansionalità, mantenendo al contempo la distribuzione di probabilità delle somiglianze tra i punti. Questo processo viene eseguito attraverso un'iterazione di minimizzazione della divergenza di Kullback-Leibler tra la distribuzione di probabilità del dataset originale e quella nel nuovo spazio a bassa dimensionalità.

Il risultato finale dell'algoritmo t-SNE è una mappe bidimensionale dei dati. In questo caso, l'algoritmo è stato applicato ai voti dei film in modo da rappresentarli in uno spazio bidimensionale, rendendo possibile la visualizzazione della distribuzione dei voti.

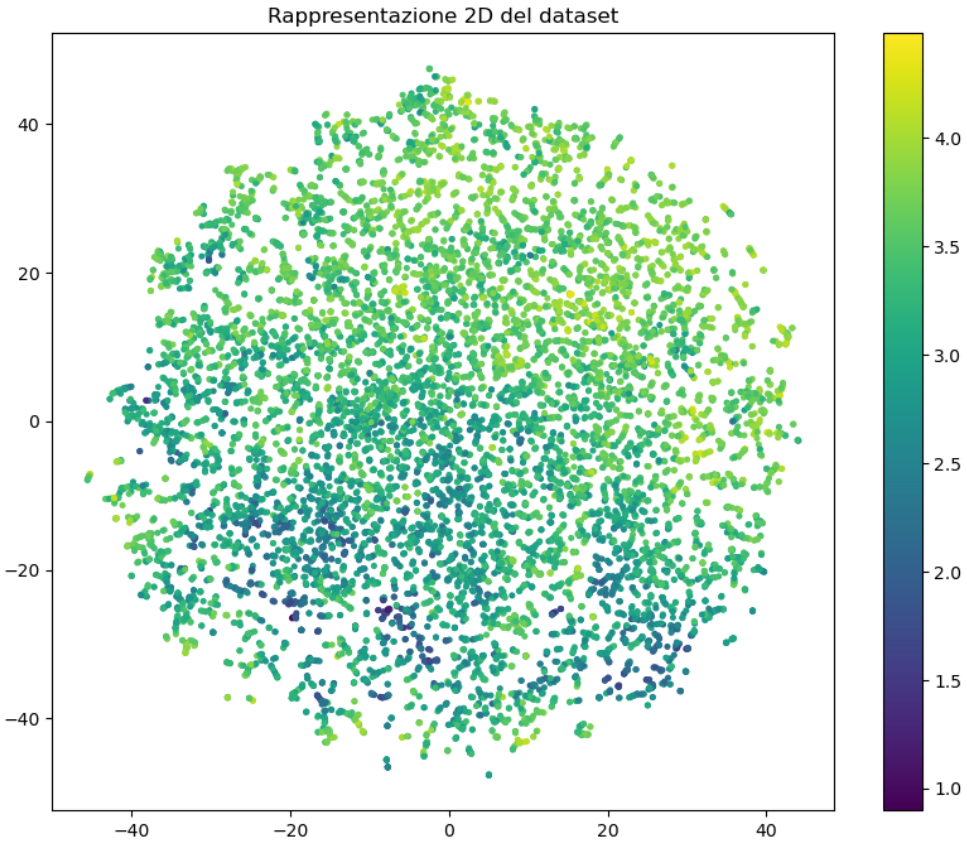


Figura 3.8: Correlazione tra generi e valutazioni

Analizzando la figura si può notare una distribuzione graduale delle valutazioni dei film. In particolare, i film che hanno una valutazione minore si trovano nella parte bassa dell'immagine e sono colorati di blu, mentre i film con le valutazioni migliori, ovvero quelli colorati di giallo, si trovano nella parte alta della figura. In generale si può osservare che la maggior parte delle valutazioni dei film sono comprese tra il 3,0 e il 3,5, e le aree dove si trovano questi film sono in generale più fitte. Questo rispecchia le valutazioni fatte precedentemente con la distibuzione.

Capitolo 4

Data Pre-processing

La fase di preprocessing consiste nel preparare i dati grezzi in modo che siano utilizzabili per le elaborazioni successive, infatti in queste fase si andranno ad eseguire delle operazioni che prepareranno i dati in modo adeguato. I dati raccolti, spesso, contengono delle imperfezioni come dati mancanti, rumore, valori anomali, che possono influenzare negativamente i risultati delle analisi. Quindi l'obiettivo principale di questa fase è quello di ripulire i dati ed eventualmente trasformarli, in modo che siano adatti alle analisi che si andranno a svolgere.

Come prima azione, in questa fase, è stato eseguito lo split del dataset in train(70), validation(10) e test(20). Il dataset utilizzato non contiene valori nulli né duplicati, quindi non c'è stato bisogno di compiere nessuna azione per questo problema. Inoltre, i dati forniti nel dataset sono tutti compresi tra 0 e 1, quindi non vi è stato necessario applicare lo scaling.

Durante questa fase è stata applicata la PCA (Principal Component Analysis), une tecnica di riduzione della dimensionalità utilizzata per semplificare la complessità ma allo stesso tempo mantenere le informazioni più significative. Questa tecnica mira a creare delle variabili linearmente indipendenti, chiamate appunto componenti principali, partendo da un insieme di variabili correlate. Attraverso la costruzione di un nuovo spazio, che verrà definito dalla componenti principali anziché dalla variabili originali, sarà possibili rappresentare la natura multi variata dei dati in un numero ridotto di dimensioni, utilizzando questa rappresentazione per identificare la struttura dei dati. Dopo aver identificato le componenti principali, sarà possibile proiettare i dati originali su un nuovo sistema di coordinate, con le componenti principale che andranno a formare i primi assi dello spazio. Questa operazione ci consente di rappresentare i dati in un nuovo spazio, con dimensioni inferiore rispetto a quello originale, senza perdere grandi quantità di informazioni.

Al train set, è stata applicata la PCA, la variabile dipendete è rappresentata dalla media dei voti, mentre tutte le restanti sono state utilizzate come variabili indipendenti. Il numero di componenti che sono state mantenute è stato impostato a 0.95, andando a conservare il 95% dell'energia dei dati. Infine le analisi svolte sul dataset verranno testate con sia con l'applicazione

della PCA che senza, in modo da capire se al variare della dimensionalità del dataset corrisponde anche una variazione nelle performance.

Capitolo 5

Modeling

La fase di modeling è la parte in cui vengono applicate le tecniche di Machine Learning supervisionato, tecniche deep basate su reti neurali e TabNet per raggiungere l'obiettivo dello studio. L'analisi svolta può essere considerata sia come un problema di regressione che di classificazione; in questo caso si è deciso di affrontarlo come regressione in quanto la variabile target assume dei valori continui.

5.0.1 Linear Regression

La regressione lineare è un tipo di modello statico, utilizzato per cercare una relazione tra una variabile dipendente e, una o più variabili indipendenti continue. L'obiettivo di questo modello è quello di cercare una retta che rappresenta al meglio la relazioni tra le variabili.

Nel caso di questo studio, la variabile dipendete viene rappresentata dal voto medio del film, mentre le caratteristiche dei film, che possono influenzare il voto medio, rappresentano le variabili indipendenti.

La regressione lineare viene definita dalla seguente funzione lineare:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$$

Dove:

- $\hat{y}$: indica la variabile dipendente.
- x: indica la variabile indipendente.
- β_0: è l'intercetta della retta di regressione.
- β_1: è la pendenza della retta di regressione.

5.0.2 Ridge Regression

La ridge regression è una regressione lineare che utilizza la penalizzazione L2 oltre ad introdurre λ che indica la regolarizzazione quadratica nella formula della regressione lineare. Al variare del valore di λ avremo due effetti: al crescere, alcuni dei coefficienti di regressione vengono "compressi" andando a diminuire il contributo delle variabili corrispondenti, contrariamente, al diminuire, i coefficienti di regressione sono lasciati più liberi rendendo il modello maggiormente adattabile ai dati di allenamento.

Questa tecnica ci permette di ridurre la dimnasionalità del modello, andando a limitare il numero di variabili che contribuiscono in modo significativo alla previsione target.

L'obiettivo principale della ridge regression è quello di cercare di minimizzare la seguente funzione:

$$\text{Ridge} = \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

Dove:

- y_i: indica la variabile di risposta dell' osservazioine.
- β_0: indica l'intercetta.
- β_j: indica il coefficiente di regressione associato alla variabile predittiva.
- x_{ij}: indica la variabile predittiva j per l'osservazione i.
- λ: indica il parametro di regolarizzazione che controlla la *penalità di shrinkage*$(\sum_{j=1}^{p}\beta_j^2)$ applicata ai coefficienti di regressione.

Di seguito vengono specificati gli iperparametri con i relativi valori utilizzati per questo progetto, successivamente verranno discusse la migliore configurazione e i risultati ottenuti:

```
param_grid= {'alpha':[0.0001, 0.001, 0.1, 0.5, 1, 2, 3, 4, 5, 6, 10, 20]}
```

5.0.3 Lasso Regression

La Regressione Lasso è una tecnica di regressione lineare che mira a identificare un sottoinsieme ristretto di variabili ritenute più rilevanti per il modello, riducendo l'impatto delle variabili meno significative. Questo approccio si basa sull'applicazione di una penalità L1, che induce l'azzeramento di coefficienti associati alle variabili meno influenti, contribuendo così a ottenere modelli più parsimoniosi. La penalità L1 agisce come uno strumento di selezione delle feature, consentendo al modello di escludere variabili meno informative, superando uno dei limiti della Ridge Regression, in cui tutte le variabili vengono mantenute nel modello indipendentemente dalla loro rilevanza.

Per il funzionamento di questa tecnica, viene introdotto un termine di regolarizzazione nella funzione di perdita del modello, noto come λ, che serve a penalizzare i coefficienti meno rilevanti. Maggiore è il valore di λ, maggiore è il grado di penalizzazione applicato ai coefficienti, portando alcuni di essi verso lo zero.

$$Ridge = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{5.1}$$

Dove:

- n: indica il numero di osservazioni sui dati.
- p: indica il numero di variabili predittori nel modello.
- y_i: indica la variabile dipendete nella i-esima osservazione.
- β_0: indica l'intercetta del modello x_{ij}, ovvero il valore della j-esima variabile esplicativa alla i-esima osservazione.
- β_j: indica il coefficiente di regressione della j-esima variabile esplicativa.
- λ: indica l'iperparametro che controlla la forza di regressione.

Di seguito vengono specificati gli iperparametri con i relativi valori utilizzati per questo progetto, successivamente verranno discusse la migliore configurazione e i risultati ottenuti:

```
param_grid= {'alpha': [1e-1, 1e-2, 1e-3, 1e-4, 1e-5]}
```

5.0.4 Random Forest Regressor

Il random forest regressor è un algoritmo di machine learning supervisionato basato sulla creazione di alberi decisionali indipendenti tra di loro. Per ciascun albero viene generato un subset casuale dei dati di training, andando ad ottenere modelli differenti tra di loro, ogni albero utilizzerà soltanto un subset casuale di dati in input per generare delle previsioni.

Di seguito vengono specificati gli iperparametri con i relativi valori utilizzati per questo progetto, successivamente verranno discusse la migliore configurazione e i risultati ottenuti:

```
param_grid= {
    'n_estimators': [10, 15, 20, 25, 30],
    'criterion': ['squared_error', 'friedman_mse'],
}
```

Dove indichiamo con:

- *n_estimators*: il numero di alberi nella foresta.
- *criterion*: misura la qualità dello split, precisamente:
  - *squared_error*: utilizza l'errore quadratico medio ed equivale alla riduzione della varianza come criterio di selezione delle caratteristiche e minimizza la perdita L2 utilizzando la media per ogni nodo terminale.
  - *friedman_mse*: utilizza l'errore quadratico medio con il punteggio di miglioramento di Friedman per il potenziale splitù.

5.0.5 KNN Regressor

La regressione K-Nearest Neighbors è una tecnica utilizzata sia per la classificazione che per la regressione. In questo caso, viene utilizzata per un problema di regressione, il che significa che l'output risultante sarà un valore numerico anziché una classe. L'obiettivo principale della regressione KNN è trovare i dati più simili tra quelli di addestramento e quelli di previsione, basandosi su un numero specifico di punti "K" da considerare durante la previsione.

Di seguito vengono specificati gli iperparametri con i relativi valori utilizzati per questo progetto, successivamente verranno discusse la migliore configurazione e i risultati ottenuti:

```
param_grid= {
    'n_neighbors': [7, 9, 11, 13, 15, 17, 19, 20],
    'weights': ['uniform', 'distance'],
}
```

Dove:

- *n_neighbors*: indica il numero di neighbors da tenere in considerazione.
- *weights*: indica in che modo deve essere assegnato un peso ai vicini durante la previsione. In particolare
  - *uniform*: tutti i vicini hanno lo stesso peso nell'influenzare la previsione.
  - *distance*: i dati più vicini avranno un peso maggiore rispetto a quelli più lontani, quindi la previsione sarà influenzata in modo proporzionale in base alla distanza del punto.

5.0.6 Support Vector Regression

Support vector machine è un algortimo utilizzato per i problemi di regressione e calssificazione. Questa tecnica ha come obiettivo quello di trovare un iperpiano in uno spazio multidimensionale

che meglio approssima i dati di input, chiamati vettori di supporto, ma allo stesso tempo sia il più distante dai punti al di fuori della boundary line.

Di seguito vengono specificati gli iperparametri con i relativi valori utilizzati per questo progetto, successivamente verranno discusse la migliore configurazione e i risultati ottenuti:

```
param_grid= {
    'kernel': ['poly', 'rbf', 'sigmoid'],
    'epsilon': [0.001, 0.01, 0.1, 1],
    'C': [0.001, 0.01, 0.1, 1]
}
```

Dove:

- *kernel*: indica la funziona kernel che deve essere utilizzata per addestrare e valutare il modello. In particolare:
  - *poly*: kernel che affronta i dati con separabilità polinomiale.
  - *rbf*: Radial Basis Function, kernel utile per dati non linearmente separabili.
  - *sigmoid*: kernel utilizzato per affrontare dati con distribuzioni sigmoidali.
- *epsilon*: indica l'epsilon-tube, ovvero una fascia che comprende gli esempi di addestramento che non contribuiscono alla funzione di perdita di SVR.
- C: parametro che controlla il compromesso tra l'aderenza ai dati di addestramento e la generalizzazione ai dati di test.

5.0.7 Neural Network

Le reti neurali sono un tipo di modello di apprendimento automatico ispirato al funzionamento del cervello umano. Sono composte da unità di elaborazione, chiamate neuroni artificiali, organizzati in strati o livelli. I neuroni ricevono informazioni in ingresso, in modo da elaborarle per trasmetterle ad altri neuroni della rete per andare a produrre un output sulla base di ciò che è stato elaborato. Per trasmettere le informazioni i neuroni, sono collegati tra loro attraverso pesi sinaptici, che vanno a determinare l'importanza delle connessioni. I pesi delle sinaptici saranno poi regolati durante la fase di addestramento dalla rete neurale stessa, in modo da produrre previsioni o output in base ai dati forniti in input.

5.0.8 Feed-forward Network

In questo progetto è stata utilizzata una rete neurale feed-forward. Questa tipologia di reti neurali, sono chiamate in questo modo perché ogni layer di input, con i relativi layer di output si muovono in una singola direzione, quindi vengono esclusi connessioni a ritroso o cicli durante le elaborazioni dei dati dei neuroni. La struttura della rete è molto semplice, infatti può essere composta da uno o più layer di neuroni, ed ogni neurone è collegato a tutti i neuroni del layer successivo. Il primo layer, ovvero quello di input, è quello dove i neuroni ricevono le informazioni da elaborare, e terminata l'elaborazione la passeranno al layer successivo. L'elaborazione consiste in una combinazione lineare degli input pesati a cui verrà applicata una funzione di attivazione non lineare. Tale funzione risulta essere molto importante in quanto introduce la non linearità alla rete, permettendo di modellare problemi più complessi.

L'addestramento di queste reti avviene principalmente attraverso l'ottimizzazione dei pesi delle connessioni tra neuroni, uno dei metodi più comuni che viene utilizzato è la discesa del gradiente, che ha come obiettivo quello di trovare il valore delle variabili del modello che minimizza la funzione di perdita, in modo da andare ad assottigliare l'errore tra output atteso e quello effettivo della rete.

Nella Figura è possibile vedere un esempio di rete neurale feed-forward.

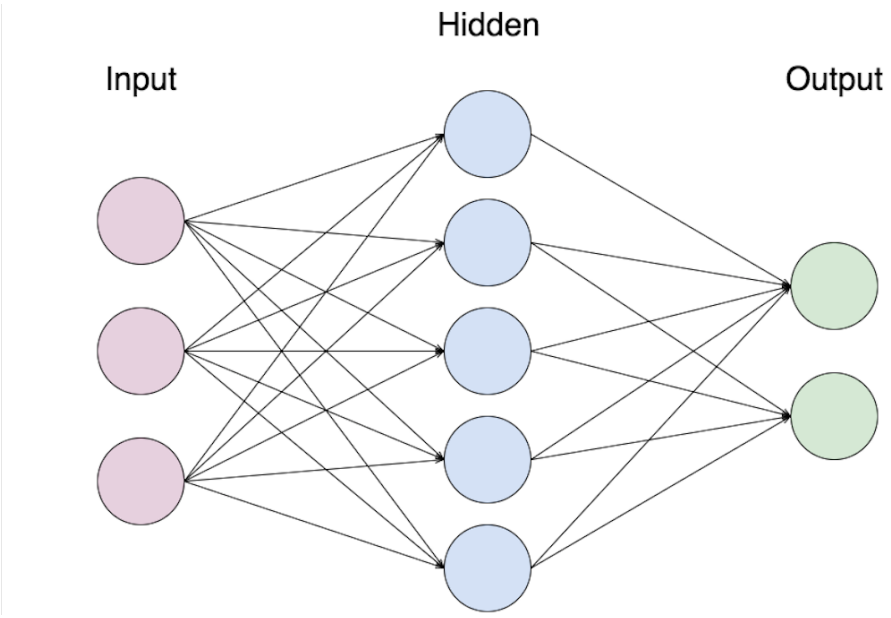


Figura 5.1: Esempio di architettura di una rete neurale feed-forward

L'implementazione in questo progetto è formata da un numero di layer variabile, con il primo layer che è fully connected e prende in input i dati, per poi andarli a mappare in un nuovo spazio di

features grazie ai pesi appresi durante l'addestramento, in mezzo c'è un numero varibile di hedden layer e l'ultimo è il layer di output.

Il codice per l'implementazione della rete è il seguente:

```
def get_model(input_size, dept= 3, hidden_size= 64, dropout_prob= 0.2):
    model= [nn.Linear(input_size, hidden_size), nn.ReLU()]
    for i in range(dept):
        model.append(nn.Linear(hidden_size, hidden_size))
        model.append(nn.ReLU())
        model.append(nn.Dropout(dropout_prob))
    model.append(nn.Linear(hidden_size, 1))
    return nn.Sequential(*model)
```

Gli iperparamtri con i relativi valori utilizzati nel progetto sono i seguenti:

- hidden_size= [64, 128, 256, 512], indica la dimensione dei hidden layer.
- dropout_prob= [0.2, 0.3], indica la probabilità che alcuni neuroni vengano disattivati durante l'addestramento della rete.
- dept= [3, 4, 5], indica il numero di hidden layer della rete.
- epochs= [200], indica il numero di epoche che è stato fissato a 200.
- batch_size= [8, 16, 32], definisce la dimensione dell'input.
- lr= [0.001, 0.01], determina la funzione del passo ad ogni iterazione, mentre ci si sposta verso un minimo della funzione di perdita.

5.0.9 TabNet

TabNet[2] è un algoritmo di machine learnig sviluppato per affrontare specificatamente problemi legati all'apprendimento dei dati tabulari. E' stato progettato nel 2019 da Google Cloud AI Reserach per migliorare le prestazioni delle reti neurali deep nell'ambito dei dati tabulari.

I dati tabulari sono una forma comune di dati strutturati, solitamente rappresentati come tabelle dove le righe rappresentano le osservazioni, mentre le colonne rappresentano le caratteristiche o le variabili.

TabNet utilizza una struttura basata su reti neurali chiamata "TabNet Architecture", che è in grado di apprendere automaticamente le relazioni complesse nei dati tabulari. Prendendo in input dati grezzi e utilizzando l'attenzione sequenziale, tecnica che viene utilizzata per dare priorità

a determinate parti di una sequenza, per selezionare le features durante il processo decisionale, permettendo una migliore interpretazione e un migliore apprendimento. Utilizza piu blocchi decisionali che creeranno un sottoinsieme di features di input, come in Figura 5.2.

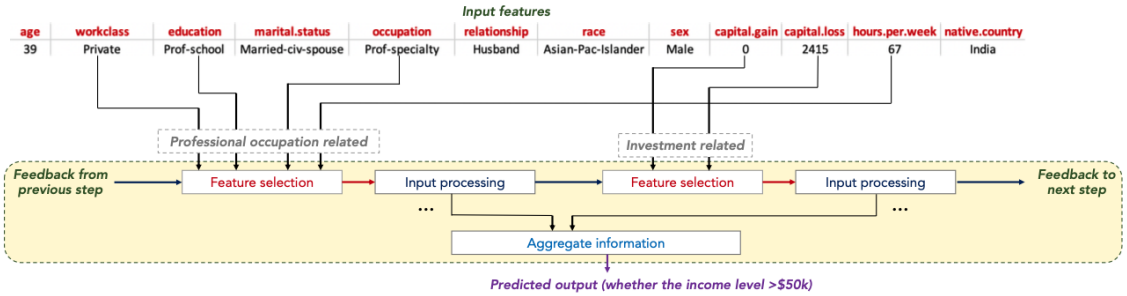
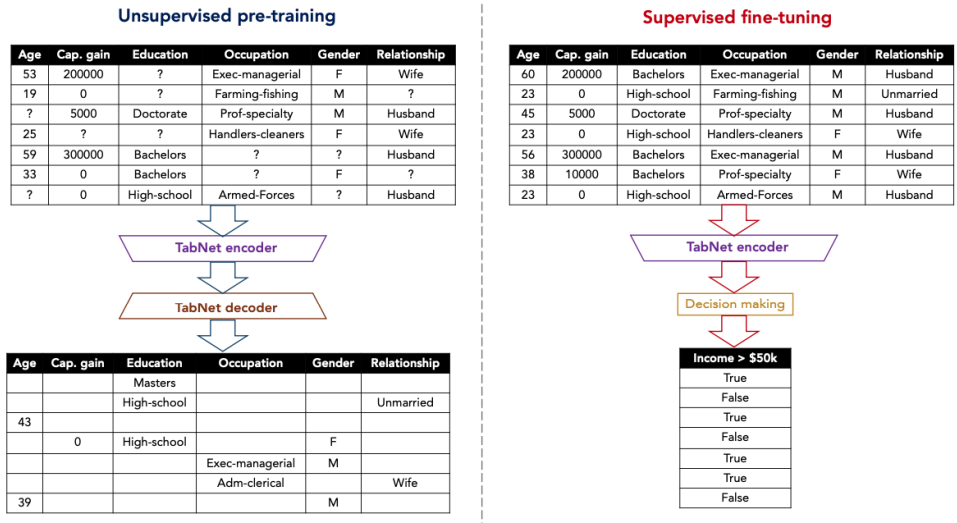


Figura 5.2: Selezione features

Questo modello, utilizza il pre-training non supervisionato, Figura5.3, andando a inizializzare i pesi del modello utilizzando conoscenze da modelli pre-addestrati su grandi dataset o su problemi simili. Applicando questa tecnica ai dati tabulari, ha dimostrato di avere dei miglioramenti nelle prestazioni.



Age	Cap. gain	Education	Occupation	Gender	Relationship
53	200000	?	Exec-managerial	F	Wife
19	0	?	Farming-fishing	M	?
?	5000	Doctorate	Prof-specialty	M	Husband
25	?	?	Handlers-cleaners	F	Wife
59	300000	Bachelors	?	?	Husband
33	0	Bachelors	?	F	?
?	0	High-school	Armed-Forces	?	Husband

Age	Cap. gain	Education	Occupation	Gender	Relationship
		Masters			
		High-school			Unmarried
43					
	0	High-school		F	
			Exec-managerial	M	
			Adm-clerical		Wife
39				M	

Age	Cap. gain	Education	Occupation	Gender	Relationship
60	200000	Bachelors	Exec-managerial	M	Husband
23	0	High-school	Farming-fishing	M	Unmarried
45	5000	Doctorate	Prof-specialty	M	Husband
23	0	High-school	Handlers-cleaners	F	Wife
56	300000	Bachelors	Exec-managerial	M	Husband
38	10000	Bachelors	Prof-specialty	F	Wife
23	0	High-school	Armed-Forces	M	Husband

Income > $50k
True
False
True
False
True
True
False

Figura 5.3: Pre-training non supervisionato

Per l'implementazione di TabNet nel progetto è stata utilizzata la libreria *tabnet_pytorch*. Tabnet è composto da tre elementi principali:

- Encoder.
- Decoder.

- Mascheramento.

Il primo elemento, l' encoder, si divide a sua volta in due unità di decisione, chiamate Decision Point, che vanno ad eleborare i dati in input in modo selettivo, per produrre delle maschere binarie che indicano quali features sono selezionate e quali sono scartate.

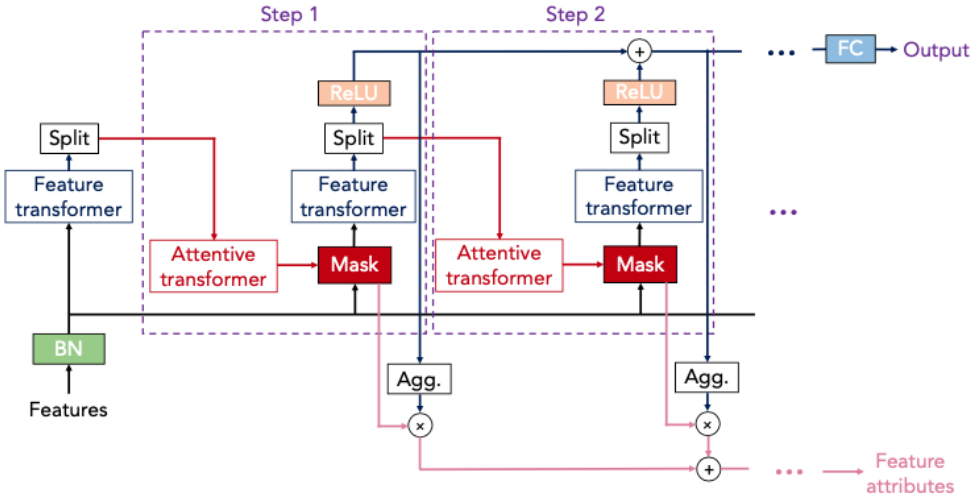


Figura 5.4: Architettura del decoder TabNet

Il decoder, invece, utilizza le maschere create dall'encorder su una rete neureale feed-forwrd per creare la previsione finale.

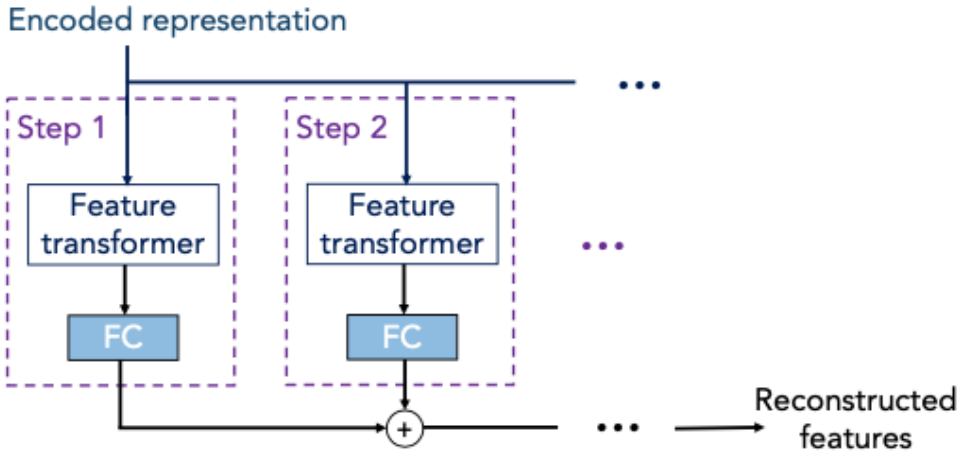


Figura 5.5: Architettura dell'encoder TabNet

Infine, il mascheramento, è una tacnica che impedisce al modello di utilizzare le stesse features per piu volte durante l'elabgorazione dei dati, andando a garantire una maggiore generalizzazione del modello e aiutando a prevenire l'overfetting.

Due componenti importanti nell'architettura Tabnet sono il *features transformer* che prende in input le features selezionate durante la fase di mascheramento e le trasforma in una nuova rappresentazione, questo componente è composto da tre layer sequenziali: Fully-Connected, Batch Normalization e Gated Linear Unit(GLU), mentre *attentive transformer* selezione le features piu rilevanti che poi verranno passate alla fese di mascheramento.

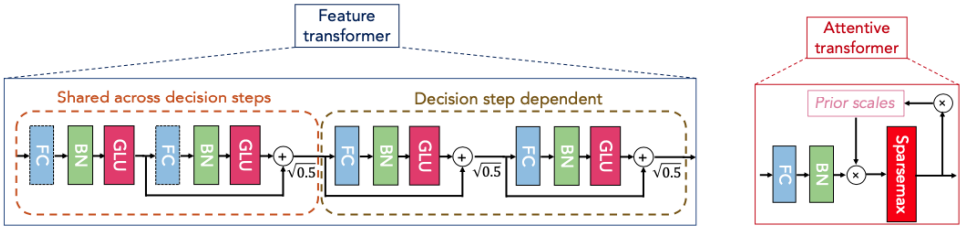


Figura 5.6: Esempi di architettetture di Fetures transformer e Attentive transformer

Il modello che viene implementato nel progetto è stato riportato di seguito:

```
def get_model(n_d, n_a, n_steps, n_indipendent):
    model = TabNetRegressor(
        n_d=n_d,
        n_a=n_a,
        n_steps=n_steps,
        n_independent=n_indipendent
    )
    return model
```

Gli iperparametri ed i relativi valori utilizzati sono riportati di seguito, mentre successivamente verrà discussa la migliore configurazione e i risultati ottentuti:

- batch_size = [256], indica il numero di sample in ogni batch.
- n_epochs = [200], indica il numero di epoche che è stato fissato a 200.
- n_d= [16, 32, 64], indica la dimensione del layer di predizione.
- n_a= [16, 32, 64], indica lo spazio di output del attentative transfornmer.
- n_steps= [3, 6, 9], indica il numero di step nell'architettura.
- n_indipendent= [2, 3], indica il numero di layer GLU indipendenti per ogni blocco GLU.

Di seguito viene riporta l'implementazione del codice di training del modello:

```
for batch_size, n_epochs, n_d, n_a, n_steps, n_indipendent in params:

    model= get_model(n_d, n_a, n_steps, n_indipendent)

    model.fit(
        X_train=X_train,
        y_train=y_train,
        eval_set=[(X_val, y_val)],
        eval_name=["mse"],
        patience= 10,
        batch_size= batch_size,
        virtual_batch_size= batch_size,
    )

    y_pred= model.predict(X_test)
    mse= mean_squared_error(y_test, y_pred)
    r2= r2_score(y_test, y_pred)

    if mse < best_mse:
        best_mse= mse
        best_model= copy.deepcopy(model)
        best_params= (batch_size, n_epochs, n_d, n_a, n_steps, n_indipendent)
        print("Best model updated")
    writer.close()
```

Capitolo 6

Performance Evaluation

6.1 Performance Evaluation

Nel seguente pragrafo si andranno a visionare i risultati ottenuti dai vari modelli utilizzati nel paragrafo Modeling. Per misurare e confrontare le performance dei modelli sono state utilizzate le seguenti metriche:

- **Mean Squared Error(MSE)**: misura l'errore quadratico medio tra le previsioni del modello e i valori osservati. Il valore resituito da questa misura ci indica le prestazioni del modello, infatti se si ottiene un risultato basso, il modello ha una buona capacità di predire i valori reali, al contrario ottenendo un valore elevato il modello ha una scarsa capacità predittiva. La formula che ci restituisce il valore è la seguente:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  Dove n indica il numero di campioni nel dataset, y_i sono i valori osservati per il campione i e $\hat{y}_i$ sono i valori previsti dal modello per il campione i.

- **Coefficiente di determinazione(R2)**: misura quanto la varianza dei dati di output previsti dal modello si avvicina alla varianza dei dati reali. Questa metrica può assumere valori compresi tra 0 e 1, dove 1 indica la perfetta aderenza dei modello ai dati di output reali, 0 indica che il modello non è in grado di spiegare la variazione nei dati di output. La formula che ci restituisce il valore è la seguente:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y}_i)^2}$$

Dove n indica il numero di campioni nel dataset, y_i sono i valori osservati per il campione i, $\hat{y}_i$ sono i valori previsti dal modello per il campione i e $\overline{y}_i$.

6.1.1 Tecniche di Machine Learning supervisionato tradizionali

Di seguito verranno discussi tutti i risultati di finetunig dei modelli presentati. Per ogni modello si vedranno le migliori configurazioni e i risultati ottenuti con e senza l'utilizzo della PCA.

Il tuning degli iperparametri è stato fatto, per tutti i modelli deep, utilizzando la *GreedSearchCV*, presente in *sklearn*. Questa tecnica prevede la prova di tutti le combinazioni possibili dei valori assegnati come iperparametri e valuta il modello per ogni combinazione trovata utilizzando la cross-validation. I parametri per la GridSearchCV utilizzati in tutti i modelli sono:

- param_grid: indica la ricerca su qualsiasi sequenza di impostazioni dei parametri.
- cv= 5: indica il numero di cross-validation per ogni combinazione del modello.
- scoring= neg_mean_squared_error: indica la metrica di valutazione che deve essere utilizzata per classificare i risultati. L'utilizzo di *neg_mean_squared_error* significa che il valore dell' MSE verrà calcolato con un segno negativo.
- return_train_score= True: indica di includere i punteggi del train.

Linear Regression

I risultati della regressione lineare sono i seguenti:

Con PCA:

- MSE: 0.00643
- R2-score: 0.97096

Senza PCA:

- MSE: 0.00544
- R2-score: 0.97543

Ridge Regression

La regressione ridge è stata implementata utilizzando la funzione di scikit-learn. Il tuning degli iperparametri, ha restituito la migliore configurazione possibile per il parametro α. Di seguito i risultati ottenuti e le migliori combinazioni:

Con PCA:

Migliore configurazione ottenuta dal training:

- α= 6

Risultati ottenuti dalla configurazione:

- MSE: 0.00639
- R2-score: 0.97114

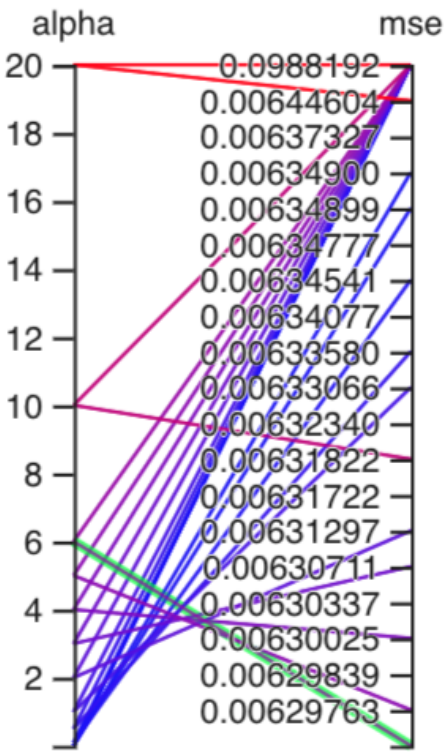
Senza PCA:

Migliore configurazione ottenuta dal training:

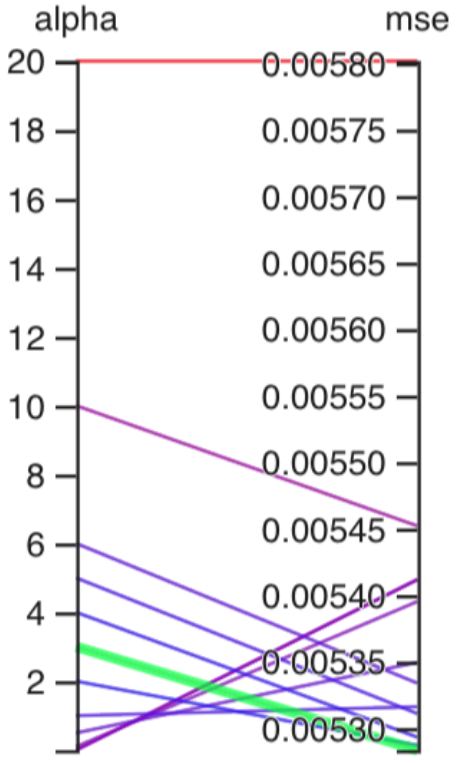
- α= 3

Risultati ottenuti dalla configurazione:

- MSE: 0.00531
- R2-score: 0.97600



(a) Parallel coordinates view ridge regression con PCA



(b) Parallel coordinates view ridge regression senza PCA

Lasso Regression

La lasso regression è stata implementata utilizzando la funsione di scikit-learn. Il tuning degli iperparametri, ha restituito la migliore configurazione possibile per il parametro α. Di seguito i risultati ottenuti e le migliori combinazioni:

Con PCA:

Migliore configurazione ottenuta dal training:

- α= 1e-05

Risultati ottenuti dalla configurazione:

- MSE: 0.00641
- R2-score: 0.97106

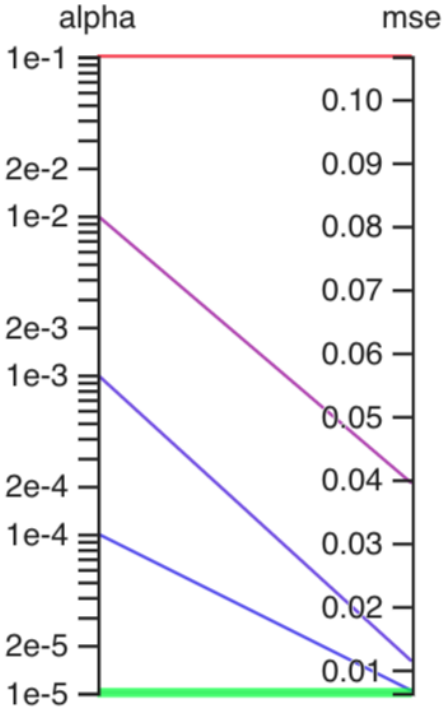
Senza PCA:

Migliore configurazione ottenuta dal training:

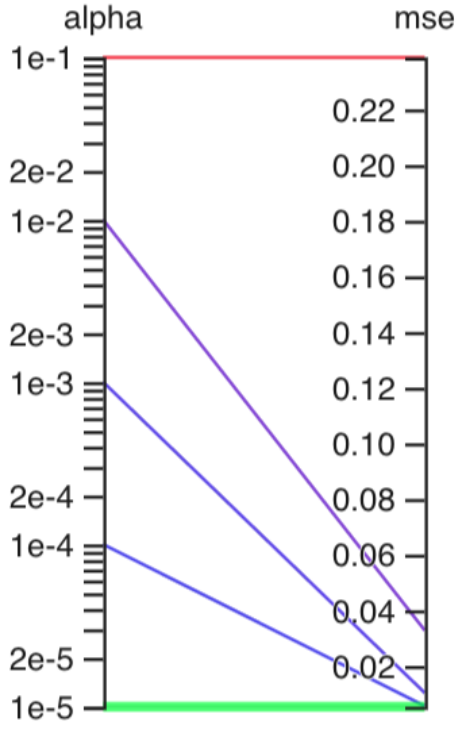
- α= 1e-05

Risultati ottenuti dalla configurazione:

- MSE: 0.00542
- R2-score: 0.97553



(a) Parallel coordinates view lasso regression con PCA



(b) Parallel coordinates view lasso regression con PCA

Siccome la lasso regression tende ad azzerare i coefficienti delle variabili meno rilevanti, di seguito è riporta la correlazione tra le prime 10 features i cui valori sono stati azzerati e il voto dei film:

Tabella 6.1: Correlazioni tra le prime 10 features con coefficienti a 0 e i voti.

Features	**Correlazione**
007	-0.097706
1920s 3	0.379278
3d	-0.153728
aardman	0.060788
afterlife	-0.079177
alcoholism	0.206545
almodovar	0.083976
amnesia	0.062696
animation	0.024383
arms dealer	-0.073337

Random Forest Regressor

Il random forest regeressor è stato implementato utilizzando la funsione di scikit-learn. Il tuning degli iperparametri, ha restituito la migliore configurazione possibile per i parametri *n_estimators* e *criterion*. Di seguito i risultati ottenuti e le migliori combinazioni:

Con PCA:

Migliore configurazione ottenuta dal training:

- **n_estimators**= 30
- **criterion**= squared_error

Risultati ottenuti dalla configurazione:

- MSE: 0.03801
- R2-score: 0.82850

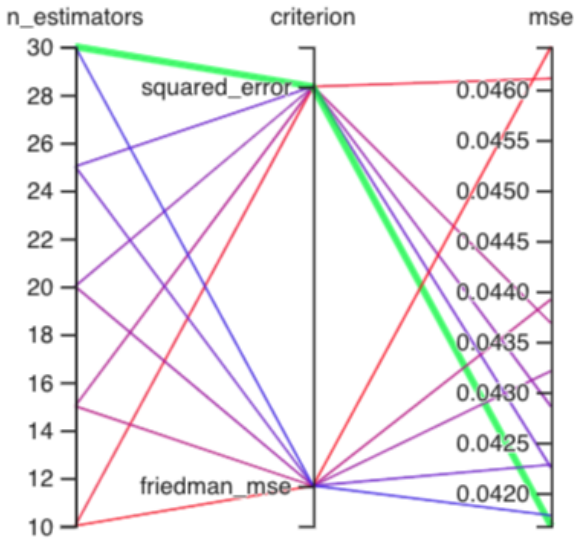
Senza PCA:

Migliore configurazione ottenuta dal training:

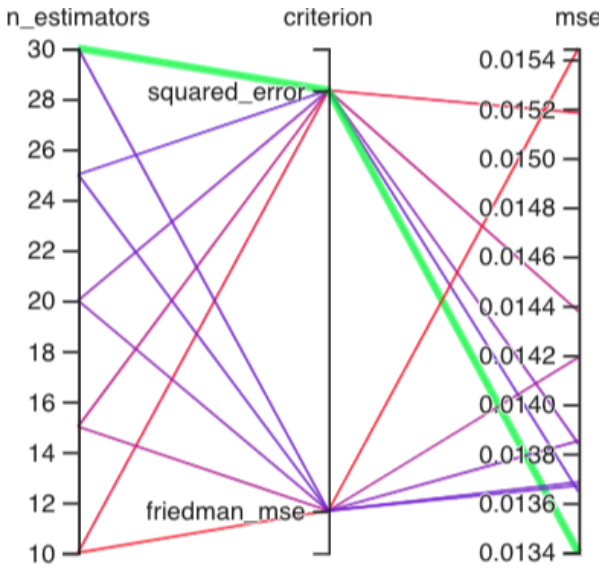
- **n_estimators**= 30
- **criterion**= squared_error

Risultati ottenuti dalla configurazione:

- MSE: 0.01234
- R2-score: 0.94431



(a) Parallel coordinates view random forest regression con PCA



(b) Parallel coordinates view random forest regression con PCA

Le prime 10 features selezionate per importanza dal random forest sono elencate di seguito:

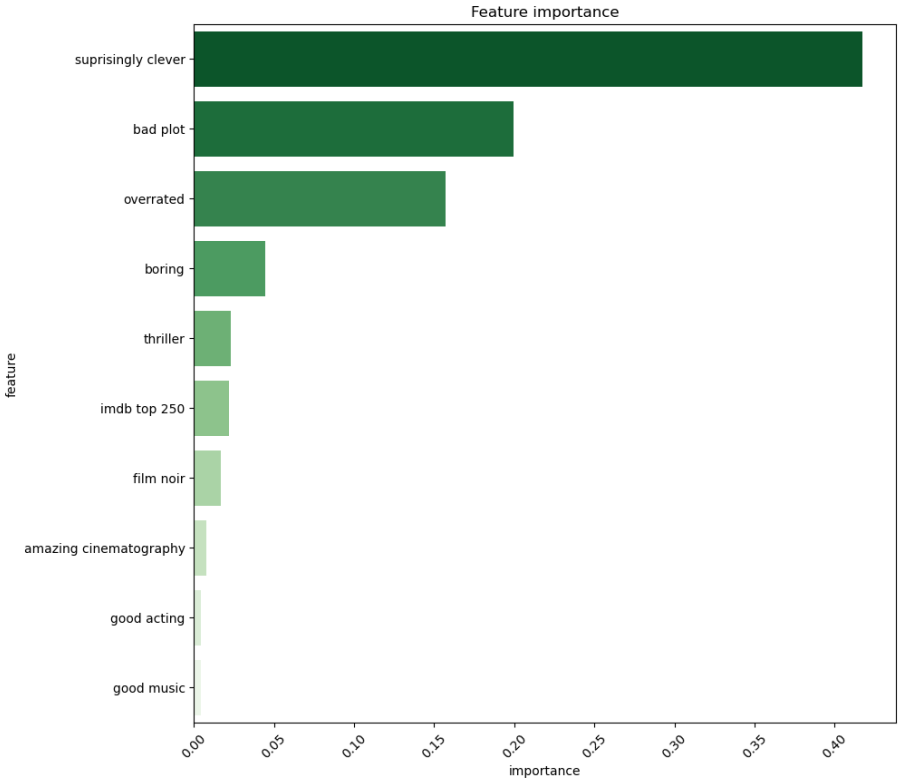


Figura 6.4: Top 10 features selezionate dal random forest.

K-Nearest Neighbors Regression

Il KNN è stato implementato utilizzando la funsione di scikit-learn. Il tuning degli iperparametri, ha restituito la migliore configurazione possibile per i parametri *n_neighbors* e *weights*. Di seguito i risultati ottenuti e le migliori combinazioni:

Con PCA:

Migliore configurazione ottenuta dal training:

- **n_neighbors**= 11
- **Wweights**= distance

Risultati ottenuti dalla configurazione:

- MSE: 0.04015
- R2-score: 0.81883

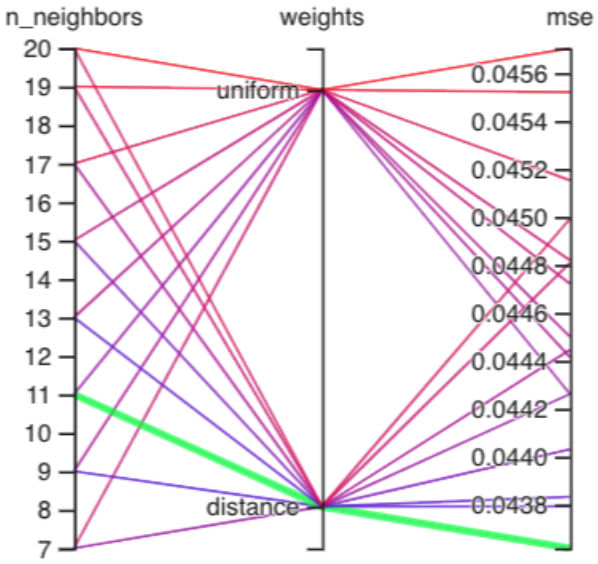
Senza PCA:

Migliore configurazione ottenuta dal training:

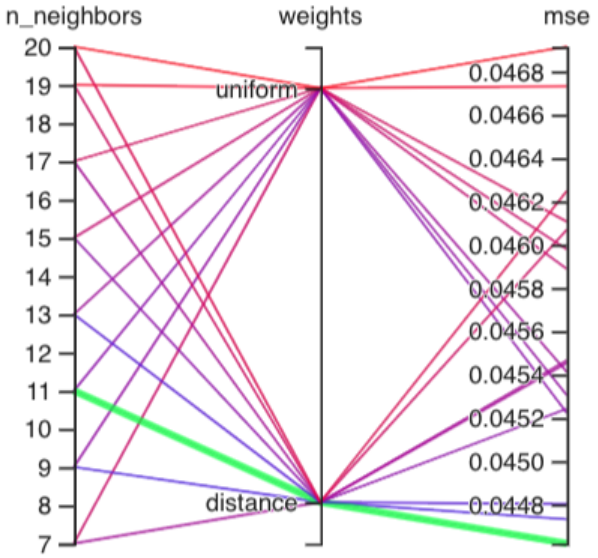
- **n_neighbors**= 11
- **weights**= distance

Risultati ottenuti dalla configurazione:

- MSE: 0.04049
- R2-score: 0.81729



(a) Parallel coordinates view KNN regression con PCA



(b) Parallel coordinates view KNN regression senza PCA

Support Vector Regressor

Il SVR è stato implementato utilizzando la funsione di scikit-learn. Il tuning degli iperparametri, ha restituito la migliore configurazione possibile per i parametri *kernel*, ϵ e *C*. Di seguito i risultati

ottenuti e le migliori combinazioni:

Con PCA:

Migliore configurazione ottenuta dal training:

- **kernel**= rbf
- ϵ= 0.001
- **C**= 1

Risultati ottenuti dalla configurazione:

- MSE: 0.00659
- R2-score: 0.97022

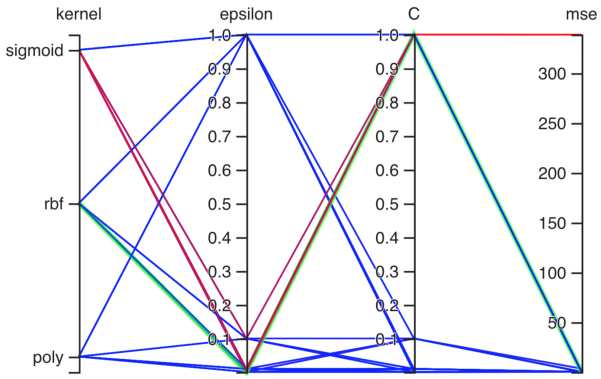
Senza PCA:

Migliore configurazione ottenuta dal training:

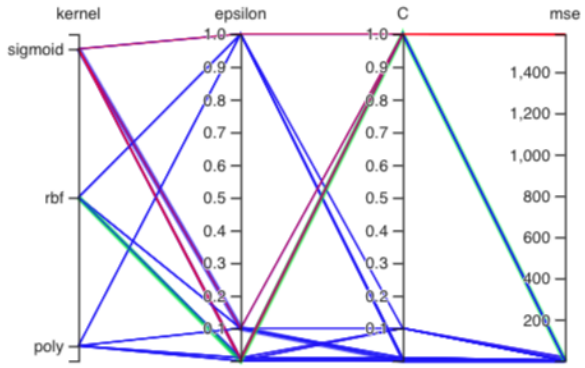
- **kernel**= rbf
- ϵ= 0.001
- **C**= 1

Risultati ottenuti dalla configurazione:

- MSE: 0.00521
- R2-score: 0.97647



(a) Parallel coordinates view SVR regression con PCA



(b) Parallel coordinates view SVR regression senza PCA

6.1.2 Neural Network

La rete neurale implementata in questo progetto è la feed-forward, come precedentemente presentata nel capitolo "Modeling".

Feed-forward network

L'implementazione della rete neurale è stata realizzata utilizzando la libreria Pytorch. Di seguito vengono indicate le migliori combinazioni degli iperparametri, tra le 144 disponibili, restituite dal tuning:

Con PCA: Migliore configurazione ottenuta dal training:

- **hidden_size**= 512
- **dropout_prob**= 0.2
- **dept**= 4
- **batch_size**= 16
- **lr**= 0.001

Risultati ottenuti dalla configurazione:

- MSE: 0.01471
- R2-score: 0.93248

Senza PCA:

Migliore configurazione ottenuta dal training:

- **hidden_size**= 128
- **dropout_prob**= 0.2
- **dept**= 3
- **batch_size**= 8
- **lr**= 0.01

Risultati ottenuti dalla configurazione:

- MSE: 0.01758
- R2-score: 0.91234

Di seguito vengono riportati le parallel coordinates view, Figure 6.7 e 6.9, e i scatter plot matrix view, Figure 6.8 e 6.10, con la PCA applicata e non applicata, per la rete neurale feed-forward implementata nel progetto. In verde viene evidenziata la migliore combinazione:

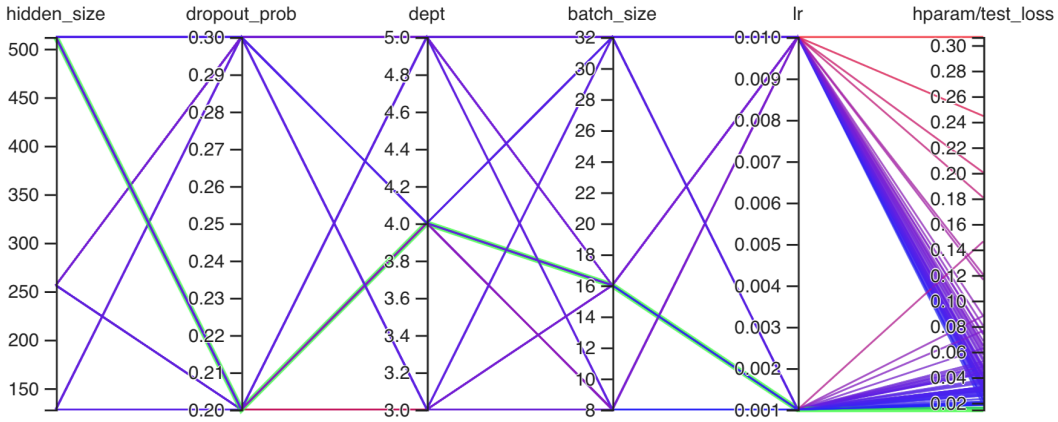


Figura 6.7: Parallel coordinates view rete neurale feed-forward con PCA

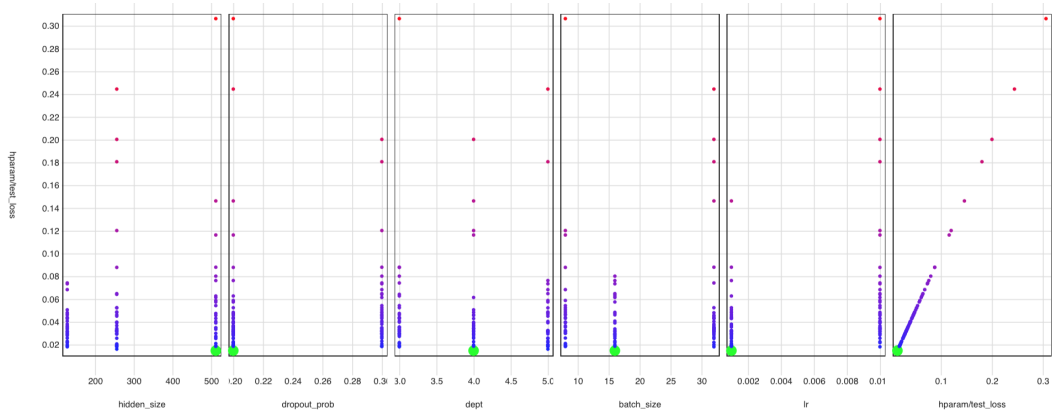


Figura 6.8: Scatter plot matrix view rete neurale feed-forward con PCA

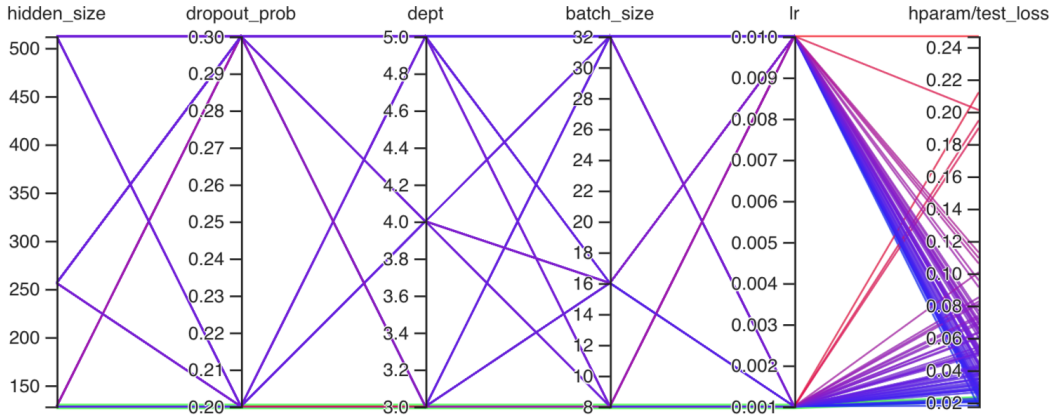


Figura 6.9: Parallel coordinates view rete neurale feed-forward senza PCA

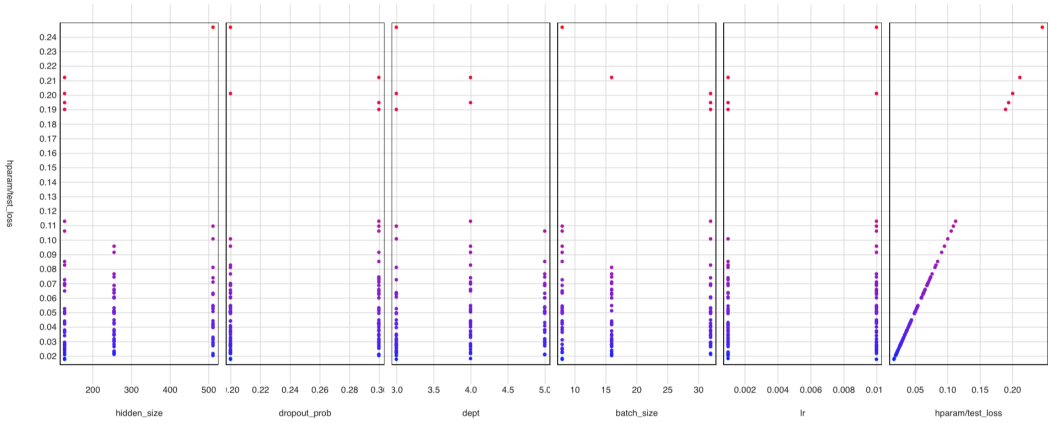


Figura 6.10: Scatter plot matrix view rete neurale feed-forward senza PCA

6.1.3 Tabnet

La rete TabNet implementata in questo progetto è stata realizzata utilizzando la libreria Pytorch, come precedentemente presentata nel capitolo "Modeling". Di seguito vengono indicate le migliori combinazioni degli iperparametri, tra le 54 disponibili, restituite dal tuning:

Con PCA:

Migliore configurazione ottenuta dal training:

- **batch_size**= 256
- **n_d**= 16
- **n_a**= 32
- **n_steps**= 3
- **n_indipendent**= 3

Risultati ottenuti dalla configurazione:

- MSE: 0.00882
- R2-score: 0.96019

Senza PCA:

Migliore configurazione ottenuta dal training:

- **batch_size**= 256
- **n_d**= 16
- **n_a**= 16
- **n_steps**= 3
- **n_indipendent**= 2

Risultati ottenuti dalla configurazione:

- MSE: 0.00546
- R2-score: 0.97537

Di seguito vengono riportati le parallel coordinates view, Figure 6.11 e 6.13, e i scatter plot matrix view, Figure 6.13 e 6.14, con la PCA applicata e non applicata, per il modello TabNet utilizzato nel progetto. In verde viene evidenziata la migliore combinazione:

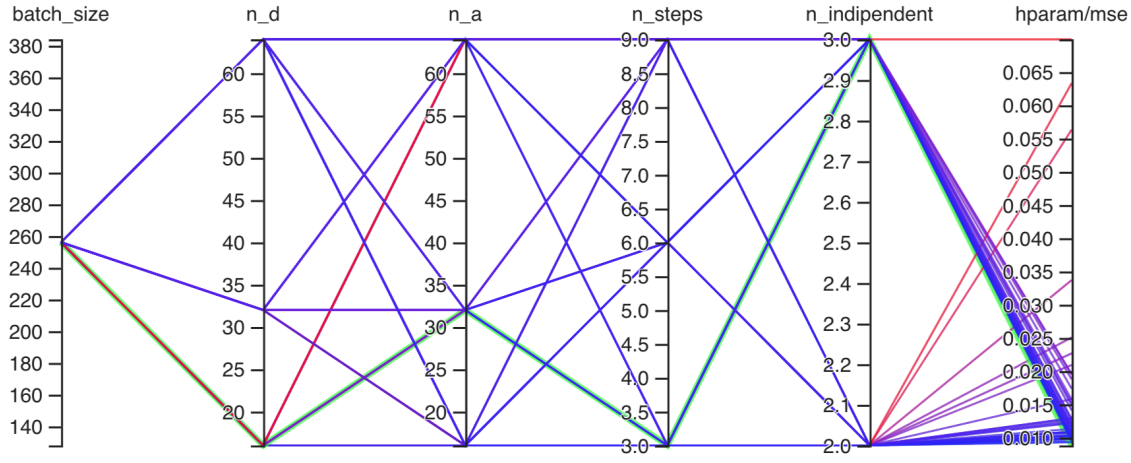


Figura 6.11: Parallel coordinates view rete TabNet con PCA

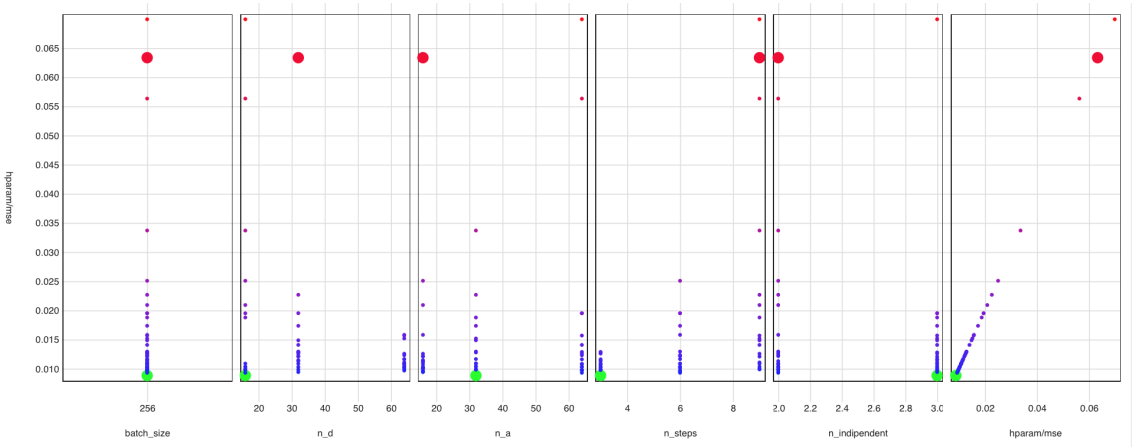


Figura 6.12: Scatter plot matrix view TabNet con PCA

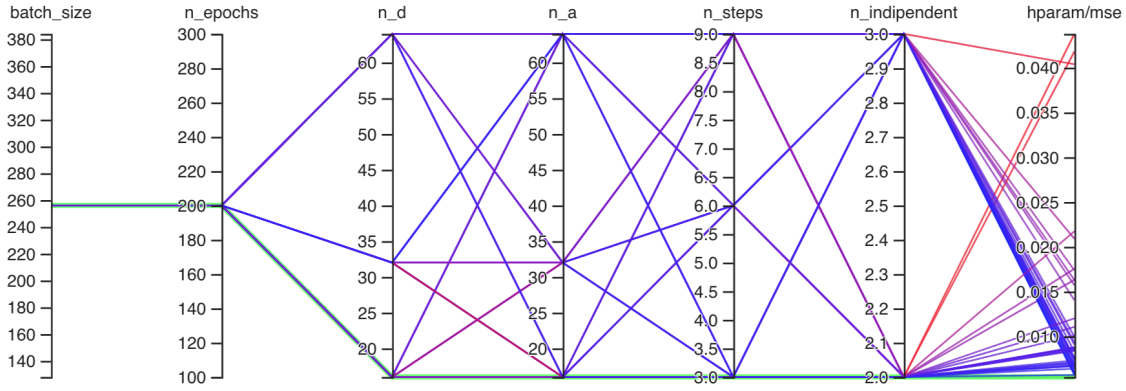


Figura 6.13: Parallel coordinates view TabNet senza PCA

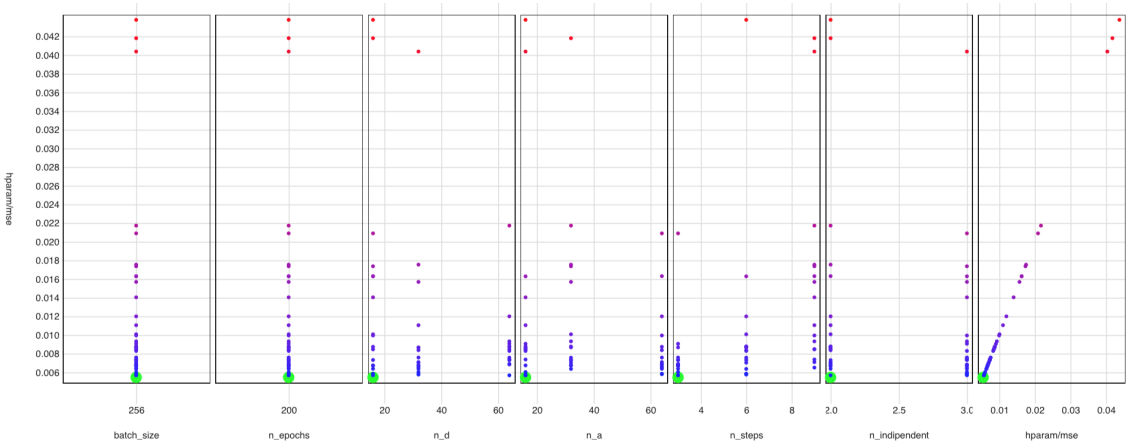


Figura 6.14: Scatter plot matrix view TabNet senza PCA

Capitolo 7

Conclusioni

Il progetto sviluppato ha lo scopo di creare un sistema che sia in grado di prevedere il voto medio dei film attraverso l'utilizzo delle tecniche di Machine Learning supervisionato. Per realizzare questo obiettivo sono state utilizzate tecniche supervisionate tradizionali, reti neurali e TabNet, un modello deep per dati tabulari.

I risultati ottenuti dimostrano che tutti i modello hanno delle buone prestazioni, che si vanno a migliore nel momento in cui non viene applicata la PCA. Quasi tutti i modelli tradizionali, fatta eccezione di KNN e Random Forest, hanno ottenuto delle prestazioni ottime.

Quindi, si intuisce che il problema di regressione di questo progetto può essere efficacemente risolto attraverso l'utilizzo di modelli lineari.

La rete neurale Feed-Forward ha ottenuto discrete performance, mentre il modello per dati tabulari TabNet ha ottenuto dei risultati leggermente migliori, rispetto ai modelli tradizionale, ma richiedendo maggiore tempo di training.

Appendice A

Riferimenti

1. Movielens: https://grouplens.org/datasets/movielens/
2. TabNet: https://arxiv.org/pdf/1908.07442.pdf