

# Capitolo 1

## Implementazione

### 1.1 Data Preprocessing

La fase di preprocessing dei dati consiste nel preparare i dati per l'addestramento del modello. Una volta acquisiti i dati e averne studiato le peculiarità e le caratteristiche, si procede con la pulizia dei dati. Abbiamo usato il metodo *pivot\_table* di Pandas per creare una tabella pivot che contiene i genome-score degli utenti per ogni film che lo possiede (andando così a ridurre la cardinalità da 60k a 13.816).

Ci siamo inoltre andati a focalizzare sui voti da parte degli utenti, raggruppandoli per film ed abbiamo calcolato la media dei voti. Quest'ultimo sarà il nostro *target* per l'addestramento del modello.

### 1.2 Modeling

La nostra fase di modellazione include uno studio di regressione basato sulla predizione del voto medio di un film dati i suoi genome-score. Abbiamo inoltre applicato una tecnica di *PCA* con l'obiettivo di ridurre la cardinalità dei dati, eseguendo l'addestramento sia con che senza la PCA su tutti i modelli al fine di confrontarne i risultati.

Eseguiamo il train-test split con un rapporto 80-20 ed in seguito addestreremo il nostro modello.

### 1.2.1 Tecniche di ML Supervisionate con Approccio non-Deep

In questa sezione verranno descritte le tecniche di ML supervisionate con approccio non-Deep utilizzate per la predizione del voto medio di un film.

- **Linear Regression:** La regressione lineare è una tecnica di ML supervisionata che permette di predire il valore di una variabile dipendente a partire da una o più variabili indipendenti.
  - **Ridge:** Ridge è un algoritmo di regressione lineare che utilizza un termine di regolarizzazione per evitare problemi di multicollinearità. Il parametro di regolarizzazione controlla l'importanza del termine di regolarizzazione nella funzione di costo
  - **Lasso:** Lasso è un algoritmo di ML supervisionato che permette di effettuare la regressione di un dato mediante la creazione di un modello lineare.
- **Decision Tree:** Decision Tree è un algoritmo di ML supervisionato che permette di effettuare la classificazione o la regressione di un dato mediante la creazione di un albero di decisione.
- **Random Forest:** Random Forest è un algoritmo di ML supervisionato che permette di effettuare la classificazione o la regressione di un dato mediante la creazione di più alberi (Metodo di Ensemble Learning).
- **SVR:** Support Vector Regression è un algoritmo che cerca di trovare una funzione che minimizzi la distanza tra i dati di training e una fascia di tolleranza definita dall'utente.
- **KNN:** K-Nearest Neighbors è un algoritmo di ML supervisionato che permette di effettuare la classificazione o la regressione di un dato cerca di stimare il valore di una variabile dipendente su nuovi dati in base alla loro vicinanza ad altri dati di training.

Per ogni modello precedentemente citato è stato applicato il Tuning dei parametri per cercare di ottimizzare il modello. Attraverso la funzione *Grid-Search* di Scikit-Learn abbiamo cercato di trovare i migliori parametri per ogni modello.

## Tuning dei Parametri

Per ogni modello precedentemente citato è stato applicato il Tuning dei parametri per cercare di ottimizzare il modello. Attraverso la funzione *GridSearch* di Scikit-Learn abbiamo cercato di trovare i migliori parametri per ogni modello, come nel seguente esempio:

```
1 def tuning(self):
2     search = GridSearchCV(estimator = self.svr, param_grid= self.
  params, cv = 3, n_jobs = 6)
3     search.fit(self.X_train_t, self.y_train)
4     print(search.best_params_)
5     print(search.best_score_)
6     self.svr = search.best_estimator_
```

### 1.2.2 Linear Regression

Il modello di Linear Regression ci permette di operare una regressione lineare tra i nostri dati (una regressione multivariable). In questo specifico modello non abbiamo particolari parametri per cui fare tuning. Di conseguenza andremo a vedere i risultati del tuning per i modelli che usano la regressione lineare: Lasso e Ridge

#### Ridge

Ridge è un algoritmo di ML supervisionato che permette di effettuare la regressione di un dato mediante la creazione di un modello lineare. Si basa su un valore di regolarizzazione che permette di evitare problemi di multicollinearità. Per cercare il valore  $\alpha$  apposito abbiamo utilizzato la funzione *RidgeCV* di Scikit-Learn, che permette di trovare il valore ottimale di  $\alpha$ , ed effettuare la Cross Validation.

Modello	senza PCA	PCA
$\alpha$	6.0	2.5

#### Lasso

Lasso è un algoritmo di ML supervisionato che permette di effettuare la regressione di un dato mediante la creazione di un modello lineare. Abbiamo cercato anche qui di ottimizzare il valore  $\alpha$  abbiamo utilizzato la funzione *LassoCV* di Scikit-Learn, che permette di trovare il valore ottimale di  $\alpha$ , ed effettuare la Cross Validation.

Modello	senza PCA	PCA
Alpha	7.425262873115622e-05	0.0005675190692206821

### 1.2.3 Decision Tree

Decision Tree è un algoritmo di ML supervisionato che permette di effettuare la regressione di un dato mediante la creazione di un albero di decisione. Abbiamo avuto bisogno di fare il tuning di più parametri, in particolare:

```

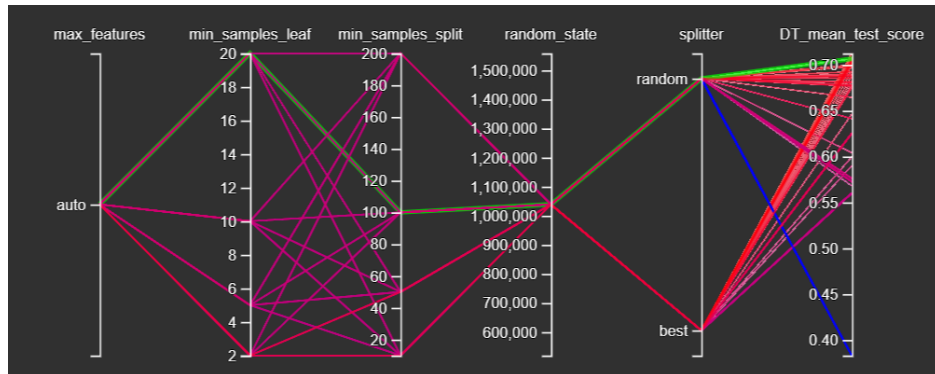
1 self.params = {
2     'splitter' : ['best', 'random'],
3     'max_depth' : [None, 20, 50, 100],
4     'min_samples_split' : [10, 50, 100, 200],
5     'min_samples_leaf' : [2, 5, 10, 20],
6     'max_features' : ['auto', 'sqrt', 'log2'],
7     'max_leaf_nodes' : [None, 5, 10, 20],
8 }

```

Abbiamo utilizzato la funzione *GridSearchCV* di Scikit-Learn per trovare i migliori parametri per il modello, utilizzando un CV di 3 volendo mantenere coerenza con altre ricerche che con una CV maggiore avrebbero portato ad un incremento considerevole del tempo. Di seguito i risultati ottenuti:

Parametri	senza PCA	PCA
<b>max_depth</b>	None	None
<b>max_features</b>	'auto'	'auto'
<b>max_leaf_nodes</b>	None	None
<b>min_samples_leaf</b>	20	20
<b>min_samples_split</b>	50	100
<b>splitter</b>	'best'	'best'

Mostriamo inoltre il grafico che mostra l'andamento del Mean Test Score in funzione dei parametri, ricordando che nei modelli successivi verrà mostrato il grafico solo con dati senza PCA:



### 1.2.4 Random Forest

Random Forest è un algoritmo di ML supervisionato che permette di effettuare la regressione di un dato mediante la creazione di più alberi (Metodo di Ensemble Learning). I parametri su cui effettueremo il tuning sono:

```

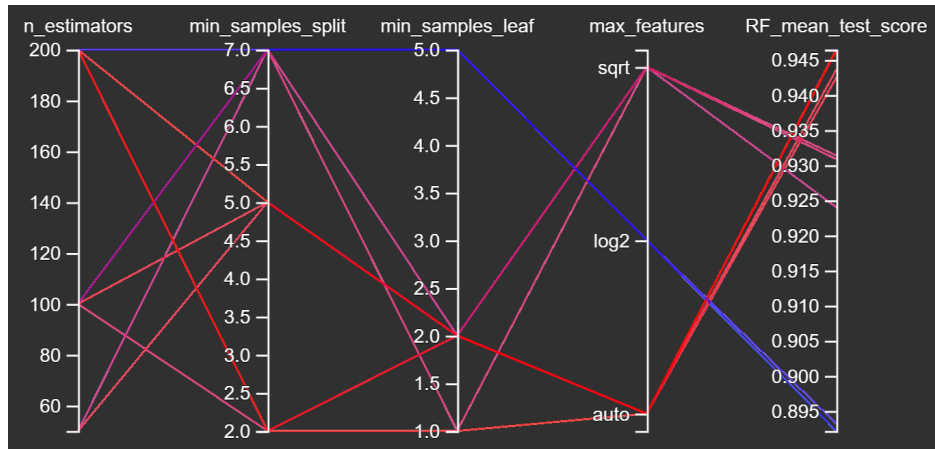
1 self.params = {
2     'n_estimators': [50,100,500],
3     'max_depth': [None ,10, 50, 100],
4     'min_samples_split': [2, 5, 10],
5     'min_samples_leaf': [1, 2, 5],
6     'max_features': ['auto', 'sqrt', 'log2'],
7 }

```

Ottenendo i seguenti risultati:

Parametri	senza PCA	PCA
max_depth	None	None
max_features	'auto'	'auto'
min_samples_leaf	1	1
min_samples_split	2	2
n_estimators	500	500

Qui di seguito inoltre riportiamo un grafico che mostra la distribuzione dei parametri e del Mean Test Score ottenuto:



## 1.2.5 SVR

SVR è un algoritmo di ML supervisionato che permette di effettuare la regressione di un dato mediante la creazione di un modello lineare. Due parametri rilevanti sono  $C$  e  $\epsilon$ :  $C$  è il parametro di regolarizzazione, mentre  $\epsilon$  è il parametro che permette di definire la tolleranza per il modello.

```

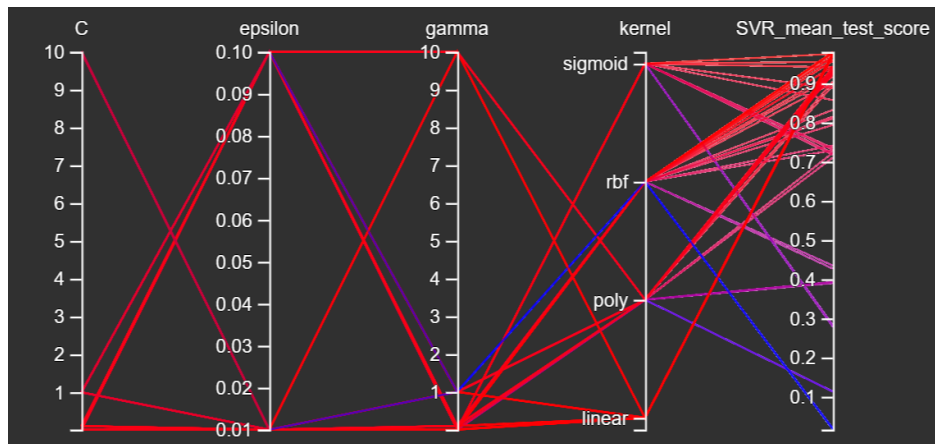
1 self.params = {
2     'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
3     'C': [0.01, 0.1, 1, 10, 100],
4     'gamma': [0.001, 0.01, 0.1, 1, 10],
5     'epsilon': [0.01, 0.1, 1, 10]
6 }

```

Ottendendo i seguenti risultati:

Parametri	senza PCA	PCA
<b>C</b>	1	100
<b>epsilon</b>	0.01	0.01
<b>gamma</b>	0.01	0.001
<b>kernel</b>	'rbf'	'rbf'

Come per il modello precedente riportiamo il grafico dell'andamento dei parametri e del Mean Test Score:



### 1.2.6 KNN

KNN è un algoritmo di ML supervisionato che permette di effettuare una regressione sui dati trovando il target predetto da un interpolazione locale dei target associati ai k più vicini. Abbiamo effettuato il tuning di parametri:

```

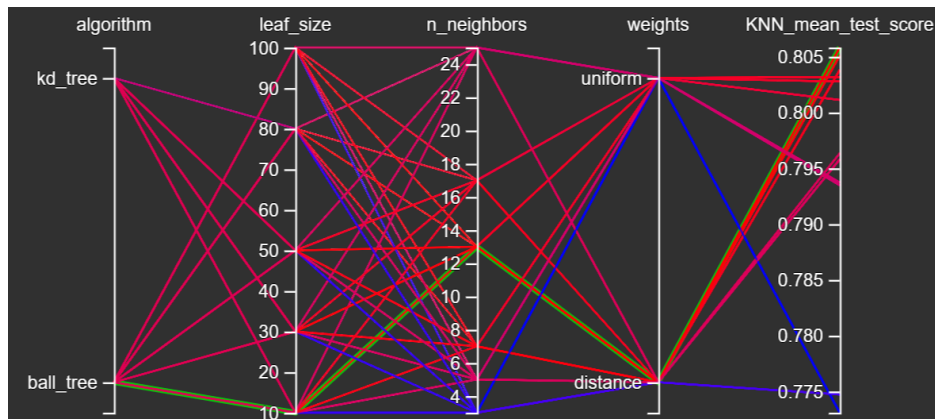
1 self.params = {
2     'n_neighbors': [3, 5, 7, 13, 17, 25],
3     'weights': ['uniform', 'distance'],
4     'algorithm': ['ball_tree', 'kd_tree', 'brute'],
5     'leaf_size': [10, 30, 50, 80, 100],
6 }

```

Con i seguenti risultati:

Parametri	senza PCA	PCA
<b>algorithm</b>	ball_tree	ball_tree
<b>leaf_size</b>	10	10
<b>n_neighbors</b>	13	13
<b>weights</b>	'distance'	'distance'

Come per i modelli precedenti riportiamo il grafico dell'andamento dei parametri e del Mean Test Score:



### 1.2.7 Tecniche di ML Supervisionate con Reti Neurali

Mostriamo di seguito la nostra Implementazione della Rete Neurale Feed Forward, che abbiamo utilizzato per effettuare la regressione dei dati. Abbiamo utilizzato il framework Pytorch per la creazione della rete neurale, che ci ha permesso di utilizzare la GPU per l'addestramento della rete.

```

1  # Define the neural network architecture
2  self.model = nn.Sequential(
3      nn.Linear(X.shape[1], hidden_size1),
4      nn.ReLU(),
5      nn.Linear(hidden_size1, hidden_size2),
6      nn.ReLU(),
7      nn.Linear(hidden_size2, 1)
8  )
9
10 self.criterion = nn.MSELoss()
11 self.optimizer = optim.Adam(self.model.parameters(), lr=lr)

```

Abbiamo effettuato il tuning degli iperparametri della rete neurale utilizzando `itertools.product`, che ci ha permesso di effettuare il tuning di tutti i parametri in maniera semplice e veloce. I parametri che abbiamo utilizzato sono:

```

1  self.params = {
2      batchsize = [128,256,512,1024]
3      hidden_size1 = [64,128,256]
4      hidden_size2 = [64,128,256]
5      lr = [0.0001,0.001,0.01]
6      num_epochs = [200,400,600]
7  }

```

I risultati ottenuti sono:



Parametri	senza PCA	PCA
<b>batchsize</b>	1024	256
<b>hidden_size1</b>	64	128
<b>hidden_size2</b>	256	256
<b>lr</b>	0.01	0.001
<b>num_epochs</b>	400	600

Visualizziamo il grafico dell'andamento dei parametri e del Mean Test Score:

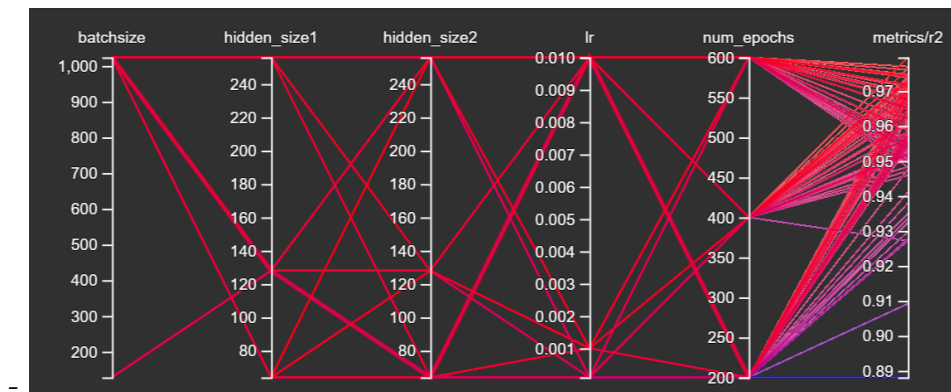


Figura 1.1: Andamento dei parametri e del Mean Test Score per la Rete Neurale (senza PCA)

## 1.3 Tabular Data

Per quanto riguarda i Tabular Data abbiamo fatto il tuning dei seguenti iperparametri (sempre utilizzando `itertools.product`):

```

1  batchsize = [512,1024,2048]
2  width = [8,16,32]
3  steps = [3,5,7]
4  learning_rate = [2e-2,1e-2,5e-3]
5  max_epochs = [70,120,150,210]
```

dove *width* corrisponde a  $n_d$  e  $n_a$  dove  $n_d$  è la Larghezza del livello di previsione delle decisioni ( $n_a$  nella documentazione viene consigliato sia uguale a  $n_d$ , per questo motivo utilizziamo un singolo valore). Viene fatto uso di Early Stopping per evitare l'overfitting. I risultati ottenuti sono:

Con un andamento degli iperparametri e del Mean Test Score come segue:

Parametri	senza PCA	PCA
batchsize	512	512
width	32	32
steps	7	5
learning_rate	0.02	0.02
max_epochs	70	150

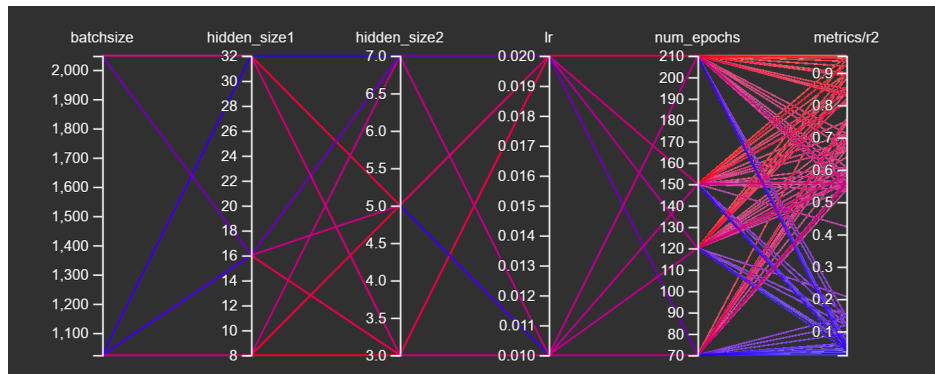


Figura 1.2: Andamento dei parametri e del Mean Test Score per i Tabular Data (senza PCA)