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 cardialità da 60k a 13.816).
In seguito a cià verrà effettuata una merge con un i generi dei film. Ci siamo inoltre andati a focalizzare sui voti da parte degli utenti, raggruppandoli per film ci abbiamo calcolato la media dei voti. Quest'ultimo capo 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 (ed in seguito anche i generi). Abbiamo inoltre applicato una tecnica di *PCA* per ridurre la cardinalità dei dati, ma senza ottenere risultati soddisfacenti, quindi è stato deciso di non farne uso ulteriormente.

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 eviare 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.
- **Deicision 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 Emsable 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 *GridSearch* 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 segue:

```
    def tuning(self):
        search = GridSearchCV(estimator = self.svr, param_grid= self.
    params, cv = 3, n_jobs = 6)
        search.fit(self.X_train_t, self.y_train)
        print(search.best_params_)
        print(search.best_score_)
        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 bisogno particolari parametri da fare tuning. Di conseguenza andremo a vedere i risultati del tuning per i modelli che usano la regressione lineare: Lasso e Ridge

Ridge

Ridge é 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 α apposito abbiamo utilizzato la funzione *RidgeCV* di Scikit-Learn, che permette di trovare il valore ottimale di α, ed effettuare la Cross Validation.

Modello	**senza PCA**	**PCA**
Lambda	6.0	2.5

Lasso

Lasso é 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 λ abbiamo utilizzato la funzione *LassoCV* di Scikit-Learn, che permette di trovare il valore ottimale di λ, ed effettuare la Cross Validation.

Modello	senza PCA	PCA
Alpha	7.425262873115622e-05	0.0005675190692206821

1.2.3 Decision Tree

Decision Tree é 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 piu parametri, in particolare:

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

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
max_depth	None	None
max_features	'auto'	'auto'
max_leaf_nodes	None	None
min_samples_leaf	20	20
min_samples_split	50	100
splitter	'best'	'best'

1.2.4 Random Forest

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

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

Ottendendo 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

1.2.5 SVR

SVR é un algoritmo di ML supervisionato che permette di effettuare la regressione di un dato mediante la creazione di un modello lineare. Due parametri rilvanti sono C e ϵ: C é il parametro di regolarizzazione, mentre ϵ é il parametro che permette di definire la tolleranza per il modello.

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

Ottendendo i seguenti risultati:

Parametri	senza PCA	PCA
C	1	100
epsilon	0.01	0.01
gamma	0.01	0.001
kernel	'rbf'	'rbf'

1.2.6 KNN

KNN é 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 piu vicini. Abbiamo effettuato il tuning di parametri:

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

Con i seguenti risultati:

Parametri	senza PCA	PCA
algorithm	ball_tree	ball_tree
leaf_size	10	10
n_neighbors	13	13
weights	'distance'	'distance'

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.

```
    # Define the neural network architecture
    self.model = nn.Sequential(
        nn.Linear(X.shape[1], hidden_size1),
        nn.ReLU(),
        nn.Linear(hidden_size1, hidden_size2),
        nn.ReLU(),
        nn.Linear(hidden_size2, 1)
    )

    self.criterion = nn.MSELoss()
    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:

```
    self.params = {
        batchsize = [128,256,512,1024]
        hidden_size1 = [64,128,256]
        hidden_size2 = [64,128,256]
        lr = [0.0001,0.001,0.01]
        num_epochs = [200,400,600]
    }
```

I risultati ottenuti sono:

Parametri	senza PCA	PCA
batchsize	1024	256
hidden_size1	256	128
hidden_size2	256	256
lr	0.01	0.001
num_epochs	400	600

1.3 Tabular Data

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

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

dove *width* corrisponde a n_d e n_a dove n_d é 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:

Parametri	**senza PCA**	**PCA**
batchsize	2048	-
width	8	-
steps	5	-
learning_rate	0.02	-
max_epochs	210	-