

REPORT DATA ANALYTICS

A.A. 2022/2023

Filippo Bartolucci
filippo.bartolucci2@studio.unibo.it,

Alfonso Esposito
alfonso.esposito5@studio.unibo.it

Indice

1	Introduzione	1
2	Data Acquisition	2
3	Data Visualization	3
3.1	Distribuzione voti film	3
3.2	Correlazione genere voto	4
3.3	Correlazione tag e voto	5
3.4	Tag più importanti	6
3.5	Similarità film per voto	7
3.6	Rappresentazione 2D dei dati	8
4	Data Preprocessing	10
4.1	Dimensionality Reduction	10
5	Classic Machine Learning	12
5.1	Support Vector Regressor	12
5.2	Random Forest	13
5.3	Linear Regressor, Lasso Regressor, Ridge regressor, Elastic Net	13
6	Deep Neural Network	15
7	TabNet: modello deep per tabular data	17
8	Risultati	20
8.1	Random Forest Regressor	21
8.2	Support Vector Regression	24
8.3	Linear Regression	25
8.4	Lasso Regression	26
8.5	Ridge Regression	28
8.6	Elastic Net Regression	30
8.7	Deep Neural Network	32
8.8	TabNet	34
9	Conclusioni	36

Elenco delle figure

1	Grafico di densità dei voti	3
2	Confronto con distribuzione gaussiana con stessi μ e σ	4
3	Voto medio per genere	5
4	Correlazione tra genere dei film e voto medio	5
5	Correlazione tra il tag e il voto	6
6	Feature più rilevanti	6
7	Correlazione tra i tag più importanti	7
8	Similarità tra i film con il voto migliore	7
9	Similarità tra i film con il voto peggiore	8
10	Rappresentazione 2D dei film	9
11	Varianza per numero di componenti	11
12	Architettura del modello TabNet	18
13	Combinazioni iperparametri con PCA: Random Forest Regressor	21
14	Risultato del finetuning con PCA: Random Forest Regressor	22
15	Combinazioni iperparametri senza PCA: Random Forest Regressor	23
16	Risultato del finetuning senza PCA: Random Forest Regressor	23
17	Combinazioni iperparametri con PCA: SVR	24
18	Combinazioni iperparametri senza PCA: SVR	25
19	Training del Lasso Regressor con PCA	26
20	Training del Lasso Regressor senza PCA	27
21	Training del Ridge Regressor con PCA	28
22	Training del Ridge Regressor senza PCA	29
23	Training dell'Elastic Net Regressor con PCA	30
24	Training dell'Elastic Net senza PCA	31
25	Tutte le combinazioni effettuate durante il finetuning per il training della rete neurale con PCA	32
26	Tutte le combinazioni effettuate durante il finetuning per il training della rete neurale senza PCA	33
27	Tutte le combinazioni effettuate durante il finetuning per il training della rete TabNet	34
28	Training e Validation Loss	35

1 Introduzione

Il seguente progetto è stato sviluppato nell'ambito del corso di "Data Analytics" della Laurea Magistrale in Informatica presso l'Università di Bologna.

L'obiettivo del progetto consiste nella realizzazione di uno studio di data analytics, che prevede l'implementazione di tutte le fasi della pipeline analitica studiate durante il corso.

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

Lo scopo principale di questo studio è quello di predire il voto medio di un film, partendo dalle sue caratteristiche. A tal fine, si è utilizzato un dataset proveniente da MovieLens, un sistema di raccomandazione di contenuti video, che contiene rating e tag per oltre 60.000 film.

Il dataset è stato raccolto nel corso degli anni dal 1995 al 2019 da più di 150.000 utenti e ogni file è dotato di un genoma che identifica una specifica caratteristica del film (ad esempio, "prima guerra mondiale") e la sua rilevanza (espressa in una scala da 0 a 1, dove 0 indica che la caratteristica non è rilevante per il film e 1 indica che la caratteristica è molto rilevante).

Funzionalità sviluppate:

- tecniche di ML supervised tradizionali non deep
- tecniche di ML supervised basate su reti neurali
- tecnica di ML supervised con modelli deep per TabularData

2 Data Acquisition

Ogni film presente nel dataset è identificato da un identificatore movieId unico. Le caratteristiche dei film sono rappresentate da dei tag che descrivono il contenuto del film. Questi tag hanno un valore di rilevanza che varia tra 0 e 1.

Il dataset MovieLens 25M è composto dai seguenti file:

- **genome-scores.csv**: contiene il valore di rilevanza associato ad ogni tag per ogni film.
- **genome-tags.csv**: contiene l'associazione tra tagId e nome del tag.
- **movies.csv**: contiene movieId, titolo del film e generi.
- **ratings.csv**: contiene tutti i voti associati ad ogni film.

I file del dataset vengono combinati con l'obiettivo di raggruppare per ogni film la rilevanza di ogni suo tag.

```
1 # Operazioni effettuate sul dataset
2 movies_df = pd.read_csv('./ml-25m/movies.csv')
3 scores_df = pd.read_csv('./ml-25m/genome-scores.csv')
4 tags_df = pd.read_csv('./ml-25m/genome-tags.csv')
5 ratings_df = pd.read_csv('./ml-25m/ratings.csv')
6
7 df = movies_df.merge(scores_df, on='movieId')
8 df = df.merge(tags_df, on='tagId')
9 df = df.pivot_table(index=['movieId', 'title'], columns='tag', values='
    relevance', fill_value=0).reset_index().rename_axis(None, axis=1)
10
11 # Voto medio per ogni film
12 ratings_df = ratings_df.groupby(['movieId'])['rating'].mean()
```

Il dataset viene unito in modo da associare a ciascun film la rilevanza di ogni tag correlato ad esso. L'obiettivo è di raggruppare le informazioni sui tag relativi ai film, in modo da ottenere una valutazione della rilevanza di ciascun tag per ogni film del dataset.

Per ogni film viene calcolata la valutazione media utilizzando le recensioni degli utenti. Obiettivo del progetto è predire, per un dato film, questo valore partendo dai tag associati. Essendo le nostre variabili target dei valori continui, il problema è stato risolto utilizzando dei modelli di regressione.

3 Data Visualization

In questa sezione del lavoro, sono stati prodotti diversi grafici e rappresentazioni visive per fornire una panoramica del dataset in questione. Poiché il dataset presenta un alto numero di feature, si è scelto di concentrarsi solo su alcune informazioni chiave e di rappresentarle visivamente per rendere più agevole la comprensione e l'analisi dei dati.

3.1 Distribuzione voti film

Il numero totale dei film è 13817, per ognuno di questi il voto medio è rappresentato da un numero decimale compreso tra 0 e 5.

La figura rappresenta un grafico di densità, che è stato utilizzato per mostrare la distribuzione dei voti. Il grafico di densità evidenzia le aree di maggiore densità di dati e quelle di minor densità, permettendo di individuare eventuali picchi o zone di concentrazione dei dati.

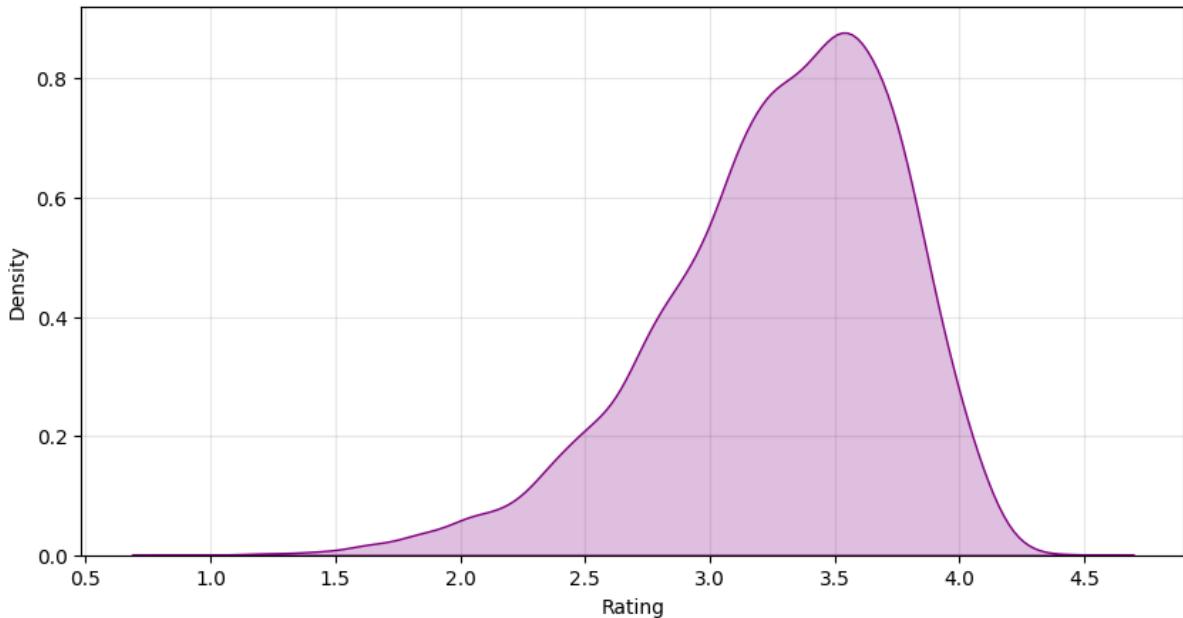


Figura 1: Grafico di densità dei voti

Analizzando il grafico di densità, si può osservare che la distribuzione dei voti presenta un'area di maggiore densità in corrispondenza del valore di circa 3.5.

Il valore atteso μ è pari a 3.27, mentre la deviazione standard σ è 0.48.

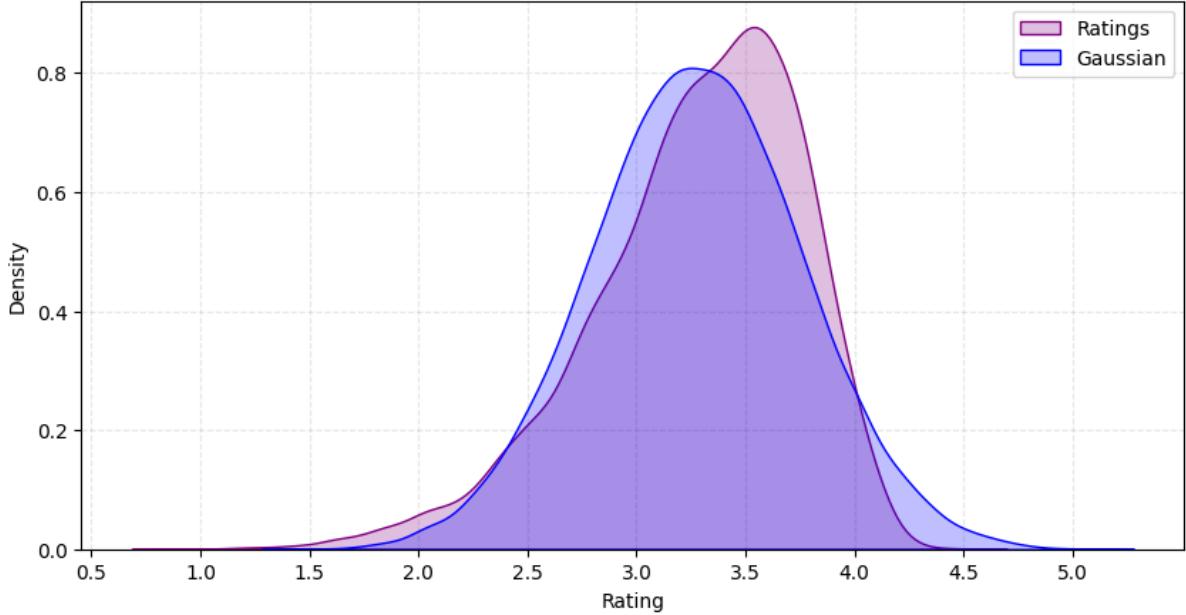


Figura 2: Confronto con distribuzione gaussiana con stessi μ e σ

Per confrontare due distribuzioni si può usare la divergenza di Kullback-Leibler (KL) che è una misura della differenza tra due distribuzioni di probabilità. Essa è basata sull'idea che la distribuzione di probabilità Q può essere usata per approssimare la distribuzione di probabilità P, e che la KL divergence rappresenta la quantità di informazione persa nell'approssimare P con Q.

La formula per la divergenza di Kullback-Leibler tra due distribuzioni di probabilità discrete P e Q è la seguente:

$$D_{KL}(P||Q) = \sum_{x \in X} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

dove X è l'insieme di tutti i possibili valori che la variabile casuale può assumere.

Il valore della divergenza di Kullback-Leibler (KL) tra la distribuzione gaussiana e la distribuzione dei voti dei film è di 0.02. Questo valore indica la differenza tra le due distribuzioni in termini di entropia relativa, dove una KL divergence di 0 indica che le due distribuzioni sono identiche.

In questo caso, un valore di 0.02 indica che le due distribuzioni sono relativamente simili, ma non sono identiche. La distribuzione dei dati potrebbe avere alcune differenze rispetto alla distribuzione gaussiana, ma tali differenze sono relativamente piccole.

3.2 Correlazione genere voto

È stata calcolata la correlazione tra genere e voto dei film e i risultati sono visibili nelle figure sottostanti. La prima mostra il voto medio per ogni genere di film presente nel dataset e la seconda mostra la correlazione tra il genere e il voto dei film.

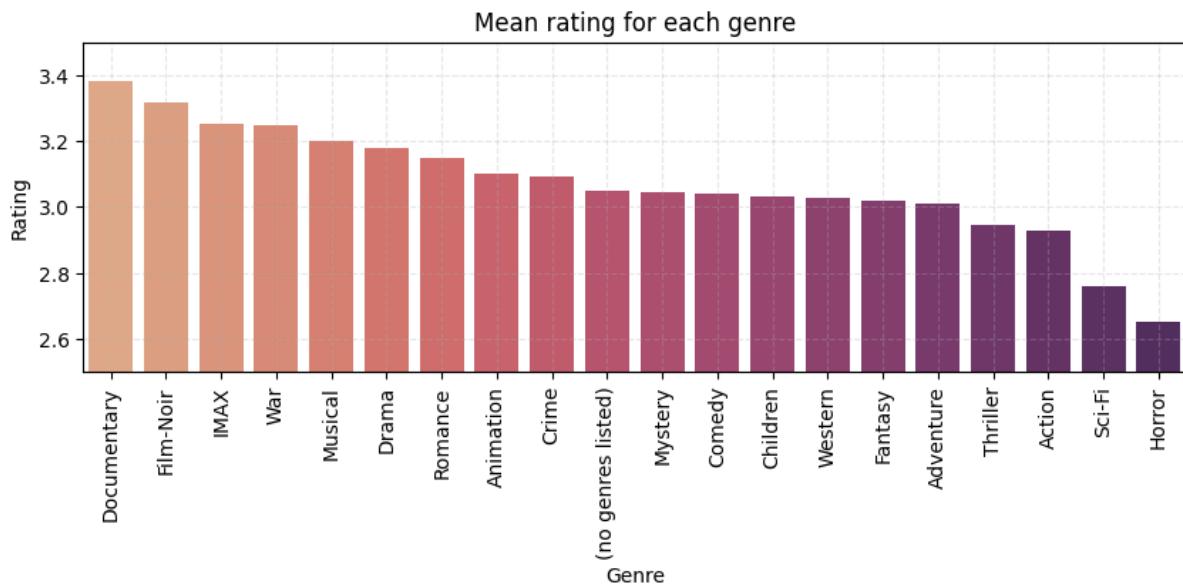


Figura 3: Voto medio per genere

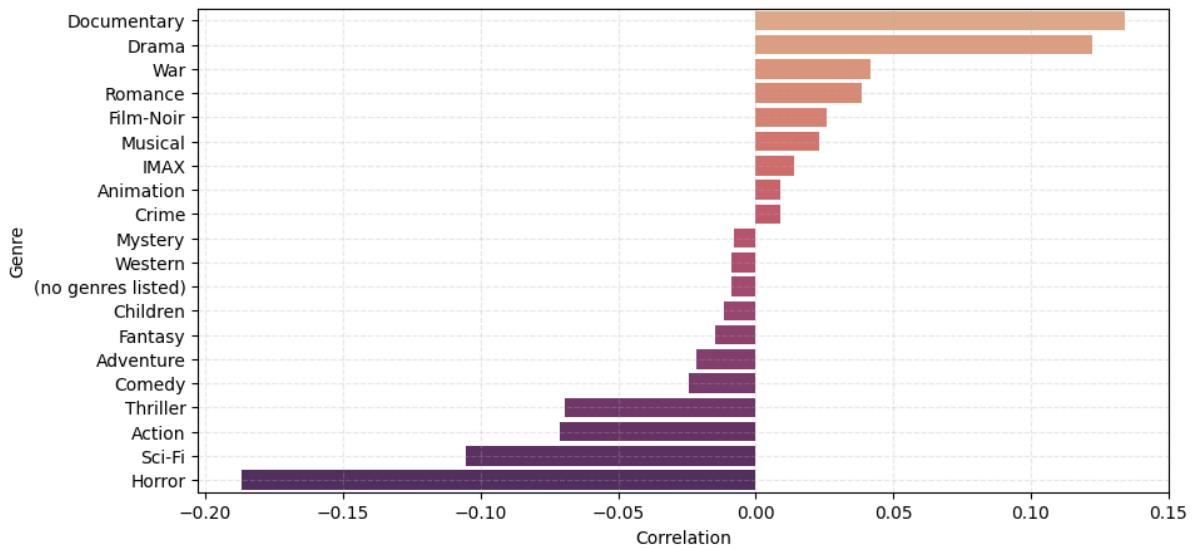


Figura 4: Correlazione tra genere dei film e voto medio

Si può notare che i generi che mostrano una forte correlazione positiva con i voti sono il documentario, drama e war, mentre i generi con una forte correlazione negativa sono l'horror, il scifi ed action. Questo suggerisce che il genere del film può influenzare il voto medio atteso.

3.3 Correlazione tag e voto

È stata calcolata la correlazione tra tag e voto dei film e i risultati sono visibili nelle figure sottostanti. A sinistra i primi 5 tag più correlati e a destra i primi 5 tag meno correlati.

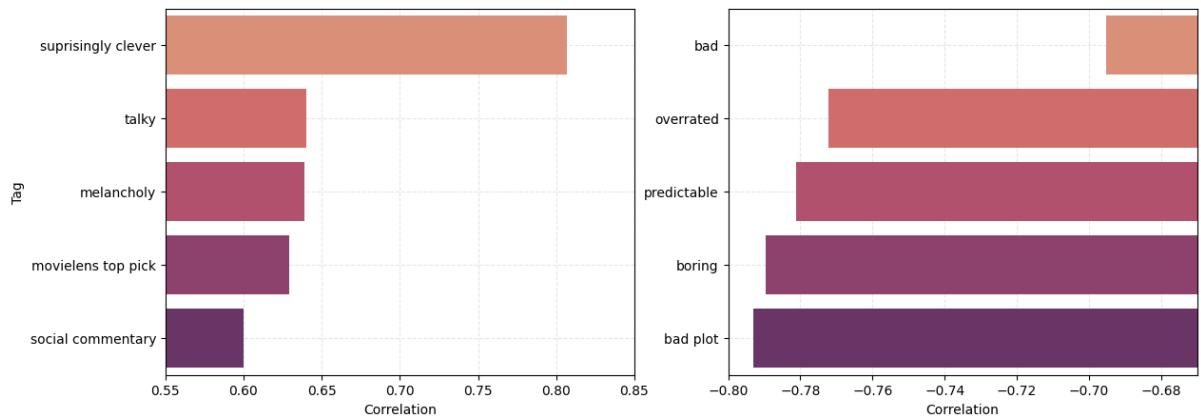


Figura 5: Correlazione tra il tag e il voto

3.4 Tag più importanti

È stata effettuata un'analisi per valutare l'importanza dei vari tag nel processo di previsione del voto medio, utilizzando una Random Forest come metodo di apprendimento automatico. Questo perché gli alberi decisionali sono in grado di fornire una spiegazione sull'importanza delle feature utilizzate dal modello, a differenza di altri modelli in cui questa operazione risulta più complessa.

Tuttavia, va tenuto presente che il valore di importanza ottenuto può non essere rappresentativo per tutti i modelli, ma offre una panoramica generale sui tag che influenzano maggiormente il risultato.

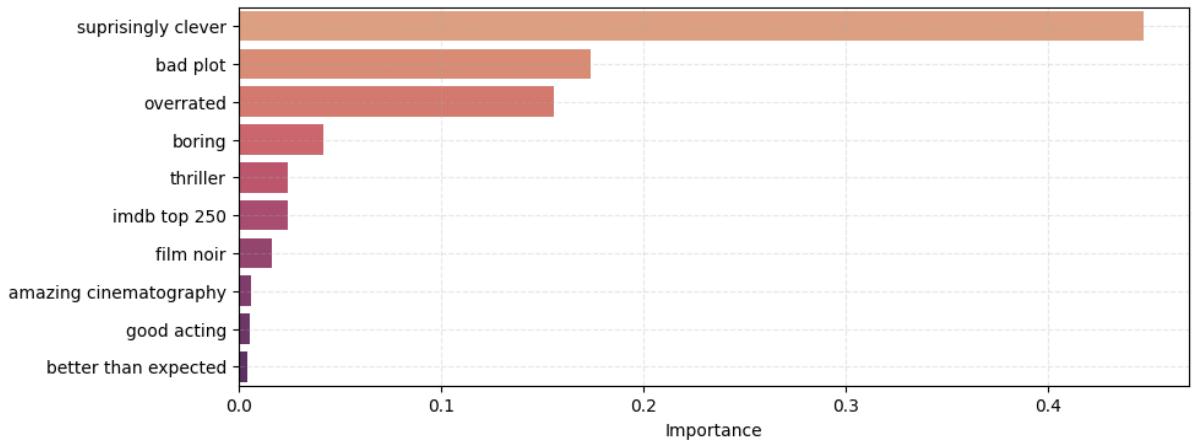


Figura 6: Feature più rilevanti

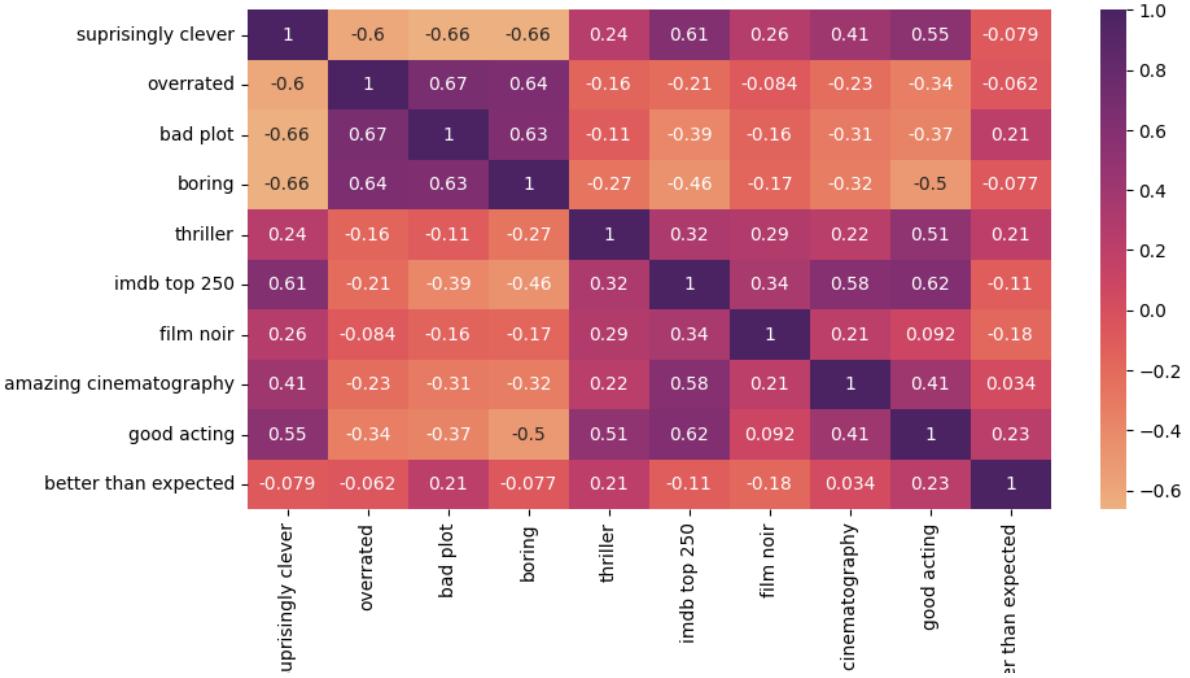


Figura 7: Correlazione tra i tag più importanti

3.5 Similarità film per voto

Sono stati effettuati dei calcoli per valutare la similarità tra i 10 film più votati, visualizzabili in Figura 8, e i 10 film meno votati, visualizzabili in Figura 9. Dalle due figure si può notare che esiste una certa somiglianza tra i vari film all'interno di ciascun grafico. Questo suggerisce che i film che hanno ottenuto voti simili hanno anche un embedding simile.

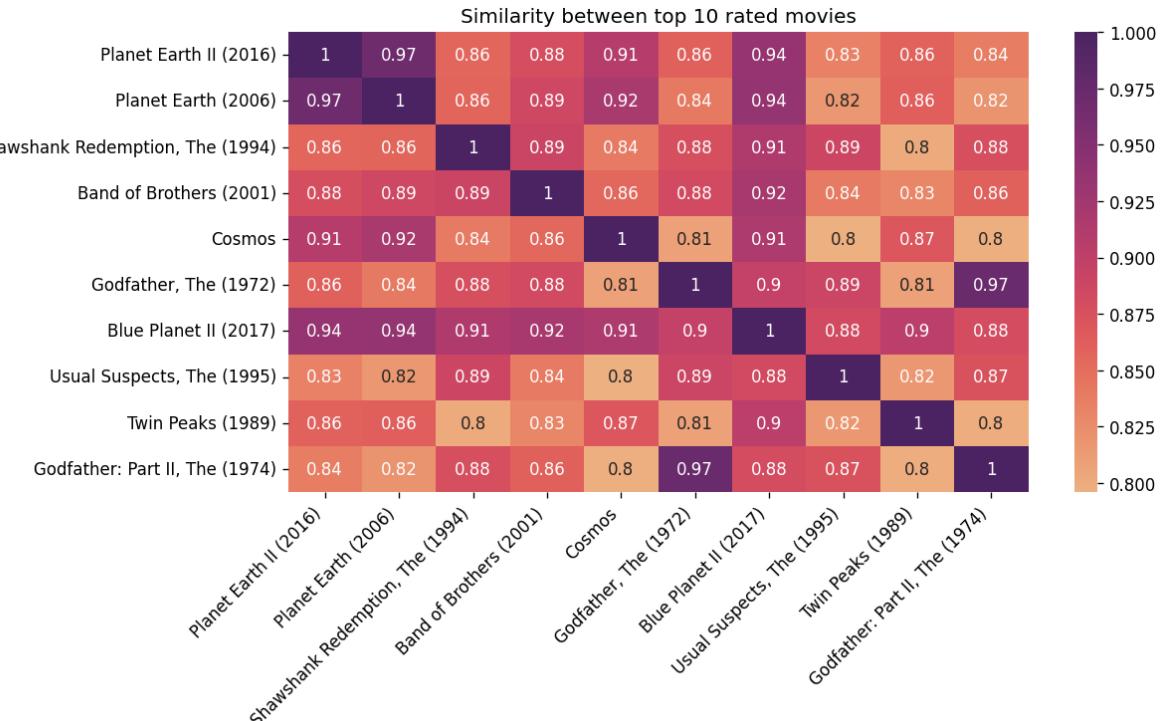


Figura 8: Similarità tra i film con il voto migliore

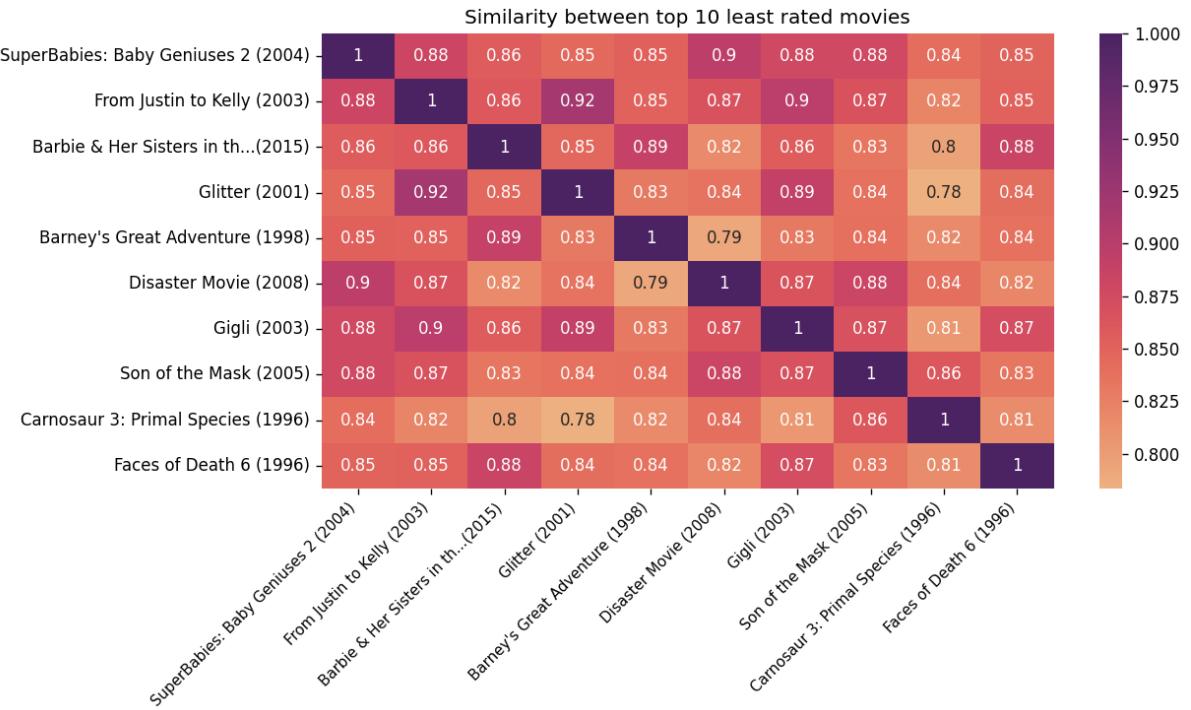


Figura 9: Similarità tra i film con il voto peggiore

3.6 Rappresentazione 2D dei dati

Per rappresentare i film sulla base del rating in uno spazio bidimensionale, è stato utilizzato un algoritmo di dimensionality reduction chiamato t-Distributed Stochastic Neighbor Embedding (t-SNE). Questo algoritmo cerca di trovare una rappresentazione a bassa dimensionalità dei dati in cui i punti che sono simili nel dataset originale sono anche vicini nello spazio a bassa dimensionalità. Questo rende possibile la visualizzazione dei dati in un grafico bidimensionale o tridimensionale, dove le relazioni tra i dati originali sono conservate.

In particolare, l'algoritmo funziona attraverso due passaggi principali: Nel primo passaggio, l'algoritmo calcola 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.

Nel secondo passaggio, 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 viene fatto attraverso un processo iterativo di minimizzazione della divergenza di Kullback-Leibler tra la distribuzione di probabilità del dataset originale e la distribuzione di probabilità nel nuovo spazio a bassa dimensionalità.

Il risultato finale è una mappa bidimensionale dei dati che consente di visualizzare le relazioni tra i dati in modo efficace e intuitivo. In questo caso specifico, l'algoritmo t-SNE è stato applicato ai dati dei voti dei film per rappresentarli in uno spazio bidimensionale, consentendo di visualizzare la distribuzione dei voti.

In Figura 10 è possibile vedere i film rappresentati in uno spazio 2D.

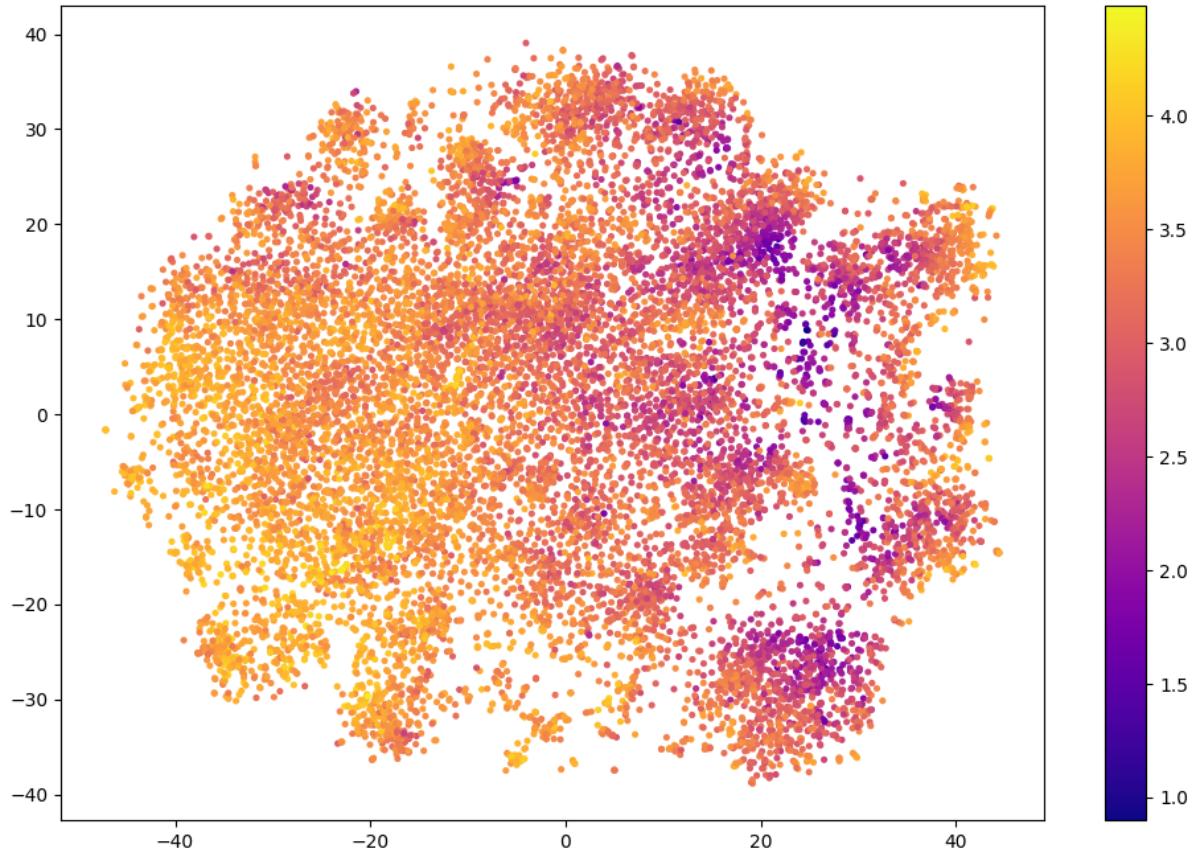


Figura 10: Rappresentazione 2D dei film

Dall'analisi della figura si può notare come la distribuzione dei film sia graduale e come i voti siano correlati con la posizione dei film nell'embedding. I film con votazione più bassa sono rappresentati dal colore blu e si trovano nella zona destra della figura, mentre i film con votazione più alta sono rappresentati dal colore giallo e si trovano nella zona sinistra della figura.

Inoltre, si può notare che la maggior parte dei film presenti nel dataset hanno un voto compreso tra 3 e 3,5, e nell'area della figura in loro corrispondenza si trovano zone più fitte, dato il maggior numero di esempi di film. Al contrario, i film con voto inferiore a 2,5 sono meno frequenti e la loro zona nella figura risulta meno fitta e con buchi, a causa della scarsità di esempi di questi film nel dataset.

Queste considerazioni sono in linea con quanto si può vedere dalla distribuzione dei voti.

4 Data Preprocessing

Prima dell'effettivo utilizzo delle tecniche di ML supervised, è stata effettuata una fase di *data preprocessing*.

Il dataset utilizzato nel progetto è stato suddiviso in due parti: train e test, utilizzando uno schema di divisione 80-20. Inoltre, è stato fissato un seed per garantire che la suddivisione del dataset sia riproducibile in modo coerente in futuro. Lo stesso test set viene usato per valutare tutti i modelli, al fine di garantire una comparabilità coerente dei risultati ottenuti dai vari modelli.

Infine il train set è stato ulteriormente suddiviso e il 10% è stato utilizzato come set di validation per la ricerca degli iperparametri ottimali.

4.1 Dimensionality Reduction

Ogni film presente nel dataset è descritto da un vettore di feature composto da 1128 elementi. Tuttavia, il grande numero di feature potrebbe rappresentare un problema per l'efficienza e la scalabilità dei modelli. Per questo motivo, è stata utilizzata una tecnica di riduzione della dimensionalità per ridurre il numero di feature dell'embedding. L'obiettivo è ottenere un embedding di dimensioni ridotte che conserva le informazioni più rilevanti del dataset, per fare ciò si è scelto di usare la PCA (Principal Component Analysis)

La PCA è una tecnica di analisi multivariata che si utilizza per ridurre la dimensionalità dei dati mantenendo il maggior grado di varianza possibile. L'idea alla base è quella di trasformare un set di variabili correlate in un nuovo set di variabili linearmente indipendenti, chiamati componenti principali.

Il calcolo delle componenti principali si basa sulla matrice di covarianza dei dati, che misura la relazione tra le variabili e la loro varianza. La matrice di covarianza viene calcolata sui dati centrati, ovvero dati a cui viene sottratta la media di ogni variabile. Questo è necessario per eliminare eventuali effetti dovuti alle diverse scale delle variabili. Gli autovettori della matrice di covarianza rappresentano le direzioni principali dei dati, le componenti principali, mentre gli autovalori rappresentano la quantità di varianza spiegata da ciascun componente principale. Per proiettare i dati sui componenti principali, si moltiplica la matrice dei dati originale per la matrice degli autovettori. Il risultato di questa operazione è una nuova matrice dei dati, dove le colonne rappresentano le nuove variabili (componenti principali). Questa nuova matrice dei dati ha lo stesso numero di righe della matrice dei dati originale (cioè il numero di osservazioni), ma un numero di colonne ridotto (cioè il numero di componenti principali).

La proiezione dei dati sulle nuove variabili permette di ridurre la dimensionalità del dataset, eliminando le variabili meno informative e mantenendo solo quelle più importanti.

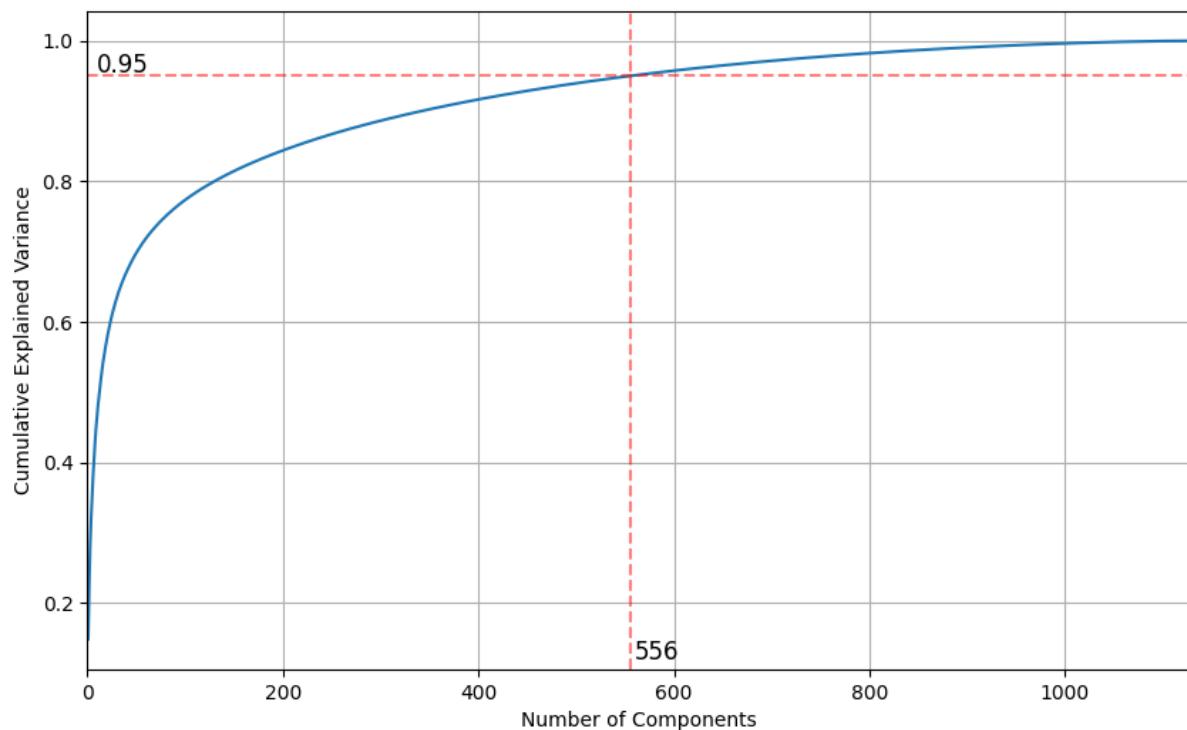


Figura 11: Varianza per numero di componenti

Nel nostro abbiamo utilizzato la PCA selezionando il numero minimo di componenti principali che spiegano il 95% della varianza totale del dataset. Il dataset verrà testato sia con che senza la tecnica di PCA, al fine di capire come le performance del modello possano variare al variare della dimensionalità del dataset.

5 Classic Machine Learning

Le tecniche di ML supervised tradizionali non deep utilizzate nel progetto sono alcune delle tecniche di regressione più comunemente usate. Queste sono state utilizzate per addestrare modelli che possono prevedere il voto medio di un film a partire dalle sue caratteristiche.

5.1 Support Vector Regressor

Il **Support Vector Regressor** (SVR) è una tecnica di regressione che utilizza un algoritmo di machine learning basato su Support Vector Machine per addestrare un modello di regressione.

SVR cerca di trovare una funzione (o iperpiano) che si adatta ai dati in modo tale che l'errore di predizione sia minimo, garantendo anche una certa tolleranza a degli errori "accettabili". Per fare ciò, la SVR utilizza i cosiddetti "vettori di supporto", ovvero i punti del dataset che si trovano più vicini all'iperpiano di separazione. L'obiettivo è quello di cercare un iperpiano che separi i vettori di supporto in modo che la distanza tra l'iperpiano e i vettori di supporto sia massima. In termini matematici, la SVR cerca di minimizzare una funzione di costo che misura la differenza tra i valori predetti e i valori effettivi del target. La scelta della funzione di kernel è un elemento importante nell'utilizzo della SVR, in quanto questo è una funzione matematica che misura la similarità tra due punti del dataset.

I kernel più comuni utilizzati con la SVR sono il kernel lineare, il kernel polinomiale e il kernel RBF. Per ottenere il miglior modello SVR, è necessario fare tuning degli iperparametri, che in un modello SVR sono il parametro di regolarizzazione C, l'iperparametro di tolleranza dell'errore ϵ , e la scelta del kernel

Il parametro di regolarizzazione C controlla il trade-off tra la complessità del modello e la sua capacità di generalizzazione. Un valore elevato di C rende il modello più complesso e aderente ai dati di addestramento, ma potrebbe portare a problemi di overfitting. Al contrario, un valore basso di C rende il modello più semplice e meno aderente ai dati di addestramento, ma potrebbe portare a problemi di underfitting.

L'iperparametro di tolleranza dell'errore epsilon controlla la distanza massima tra i valori di output previsti e quelli reali che viene considerata accettabile. Un valore elevato di epsilon consente al modello di avere un margine più ampio, ma potrebbe portare a una riduzione della precisione delle predizioni.

```
1 # iperparametri testati
2 c = [0.001, 0.01, 0.1, 1]
3 epsilon = [0.001, 0.01, 0.1, 1]
4 kernel = ['linear', 'poly', 'rbf']
5
6 for c_, epsilon_, kernel_ in itertools.product(c, epsilon, kernel):
7     svr = SVR(C=c_, epsilon=epsilon_, kernel=kernel_)
8     svr.fit(X_train, Y_train)
9     Y_pred = svr.predict(X_val)
10    mse = np.mean((Y_pred - Y_val)**2)
11
12    if mse < svr_best_mse:
13        svr_best_mse = mse
14        svr_best_c = c_
15        svr_best_epsilon = epsilon_
16        svr_best_kernel = kernel_
17        svr_best = svr
```

5.2 Random Forest

Il Random Forest Regressor (RFR) è una tecnica di regressione basata su un insieme di alberi decisionali. In questo approccio, il modello di regressione è costruito a partire da un insieme di alberi decisionali, in cui ciascun albero è costruito su un sottoinsieme casuale del dataset. Il voto medio di un film può quindi essere previsto attraverso la combinazione delle previsioni di tutti gli alberi decisionali nel modello. Anche in questo caso, come nell'applicazione dell'SVR, abbiamo cercato la configurazione che ci permetta di ottenere il miglior risultato.

Iterando sul numero di *estimators*, ossia sul numero di alberi nell' RFR, partendo da un minimo di 50 fino ad un massimo di 100, per ogni criterio utilizzato creiamo un nuovo modello e lo addestriamo sui dati di training. Successivamente viene eseguita una previsione su dati di test e calcoliamo l'errore quadratico medio (MSE) tra le previsioni e i valori reali.

Il valore dell'iperparametro *n_estimators* che produce il valore dell'MSE più basso viene selezionato come il migliore e lo utilizzato per addestrare un nuovo modello di RFR.

```
1 n_tree = [ i for i in range(50, 100, 5)]
2 criterion = ["squared_error", "friedman_mse", "poisson"]
3
4 for n, c in itertools.product(n_tree, criterion)
5     rf = RandomForestRegressor(n_estimators=n, criterion=c)
6     rf.fit(X_train, Y_train)
7     Y_pred = rf.predict(X_val)
8     mse = np.mean((Y_pred - Y_val)**2)
9
10    if mse < rfr_best_mse:
11        rfr_best_mse = mse
12        rfr_best_n = n
13        rfr_best_c = c
14        rfr_best = rf
```

5.3 Linear Regressor, Lasso Regressor, Ridge regressor, Elastic Net

Il Linear Regressor, il Lasso Regressor, il Ridge Regressor e l'Elastic Net sono tutte tecniche di regressione lineare. Questi modelli sono costruiti utilizzando una funzione lineare che modella la relazione tra le caratteristiche del film e il voto medio. L'obiettivo della regressione è trovare i coefficienti della funzione che minimizzano l'errore di predizione. Queste tecniche differiscono nella regolarizzazione utilizzata per evitare l'overfitting del modello. L'iperparametro utilizzato per controllare la regolarizzazione è α .

Un valore maggiore di alfa indica una maggiore regolarizzazione e una riduzione della complessità del modello. Un valore minore di alfa indica una regolarizzazione più debole e una maggiore complessità del modello.

Inoltre, il valore di alfa influisce sulla proporzione di L1 e L2 nella regolarizzazione, con valori più alti di alfa che favoriscono L1 (nel caso della regressione Lasso) e valori più bassi di alfa che favoriscono L2 (nel caso della regressione Ridge). Nel caso dell'Elastic Net, l'iperparametro alfa controlla il peso relativo di L1 e L2 nella regolarizzazione e quindi controlla la proporzione di selezione delle variabili (sparsity) e la stabilizzazione della magnitudine dei coefficienti.

In tutti e quattro i modelli, seguiamo lo stesso approccio

- Creiamo un modello di regressione
- Alleniamo il modello di regressione sui dati di training, iterando su alfa per ottenere la migliore configurazione
- Effettuiamo la predizione sui dati di test
- Calcoliamo la loss e il coefficiente di determinazione

Di seguito una rappresentazione in Python dei passi precedentemente descritti.

```
1 alpha = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6]
2
3 for a in alpha:
4     model = RegressorModel(alpha=a)
5     model.fit(X_train, Y_train)
6     Y_pred = model.predict(X_val)
7     mse = np.mean((Y_pred - Y_val)**2)
8
9     if mse < best_mse:
10         best_mse = mse
11         best_a = a
12         best_model = model
```

6 Deep Neural Network

La rete neurale è stata realizzata utilizzando il framework PyTorch. Il 10% del train set viene utilizzato come validation set per valutare le prestazioni del modello su dati non visti durante l’addestramento e per scegliere i parametri del modello in modo da ottimizzare le prestazioni sul dataset di test.

La rete utilizzata è un modello FeedForward Network con un numero variabile di layer. Ogni layer è composto da una successione di Linear, ReLU e dropout.

Di seguito il codice Python del modello:

```
1 # Modello PyTorch
2 def get_model(input_size, hidden_size, dropout_prob=0, depth=1):
3     model = [
4         torch.nn.Linear(input_size, hidden_size),
5         torch.nn.ReLU(),
6         torch.nn.Dropout(dropout_prob)
7     ]
8
9     for i in range(depth):
10        model.append(torch.nn.Linear(hidden_size, hidden_size))
11        model.append(torch.nn.ReLU())
12        model.append(torch.nn.Dropout(dropout_prob))
13
14    model.append(torch.nn.Linear(hidden_size, 1))
15
16    return torch.nn.Sequential(*model)
```

Gli iperparametri non fissati del modello sono i seguenti:

- **hidden_size** in [256, 512, 1024]
- **depth** in [3,4,5]
- **batch_size** in [8,16,32]
- **learning_rate** in [0.1, 0.01]
- **step_size_lr_decay** in [10, 20]

Per trovare la combinazione ottimale degli iperparametri del modello, è stata utilizzata la tecnica del finetuning, ovvero l’iterazione attraverso le diverse combinazioni di iperparametri per addestrare il modello, seguita dalla valutazione delle performance ottenute al fine di trovare la combinazione di iperparametri che massimizza le prestazioni.

Di seguito viene riportato lo pseudocodice del training del modello:

```
1 # Pseudo codice training
2 def train():
3     best_mse = +inf
4     best_model = None
5
6     for hidden_size, depth, num_epochs, batch, lr, step_size, momentum in
7         hyperparameters:
8         model = get_model(N_features, hidden_size, dropout_prob, depth=depth)
9         optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=
10             momentum)
```

```

9     scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=
10    step_size, gamma=0.1)
11
12    model.fit(criterion, optimizer, scheduler, num_epochs, train_set,
13    val_set)
14    mse = test(model, test_set)
15
16    if mse < best_mse:
17        best_mse = mse
18        best_model = model
19
20    return best_model

```

Il numero di epoche utilizzato durante l’addestramento del modello è stato scelto in modo arbitrario e fissato a 200. Tuttavia, per evitare di addestrare il modello oltre il necessario, è stata utilizzata la tecnica dell’early stopping che permette di interrompere l’addestramento del modello prima della fine delle epoche fissate, qualora non si osservi alcun miglioramento nella validation loss.

Nel caso in cui la validation loss non migliori per un numero di epoche prestabilito, chiamato *patience*, l’addestramento viene interrotto e viene restituito il modello migliore raggiunto durante il training.

Così facendo si evita di addestrare il modello oltre il necessario e si garantisce di restituire il modello migliore possibile in termini di prestazioni sul validation set.

Nel nostro caso, il valore di patience massimo è stato fissato a 10.

7 TabNet: modello deep per tabular data

I modelli di deep learning per tabular data sono stati sviluppati per gestire dati tabulari complessi, ovvero dati che contengono molte colonne e molte righe. Questi modelli sono in grado di effettuare la selezione delle caratteristiche in modo automatico e di eseguire la classificazione o la regressione sui dati di input. Essi sono in grado di gestire grandi quantità di dati e di trovare relazioni complesse tra le variabili, migliorando la capacità di previsione del modello.

Dato che questo tipo di modelli sono in grado di effettuare la selezione delle caratteristiche in modo automatico, eliminando la necessità di selezionare manualmente le variabili di input più rilevanti per la previsione, consentono di ridurre il rischio di overfitting e di migliorare la generalizzazione del modello.

TabNet è un algoritmo di deep learning sviluppato da Google AI per l'apprendimento di modelli predittivi sui dati tabulari. Il modello utilizza un'architettura di rete neurale che si basa sull'attenzione selettiva, ovvero è in grado di selezionare le caratteristiche (features) più rilevanti dei dati in input, rendendolo un algoritmo molto efficace nella risoluzione di problemi di classificazione e regressione su dati tabulari.

La particolarità di TabNet è che, invece di utilizzare una singola rete neurale, il modello è composto da diverse reti neurali a cascata, ciascuna delle quali elabora i dati in modo selettivo e ne estrae informazioni sempre più rilevanti e specifiche. Inoltre, TabNet utilizza un meccanismo di mascheramento, che impedisce alla rete neurale di utilizzare le stesse caratteristiche per più volte, garantendo così una migliore generalizzazione del modello.

Per l'implementazione di TabNet è stata usata la libreria **tabnet_pytorch** di *PyTorch*. L'architettura di TabNet è basata su un'insieme di reti neurali a cascata che utilizzano l'attenzione selettiva per selezionare le caratteristiche più rilevanti dei dati in input. In particolare, l'architettura di TabNet si compone di tre elementi principali:

- Encoder.
- Decoder.
- Mascheramento.

L'encoder è composto da diverse unità di decisione (Decision Point, DP) che elaborano i dati in input in modo selettivo e producono una serie di maschere binarie che indicano quali caratteristiche vengono selezionate e quali vengono scartate. Il **decoder**, invece, si compone di una rete neurale feedforward che utilizza le maschere prodotte dall'encoder per eseguire la previsione finale.

Infine, il **mascheramento** è un meccanismo che impedisce alla rete neurale di utilizzare le stesse caratteristiche per più volte durante l'elaborazione dei dati garantendo una migliore generalizzazione del modello e aiutando a prevenire l'overfitting.

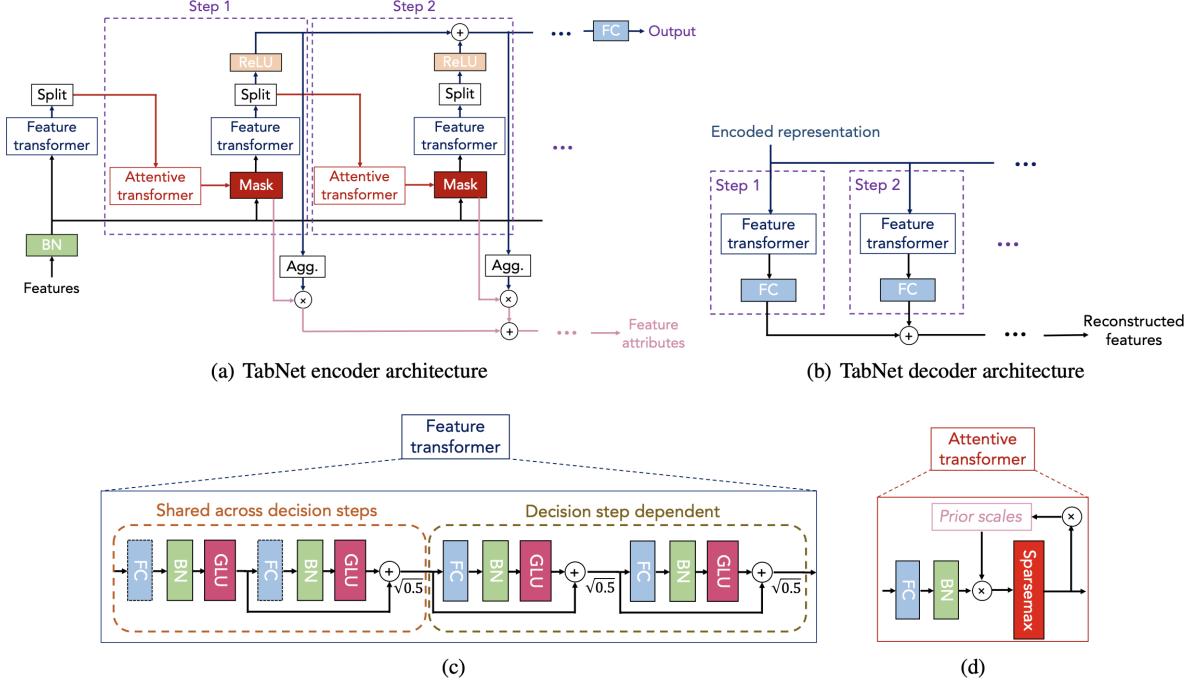


Figura 12: Architettura del modello TabNet

Di seguito il codice Python del modello:

```

1 def get_model(n_d_a, n_step, n_independent, n_shared, gamma, epsilon):
2     model = TabNetRegressor(
3         n_d=n_d_a,
4         n_a=n_d_a,
5         n_steps=n_step,
6         gamma=gamma,
7         optimizer_fn=torch.optim.Adam,
8         n_independent=n_independent,
9         n_shared=n_shared,
10        epsilon=epsilon,
11        seed=42
12    )
13    return model

```

Di seguito vengono presentati gli iperparametri del modello. Alcuni non sono fissati, altri invece, meno incisivi sul risultato finale, è stato deciso di lasciarli fissati. Di seguito una breve presentazione e l'intervallo dei valori possibili:

- n_d : la dimensionalità dello spazio di output del transformer network.
Il range dei valori scelto è [8, 16, 32, 64]
- n_a : la dimensionalità dello spazio di output dell'attention network
Il range dei valori scelto è [8, 16, 32, 64]
- n_{steps} : il numero di passi sequenziali nel meccanismo di attenzione
Il range dei valori scelto è [2, 3, 4, 5, 6, 7, 8, 9, 10]
- γ : Lo scaling factor per le feature del transformer network
Il parametro è stato fissato al valore [1.3]

- *optimizer*: L'optimizer della rete
Il parametro è stato fissato al valore [*Adam*]
- $n_{independent}$: il numero di feature indipendenti da usare del transformer network
Il range dei valori scelto è [2, 3]
- n_{shared} : il numero di feature condivise del transformer network
Il range dei valori scelto è [2, 3]
- ϵ : un valore piccolo da aggiungere al denominatore del calcolo dell'importanza delle caratteristiche per evitare la divisione per zero
Il parametro è stato fissato al valore [1e-08]
- *seed*: il seed casuale da utilizzare per la riproducibilità
Il parametro è stato fissato al valore [42]

Analogamente al metodo utilizzato per le tecniche di machine learning classiche e per le tecniche di deep learning, per trovare la combinazione ottimale degli iperparametri del modello, è stata utilizzata la tecnica del finetuning.

Di seguito, viene riportato lo pseudocodice del training del modello:

```

1 for n_d_a , n_step,n_independent,n_shared , gamma , epsilon ,nums_epochs ,
2   batch_sizes in hyperparameters:
3
4   model = get_model(n_d_a , n_step , n_independent , n_shared , gamma , epsilon)
5
6   model.fit(
7     X_train=X_train ,
8     y_train=Y_train ,
9     eval_set=[(X_val , Y_val)],
10    eval_metric=[ 'mse' ],
11    patience=10 ,
12    batch_size=batch_sizes ,
13    virtual_batch_size=128 ,
14    num_workers=0 ,
15    drop_last=False ,
16    max_epochs=nums_epochs ,
17  )
18
19  preds = model.predict(X_test)
20  mse = mean_squared_error(Y_test , preds)
21
22  if mse < best_mse:
23    best_mse = mse
24    best_n_d = n_d_a
25    best_n_a = n_d_a
26    best_n_step = n_step
27    best_n_independent = n_independent
28    best_n_shared = n_shared
29    best_gamma = gamma
30    best_batch_size = batch_sizes
31    best_model = model

```

Il numero di epoche utilizzato durante l'addestramento del modello è stato fissato arbitrariamente a 200. Il training è regolato dalla *patience* che termina l'addestramento quando il non c'è un miglioramento della loss dopo un numero fissato di passi evitando di addestrare il modello oltre

quanto è necessario, analogamente a quanto accade nel modello deep. In questo caso si è scelto di fissare il valore patience a 10.

8 Risultati

Per misurare e confrontare le performance dei vari modelli abbiamo usato due metriche: l'errore quadratico medio (MSE) e l' R2 score.

l'MSE calcola la media dei quadrati degli errori tra i valori previsti dal modello e i valori reali osservati nei dati di test. Un valore di MSE 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.

La formula per calcolarlo è la seguente:

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

dove n è il numero di osservazioni, y_i è il valore osservato e \hat{y}_i è il valore previsto dal modello per l'osservazione i .

Un'altra misura utilizzata è il coefficiente di determinazione, noto anche come R2 score. Questa misura indica quanto la varianza dei dati di output previsti dal modello si avvicina alla varianza dei dati di output reali. Il punteggio R2 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.

Di seguito la formula per calcolarlo:

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

dove n è il numero di osservazioni, y_i è il valore osservato, \hat{y}_i è il valore previsto dal modello per l'osservazione i e \bar{y} è la media dei valori osservati.

Di seguito verranno illustrati tutti i risultati del finetuning dei modelli presentati nelle precedenti sezioni, con particolare enfasi sulla configurazione migliore. Ogni modello è stato allenato e testato due volte. Una volta utilizzando il set di training al quale è stato applicata la tecnica PCA, e una seconda volta in cui questa tecnica non è stata applicata.

8.1 Random Forest Regressor

Il Random Forest Regressor è stato allenato cercando la migliore configurazione tra il criterion e il numero di estimator.

Con PCA

La migliore configurazione ottenuta dal training con l'utilizzo della pca è:

- numero di estimator: 85
- criterion: Squared Error

I risultati ottenuti da questa configurazione sono:

- MSE: 0.03276
- R2 score: 0.8378

La figura mostra i risultati di ogni combinazione provata dal modello. I primi due assi rappresentano un iperparametro, l'ultimo asse rappresenta il valore della loss di quella combinazione. La linea che unisce i tre assi rappresenta la combinazione ed il risultato di quest'ultima in termini di errore. Evidenziata in verde possiamo apprezzare la combinazione che ha dato il risultato migliore.

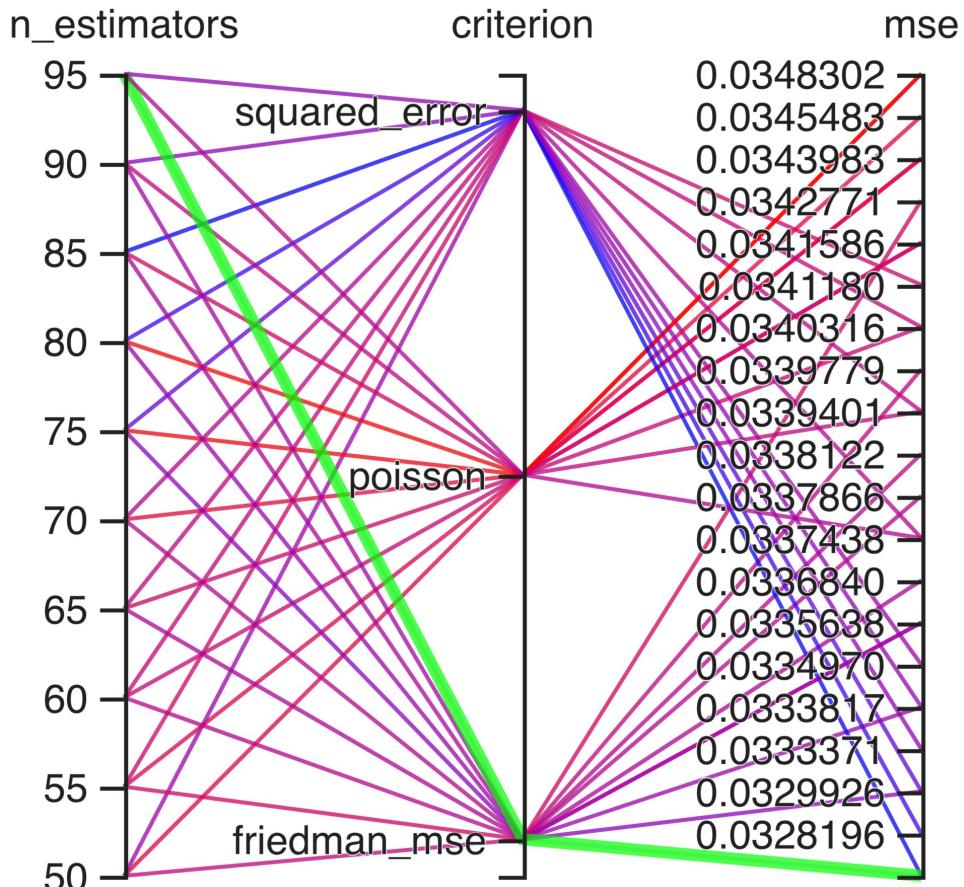


Figura 13: Combinazioni iperparametri con PCA: Random Forest Regressor

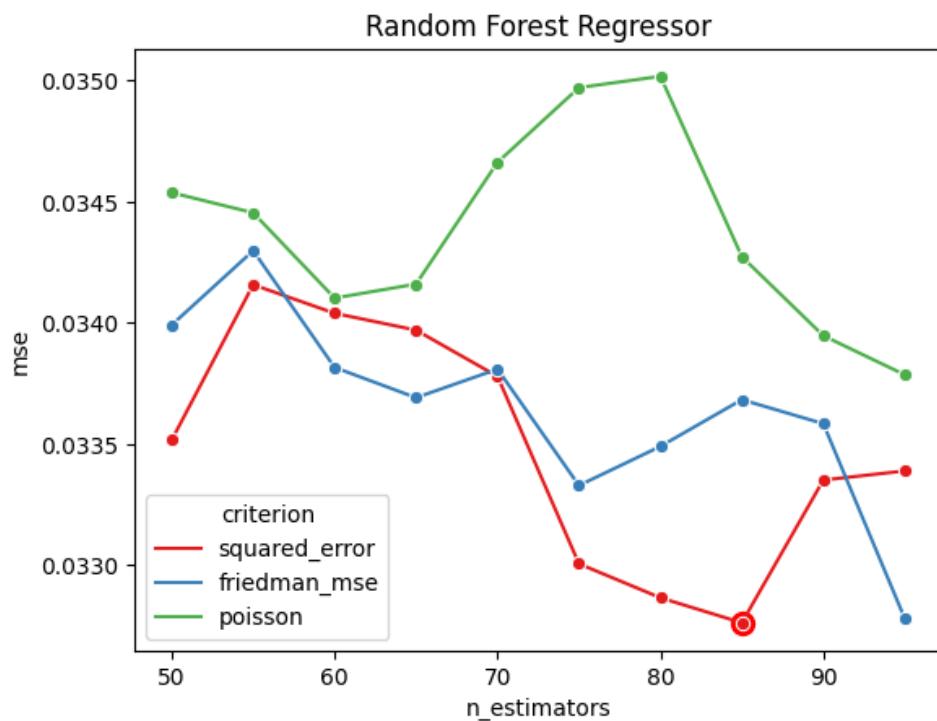


Figura 14: Risultato del finetuning con PCA: Random Forest Regressor

Senza PCA

La migliore configurazione ottenuta dal training senza l'utilizzo della pca è:

- numero di estimator: 60
- criterion: squared error

I risultati ottenuti da questa configurazione sono:

- MSE: 0.01194
- R2 score: 0.9446

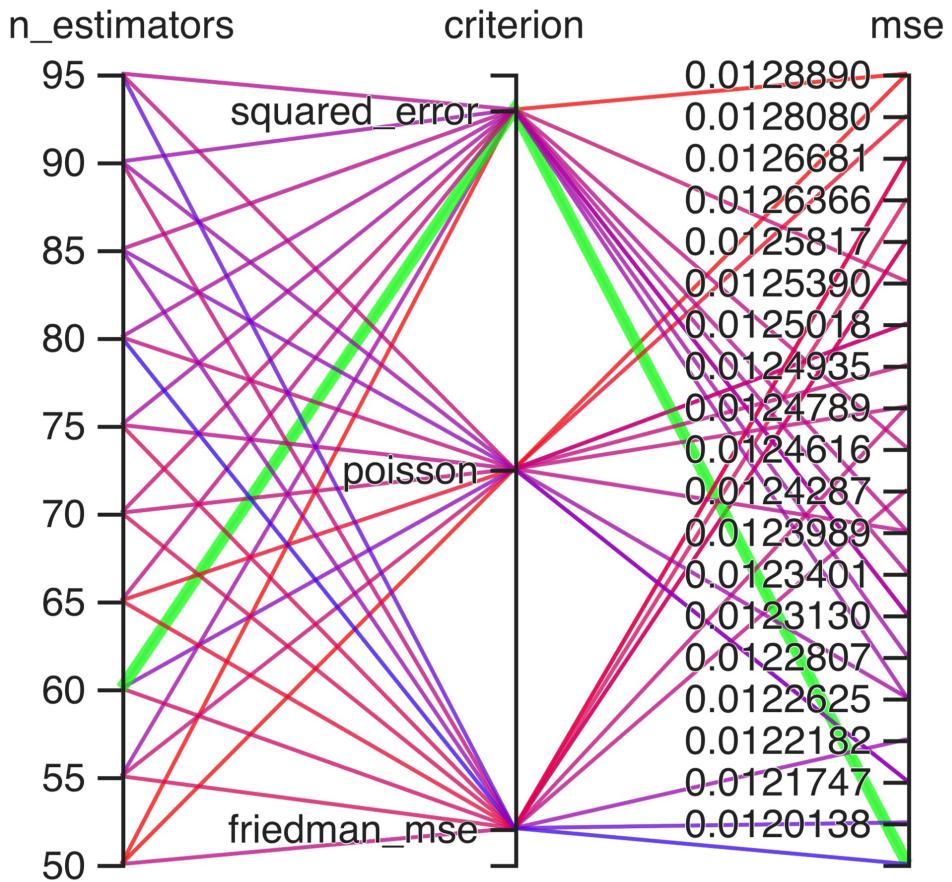


Figura 15: Combinazioni iperparametri senza PCA: Random Forest Regressor

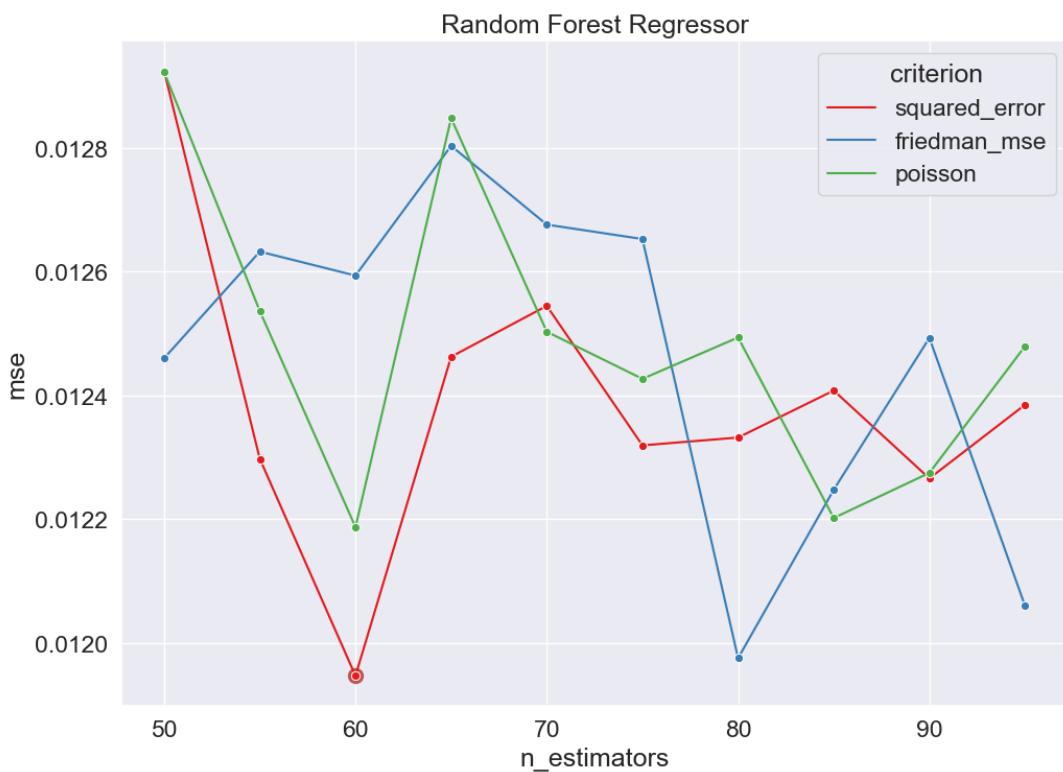


Figura 16: Risultato del finetuning senza PCA: Random Forest Regressor

8.2 Support Vector Regression

Il Support Vector Regressor è stato allenato cercando la migliore configurazione tra ε , C e Kernel

Con PCA

La migliore configurazione ottenuta dal training con l'utilizzo della pca è:

- Kernel: rbf
- C: 1
- ε : 0.001

I risultati ottenuti da questa configurazione sono:

- MSE: 0.006359
- R2 score: 0.97131

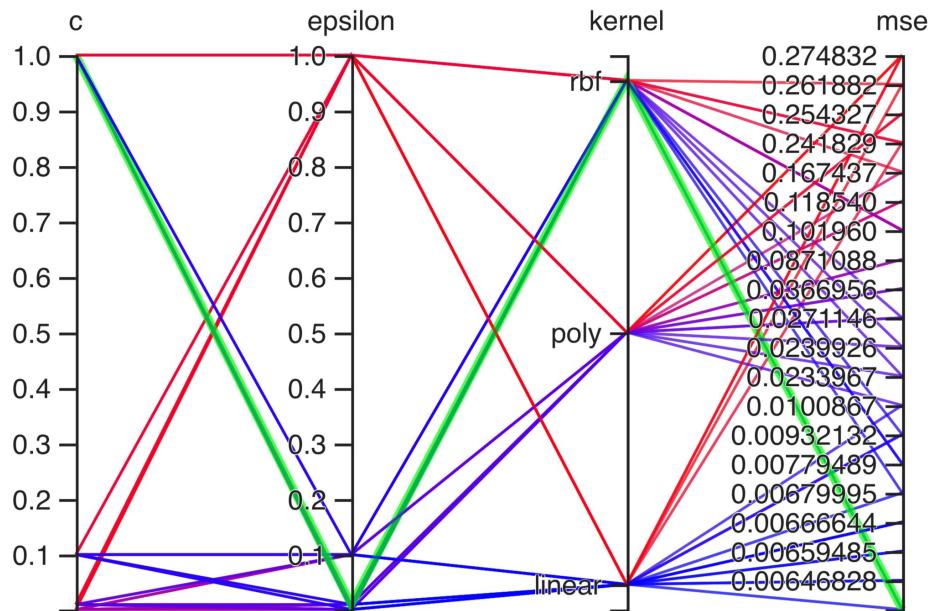


Figura 17: Combinazioni iperparametri con PCA: SVR

Senza PCA

La migliore configurazione ottenuta dal training senza l'utilizzo della pca è:

- Kernel: RBF
- C: 1.0
- ε : 0.001

I risultati ottenuti da questa configurazione sono:

- MSE: 0.004959
- R2 score: 0.9764

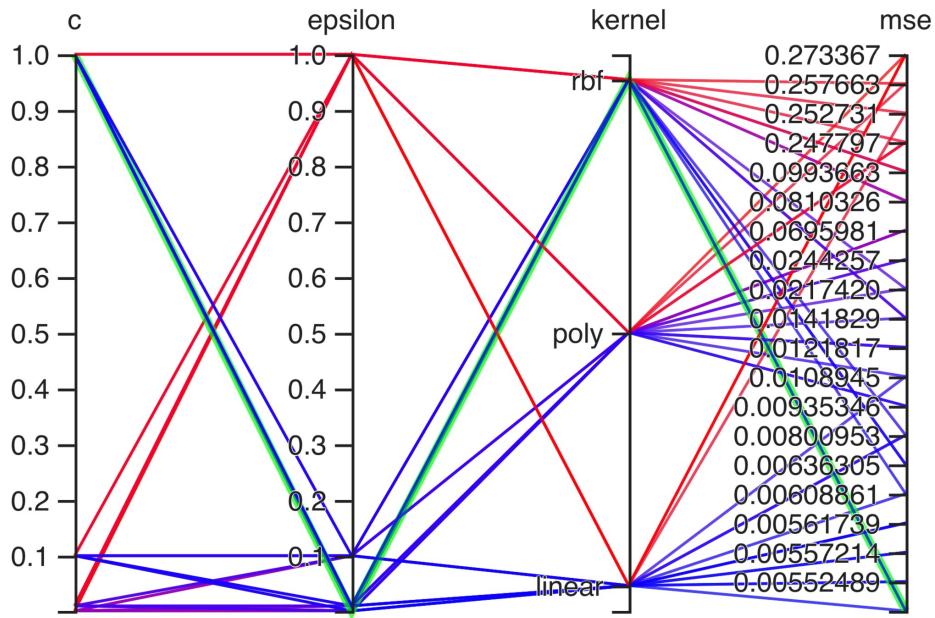


Figura 18: Combinazioni iperparametri senza PCA: SVR

8.3 Linear Regression

Con PCA

I risultati della Linear Regression senza utilizzare la tecnica della PCA sono i seguenti:

- MSE: 0.006436
- R2 Score: 0.9709

Senza PCA

I risultati della Linear Regression utilizzando la tecnica della PCA sono i seguenti:

- MSE: 0.005445
- R2 Score: 0.9754

8.4 Lasso Regression

Il Lasso Regressor è stato allenato cercando la migliore configurazione iterando sui possibili valori di α . Di seguito verranno illustrate le migliori combinazioni e i risultati derivanti dal training di quest'ultime

Con PCA

La migliore configurazione ottenuta dal training con l'utilizzo della pca è:

- α : 1e-06

I risultati ottenuti da questa configurazione sono:

- MSE: 0.006421
- R2 score: 0.9710

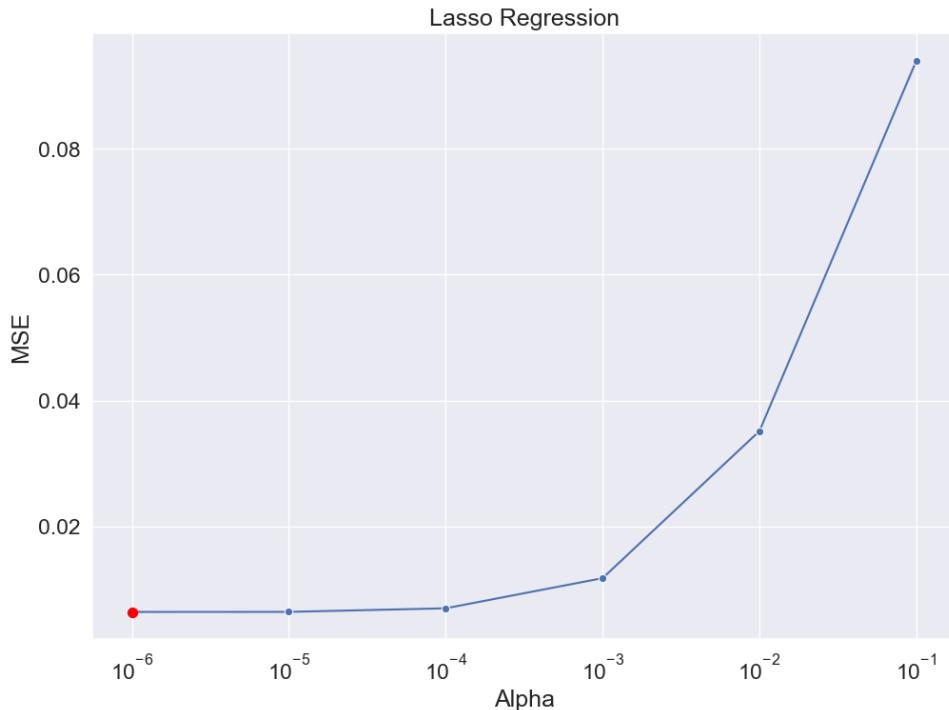


Figura 19: Training del Lasso Regressor con PCA

Senza PCA

La migliore configurazione ottenuta dal training senza l'utilizzo della pca è:

- α : 1e-05

I risultati ottenuti da questa configurazione sono:

- MSE: 0.005404
- R2 score: 0.9755

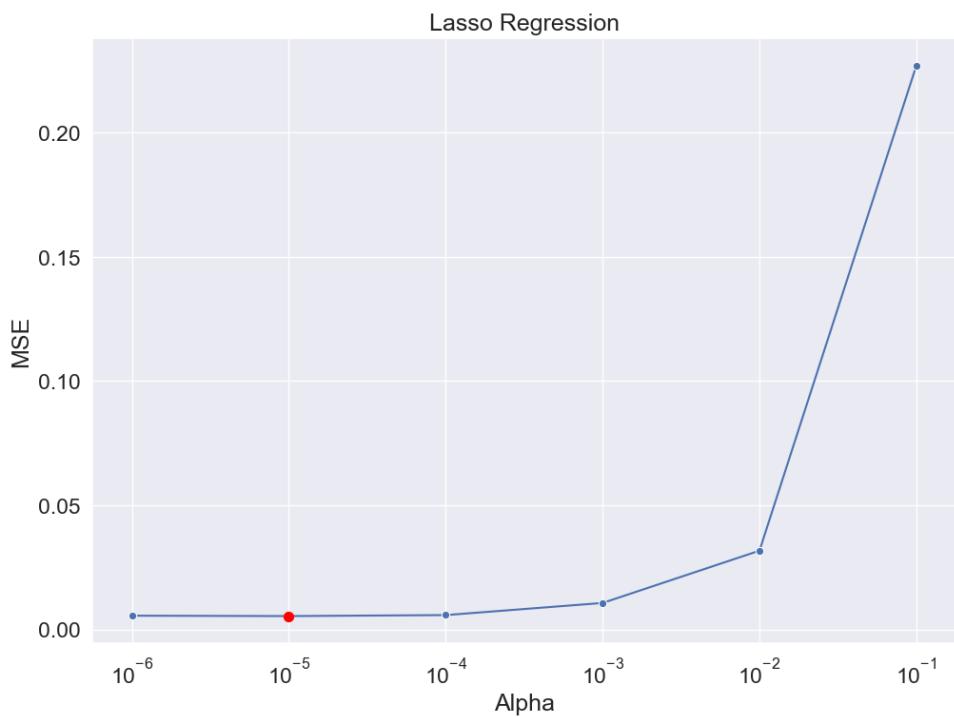


Figura 20: Training del Lasso Regressor senza PCA

8.5 Ridge Regression

Il Ridge Regressor è stato allenato cercando la migliore configurazione iterando sui possibili valori di α . Di seguito verranno illustrate le migliori combinazioni e i risultati derivanti dal training di quest'ultime

Con PCA

La migliore configurazione ottenuta dal training con l'utilizzo della pca è:

- α : 1e-6

I risultati ottenuti da questa configurazione sono:

- MSE: 0.006421
- R2 score: 0.9710

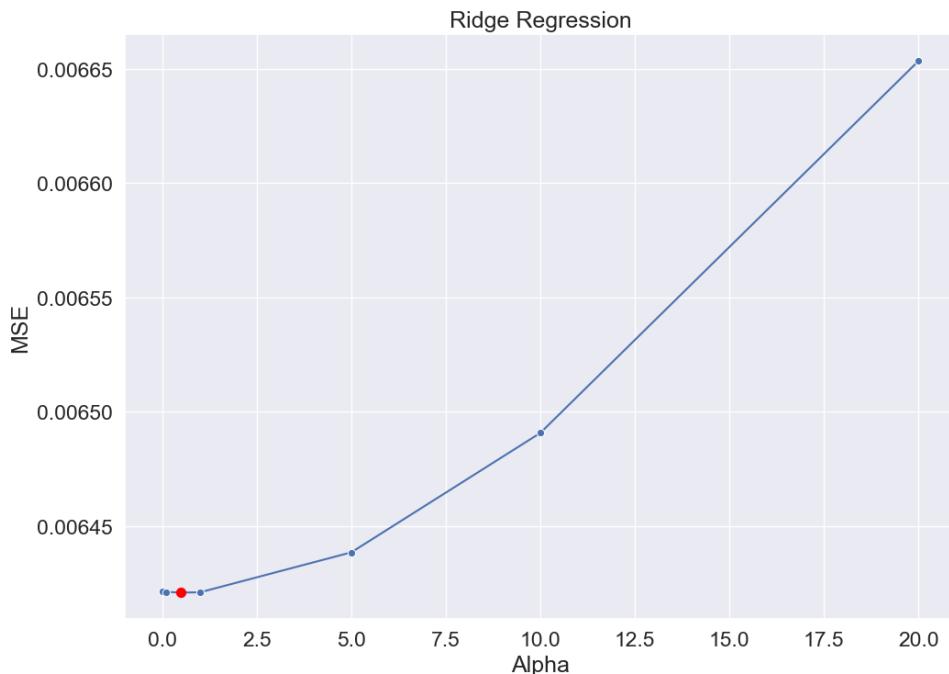


Figura 21: Training del Ridge Regressor con PCA

Senza PCA

La migliore configurazione ottenuta dal training senza l'utilizzo della pca è:

- α : 5

I risultati ottenuti da questa configurazione sono:

- MSE: 0.005398
- R2 score: 0.9760

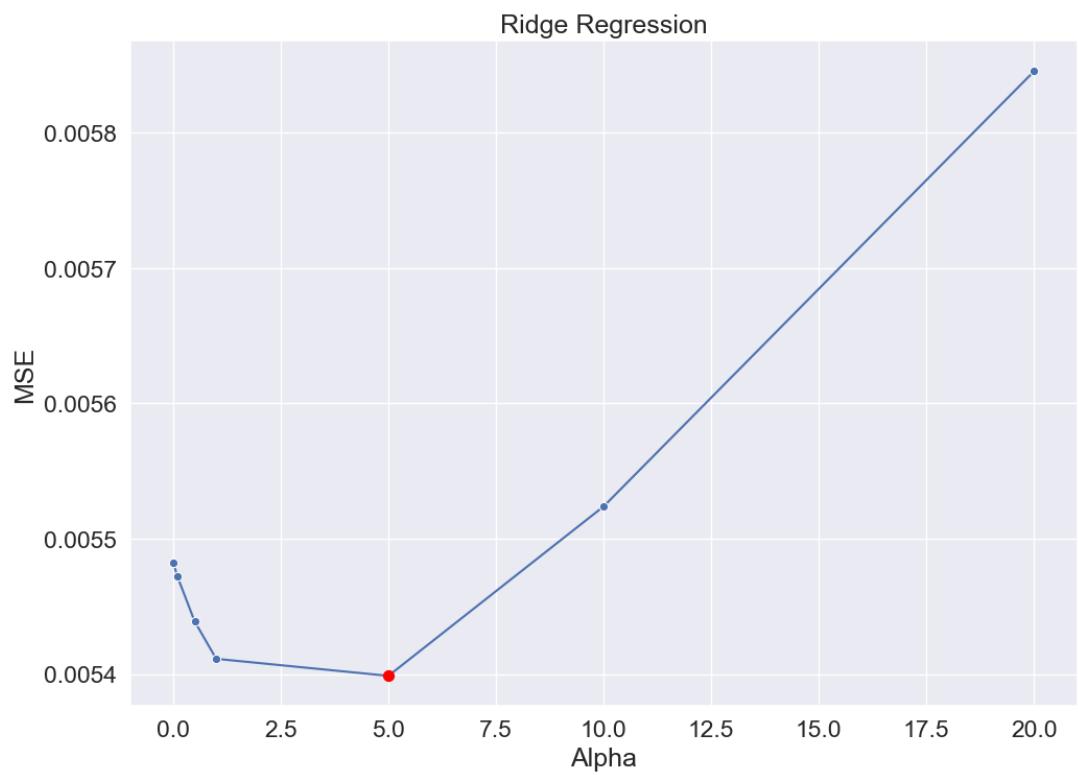


Figura 22: Training del Ridge Regressor senza PCA

8.6 Elastic Net Regression

L'Elastic Net Regressor è stato allenato cercando la migliore configurazione iterando sui possibili valori di α . Di seguito verranno illustrate le migliori combinazioni e i risultati derivanti dal training di quest'ultime

Con PCA

La migliore configurazione ottenuta dal training con l'utilizzo della pca è:

- α : 1e-06

I risultati ottenuti da questa configurazione sono:

- MSE: 0.006421
- R2 score: 0.9710

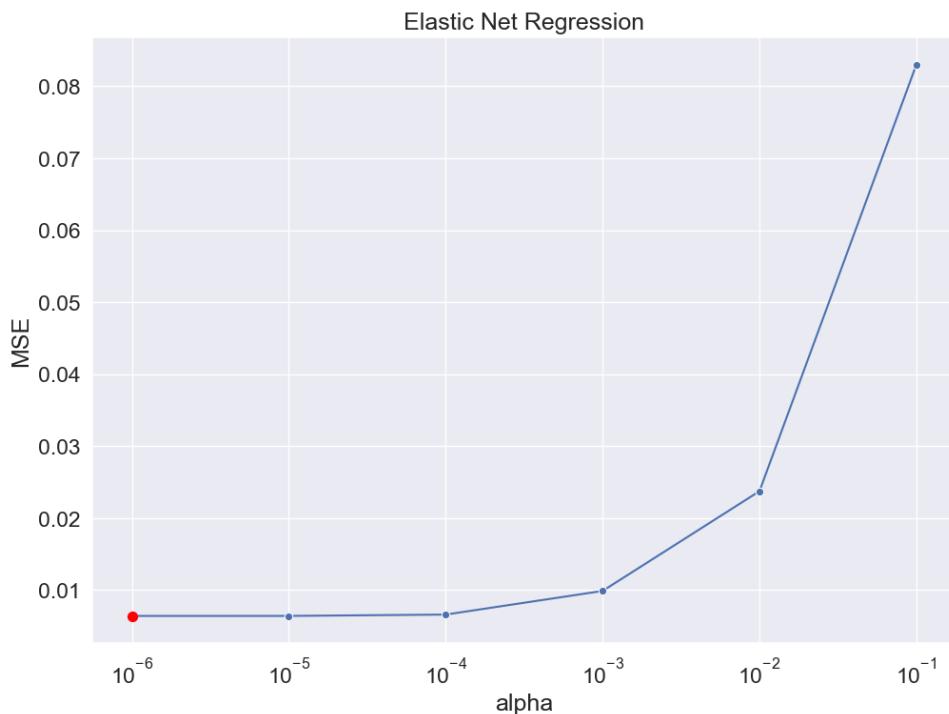


Figura 23: Training dell'Elastic Net Regressor con PCA

Senza PCA

La migliore configurazione ottenuta dal training senza l'utilizzo della pca è:

- α : 1e-04

I risultati ottenuti da questa configurazione sono:

- MSE: 0.005432
- R2 score: 0.9757

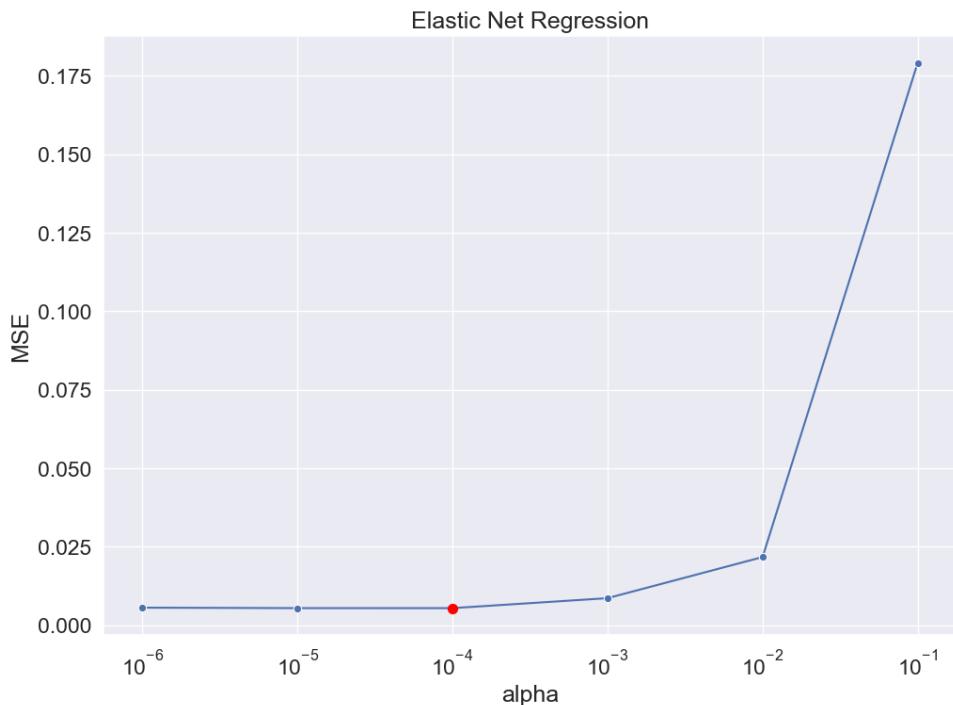


Figura 24: Training dell'Elastic Net senza PCA

8.7 Deep Neural Network

La rete neurale è stata addestrata cercando la migliore configurazione iterando sui possibili valori degli iperparametri illustrati nella sezione 6. Di seguito verrà illustrata le migliore combinazione e i risultati derivanti dal training di quest'ultima. Anche in questo caso si è deciso di effettuare il training sia utilizzando la PCA sul set di training, sia senza utilizzarla.

Con PCA

La migliore configurazione ottenuta dal training è:

- **hidden_size**: 1024
- **depth**: 5
- **batch_size**: 8
- **learning_rate**: 0.01
- **step_size_lr_decay**: 20

I risultati ottenuti da questa configurazione sono:

- MSE: 0.004927
- R2 score: 0.9778

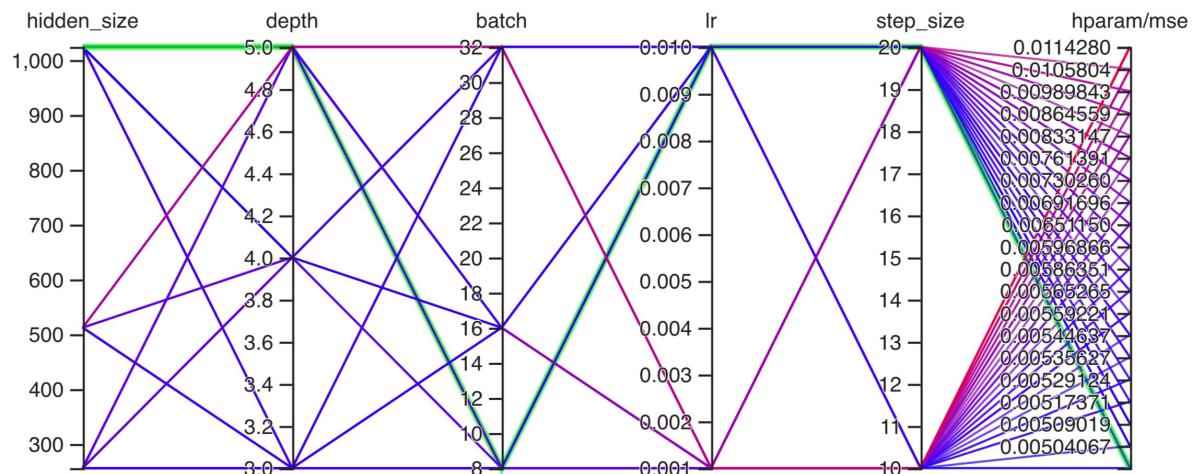


Figura 25: Tutte le combinazioni effettuate durante il finetuning per il training della rete neurale con PCA

Senza PCA

La migliore configurazione ottenuta dal training è:

- **hidden_size**: 1024
- **depth**: 5
- **batch_size**: 8
- **learning_rate**: 0.01
- **step_size_lr_decay**: 20

I risultati ottenuti da questa configurazione sono:

- MSE: 0.004480
- R2 score: 0.9798

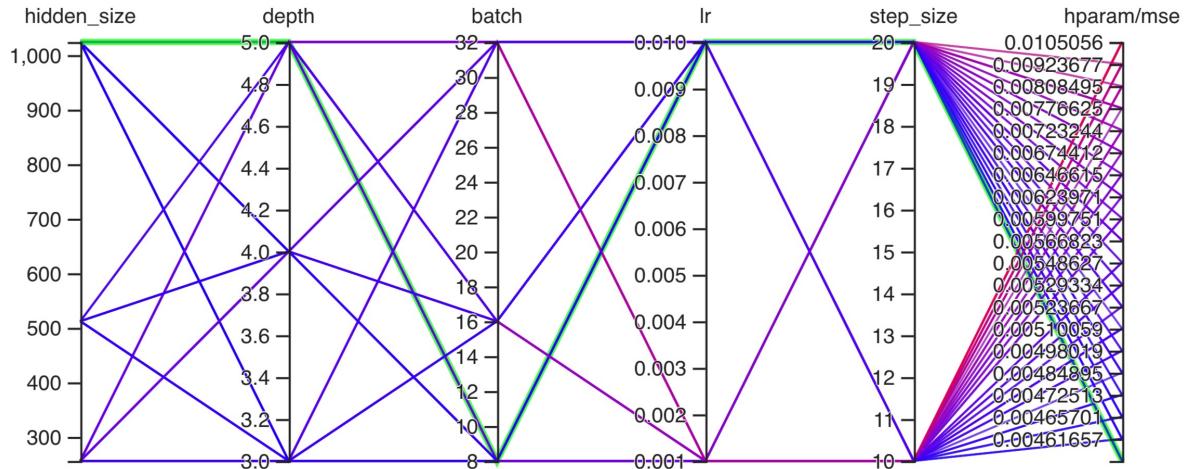


Figura 26: Tutte le combinazioni effettuate durante il finetuning per il training della rete neurale senza PCA

8.8 TabNet

La rete TabNet è stata addestrata cercando la migliore configurazione iterando sui possibili valori degli iperparametri illustrati nella sezione 7. Di seguito verrà illustrata le migliore combinazione e i risultati derivanti dal training di quest'ultima. In questo caso non è stata applicata la tecnica della PCA sul set di training in quanto la scelta delle feature caratteristiche ed importanti nel training del modello è una proprietà intrinseca della rete.

La migliore configurazione ottenuta dal training è:

- *batch_size*: 128
- n_d : 16
- n_a : 16
- n_{steps} : 6
- γ : 1.3
- *optimizer*: Adam
- $n_{independent}$: 3
- n_{shared} : 2
- ϵ : 1e-08
- *seed*: 42

In Figura 27 si visualizzano tutte le combinazioni effettuate dall'itertool. Si noti che gli unici iperparametri che vengono mostrati sono quelli che vengono iterati in un range di valori e dunque non fissati.

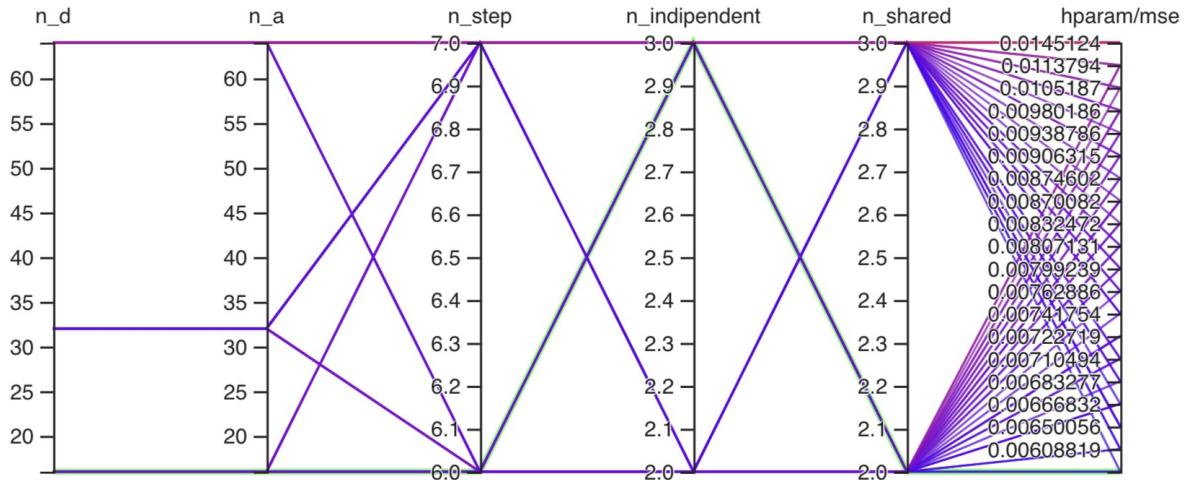


Figura 27: Tutte le combinazioni effettuate durante il finetuning per il training della rete TabNet

I risultati ottenuti da questa configurazione sono:

- MSE: 0.005215
- R2 score: 0.9765

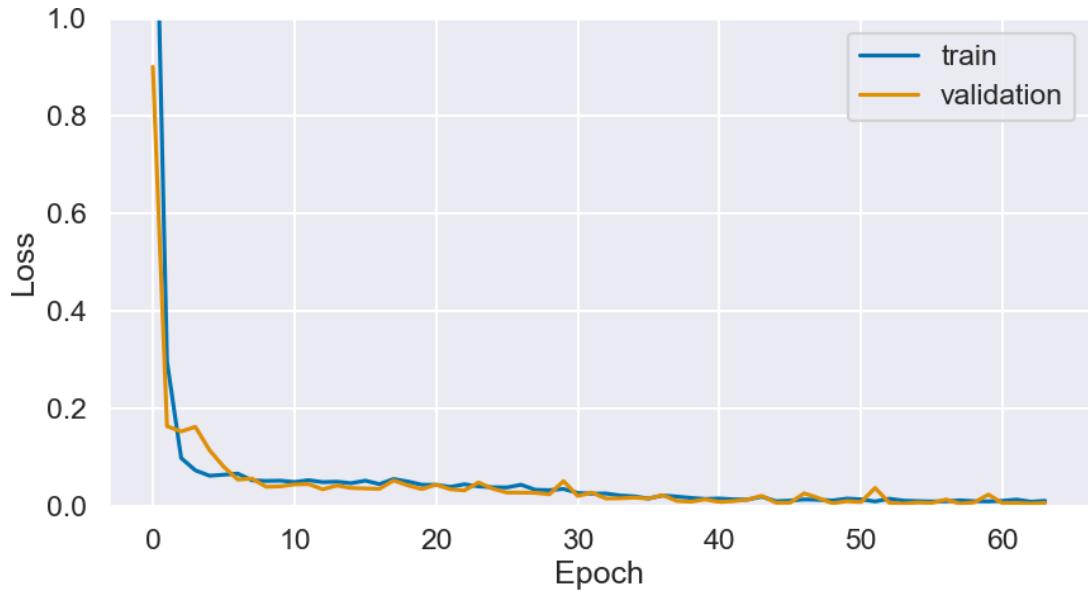


Figura 28: Training e Validation Loss

9 Conclusioni

Analizzando i risultati ottenuti, possiamo trarre le seguenti conclusioni. Tenendo in considerazione il task richiesto, tutti i modelli addestrati con i dati trasformati tramite PCA mostrano risultati leggermente inferiori rispetto ai modelli addestrati sui dati non trasformati. Tuttavia, l'uso della tecnica PCA offre il vantaggio di ridurre il numero di feature del dataset, riducendo così la pressione sulla memoria durante il training dei modelli.

In generale, tutti i modelli, ad eccezione dei random forest regressor, hanno mostrato ottime prestazioni. Ciò suggerisce che questo problema di regressione può essere risolto efficacemente con modelli lineari. I modelli Deep Learning sono stati in grado di ottenere risultati leggermente migliori, ma richiedono tempi di training significativamente più lunghi rispetto alla regressione lineare che offre prestazioni simili in pochi secondi. Le stesse considerazioni valgono per la tecnica tabulare, in cui il modello è in grado di esprimere una maggiore complessità rispetto a quella richiesta dal task del dataset.