

Report Data Analytics

Isabella Marasco - 1040993

isabella.marasco3@studio.unibo.it

Laurea Magistrale in Informatica

A.A. 2022/2023

Contents

1	Introduzione	3
2	Data Acquisition	4
3	Data Visualization	5
4	Data Preprocessing	9
5	Modeling	10
5.1	Tecniche di ML supervisionato tradizionali	10
5.1.1	Linear Regression	10
5.1.2	Ridge Regression	10
5.1.3	Lasso Regression	11
5.1.4	Random Forest Regression	12
5.1.5	K-Nearest Neighbors Regression	12
5.1.6	Support Vector Regressor	12
5.2	Neural network	13
5.2.1	Feed-forward network	13
5.3	TabNet	14
6	Performance Evalution	18
6.1	Tecniche di ML supervisionato tradizionali	18
6.1.1	Linear Regression	19
6.1.2	Ridge Regression	19
6.1.3	Lasso Regression	20
6.1.4	Random Forest Regression	21
6.1.5	K-Nearest Neighbors Regression	23
6.1.6	Support Vector Regressor	23
6.2	Neural network	25
6.2.1	Feed-forward network	25
6.3	TabNet	28
7	Conclusioni	30

1 Introduzione

Questo progetto è parte della prova d'esame per il corso di Data Analytics A.A. 2022/2023. Lo scopo è quello di predire il voto medio di un film a partire dalle sue caratteristiche. Il dataset utilizzato è MovieLens [1], un recommendation system, al cui interno sono presenti più di 60000 film.

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

Per la realizzazione dello studio sono state implementate tutte le fasi della pipeline viste a lezione:

- Data Acquisition
- Data Visualization
- Data Preprocessing
- Modeling
- Performance Evaluation

2 Data Acquisition

La fase di acquisizione dei dati consiste nella raccolta dei dati che si vogliono analizzare. L'aquisizione può avvenire in diversi modi, tra cui l'acquisizione statica, utilizzata in questo progetto.

Il dataset utilizzato è MovieLens [1], un recommendation system per i film, al cui interno sono presenti più di 60000 film con le relative valutazioni e tag inoltre, ogni film dispone di un genoma che ne identifica una caratteristica e la sua rilevanza.

Il dataset è composto da sei file in formato csv:

- ***genome-scores***: ogni riga del file rappresenta la rilevanza di ogni film. Contiene l'ID del film, l'ID del tag e la rilevanza del tag.
- ***genome-tags***: contiene i tag utilizzati per valutare i film nel file genome-scores.csv. Ogni riga del file rappresenta un tag e contiene l'ID del tag e la sua descrizione.
- ***links***: contiene i link tra gli ID dei film nel dataset Movielens e gli ID dei film su altri siti web. Ogni riga del file rappresenta un film e contiene l'ID del film nel dataset Movielens, l'ID del film su IMDb e l'ID del film su TMDb.
- ***movies***: contiene informazioni sui film, tra cui il loro titolo con l'anno di uscita e il genere. Ogni riga del file rappresenta un film e contiene l'ID del film, il titolo con il loro anno di uscita e il genere.
- ***ratings***: contiene le valutazioni degli utenti per i film. Ogni riga del file rappresenta una valutazione e contiene l'ID dell'utente, l'ID del film, la valutazione e la data della valutazione.
- ***tags***: contiene le etichette aggiunte dagli utenti ai film. Ogni riga del file rappresenta un'etichetta e contiene l'ID dell'utente, l'ID del film e l'etichetta.

Per l'analisi, non sono stati utilizzati tutti i file disponibili, ma solo i file csv *movies*, *genome-scores*, *genome-tags* e *ratings*. Dato che i dati erano distribuiti in più file, si è deciso di crearne uno unico. In questo modo, si è ottenuto un dataset in cui a ciascun film è stato associato la rilevanza di ogni tag ad esso correlato. Per ogni film è stata calcolata la valutazione media delle recensione degli utenti, ottenendo dei valori continui. Per tale motivo si è deciso di affrontare tale problema come uno di regressione.

3 Data Visualization

La data visualization mira a comunicare i dati in modo chiaro ed efficace attraverso l'uso di grafici. È utile per esplorare i dati in modo efficiente e per scoprire delle loro possibili relazioni. In questo paragrafo vengono proposti diversi grafici.

Di seguito, in Figura 1 si può osservare la distribuzione dei voti all'interno del dataset. Osservando il grafico si può notare che la distribuzione dei voti presenta un'area di maggiore densità intorno al valore 3.5.

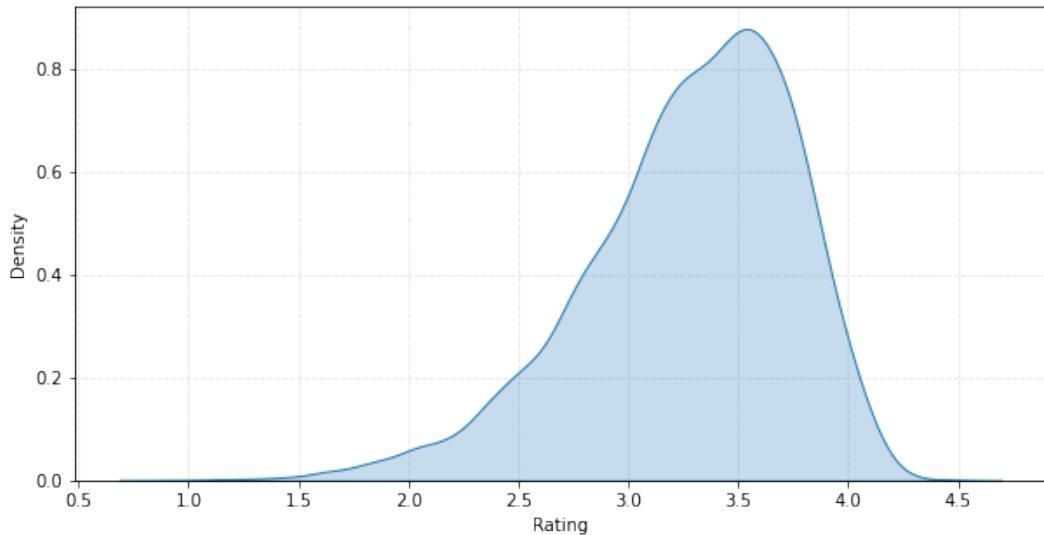


Figure 1: Distribuzione dei voti

Il seguente grafico, visibile in Figura 2, rappresenta la distribuzione della correlazione tra tag e voti. Dove si può notare che ci sono alcuni tag che hanno una maggiore correlazione con il voto rispetto ad altri.

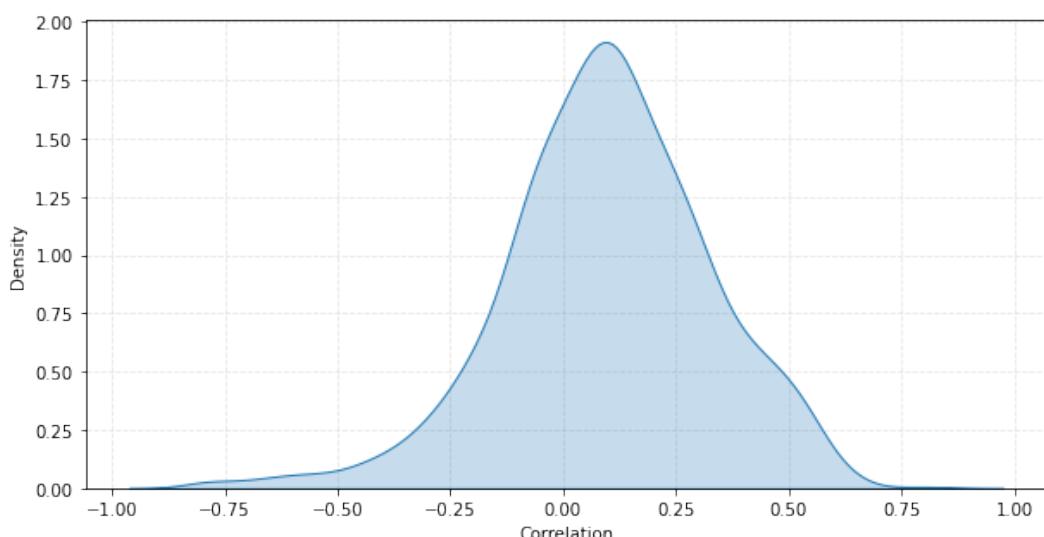


Figure 2: Distribuzione della correlazione tra tag e voti

Successivamente, sono stati realizzati due grafici, visualizzabili in Figura 3, per comprendere i primi 5 tag più e meno correlati con i voti, in modo da comprendere quali tag dei film potrebbero influenzare maggiormente la valutazione degli utenti.

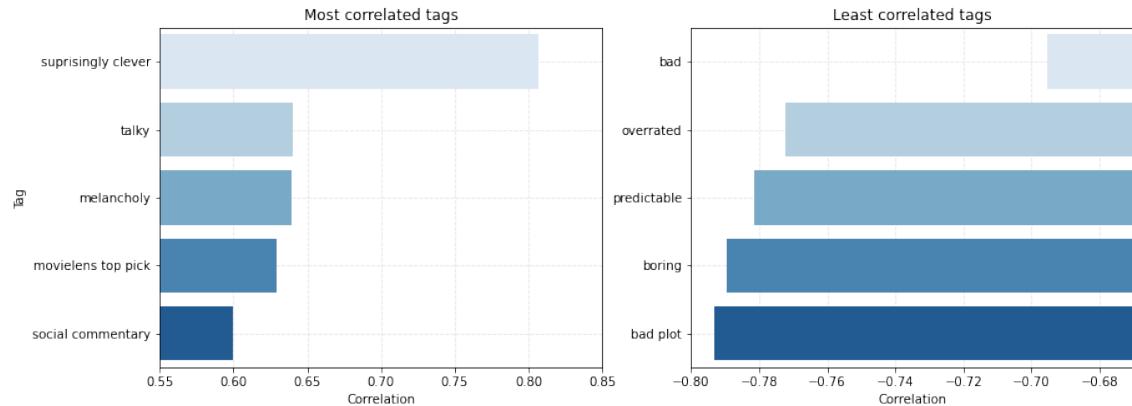


Figure 3: Correlazione tra tag e voti

Un altro grafico realizzato, visibile in Figura 4, indica la correlazione tra i generi e i voti. Si può notare che i generi che mostrano una correlazione maggiore positiva con i voti sono il documentario, drama e war, mentre i generi con una maggiore correlazione negativa sono l'horror, il scifi ed action.

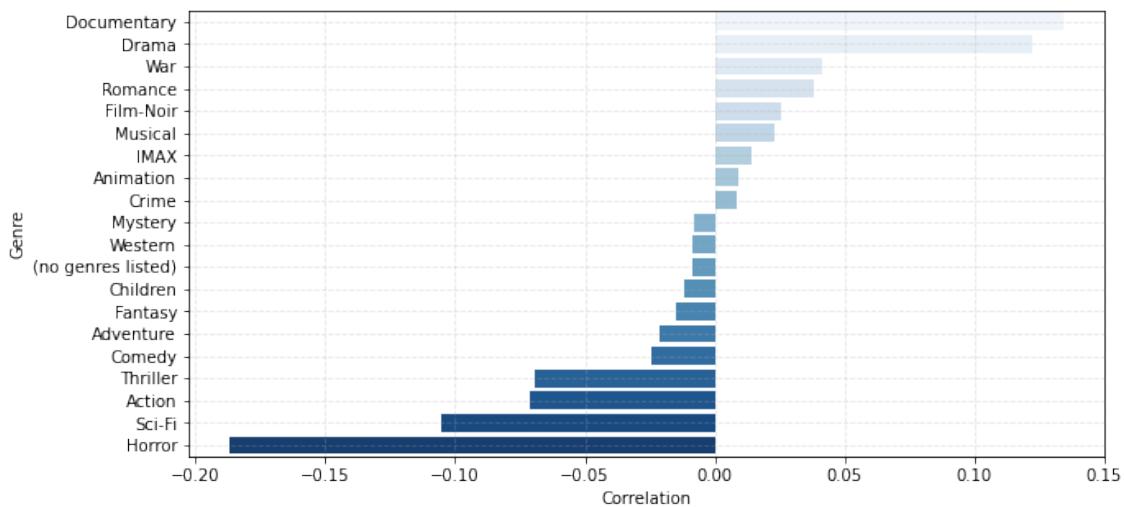


Figure 4: Correlazione tra generi e voti

Un ulteriore grafico, visibile in Figura 5, mostra il voto medio per ogni genere di film.

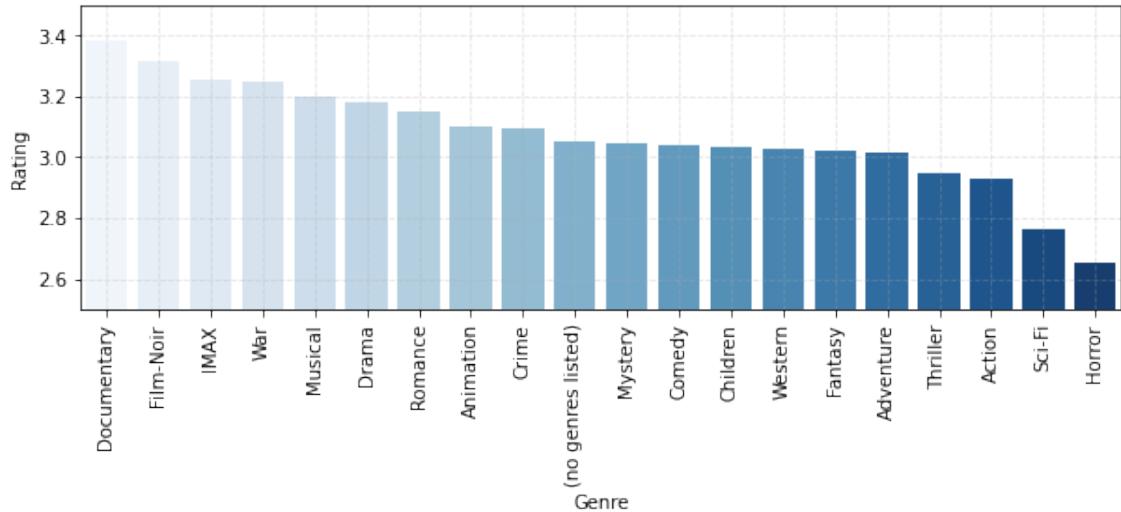


Figure 5: Voto medio per ogni genere

Infine, è stato realizzato un grafico utilizzando un algoritmo di riduzione di dimensionalità chiamato t-Distributed Stochastic Neighbor Embedding (t-SNE), visibile in Figura 6, per rappresentare i film sulla base del voto in uno spazio bidimensionale.

L’obiettivo dell’algoritmo è quello di trovare una rappresentazione a bassa dimensionalità dei dati, in cui i punti simili nel dataset originale sono anche vicini nello spazio a bassa dimensionalità. Questo processo consente di visualizzare i dati in un grafico bidimensionale o tridimensionale in cui le relazioni tra i dati originali sono conservate.

Il processo di t-SNE avviene in due passaggi principali. Inizialmente, viene calcolata una matrice di somiglianza tra i punti del dataset originale utilizzando una funzione di distanza come la distanza euclidea o la distanza cosenica. Questa matrice di somiglianza viene poi convertita in una distribuzione di probabilità utilizzando una funzione di similarità.

In seguito, l’algoritmo cerca di rappresentare i punti del dataset in uno spazio a bassa dimensionalità, 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 mappa bidimensionale dei dati. In questo specifico caso, l’algoritmo è stato applicato ai voti dei film per rappresentarli in uno spazio bidimensionale, consentendo di visualizzare la distribuzione dei voti.

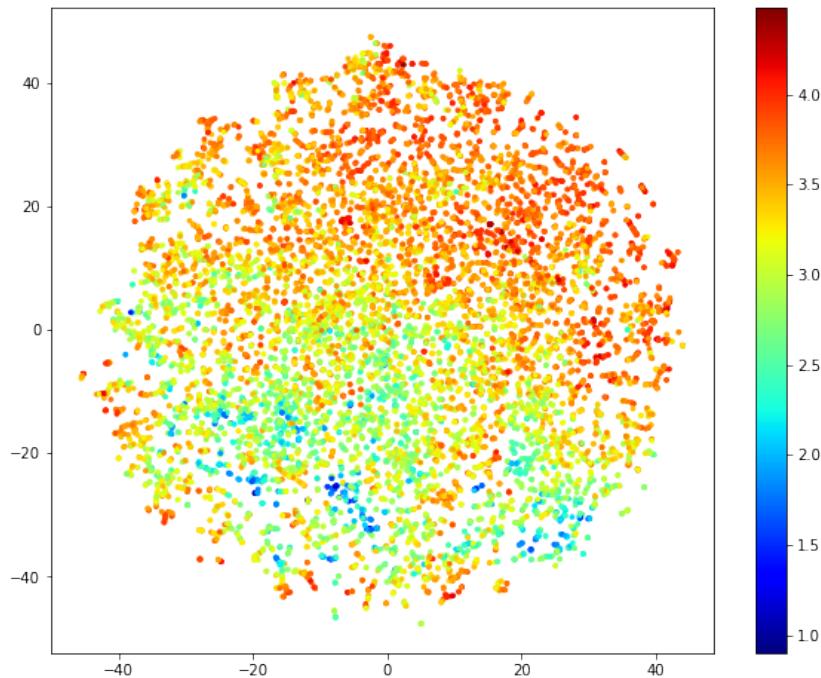


Figure 6: Rappresentazione 2D dei dati

Osservando la Figura ottenuta, si può notare che la distribuzione delle valutazione dei film sia graduale. I film con la votazione più bassa, sono colorati in blu e si trovano nella zona in basso, mentre i film con la votazione più alta sono rappresentati in rosso e si trovano nella parte superiore della Figura.

Inoltre, si può osservare che gran parte dei film hanno una valutazione compresa tra 3 e 3.5, queste sono aree più fitte poiché sono formate da un maggior numero di esempi. Tutto ciò è in linea con quanto visto nella distribuzione dei voti.

4 Data Preprocessing

Il preprocessing è il termine con cui ci si riferisce a tutte le attività che vengono eseguite sui dati grezzi al fine di renderli pronti per l'elaborazione successiva.

Questa fase è particolarmente importante perché i dati raccolti spesso contengono rumore, dati mancanti, valori anomali e altre imperfezioni che possono influire negativamente sui risultati dell'analisi. Attraverso questa fase, è possibile pulire e trasformare i dati in modo che siano più adatti alle esigenze dell'analisi.

In questa fase, per prima cosa, è stato fatto lo split del dataset in train, validation e test (70-10-20).

Il dataset utilizzato, non contiene né valori nulli, né valori duplicati inoltre, non vi è necessità di effettuare lo scaling dei dati perché questi sono già tutti compresi tra 0 e 1. Durante questa fase è stata applicata la Principal Component Analysis (PCA) una tecnica di riduzione della dimensionalità che mira a rappresentare le variazioni delle variabili del dataset utilizzando un numero inferiore di componenti principali.

Questa tecnica si basa sulla trasformazione di un insieme di variabili correlate in un nuovo insieme di variabili linearmente indipendenti, chiamate componenti principali. Attraverso la costruzione di un nuovo spazio, definito dalle componenti principali anziché dalle variabili originarie, è possibile rappresentare la natura multivariata dei dati in un numero relativamente ridotto di dimensioni, utilizzando tale rappresentazione per identificare la struttura dei dati. Dopo aver individuato le componenti principali, è possibile proiettare i dati originali su un nuovo sistema di coordinate, dove i primi assi sono i componenti principali calcolati. Ciò consente di rappresentare i dati in uno spazio a dimensioni inferiori rispetto a quello originale, senza perdere un'eccessiva quantità di informazione.

Al train set è stata applicata la PCA in cui la variabile dipendente è rappresentata dalla media dei voti per ogni film mentre tutte le altre sono utilizzate come variabili indipendenti, il numero di componenti da mantenere è stato settato a 0.95 ovvero, è stata conservata il 95% dell'energia dei dati.

Il dataset verrà testato sia con l'applicazione della PCA che senza per comprendere come, al variare della dimensionalità delle dataset, possano variare le performance.

5 Modeling

Nella fase di modeling, per predire il voto medio dei film sono stati utilizzati diverse tecniche di Machine Learning supervisionato, tra cui quelle tradizionali, tecniche deep basate su reti neurali ed infine, TabNet un modello deep per Tabular Data.

Questa analisi può essere considerata sia come un problema di regressione che di classificazione. In questa ricerca si è scelto di affrontarlo come un problema di regressione dato che la variabile target è un valore continuo.

5.1 Tecniche di ML supervisionato tradizionali

Le tecniche di Machine Learning supervisionato tradizionale sono metodi utilizzati per addestrare modelli di machine learning a partire da un set di dati etichettati, cioè un insieme di dati per cui sono noti sia i dati di input che l'output corrispondente.

5.1.1 Linear Regression

La regressione lineare è un metodo statistico utilizzato per studiare la relazione tra una variabile dipendente e, una o più variabili indipendenti continue. In particolare, ha l'obiettivo di trovare la retta che meglio rappresenta la relazione tra le variabili.

In questo caso specifico, la variabile dipendente è rappresentata dal voto medio dei film mentre le variabili indipendenti sono le caratteristiche dei film che possono influenzare la predizione ovvero, la valutazione dei diversi tag.

La regressione lineare è definita dalla funzione lineare:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p \quad (1)$$

Dove:

- \hat{y} è la variabile dipendente o variabile da predire
- x è la variabile indipendente o variabile di input
- β_0 è l'intercetta della retta di regressione
- β_1 è la pendenza della retta di regressione

5.1.2 Ridge Regression

La Ridge Regression è una tecnica di regressione lineare che utilizza la penalizzazione L2 andando ad introdurre un termine di regolarizzazione quadratico (λ) nella formula della regressione lineare. Aumentando il valore di λ , si limita l'importanza del contributo di alcune variabili attraverso la riduzione dei valori dei coefficienti di regressione (β). In altre parole, al crescere di λ , alcuni coefficienti di regressione vengono "schiacciati" e il contributo delle variabili corrispondenti diventa meno rilevante.

Questo approccio consente di ridurre la dimensionalità del modello, ovvero di limitare il numero di variabili che contribuiscono significativamente alla previsione del target.

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

Dove:

- y_i è la variabile di risposta per l'osservazione
- β_0 è l'intercetta
- β_j rappresenta il coefficiente di regressione associato alla variabile predittiva
- x_{ij} rappresenta il valore della variabile predittiva j per l'osservazione i
- λ è il parametro di regolarizzazione, che controlla la quantità di *shrinkage* applicata ai coefficienti di regressione. Aumentando il valore di λ , la regolarizzazione diventa più forte e i coefficienti di regressione vengono ridotti verso lo zero.

Di seguito vengono specificati gli iperparametri e i relativi valori utilizzati all'interno di questo progetto, mentre nel paragrafo 6.1.2 verrà specificata la configurazione migliore e i risultati ottenuti:

- alpha = [0.0001, 0.001, 0.1, 0.5, 1, 5, 10, 20]:

5.1.3 Lasso Regression

La Lasso Regression è una tecnica di regressione lineare che si concentra sulla selezione delle variabili più rilevanti per un modello predittivo mentre penalizza le variabili meno rilevanti, andando ad utilizzare la penalità L1 che va ad azzerrare i pesi che non contribuiscono molto nel modello.

Quindi, introduce un termine di regolarizzazione alla funzione di perdita del modello, che penalizza i coefficienti delle variabili meno rilevanti, questo è controllato da un parametro di regolarizzazione chiamato λ che determina la quantità di penalizzazione da applicare. Più grande è α , maggiore sarà la penalizzazione e più variabili verranno "contratte" verso zero.

$$Lasso = \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^p |\beta_j| \quad (3)$$

Dove:

- n è il numero di osservazioni nel set di dati
- p è il numero di variabili esplicative nel modello
- y_i è la variabile dipendente nella i -esima osservazione
- β_0 è l'intercetta del modello x_{ij} è il valore della variabile esplicativa j -esima nella i -esima osservazione
- β_j è il coefficiente di regressione della variabile esplicativa j -esima
- λ è un iperparametro che controlla la forza della regolarizzazione

Di seguito vengono specificati gli iperparametri e i relativi valori utilizzati all'interno di questo progetto, mentre nel paragrafo 6.1.3 verrà specificata la configurazione migliore e i risultati ottenuti:

- alpha = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5]:

5.1.4 Random Forest Regression

Il random forest è un algoritmo di apprendimento supervisionato basato sulla creazione di un insieme di alberi decisionali.

Durante l'addestramento, vengono creati numerosi alberi decisionali indipendenti tra loro. Ciascun albero viene generato su un sottoinsieme casuale dei dati di training, in modo da ottenere modelli differenti tra loro. Inoltre, ogni albero utilizza soltanto un sottoinsieme casuale delle variabili di input per effettuare le previsioni.

Di seguito vengono specificati gli iperparametri e i relativi valori utilizzati all'interno di questo progetto, mentre nel paragrafo 6.1.4 verrà specificata la configurazione migliore e i risultati ottenuti:

- `n_estimator = [10, 15, 20, 25, 30]`: indica il numero di alberi nella foresta.
- `criterion = ['squared_error', 'friedman_mse']`: misura la qualità dello split. Dove:
 - "squared_error": per l'errore quadratico medio, che equivale alla riduzione della varianza come criterio di selezione delle caratteristiche e minimizza la perdita L2 utilizzando la media di ciascun nodo terminale.
 - "friedman_mse": utilizza l'errore quadratico medio con il punteggio di miglioramento di Friedman per il potenziale split.

5.1.5 K-Nearest Neighbors Regression

L'algoritmo K-Nearest Neighbors (K-NN) cerca i k dati più vicini presenti all'interno del train set.

Di seguito vengono specificati gli iperparametri e i relativi valori utilizzati all'interno di questo progetto, mentre nel paragrafo 6.1.5 verrà specificata la configurazione migliore e i risultati ottenuti:

- `n_neighbors = [7, 8, 9, 10, 15, 20]`: indica il numero di neighbors da prendere in considerazione.
- `weights = ['uniform', 'distance']`: indica il peso che viene utilizzato nella previsione. Dove:
 - uniform: i pesi sono uniformi, tutti i punti sono pesati allo stesso modo.
 - distance: il peso dei punti è inverso alla loro distanza. Quindi, i vicini più vicini di un punto di interrogazione avranno un'influenza maggiore rispetto ai vicini più lontani.

5.1.6 Support Vector Regressor

L'algoritmo Support Vector Regressor (SVR) cerca il piano che meglio approssima i dati di input, ma che al contempo sia il più possibile lontano dai punti che si trovano al di fuori del boundary line, questi punti sono chiamati vettori di supporto.

Il modello può essere configurato con diversi iperparametri che possono portare a dei miglioramenti delle performance.

Di seguito vengono specificati gli iperparametri e i relativi valori utilizzati all'interno di questo progetto, mentre nel paragrafo 6.1.6 verrà specificata la configurazione migliore e i risultati ottenuti:

- `kernel = ['linear', 'poly', 'rbf']`: indica il tipo di kernel da utilizzare.
- `epsilon = [0.001, 0.01, 0.1, 1]`: indica l'epsilon-tube, ovvero la larghezza della fascia di errore consentita intorno alla linea di regressione.

5.2 Neural network

Le reti neurali sono un tipo di algoritmo di apprendimento automatico ispirato al funzionamento del cervello umano. La rete neurale è formata da un insieme di neuroni artificiali, i quali sono in grado di elaborare informazioni in ingresso, trasmetterle ad altri neuroni della rete e produrre un'output in base a ciò che è stato elaborato. Durante il train, la rete neurale si adatta ai dati di train attraverso l'ottimizzazione dei pesi delle connessioni tra i neuroni, in modo da minimizzare l'errore di predizione.

5.2.1 Feed-forward network

La rete neurale, utilizzata in questo progetto, è la feed-forward in cui ogni layer di input ed il relativo layer di output si muovono in una sola direzione, ovvero non sono presenti cicli o connessioni a ritroso. Questa rete è composta da uno o più layer di neuroni dove, ogni neurone è collegato a tutti i neuroni del layer successivo. I neuroni del layer di input ricevono l'input, elaborano l'informazione e la passano al layer successivo. Questa elaborazione consiste in una combinazione lineare degli input pesati e l'applicazione di una funzione di attivazione non lineare. La funzione di attivazione è importante perché introduce la non linearità nella rete, consentendo di modellare problemi più complessi.

La rete feedforward, solitamente, viene addestrata utilizzando l'algoritmo di discesa del gradiente, che permette di aggiornare i pesi dei neuroni per ridurre l'errore tra l'output atteso e quello effettivo della rete. Quindi, l'obiettivo dell'algoritmo di discesa del gradiente è quello di trovare i valori delle variabili del modello che minimizzano la funzione di perdita. Nell'immagine 7 è possibile vedere un esempio di rete FeedForward.

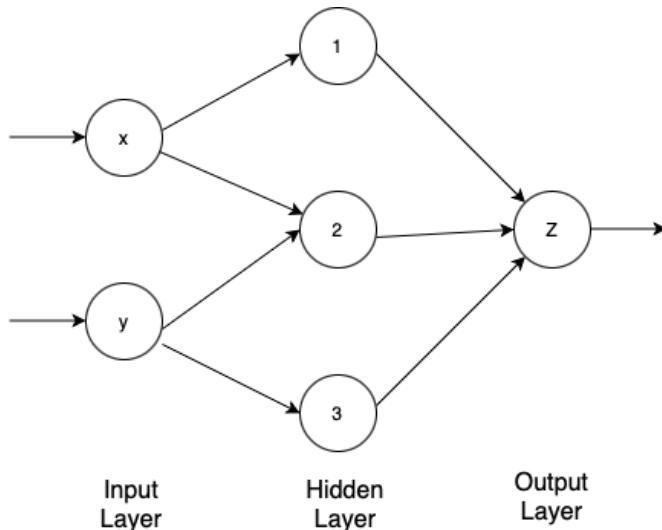


Figure 7: Architettura rete feed-forward

La rete implementata in questo progetto è composta da un numero di layer variabile. Il primo layer della rete è fully connected che prende in input i dati dell'input e li mappa in un nuovo spazio di feature utilizzando dei pesi appresi durante l'addestramento della rete mentre l'ultimo è il layer di output. In mezzo c'è un numero variabile di hidden layer.

Di seguito è possibile osservare il codice implementato per realizzare la rete.

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

Di seguito vengono specificati gli iperparametri e i relativi valori utilizzati all'interno di questo progetto mentre, nel paragrafo 6.2 verrà specificata la configurazione migliore e i risultati ottenuti:

- `hidden_size` = [128, 256, 512]: indica la dimensione dei hidden layer.
- `dropout_prob` = [0.2, 0.3, 0.4]: indica la probabilità di disattivare casualmente alcuni neuroni durante l'addestramento della rete.
- `dept` = [3, 4, 5]: indica il numero di hidden layer nella rete.
- `batch_size` = [8, 16, 32]: indica la dimensione dell'input
- `learning_rate` = [0.001, 0.01]: determina la dimensione del passo a ogni iterazione mentre ci si sposta verso un minimo della funzione di perdita.

Il numero di epoch è stato fissato, in modo arbitrario, a 200.

5.3 TabNet

TabNet [2] è un algoritmo di machine learning utilizzato per l'elaborazione di dati tabulari, nato in risposta alle limitate prestazioni delle reti neurali deep applicate ai dati tabulati, è stato sviluppato nel 2019 da diversi ricercatori di Google Research.

TabNet prende in input i dati grezzi e utilizza l'attenzione sequenziale per selezionare le feature da utilizzare in ogni step decisionale, ciò permette una migliore interpretazione e un migliore apprendimento. Utilizza più blocchi decisionali che si concentrano sull'elaborazione di un sottoinsieme di features di input, visibile in Figura 8.

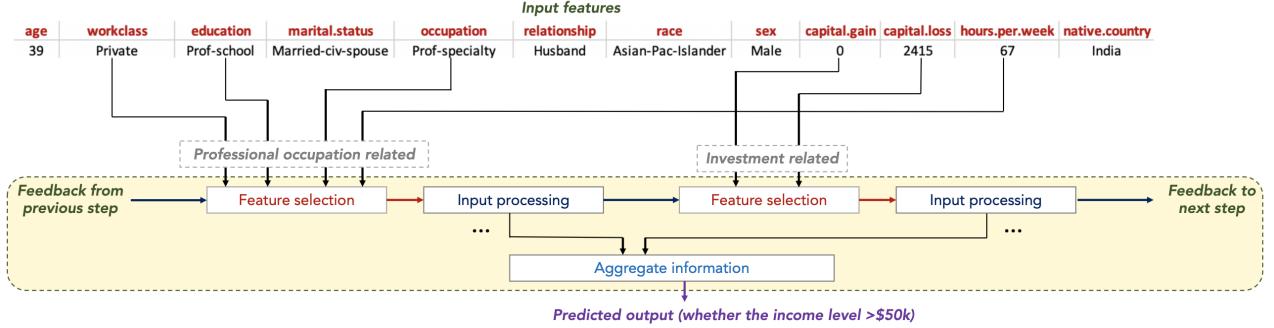


Figure 8: Selezione feature

Inoltre, utilizza il pre-training non supervisionato, visibile in Figura 9, questo perchè da alcuni studi realizzati è emerso che con i dati tabulati ci sono dei miglioramenti delle prestazioni.

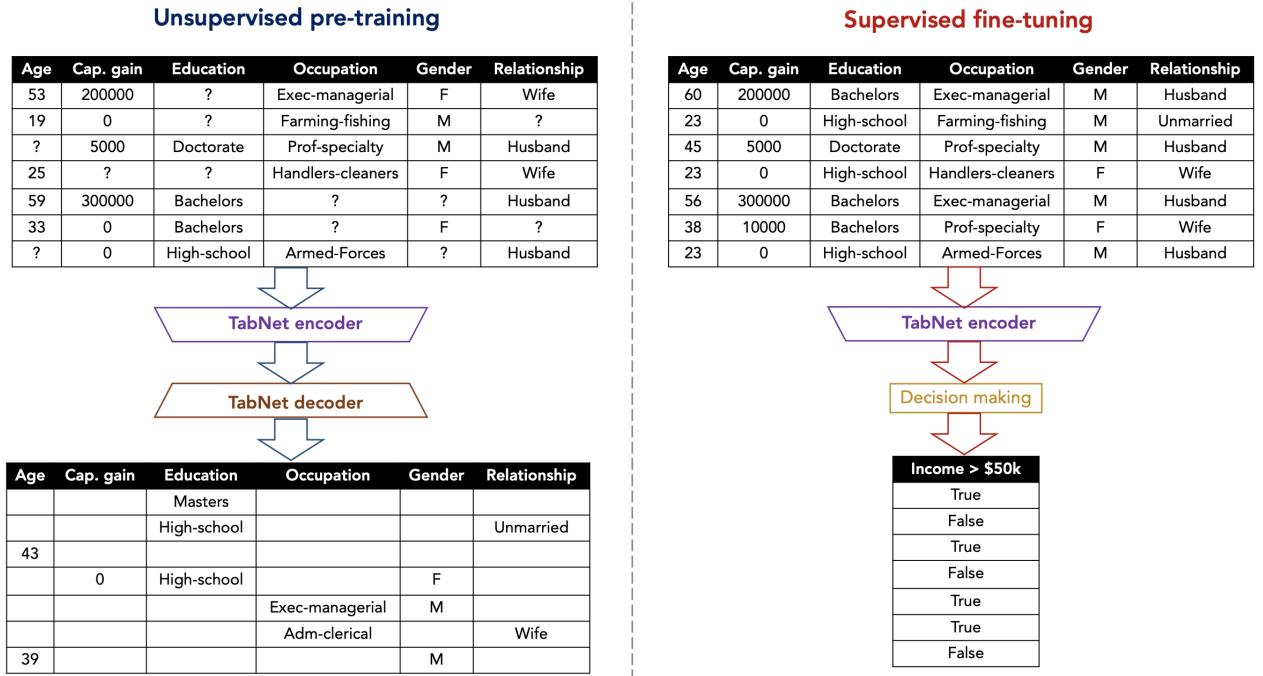


Figure 9: Pre-training non supervisionato

Durante la fase di training, le feature vengono passate all'encoder composto dall'*attentive transformer* il quale, si occupa di selezionare le feature più rilevanti per poi passarle al *mask* che utilizza le informazioni precedenti per aggregare quanto ogni feature è stata utilizzata nelle fasi decisionali precedenti. Ciò consente di identificare le caratteristiche salienti in modo più efficiente e accurato. La normalizzazione dei coefficienti viene fatta utilizzando sparsemax, che produce una distribuzione sparsa ovvero, una distribuzione in cui sono presenti pochi valori non nulli, così da andare a selezionare solo le informazioni più rilevanti ed escludere le altre.

Infine, le feature selezionate da *mask* vengono passate in input al *feature transformer* che le trasforma in una nuova rappresentazione, utilizzata per prendere decisioni utilizzando una serie di layer di trasformazione. Infatti, il *feature transformer* è composto da tre layer sequenziali ovvero, Fully-Connected, Batch Normalization e Gated Linear Unit (GLU) ed è diviso in due blocchi *shared across decision step* e *decision step dependent*, dove i primi catturano le

informazioni comuni tra i diversi step decisionali, mentre i secondi catturano le informazioni specifiche a ciascuna fase decisionale.

Il decoder è formato da un blocco di feature trasformer per ogni step ed utilizza un’architettura a blocchi inversa rispetto al *feature transformer*. Ogni blocco di trasformazione prende in input la rappresentazione intermedia prodotta dal *feature transformer* e la trasforma in una nuova rappresentazione con dimensioni progressivamente maggiori, fino a raggiungere l’output finale del modello.

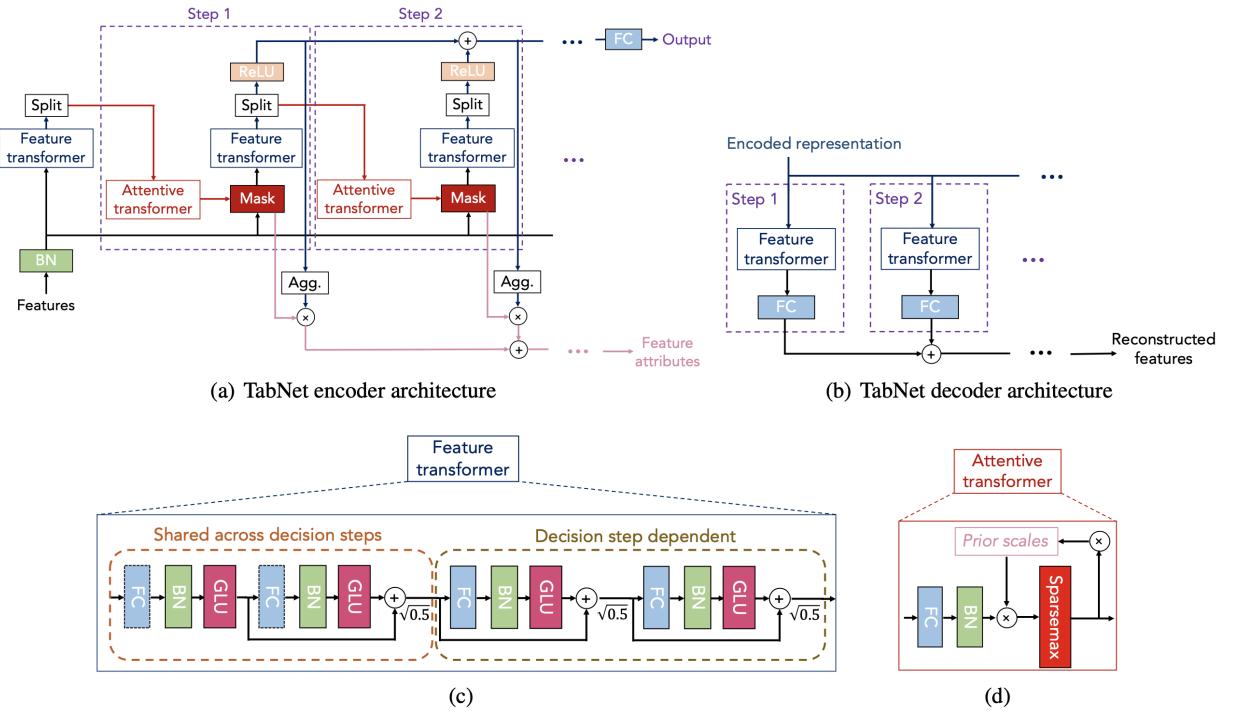


Figure 10: Architettura TabNet

Di seguito, viene riportato il codice del modello implementando:

```

1 def get_model(n_d, n_a, n_steps, n_independent):
2     model = TabNetRegressor(
3         n_d=n_d,
4         n_a=n_a,
5         n_steps=n_steps,
6         n_independent=n_independent
7     )
8     return model

```

Di seguito vengono specificati gli iperparametri e i relativi valori utilizzati all’interno di questo progetto, mentre nel paragrafo 6 verrà specificata la configurazione migliore e i risultati ottenuti:

- `batch_size = [256]`: numero di sample contenuti in ogni batch.
- `n_d = [8, 16, 32]`: determina la dimensione del layer di predizione.
- `n_a = [8, 16, 32]` : determina la dimensione dello spazio di output del attentive transformer.
- `n_steps = [3, 5, 7]`: numero di step nell’architettura.

- `n_independent = [2, 3]` : numero di layer GLU indipendenti per ogni blocco GLU.

Di seguito, è possibile osservare l'implementazione del codice del training del modello:

```

1 for b, n_e, n_d, n_a, n_s, n_i in params:
2     model = get_model(b, n_e, n_d, n_a, n_s, n_i)
3     model.fit(
4         X_train=X_train,
5         y_train=y_train,
6         eval_set=[(X_val, y_val)],
7         eval_name=["mse"],
8         patience=10,
9         batch_size=b
10        virtual_batch_size = 128
11    )
12
13    y_pred = model.predict(X_test)
14    mse = mean_squared_error(y_test, y_pred)
15    r2 = r2_score(y_test, y_pred)
16
17    print('MSE:', mse)
18    print('R2 Score:', r2)
19    if mse < best_mse:
20        best_mse = mse
21        best_model = copy.deepcopy(model)
22        best_params = (b, n_e, n_d, n_a, n_s, n_i)
23        print('Best model updated')

```

Il numero di epoch è stato fissato, in modo arbitraio, a 200.

6 Performance Evaluation

Nel seguente paragrafo si andranno a vedere i risultati ottenuti dai diversi modelli utilizzati, di cui si è parlato nel paragrafo 5. Le misure utilizzate per valutare tutti i modelli sono:

- **Mean Squared Error (MSE):** misura l'errore quadratico medio tra le previsioni del modello e i valori osservati. Un valore basso indica che il modello ha una buona capacità di predire i valori reali, mentre un valore alto indica che il modello ha una scarsa capacità di predizione. Di seguito la formula per calcolarlo:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (4)$$

dove n è il numero di campioni, y_i è il valore reale del campione i e \hat{y}_i è il valore previsto per il campione i .

- Il **coefficiente di determinazione (R2):** misura quanto bene il modello si adatta ai dati. Può assumere valori compresi tra 0 e 1, dove 1 indica una perfetta aderenza del modello ai dati di output reali e 0 indica che il modello non è in grado di spiegare la variazione nei dati di output. La formula per calcolarlo è la seguente:

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (5)$$

dove n è il numero di campioni, y_i è il valore reale del campione i , \hat{y}_i è il valore previsto per il campione i e \bar{y} è la media dei valori reali.

Ogni modello è stato allenato e testato due volte, una prima volta con l'applicazione della PCA sui dati e una seconda volta senza.

6.1 Tecniche di ML supervisionato tradizionali

Le tecniche di Machine Learning supervisionato tradizionale sono la linear regression, il ridge regression, il random forest regression, il KNN e SVR. Per ogni modello si vedranno le migliori configurazioni e i risultati ottenuti con e senza l'applicazione della PCA.

Il tuning degli iperparametri, per tutti i modelli deep, è stato realizzato utilizzando GridSearchCV, presente in sklean. GridSearchCV prova tutte le combinazioni dei valori passati agli iperparametri e valuta il modello per ciascuna combinazione utilizzando il metodo di convalida incrociata. I parametri utilizzati nel GridSearchCV sono:

- param_grid: consente la ricerca su qualsiasi sequenza di impostazioni dei parametri.
- cv=5: numero di cross-validation per ogni combinazione di parametri.
- scoring=neg_mean_squared_error: metrica di valutazione da utilizzare per classificare i risultati.
- return_train_score=True: include i punteggi del train.

6.1.1 Linear Regression

La linear regression, di cui parlato nel paragrafo 5.1.1, è stata implementata utilizzando la funzione di scikit-learn.

Di seguito, in tabella 1, si possono osservare i risultati ottenuti dal modello.

Metrica	PCA	No PCA
mse	0.00643	0.00544
R2	0.97096	0.97543

Table 1: Risultati linear regression

6.1.2 Ridge Regression

La ridge regression, di cui parlato nel paragrafo 5.1.2, è stata implementata utilizzando la funzione di scikit-learn.

Dal tuning degli iperparametri fatto, è risultato che il valore migliore per il parametro *alpha* è 5, sia per i dati con PCA che senza.

Di seguito, in tabella 2, si possono osservare i risultati ottenuti dal modello.

Metrica	PCA	No PCA
mse	0.00639	0.00532
R2	0.97113	0.97598

Table 2: Risultati ridge regression

In Figura 11 e 12, è possibile osservare i parallel coordinates view ottenuti. In verde è indicata la configurazione migliore.

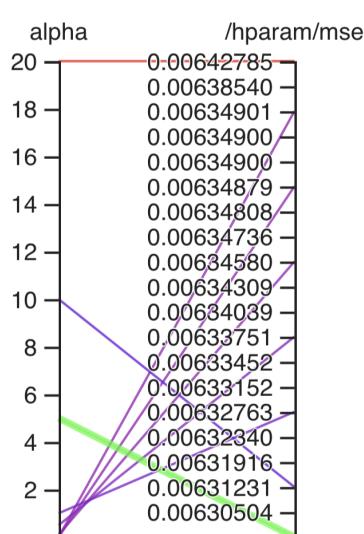


Figure 11: Parallel coordinates view ridge regression con PCA

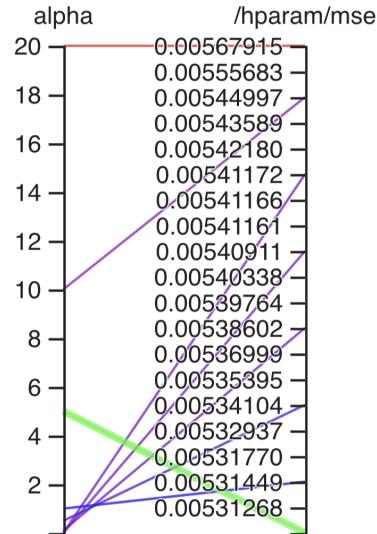


Figure 12: Parallel coordinates view ridge regression senza PCA

6.1.3 Lasso Regression

Il lasso regression, di cui parlato nel paragrafo 6.1.3, è stata implementata utilizzando la funziona di scikit-learn.

Dal tuning dei parametri fatto, è risultato che il valore migliore per il parametro *alpha* è 1e-05 sia per i dati in cui viene applicata la PCA sia in quelli in cui non viene applicata.

Di seguito, in tabella 3, si possono osservare i risultati ottenuti dal modello.

Metrica	PCA	No PCA
mse	0.00641	0.00542
R2	0.97106	0.97553

Table 3: Risultati lasso regression

In Figura 13 e 14, è possibile osservare i parallel coordinates view ottenuti. In verde è indicata la configurazione migliore.

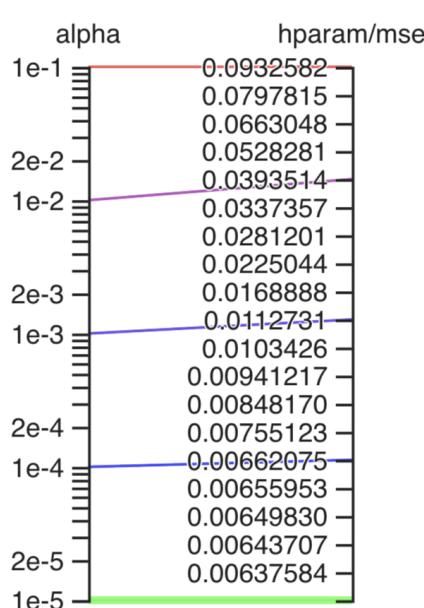


Figure 13: Parallel coordinates view lasso regression con PCA

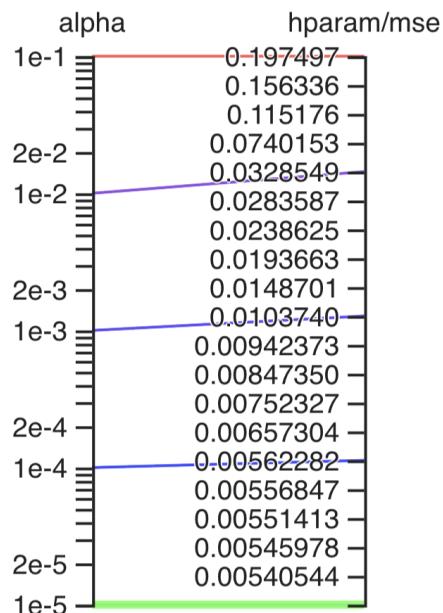


Figure 14: Parallel coordinates view lasso regression senza PCA

Dato che lasso regression tende ad azzerare i coefficienti delle variabili meno rilevanti, nella tabella 4 è possibile osservare la correlazione tra le prime 10 features i cui coefficienti sono stati azzerati e i voti dei film.

Feature	Correlazione
007	-0.097706
1920s	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

Table 4: Correlazione delle prime 10 feature con i coefficienti settati a 0 e i voti

6.1.4 Random Forest Regression

Il random forest, spiegato nel paragrafo 5.1.4, è stato implementato utilizzando scikit-learn. Dal tuning degli iperparametri fatto, è risultato che i valori migliori sono quelli visibili in tabella 5

Parametro	Migliore configurazione (PCA)	Migliore configurazione (senza PCA)
n_estimators:	30	30
criterion	squared_error	friedman_mse

Table 5: configurazione migliore Random forest

Di seguito, in tabella 6, si possono osservare i risultati ottenuti dal modello.

Metrica	PCA	No PCA
mse	0.03735	0.01255
R2	0.83150	0.94334

Table 6: Risultati Random forest

In Figura 15 e 16, è possibile osservare i parallel coordinates view ottenuti. In verde è indicata la configurazione migliore.

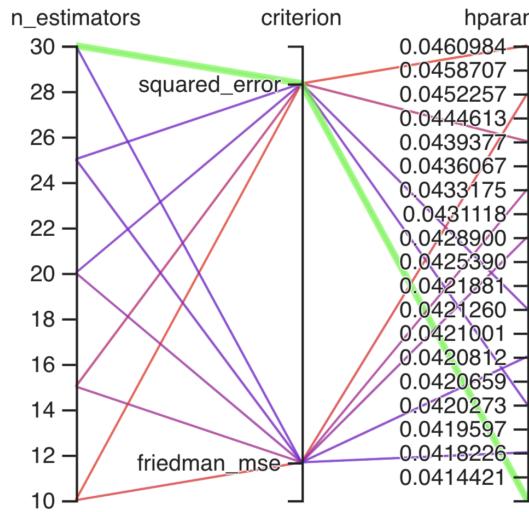


Figure 15: Parallel coordinates view random forest con PCA

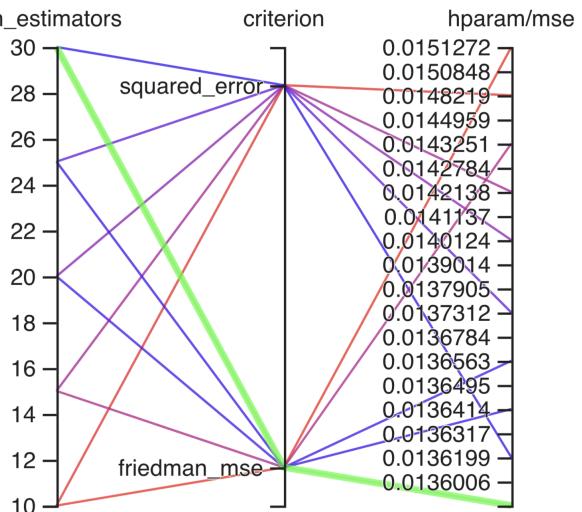


Figure 16: Parallel coordinates view random forest senza PCA

Di seguito, in Figura 17, è possibile osservare le prime 10 feature più importanti selezionati dal random forest.

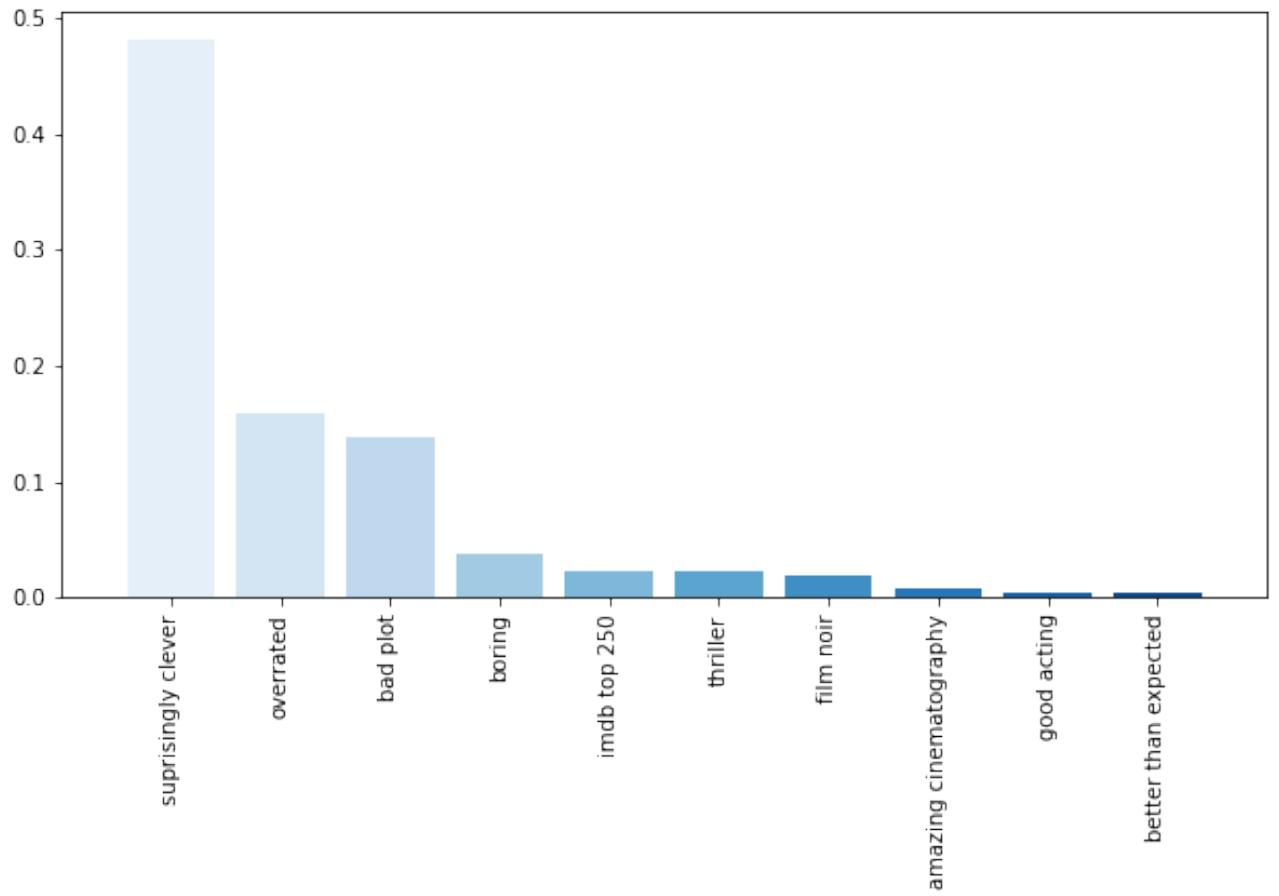


Figure 17: Le 10 feature più importanti

6.1.5 K-Nearest Neighbors Regression

Il KNN, spiegato nel paragrafo 5.1.5, è stato implementato utilizzando scikit-learn.

Dal tuning degli iperparametri fatto, è risultato che i valori migliori per i parametri sono quelli visibili in tabella 7

Parametro	Migliore configurazione (PCA)	Migliore configurazione (senza PCA)
n_neighbors	9	10
weights	distance	distance

Table 7: configurazione migliore KNN

Di seguito, in tabella 8, si possono osservare i risultati ottenuti dal modello.

Metrica	PCA	No PCA
mse	0.04028	0.04050
R2	0.81828	0.81727

Table 8: Risultati KNN

In Figura 18 e 19, è possibile osservare i parallel coordinates view ottenuti. In verde è indicata la configurazione migliore.

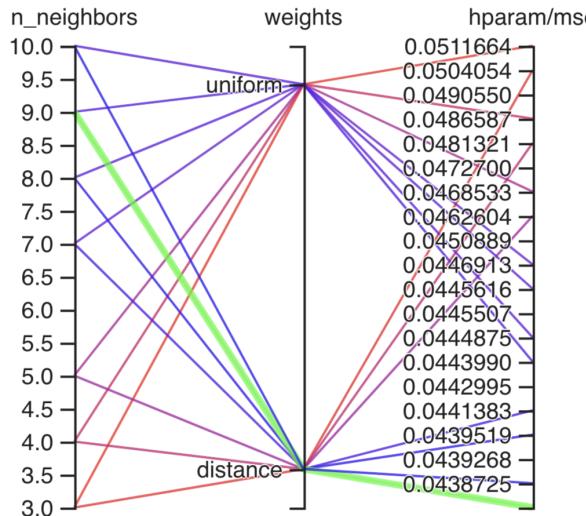


Figure 18: Parallel coordinates view KNN con PCA

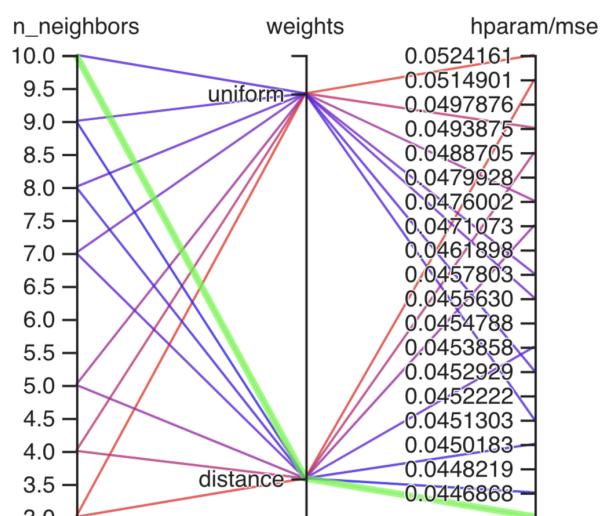


Figure 19: Parallel coordinates view KNN senza PCA

6.1.6 Support Vector Regressor

SVR, spiegato nel paragrafo 5.1.6, è stato implementato utilizzando scikit-learn.

Dal tuning degli iperparametri fatto, è risultato che i valori migliori per i parametri sono quelli visibili in tabella 9

Parametro	Migliore configurazione (PCA)	Migliore configurazione (senza PCA)
epsilon	0.01	0.001
kernel	linear	rbf

Table 9: configurazione migliore SVR

Di seguito, in tabella 10, si possono osservare i risultati ottenuti dal modello.

Metrica	PCA	No PCA
mse	0.00645	0.00521
R2	0.97086	0.97647

Table 10: Risultati SVR

In Figura 20 e 21, è possibile osservare i parallel coordinates view ottenuti. In verde è indicata la configurazione migliore.

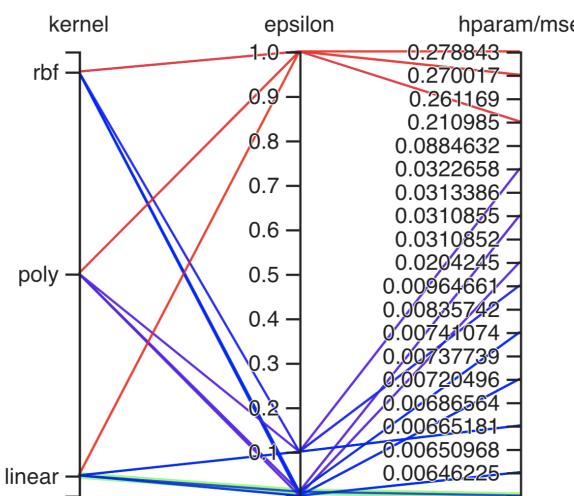


Figure 20: Parallel coordinates view SVR con PCA

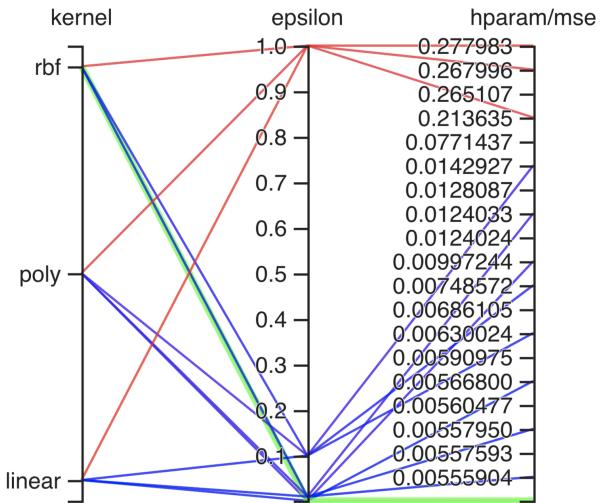


Figure 21: Parallel coordinates view SVR senza PCA

6.2 Neural network

Le rete neurale implementata è la feed-forward.

6.2.1 Feed-forward network

L'implementazione della rete è stata realizzata utilizzando Pytorch, il codice realizzato è visibile nel paragrafo 5.2.

Durante l'esecuzione della rete sono stati effettuate 162 combinazioni per il tuning degli iperparametri, da cui è emersa la migliore configurazione, visibile in tabella 11.

Parametro	Migliore configurazione PCA	Migliore configurazione senza PCA
hidden_size	512	256
dropout_prob	0.3	0.4
dept	3	5
batch_size	8	8
learning_rate	0.001	0.001

Table 11: configurazione migliore feed-forward network

Di seguito, in tabella 12, si possono osservare i risultati ottenuti dal modello.

Metrica	PCA	No PCA
mse	0.00519	0.00501
R2	0.97655	0.97735

Table 12: Risultati feed-forward network

Di seguito, in Figura 22 e 24, è possibile osservare i parallel coordinates view ottenuti, mentre in Figura 23 e 25 si può vedere gli scatter plox matrix view dei dati con e senza l'applicazione della PCA. In verde è indicata la configurazione migliore.

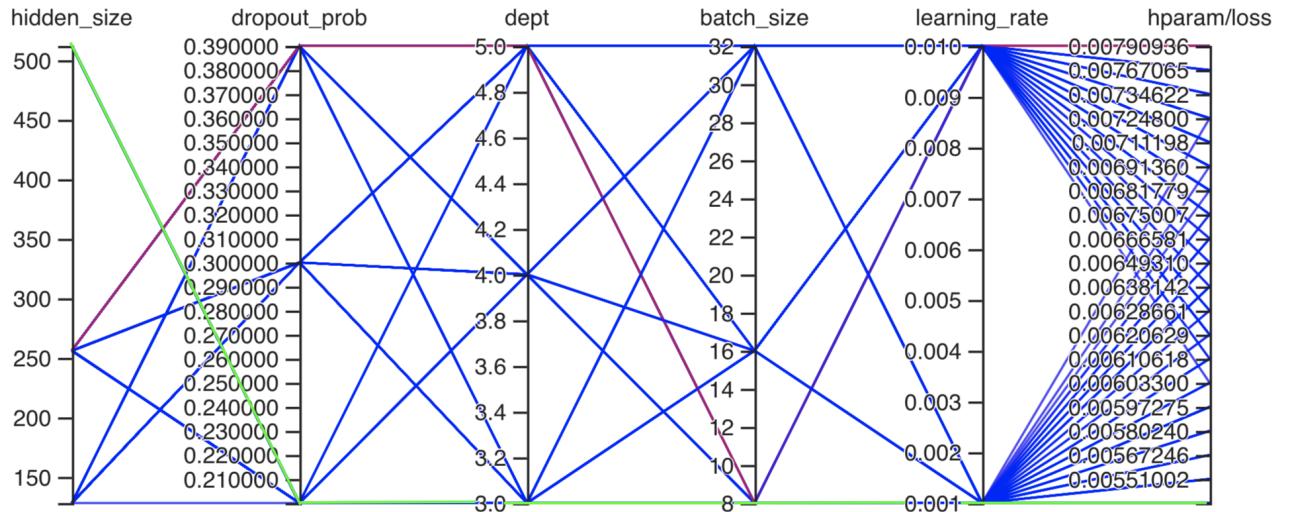


Figure 22: Parallel coordinates view feed-forward network con PCA

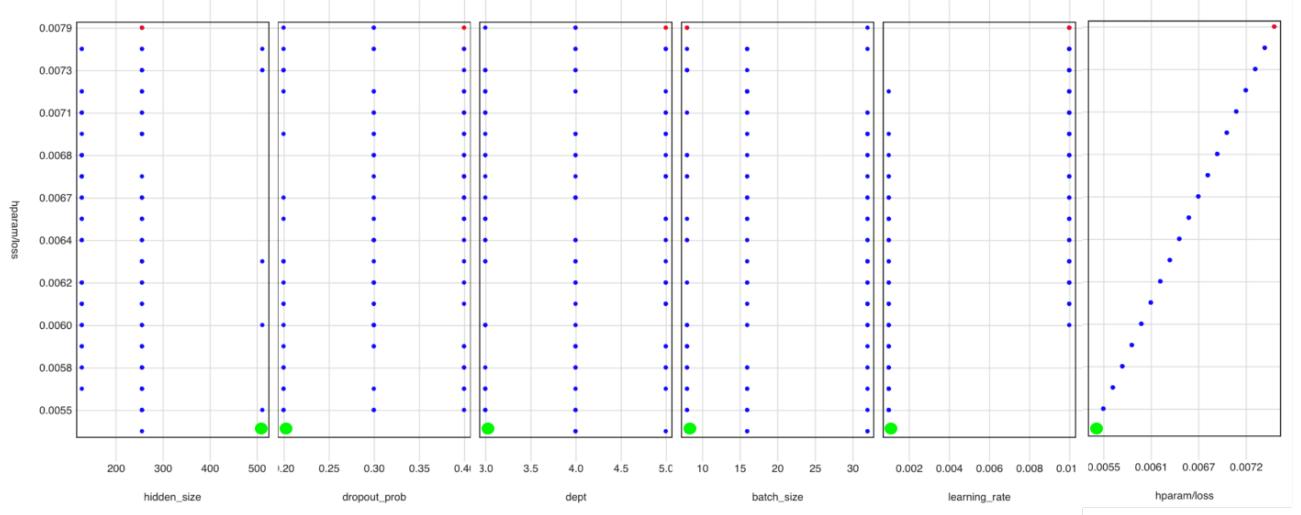


Figure 23: Scatterplot matrix view feed-forward network con PCA

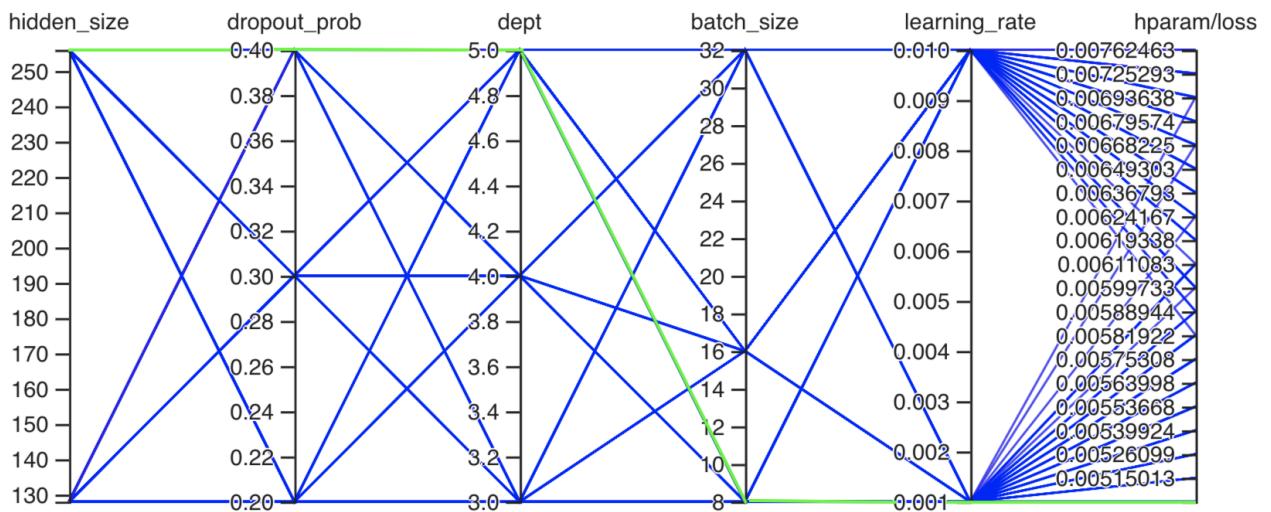


Figure 24: Parallel coordinates view feed-forward network senza PCA

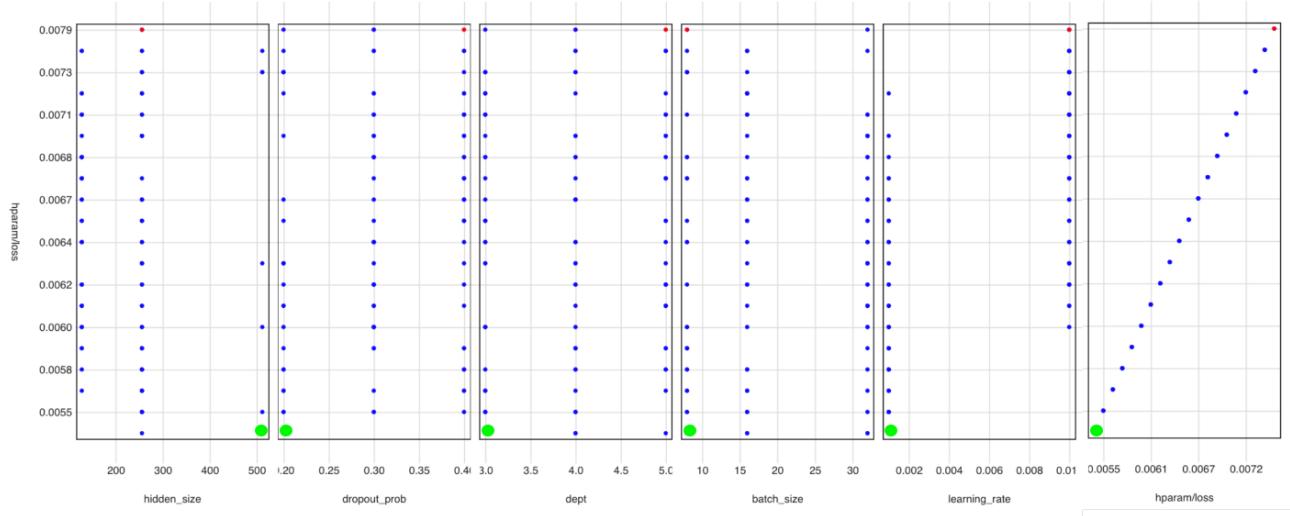


Figure 25: Scatterplot matrix view feed-forward network senza PCA

6.3 TabNet

L'implementazione della rete è stata realizzata utilizzando Pytorch. Il suo funzionamento è spiegato nel paragrafo 5.3.

Durante l'esecuzione della rete sono stati effettuate 54 combinazioni per il tuning degli iperparametri, da cui è emersa la migliore configurazione, visibile in tabella 13.

Parametro	Migliore configurazione con PCA	Migliore configurazione senza PCA
batch_size	256	256
n_d	8	8
n_a	16	16
n_steps	5	3
n_independent	2	2

Table 13: configurazione migliore TabNet

Di seguito, in tabella 14, si possono osservare i risultati ottenuti dal modello.

Metrica	PCA	No PCA
mse	0.00664	0.00535
R2	0.97001	0.97583

Table 14: Risultati TabNet

Di seguito, in Figura 26 e 28, è possibile osservare i parallel coordinates view ottenuti, mentre in Figura 27 e 29 si può vedere lo scatter plox matrix view. In verde è indicata la configurazione migliore.

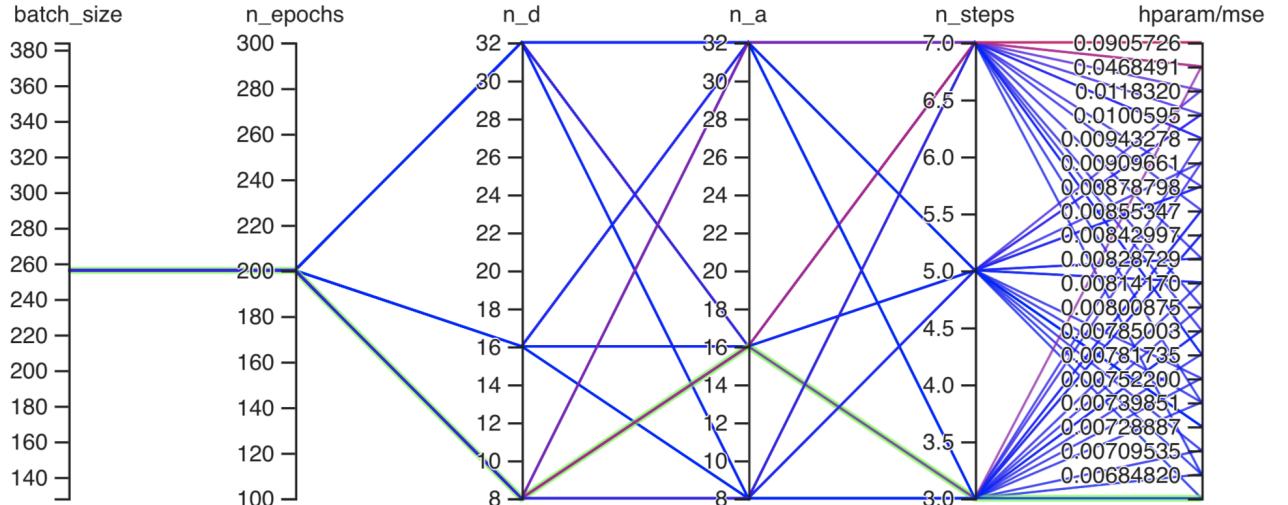


Figure 26: Parallel coordinates view TabNet con PCA

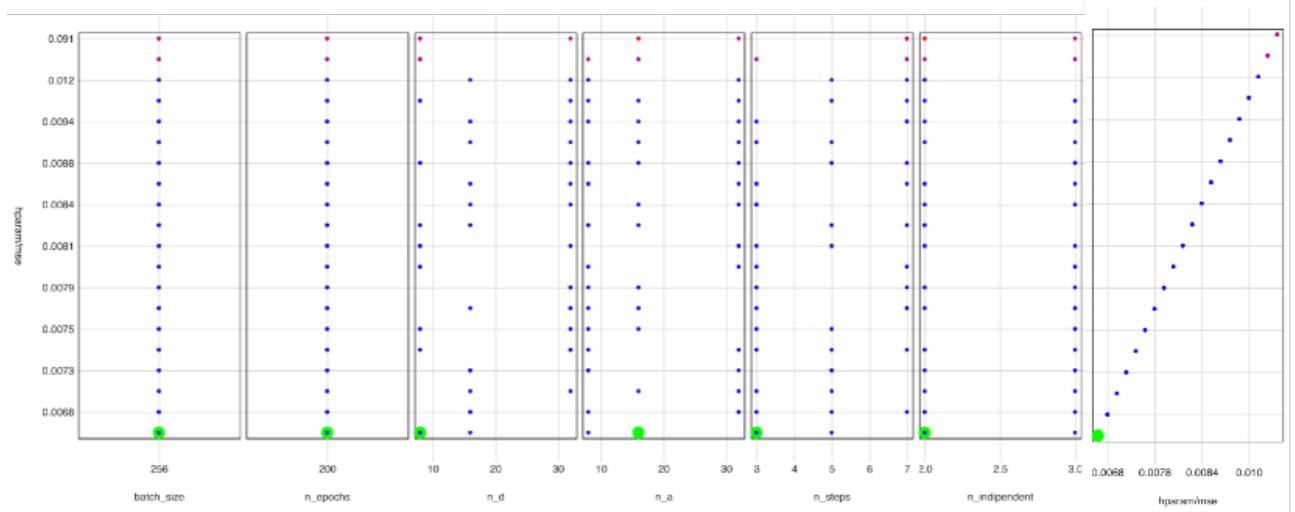


Figure 27: Scatterplot matrix view TabNet con PCA

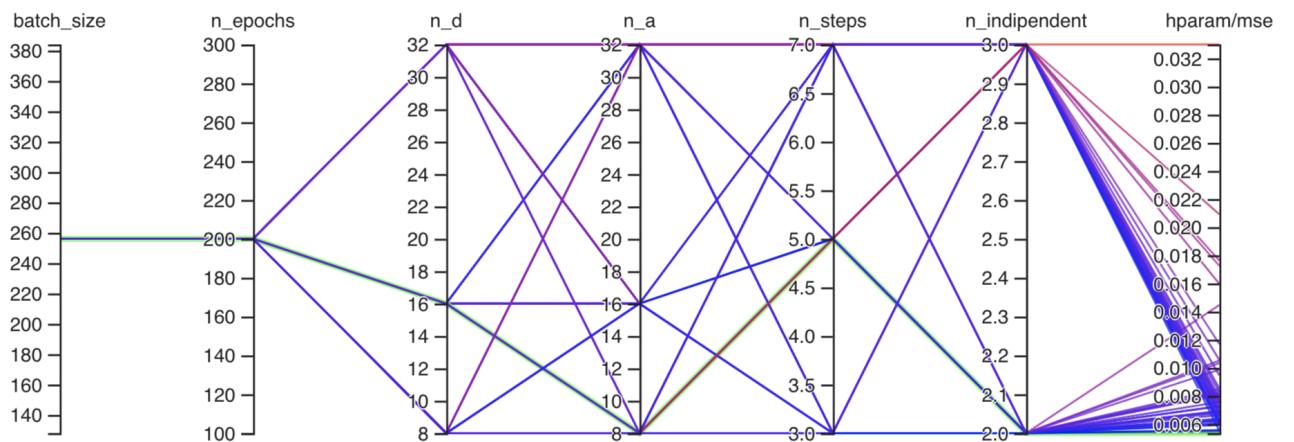


Figure 28: Parallel coordinates view TabNet senza PCA

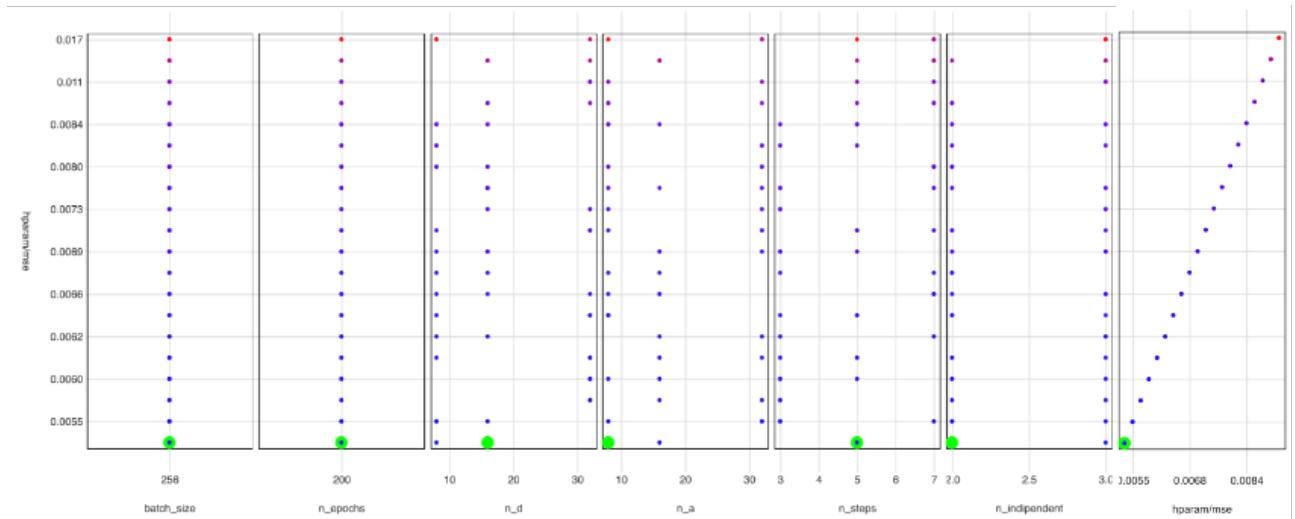


Figure 29: Scatterplot matrix view TabNet senza PCA

7 Conclusioni

Il presente progetto ha lo scopo di sviluppare un sistema in grado di prevedere il voto medio dei film attraverso l'implementazione di diverse tecniche di Machine Learning supervisionato. In particolare, sono state utilizzate tecniche supervisionate tradizionali, reti neurali e il modello TabNet, un algoritmo deep per dati tabulari.

I risultati ottenuti mostrano che tutti i modelli hanno prestazioni migliori con l'utilizzo dei dati a cui non è stata applicata la PCA. Inoltre, quasi tutti i modelli tradizionali, ad eccezione di KNN e Random Forest, hanno ottenuto ottime performance. Da ciò possiamo dedurre che questo problema di regressione può essere risolto in modo efficiente utilizzando i modelli lineari.

La rete Feed-Forward e il TabNet hanno ottenuto risultati leggermente migliori rispetto i modelli non deep ma, questi richiedono un tempo maggiore di training rispetto ai primi.

References

- [1] *Dataset MovieLens*. URL: <https://grouplens.org/datasets/movielens/>.
- [2] *TabNet*. URL: <https://arxiv.org/pdf/1908.07442.pdf>.