DATA ANALYTICS

Relazione Progetto d'Esame

Anno Accademico 2023/24

Gabriele Raciti
Matricola : 0001102147
gabriele.raciti2@studio.unibo.it

0. INDICE

1. INTRODUZIONE

L'obiettivo di questo progetto è il riconoscimento dell'anno di pubblicazione di una canzone basandosi su feature estratte dalla sua traccia audio. In particolare, il progetto prevede l'implementazione di modelli in grado di predirre l'anno di pubblicazione utilizzando un dataset contenente 90 feature, attraverso tre diverse funzionalità:

- **Funzionalità 1:** utilizzo di tecniche di Machine Learning supervised tradizionali non deep (Random Forest, Linear Regression, Support-Vector Machines, K-Nearest Neighbors)
- **Funzionalità 2:** utilizzo di tecniche di Machine Learning supervised basate su reti neurali ed architettura Feed-Forward
- **Funzionalità 3:** utilizzo di tecniche di Machine Learning supervised con modelli deep per Tabular Data (TabNet & TabTransformer)

Il progetto è stato strutturato in due componenti software distinte ed autonome dal punto di vista dell'esecuzione:

- **Training Module:** componente che effettua il pre-processing, l'addestramento degli algoritmi implementati, il tuning degli iper-parametri e il salvataggio dei modelli risultanti.
- **Inference Module**: componente che istanzia le tecniche implementate precedentemente usando la configurazione migliore degli iper-parametri identificata durante la fase di Training. Inoltre, implementa un API descritta successivamente.

L'API implementata offre 4 metodi accessibili dall'esterno:

- ***getName()***: restituisce nome e matricola
- ***preprocess(df, clfName)***: prende in input un DataFrame Pandas contenente i dati di test e una stringa che identifica la tecnica di ML da utilizzare. Il metodo effettua il pre-processing dei dati di test in base alla tecnica di ML indicata. L'attributo *clfName* è una stringa con i seguenti valori ammissibili:
    - 'LR': identifica regressore *Linear Regression*
    - 'RF': identifica regressore *Random Forest Regressor*
    - 'KNR': identifica regressore basato su *KNN*
    - 'SVR': identifica regressore *Support Vector Regressor*

    - ‘FF’: identifica regressore con reti neurali, architettura *Feed Forward*
    - ‘TB’: identifica regressore con reti neurali, architettura *TabNet*
    - ‘TF’: identifica regressore con reti neurali, architettura *TabTransformer*

- ***load(clfName)***: prende in input una stringa che identifica la tecnica di ML da utilizzare (tra i valori specificati prima), istanzia la tecnica di ML specificata e restituisce un oggetto Python relativo all’istanza dell’algoritmo di ML, con iper-parametri determinati durante la fase di training.
- ***predict(df, clfName, clf)***: prende in input un dataframe di test (ottenuto come output del metodo di preprocess descritto precedentemente), una stringa che identifica la tecnica di ML da utilizzare e l’istanza dell’algoritmo di ML (ottenuto come output del metodo *load* descritto precedentemente). Restituisce un dizionario Python che contiene i valori delle prestazioni dell’algoritmo di ML quando viene eseguito sui dati di test pre-processati. In particolare, le chiavi del dizionario restituite sono:
    - ‘mse’: valore del Mean Squared Error
    - ‘mae’: valore del Mean Absolute Error
    - ‘mape’: valore del Mean Absolute Percentage Error
    - ‘r2score’: valore di R2 score

2. METODOLOGIA

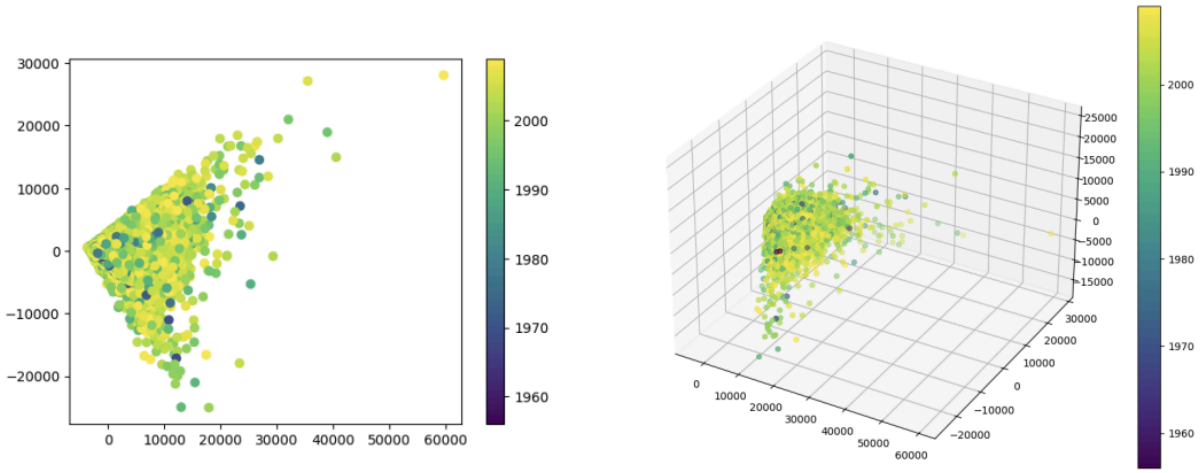
Di seguito verrà mostrata la metodologia utilizzata per lo sviluppo delle funzionalità richieste.
La prima fase ha riguardato l'acquisizione dei dati dal dataset fornito. Successivamente, si è proceduto con la visualizzazione dei dati. Questi sono stati visualizzati in 2 e 3 dimensioni grazie all'utilizzo della PCA, permettendoci di comprendere la distribuzione dei dati a nostra disposizione ed evidenziare eventuali outlier. Inoltre, è stata visualizzata la varianza cumulativa rispetto al numero di componenti principali, consentendoci di valutare se fosse possibile ridurre la dimensionalità dei dati mantenendo comunque alte percentuali della varianza totale.
Infine si è proceduto con la ricerca e l'identificazione degli outliers analizzando le distanze dei vicini di ogni punto tramite KNN.
Successivamente, il dataset iniziale, composto da 252175 campioni, è stato suddiviso in un Train-Set di 226910 campioni e un Validation-Set di 25213 campioni, seguendo una politica di train_test_split con rapporto 0.1. Si è deciso di non utilizzare un Test-Set per poter massimizzare la capacità di Learning dei modelli (permettendo l'addestramento con un numero maggiore di campioni) e per via del successivo test effettuato dai docenti per valutare i risultati ottenuti.
Si è successivamente proceduto allo svolgimento della fase di Pre-processing. Durante questa fase, sono state esplorate varie alternative di pre-processing per poterle successivamente utilizzare con i modelli. Le opzioni rivelatosi utili e salvate per un utilizzo successivo sono state: *Min-max Scaling*, *Standard Scaling*, *Norma L2* (in combinazione con Standard Scaling) e applicazione della *PCA con 30* e *con 70 componenti principali*.
Infine, nella fase di Learning, sono stati addestrati i modelli richiesti utilizzando metodologie come RandomSearch e GridSearch per trovare la combinazione di iperparametri che offrisse i risultati migliori. I modelli sono stati valutati sul Validation-set per la scelta degli iperparametri migliori.
Durante la fase di Pre-Processing e Learning sono state valutate diverse strategie basate sui risultati della fase di visualizzazione, inclusa la rimozione degli outliers. Tuttavia, poiché la rimozione degli outliers ha peggiorato le prestazioni dei modelli, si è deciso di mantenerli per consentire una massimizzazione dei risultati dei vari modelli addestrati.

3. IMPLEMENTAZIONE

Di seguito verrà descritta la fase di sviluppo e implementazione dei vari modelli. In particolare, verranno analizzate le diverse fasi della pipeline con le modalità operative adottate, per poi concentrarsi sulla fase di Learning e sull'esplorazione dei vari iperparametri per ciascun modello.

3.1. Acquisizione e visualizzazione dati

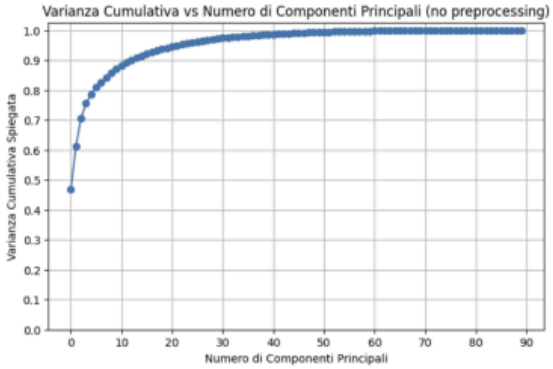
Di seguito verrà fornita una breve descrizione delle fasi di acquisizione e visualizzazione dei dati.
Il file .csv contenente i dati di interesse è stato inizialmente acquisito tramite la libreria **pandas**. Dopo averlo caricato, si è proceduto a rimuovere i valori nulli e i valori duplicati per evitare dati inconsistenti. Successivamente, è stata stampata la shape del dataframe prima e dopo la rimozione dei dati inconsistenti. Il dataframe è stato poi suddiviso in dati di input X e label y (colonna 'Year').
La fase successiva è stata la visualizzazione dei dati, con l'obiettivo di comprendere la loro distribuzione e identificare eventuali outlier, per capire come procedere nelle successive analisi.
Il primo step in questa fase è stato l'utilizzo dell'analisi delle componenti principali (PCA), una tecnica di riduzione della dimensionalità che consente di ridurre dati ad alta dimensionalità a uno spazio bidimensionale e tridimensionale, facilitandone così la visualizzazione. Poiché i nostri dati presentano un'elevata dimensionalità (90 feature), è fondamentale ridurla per poterli rappresentare in 2D o 3D e analizzare la loro distribuzione. La PCA permette di preservare, per quanto possibile, le informazioni essenziali e le relazioni tra i punti dati originali mentre riduce le dimensioni dello spazio dei dati. Questa tecnica ci consente di esplorare la struttura sottostante dei dati prima di intraprendere ulteriori analisi, offrendo una panoramica generale della disposizione e della varianza tra i dati. Una volta ottenute le componenti principali per la rappresentazione in 2D e 3D, si è proceduto alla visualizzazione dei dati tramite scatter plot. Mostriamo di seguito i risultati ottenuti.



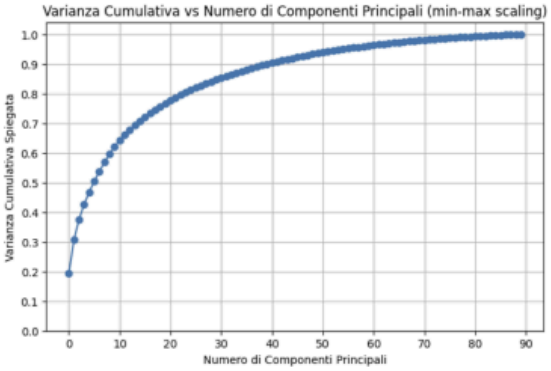
(a) Visualizzazione PCA 2D (b) Visualizzazione PCA 3D

Figure 1: Visualizzazione componenti tramite PCA

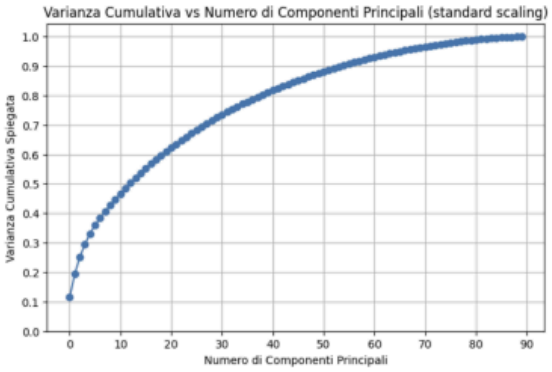
Una volta compresa la distribuzione dei dati, si è deciso di continuare a esplorare le dimensionalità di essi per comprendere quante di queste sono davvero necessarie per la loro analisi. In particolare, si è utilizzato nuovamente l'algoritmo PCA per visualizzare la varianza cumulativa spiegata da ogni componente. Questo ha permesso di identificare quanti componenti principali sono necessari per catturare una certa percentuale della varianza nei dati. Ciò è utile per capire se è possibile ridurre la dimensionalità dei dati (ovvero il numero di feature) mantenendo comunque alte percentuali della varianza totale. Questo procedimento è stato effettuato sui dati in forma grezza e dopo averli pre-processati tramite Min-Max Scaling e Standard Scaling.
Mostriamo di seguito i grafici ottenuti.



(a) Varianza cumulativa dati no-preprocessing.



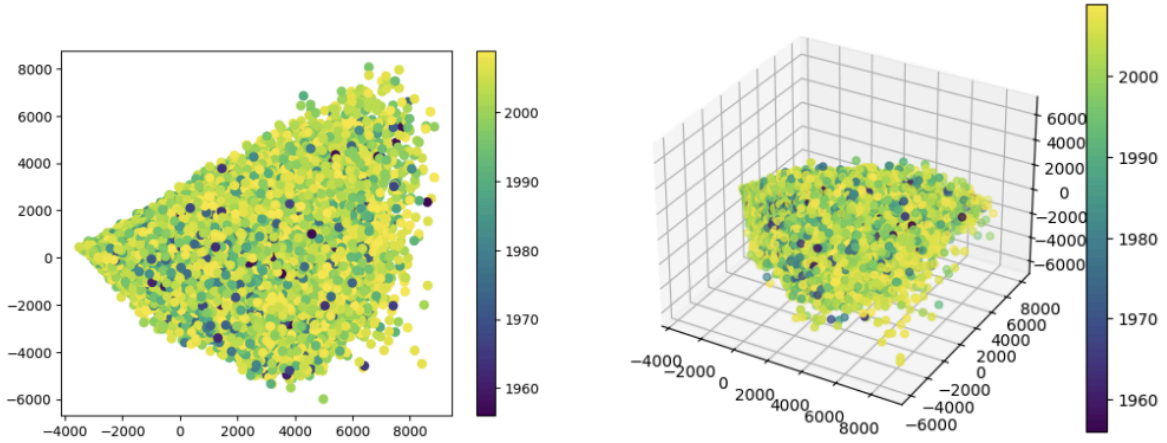
(b) Varianza cumulativa dati Min-Max Scaling



(c) Varianza cumulativa dati Standard Scaling

Figure 2: Varianza cumulativa per numero di componenti

Si è infine proceduto all'identificazione degli outliers calcolando inizialmente le distanze e gli indici dei 10 punti più vicini per ogni punto nel dataset, per poi calcolare una soglia basata sulla distribuzione delle distanze per identificare gli outliers. Si è poi proceduto alla visualizzazione dei dati senza outliers, seguendo lo stesso procedimento applicato in precedenza. Mostriamo di seguito lo scatter plot risultante.



(a) Visualizzazione PCA 2D (b) Visualizzazione PCA 3D

Figure 3: Visualizzazione componenti tramite PCA (senza outliers)

3.2. Preprocessing

Per quanto riguarda la fase del pre-processing, sono state utilizzate diverse tecniche sui dati di addestramento prima dell'uso con i vari modelli, e salvate tramite la libreria *pickle* per poterle successivamente riutilizzare nella fase di Test. Ogni modello utilizza tecniche di pre-processing differenti, scelte in base ai migliori risultati ottenuti dal modello. Di seguito descriviamo tutte le operazioni eseguite e salvate.
La prima operazione di pre-processing effettuata è l'applicazione del Min-Max Scaling, che ha permesso di normalizzare i dati affinché tutti i valori rientrassero nell'intervallo [0,1].
La seconda operazione di pre-processing svolta è l'applicazione dello Standard Scaling. Questo ha permesso di standardizzare i dati in modo che avessero una media zero e una deviazione standard pari a 1.
La terza operazione di pre-processing è stata l'applicazione della norma L2. Questa tecnica modifica i dati in modo tale che la lunghezza (o norma) di ogni vettore di caratteristiche nel dataset sia pari a 1. Ciò si ottiene dividendo ciascun valore di un vettore per la radice quadrata della somma dei quadrati di tutti i valori nel vettore.
Le restanti operazioni di pre-processing riguardano l'applicazione della PCA con un determinato numero di componenti principali. La PCA infatti riduce la dimensionalità del dataset originale, proiettandolo in uno spazio di dimensioni inferiori definito dalle componenti principali. La selezione di

un numero specifico di queste componenti consente di catturare la maggior parte della varianza presente nei dati originali, minimizzando al contempo la perdita di informazioni. Questo processo non solo aiuta a identificare le caratteristiche più significative dei dati, ma migliora anche l'efficienza computazionale e può contribuire a ridurre il rischio di overfitting.
In particolare, abbiamo memorizzato 2 diverse versioni della PCA, utilizzate su modelli differenti: la prima, addestrata su dati precedentemente normalizzati tramite min-max Scaling, con 30 componenti principali; la seconda, anch'essa addestrata su dati precedentemente normalizzati tramite min-max Scaling, con numero di componenti principali pari a 70.
Infine, si è deciso di non rimuovere gli outlier identificati durante la fase di visualizzazione, poiché la loro rimozione peggiorava le prestazioni dei modelli. Si è quindi deciso di mantenere gli outlier nel dataset, permettendo così ai vari modelli di ottenere prestazioni migliori.

3.3. Modeling e tuning degli iperparametri (Training Module)

Dopo aver acquisito, visualizzato e pre-processato i dati è iniziata la fase di modeling e tuning degli iperparametri. In particolare, si è deciso di utilizzare l'oggetto ***Pipeline*** di scikit-learn in combinazione con tecniche di ottimizzazione dei parametri come ***GridSearchCV*** e ***RandomSearchCV***. L'utilizzo della Pipeline in combinazione con le tecniche per l'ottimizzazione dei parametri ha permesso di esplorare le diverse metodologie di pre-processing descritte precedentemente e i diversi iperparametri prevenendo il problema del Data Leakage. In particolare, la *RandomSearchCV* è stata utilizzata nei modelli più lenti da addestrare (come Random Forest e SVM) al fine di ottenere inizialmente una panoramica generale delle combinazioni potenzialmente migliori di parametri. Successivamente, è stata utilizzata la *GridSearchCV* con un insieme ristretto di possibili valori dei parametri per una ricerca più dettagliata. Si è infine proceduto all'addestramento del modello migliore trovato (ovvero con la combinazione di iperparametri migliore), per poi salvarlo tramite la libreria *pickle*. Mostriamo di seguito nel dettaglio il processo di addestramento dei vari modelli richiesti.

3.3.1 Linear Regression

L'esplorazione della modellazione e dell'ottimizzazione degli iperparametri del modello Linear Regressor è proceduta utilizzando direttamente una *GridSearchCV*, data la velocità di addestramento del modello e la mancanza di numerosi iperparametri da esplorare. In particolare, si è deciso di esplorare le tecniche di Min-Max Scaling e Standard Scaling, in combinazione con l'utilizzo delle normalizzazioni L1, L2 e LMAX. Inoltre, si è esplorato il parametro *fit_intercept*.

I migliori parametri individuati sono stati i seguenti:

preprocess: Standardization + Norma L2
fit_intercept: True

3.3.2 Random Forest

L'esplorazione della modellazione e dell'ottimizzazione degli iperparametri del modello Random Forest è iniziata utilizzando una *RandomSearchCV*. In particolare, sono stati esplorati i seguenti iperparametri: *n_estimators*, *max_depth*, *min_samples_split*, *min_samples_leaf*. Inoltre si è deciso di esplorare le tecniche di Min-Max Scaling e Standard Scaling per individuare quella con i risultati migliori. Infine è stato esplorato anche l'utilizzo della PCA con il relativo parametro *n_components*, per comprendere se una riduzione della dimensionalità dei dati avrebbe permesso una migliore generalizzazione da parte del modello.
Dopo aver ottenuto una panoramica generale delle combinazioni potenzialmente migliori, si è proceduto ad utilizzare una *GridSearchCV* per esplorare i valori "vicini" a quelli trovati precedentemente.

Gli iperparametri migliori trovati sono i seguenti:

preprocess: Min-Max Scaling
pca_n_components: 30
n_estimators: 500
max_depth: None
min_samples_split: 3
min_samples_leaf: 8

I restanti iperparametri sono stati mantenuti ai valori di default.

3.3.3 Support Vector Regression

L'esplorazione della modellazione e dell'ottimizzazione degli iperparametri per il modello *Support Vector Regressor* è iniziata utilizzando RandomSearchCV, al fine di ottenere una panoramica delle combinazioni potenzialmente migliori. In particolare, sono stati esplorati i seguenti iperparametri: *C*, *gamma*, *kernel*. Inoltre, sono state esplorate le tecniche di Min-Max Scaling e Standard Scaling e l'utilizzo della PCA. Dopo aver ottenuto una panoramica iniziale dei parametri migliori, si è completata l'esplorazione tramite una *GridSearchCV* dei valori vicini a quelli trovati in precedenza.

I parametri migliori trovati sono i seguenti:

preprocess: Min-Max Scaling
C: *10*
gamma: *10*
kernel: *'rbf'*

I restanti iperparametri sono stati mantenuti ai valori di default.

3.3.4 K-Nearest Neighbors

L'esplorazione della modellazione e dell'ottimizzazione degli iperparametri del modello K-Nearest Neighbors è avvenuta tramite l'utilizzo di *GridSearchCV*. Gli iperparametri esplorati sono: *n_neighbors, weights, metric*. Sono state successivamente esplorate anche le tecniche di Min-Max Scaling e Standard Scaling e l'utilizzo della PCA.

I parametri migliori trovati sono i seguenti:

preprocess: Min-Max Scaling
pca_n_components: 70
n_neighbors: *30*
weights: *distance*
metric: *euclidean*

I restanti iperparametri sono stati mantenuti ai valori di default.

3.3.5 Rete Neurale, architettura Feed Forward

L'implementazione della rete neurale è avvenuta tramite la libreria *PyTorch*. In particolare, prima di procedere con la ricerca degli iperparametri migliori per la rete neurale, è stato necessario implementare alcuni metodi e classi utili alla definizione del Dataset, al Training della rete e al suo Test. Inoltre, è stato utilizzato *TensorBoard*, un insieme di strumenti di visualizzazione e monitoraggio, per tenere traccia dell'evoluzione del modello durante le fasi di training e validation e per identificare quale configurazione forniva risultati migliori.
La prima classe definita è stata '*MyDataset()'*, che permette di trasformare il nostro dataset, rappresentato come array *NumPy,* in Tensori, strutture necessarie per soddisfare i requisiti di PyTorch durante il processo di addestramento. In particolare, i dati X e le label y sono stati trasformati rispettivamente in un Tensore Float e in un Tensore Long.
Successivamente è stata implementata la classe '*FeedForward()'*, che definisce l'architettura della Rete Neurale utilizzata. Poi sono stati implementati i metodi '*train_model()'* e '*test_model()'*, che si occupano rispettivamente dell'addestramento e del test della rete neurale.
Infine, è iniziata l'esplorazione degli iperparametri ottimali per il modello. In particolare, gli iperparametri testati sono stati:

- Architettura della rete neurale (numero di hidden-layer e numero di neuroni per ogni hidden-layer)
- Numero di epoche
- Learning rate
- Batch size
- Ottimizzatore (con relativi iperparametri)
- Dropout

Per ogni combinazione di questi iperparametri, è stato definito un modello con le caratteristiche scelte e successivamente addestrati. I risultati sono stati osservati tramite TensorBoard, per identificare quale architettura otteneva i risultati migliori.

L’architettura che ha ottenuto i risultati migliori è la seguente:

nn.Linear(90, 128),
nn.BatchNorm1d(128),
nn.ReLU(),
nn.Linear(128, 64),
nn.BatchNorm1d(64),
nn.ReLU(),
nn.Linear(64, 128),
nn.BatchNorm1d(128),
nn.ReLU()

Gli iperparametri scelti sono stati:

preprocess: min-max
num_epoch: 100 (con scelta epoca con val-loss migliore)
learning_rate: 0.01
batch_size: 32
optimizer: Adam, betas = (0.999, 0.999)

3.3.6 Rete Neurale, architettura TabNet

L'implementazione della rete TabNet è avvenuta tramite la libreria *PyTorch_tabular*. La scelta degli iperparametri ottimali per il modello è avvenuta tramite l'utilizzo iniziale di una RandomSearch, grazie alla quale si sono compresi i valori intorno ai quali la rete dava dei risultati migliori. E' stata successivamente utilizzata una GridSearch, grazie alla quale si sono esplorati i valori vicini a quelli precedentemente trovati per poter comprendere la combinazione di iperparametri che forniva dei risultati ottimali.

La configurazione finale del modello TabNet presenta i seguenti iperparametri:

preprocess: Standard Scaling
batch size: *32*
max_epochs: 25
n_d: *8*
n_a: *8*
n_steps: 3
gamma: 1.5
n_indipendent: 3
n_shared: 1
learning_rate: 0.01

I restanti iperparametri sono stati mantenuti ai valori di default.

3.3.7 Rete Neurale, architettura TabTransformer

L'implementazione della rete TabTransformer è avvenuta tramite la libreria *PyTorch_tabular*. Anche in questo caso, avendo molti iperparametri da esplorare, la ricerca iniziale di essi è avvenuta tramite l'utilizzo iniziale di una RandomSearch, grazie alla quale si sono compresi i valori intorno ai quali la rete dava dei risultati migliori. E' stata successivamente utilizzata una GridSearch, grazie alla quale si sono esplorati i valori vicini a quelli precedentemente trovati per poter comprendere la combinazione di iperparametri che forniva dei risultati ottimali.

La configurazione finale del modello TabTransformer presenta i seguenti iperparametri:

preprocess: Standard Scaling
batch size: 2048
max_epochs: 100
num_head: 2
num_attn_block: 11
ff_hidden_multiplier: 6
batch_norm_continuous_input: True
transformer_head_dim: 64
transformer_activation: 'GEGLU'
add_norm_dropout: 0.1
attn_dropout: 0.1
ff_dropout: 0.1

I restanti iperparametri sono stati mantenuti ai valori di default.

4. RISULTATI OTTENUTI

Di seguito verranno mostrati i risultati ottenuti dai vari modelli sul Validation-Set.
La valutazione delle prestazioni è stata condotta attraverso un processo sistematico per confrontare i diversi modelli sviluppati e valutare la loro capacità di generalizzazione e precisione nelle previsioni. Sono state utilizzate diverse metriche per avere una visione completa delle performance dei modelli. In particolare, le metriche utilizzate sono state: MSE, MAE, MAPE e R^2 Score.
Il processo di valutazione nel modulo di Test prevede l'importazione dei modelli definiti in precedenza, insieme ai relativi pre-processing applicati sui dati. Questi modelli vengono caricati ed utilizzati per effettuare delle previsioni sul validation set. Questo procedimento viene ripetuto per ciascun modello, consentendo di mostrare i risultati ottenuti per ogni configurazione testata.

I risultati ottenuti dai vari modelli implementati sono i seguenti:

Linear Regression:
- 'MSE': 77.12388430135948
- 'MAE': 6.30986623368177
- 'MAPE': 0.0031666652333647582
- 'R^2 Score': 0.2835460075656998

KNN:
- 'MSE': 74.40238744606964
- 'MAE': 6.269314179420018
- 'MAPE': 0.0031452430798335
- 'R^2 Score': 0.30882776437855564

Random Forest:
- 'MSE': 76.3739501528225
- 'MAE': 6.252239817342722
- 'MAPE': 0.0031378714564361267
- 'R^2 Score': 0.290512634307199

SVR:

‘MSE’: 70.41183485052065
‘MAE’: 5.53360882747824
‘MAPE’: 0.0027802649616192458
‘R^2 Score’: 0.34589860650482196

Rete Neurale Feed Forward:

‘MSE’: 68.86982727050781
‘MAE’: 5.730935573577881
‘MAPE’: 0.0028767965268343687
‘R^2 Score’: 0.3596232533454895

TabNet:

‘MSE’: 68.62740126952995
‘MAE’: 5.7732044175034956
‘MAPE’: 0.002898104313781601
‘R^2 Score’: 0.36247537224886894

TabTransformer:

‘MSE’: 82.50808369580339
‘MAE’: 6.573350120450326
‘MAPE’: 0.0032990011221899654
‘R^2 Score’: 0.23352867263559574

5. CONCLUSIONI

Avendo a disposizione solo una parte del Dataset completo, lo sviluppo e i risultati ottenuti dai vari modelli sono stati in linea con le aspettative. Si è osservato un miglioramento delle prestazioni in linea con l'aumento della complessità del modello utilizzato. Infatti, i modelli più semplici come la regressione lineare hanno fornito risultati iniziali che, pur essendo adeguati, non erano ottimali. Con l'incremento della complessità dei modelli si è assistito a un miglioramento nelle metriche di valutazione (MSE, MAE, MAPE e R^2 Score), che hanno mostrato una maggiore precisione e capacità di generalizzazione da parte dei modelli, a differenza del TabTransformer che non ha ottenuto degli ottimi risultati. L'uso di tecniche di pre-processing come la PCA e la normalizzazione ha ulteriormente contribuito a ottimizzare le performance dei modelli.
Inoltre, l'adozione di strategie di ottimizzazione degli iperparametri come RandomSearch e GridSearch e il monitoraggio delle prestazioni tramite strumenti come TensorBoard ha permesso di ottenere risultati più robusti e affidabili.
In conclusione, anche con un dataset parziale, il processo di sviluppo ha dimostrato che l'aumento della complessità del modello utilizzato ha portato nella maggior parte dei casi a un miglioramento delle prestazioni, confermando le aspettative iniziali.