Alma Mater Studiorum - Università di Bologna
Dipartimento Informatica - Scienza e Ingegneria - DISI
Laurea Magistrale in Informatica
Corso di Data Analytics

Report progetto data analytics

Martina Daghia - 0001097932 - martina.daghia@studio.unibo.it
Martina Zauli - 0001097933 - martina.zauli@studio.unibo.it

4 settembre 2024
Anno scolastico 2023-2024

Indice

1 Introduzione

Questo progetto consiste nel predire l'anno di pubblicazione di una canzone considerando le feature della sua traccia audio.

Questo task di regressione sfrutta tecniche avanzate di machine learning per analizzare un dataset di canzoni, con l'obiettivo di riconoscere l'anno di pubblicazione basandosi su feature audio fornite, identificando la variabile target ovvero l'anno di pubblicazione della canzone.

Il progetto è stato sviluppato seguendo una pipeline di Data Analytics completa. Sono state implementate diverse tecniche includendo modelli tradizionali come Linear Regression, Random Forest, Support Vector Regressor e K-Nearest Neighbors, reti neurali con architettura Feed-Forward e modelli deep specifici per dati tabulari come TabNet e TabTransformer.

La relazione spiega la metodologia applicata per lo sviluppo di questo task, l'implementazione dei modelli e i risultati finali ottenuti.

2 Metodologia

Il progetto sviluppato si è basato su un'analisi di data analytics suddivisa in diverse fasi, a partire dalla data acquisition.

2.1 Data Acquisition

In questa fase, abbiamo scaricato il dataset `train.csv` e lo abbiamo importato nel nostro script. Il dataset risultante è composto da **252.175 righe** e **91 colonne**.

Ogni riga del dataset corrisponde a una singola canzone, mentre le colonne rappresentano le diverse caratteristiche audio estratte per ciascuna traccia, come ad esempio le frequenze, le intensità e altre metriche audio. La tabella seguente mostra un estratto del dataset organizzato per ogni anno (colonna `Year`):

Year	S0	S1	S2	S3	...	S89
2007	44.76752	114.82099	3.83239	27.99928	...	31.32820
2004	52.28942	75.73319	11.35941	-6.20582	...	3.86143
2005	33.81773	6.18222	9.45456	17.85216	...	35.74749
1998	41.60866	3.17811	-3.97174	23.53564	...	3.60432
1987	44.49525	-32.25270	58.08217	3.73684	...	30.11015

Tabella 1: Dataset.

2.2 Data Visualization

Per cominciare, abbiamo studiato la **distribuzione delle canzoni** in base all'anno della loro pubblicazione, ossia ogni anno, quante canzoni sono state pubblicate. Questo è stato utile per visualizzare quanto sia cambiato il volume della produzione musicale nel tempo.

Per fare questo, abbiamo creato un grafico a barre grazie alla libreria Matplotlib dove viene mostrato una variazione nel numero di canzoni nel tempo, con alcuni anni caratterizzati da un numero maggiore di pubblicazioni rispetto ad altri.

Dalla figura 1 si nota come alcuni anni abbiano visto un numero maggiore di pubblicazioni rispetto ad altri, suggerendo potenziali tendenze nella produzione musicale in quel determinato periodo.

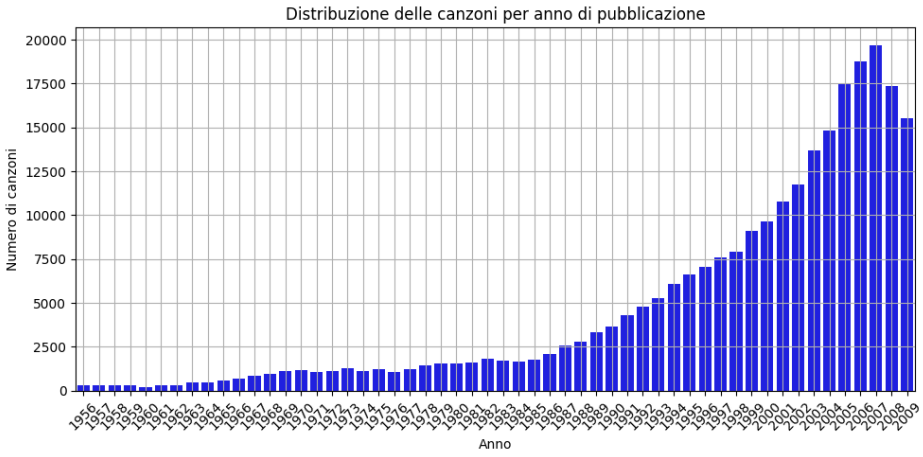

Figura 1: Distribuzione volume canzoni in base agli anni.

Abbiamo deciso di **suddividere i dati per decenni** calcolando il numero di canzoni per decenni. Grazie a questa suddivisione abbiamo notato che il decennio più rappresentativo è quello compreso tra il 2000 e il 2009, con un totale di 139.859 canzoni, seguito dal decennio 1990-1999 con 68.371 canzoni.

Abbiamo creato i **grafici di densità per ogni features audio**, i quali mostrano la distribuzione dei valori delle caratteristiche audio. Inoltre, per ogni grafico vengono segnate tramite una riga verticale la media, la mediana e la moda.

Dai grafici notiamo che è presente una distribuzione gaussiana quasi per tutte le features audio contenute nel nostro dataset. Ci sono alcune features che hanno curve più larghe, come quelle di S2 e S5, indicano una maggiore variabilità dei valori, mentre curve più strette, come in S10 e S11,

indicano dati più concentrati attorno alla media. Alcune features come S13, S14, S15 mostrano code lunghe, indicando la presenza di outliers.

Mentre per quanto riguarda la simmetria, le feature S0, S3 e S4 hanno una distribuzione simmetrica, dove media, mediana e moda sono vicine tra loro. Mentre, ad esempio S2, S5, S12 e S13, si osserva una asimmetria, con una distribuzione che tende verso valori più elevati o più bassi. Questo fenomeno è evidente quando media, mediana e moda non coincidono e la curva è sbilanciata su un lato.

In generale, i boxplot consentono di visualizzare la mediana (la linea orizzontale all'interno del box), l'intervallo interquartile (la larghezza del box che rappresenta il 50% dei dati), e la presenza di outliers (i punti al di fuori dei "baffi"). Il **boxplot** mostrato nella figura 2 si mostra la distribuzione dei valori delle diverse feature nel dataset, raggruppate per anno.

Le features da S13 a S19 presentano una distribuzione positiva molto ampia con numerosi outliers estremamente distanti dai valori centrali. Questo suggerisce una significativa varianza all'interno di questi attributi, ma la mancanza di variazioni negative potrebbe rappresentare un limite per l'apprendimento delle relazioni più complesse nei dati.

Altre feature, come quelle da S0 a S12, mostrano distribuzioni molto più compatte (intorno allo zero), indicando una variabilità bassa che potrebbe non essere rilevante per la predizione dell'anno di pubblicazione delle canzoni.

Infine, le altre features mostrano valori sia positivi che negativi rappresentando una distribuzione più ampia, quindi una maggiore variabilità e un bilanciamento rispetto alle altre features.

Già da questa figura notiamo che sono presenti diversi outliers per ogni features infatti successivamente abbiamo effettuato un'analisi a riguardo.

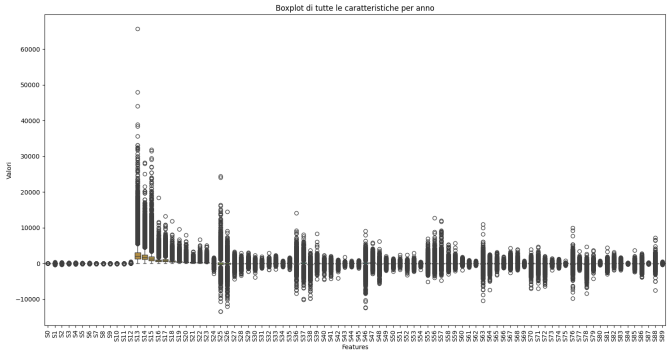

Figura 2: Boxplot dell'intero dataset.

Dopo aver esaminato il boxplot per tutto il dataset, abbiamo deciso per una maggiore visione chiara di effettuare il **boxplot per ciascuna caratteristica audio** del dataset.

In ogni boxplot è presente una linea centrale (la mediana) che divide il box in due parti e rappresenta il valore mediano della feature, con il 50% dei dati che si trova sopra di essa e il 50% sotto. Le dimensioni del box (che rappresenta l'intervallo interquartile, IQR) indicano la dispersione dei dati: un IQR ampio suggerisce una grande variabilità nella feature, mentre un IQR stretto indica una distribuzione più concentrata attorno alla mediana.

Grazie a questi grafici abbiamo notato che ci sono diversi outliers per differenti features e perciò nella fase di pre process abbiamo poi deciso di sostituire gli outliers con la mediana. Ad esempio, nelle features S14, S16, S18, e altre, si osserva una concentrazione di outliers verso l'alto, suggerendo che queste variabili contengono valori estremamente alti rispetto alla maggior parte dei dati.

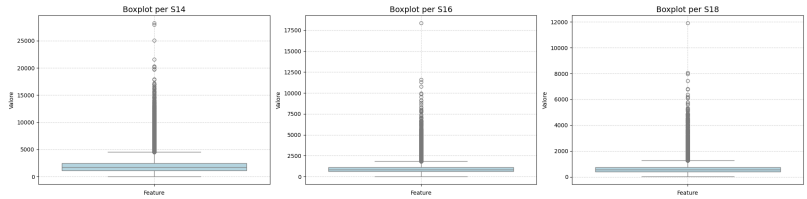

Figura 3: Boxplot singoli delle features S14, S16 e S18.

Inoltre, si nota che alcune features mostrano una distribuzione centrata attorno a zero con un IQR ridotto, mentre altre presentano distribuzioni più ampie con outliers più estremi.

La **matrice di correlazione** mostrata evidenzia la relazione tra le diverse features nel dataset. La correlazione mostrata dalla matrice misura la forza e la direzione della relazione lineare tra due variabili e varia tra -1 e 1, dove:

- 1 indica una correlazione positiva perfetta (quando una variabile aumenta, l'altra aumenta in proporzione);
- -1 indica una correlazione negativa perfetta (quando una variabile aumenta, l'altra diminuisce proporzionalmente);
- 0 indica assenza di correlazione lineare.

Per facilitare l'interpretazione della matrice abbiamo sfruttato una scala di colori ("coolwarm"): i valori uguali a 1 sono rossi mentre quelli vicini a -1 diventano più blu.

Alcuni gruppi di feature mostrano una forte correlazione positiva, evidenziati dai blocchi rossi nella matrice (come il gruppo di features che va da S12 a S23). Ciò indica che queste caratteristiche sono fortemente correlate tra loro e potrebbero essere ridondanti per l'analisi, in quanto potrebbero trasmettere informazioni simili.

Notiamo che la diagonale principale è composta solamente da valori di correlazione pari a 1, poiché una feature è sempre perfettamente correlata con se stessa.

Infine, dalla matrice di correlazione non spiccano altri cluster fortemente correlati, la maggior parte delle features mostra correlazioni deboli o assenti con le altre. Queste features possono essere utili per mantenere una rappresentazione completa del dataset.

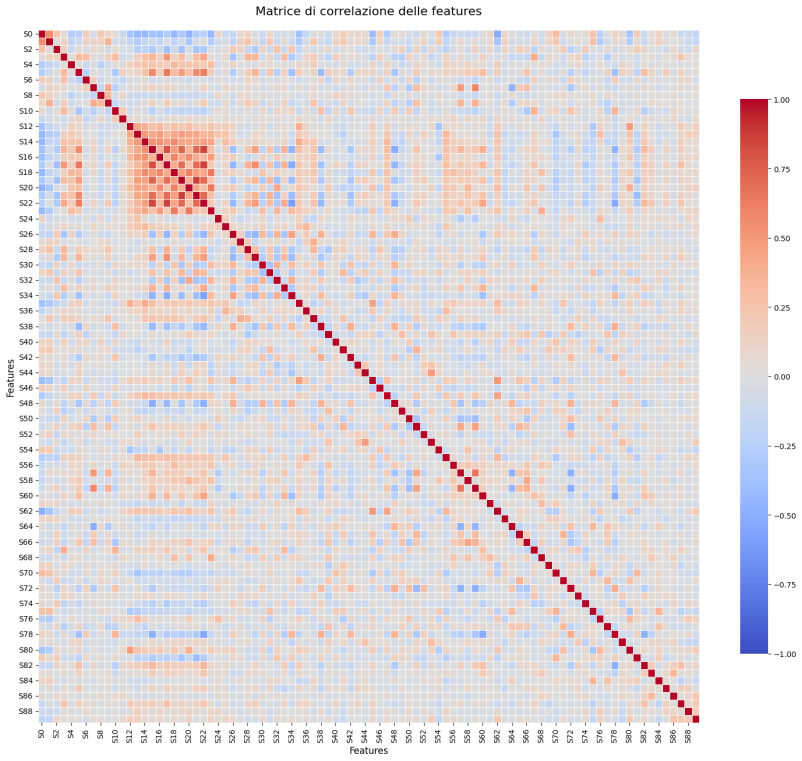

Figura 4: Matrice di correlazione.

2.3 Data Preprocess

Prima di testare i modelli Scikit-learn è stata effettuata la fase di preprocessing del nostro dataset. Il **preprocessing dei dati** è un passaggio fondamentale per garantiere l'integrità e la qualità dei dati.

Inizialmente, abbiamo effettuato una ricerca dei **valori mancanti** nel dataset, che non sono stati rilevati. Tuttavia, abbiamo trovato 52 **duplicati** e proceduto alla loro rimozione. Infine, abbiamo effettuato una verifica per confermare l'eliminazione dei duplicati.

Dati i passaggi precedentemente effettuati e notando i diversi **outliers** presenti abbiamo deciso di **rimuoverli** perché potrebbero influenzare le prestazioni dei modelli. Per trovare e gestire gli outlier abbiamo utilizzato un approccio che si basa sull'analisi del range interquartile (IQR), che

è una tecnica statistica comune per rilevare gli outlier in un dataset. Per ogni anno del dataset, abbiamo calcolato la mediana e i limiti inferiori e superiori degli outlier per ogni variabile; successivamente, gli outlier sono stati sostituiti con la mediana calcolata.

Colonna	**Numero outliers**	**Percentuale outliers**	**Numero nuovi outliers**	**Percentuale nuovi outliers**
S0	5306	2.10%	1492	0.59%
S1	7313	2.90%	2684	1.06%
S2	7484	2.97%	2657	1.05%
...	...	...	...	...
S88	19031	7.55%	9973	3.96%
S89	19330	7.67%	10960	4.35%

Tabella 2: Tabella dei risultati sugli outliers (prima del preprocess) e i nuovi outliers.

2.3.1 Standard Scaler

Per **preparare i dati alla fase di modellazione**, abbiamo utilizzato lo Standard Scaler per standardizzare il dataset. Questo processo è fondamentale per garantire che tutte le features del dataset abbiano una distribuzione uniforme, con media pari a zero e deviazione standard pari a uno. La standardizzazione aiuta a prevenire che variabili con scale diverse influenzino in modo sproporzionato il processo di addestramento dei modelli, assicurando che ogni variabile contribuisca equamente alla costruzione del modello.

Abbiamo applicato la standardizzazione sia al set di addestramento che a quello di test. In particolare, il Standard Scaler è stato addestrato sui dati di addestramento per calcolare la media e la deviazione standard; successivamente, queste statistiche sono state utilizzate per trasformare entrambi i set di dati. Questo approccio garantisce che i dati (le features) siano scalati in modo coerente, evitando il rischio di data leakage e assicurando che il modello possa generalizzare bene su dati non visti.

Standard scaling o Min-Max scaling?

Dopo vari esperimenti, abbiamo scelto di applicare lo Standard Scaler per normalizzare i dati, poiché ha mostrato le migliori prestazioni sui nostri modelli. Questa scelta è stata supportata dalla distribuzione generale dei dati, che si avvicina a una distribuzione normale, rendendo lo Standard Scaler l'opzione più adatta per ottenere una standardizzazione efficace.

Anche se per alcuni modelli la scelta del Min-Max scaling sarebbe potuta essere più opportuna abbiamo deciso di non utilizzarla per due motivi.

Il *primo motivo* riguarda la sensibilità agli outlier da parte del Min-Max Scaling in quanto normalizza i dati mappando i valori minimi e massimi alle estremità dell'intervallo specificato (tipicamente [0, 1]). Se sono presenti outlier che non sono stati eliminati questi potrebbero influenzare la scala dei dati. Invece, lo Standard Scaler riduce l'influenza degli outlier, centrando i dati sulla media e ridimensionandoli in base alla deviazione standard.

Il *secondo motivo* riguarda il fatto che alcuni algoritmi di apprendimento, come i modelli basati sulla regressione lineare o sui SVM, funzionano meglio quando le feature hanno una distribuzione simile a quella normale. A tal proposito lo Standard Scaler è più adatto per trasformare i dati in una forma che si avvicina a questa distribuzione, rendendolo la scelta preferita per questi modelli.

2.3.2 PCA

Abbiamo applicato la Principal Component Analysis (PCA) come parte del nostro processo di preprocessing per ridurre la dimensionalità del dataset, mantenendo il 95% della varianza spiegata. Nonostante l'applicazione della PCA, abbiamo osservato che questa tecnica non ha migliorato le prestazioni dei modelli di regressione testati. In particolare, le metriche di valutazione come l'errore quadratico medio (MSE) e l'errore assoluto medio (MAE) non hanno mostrato miglioramenti significativi rispetto ai risultati ottenuti con i dati standardizzati senza PCA. La riduzione della dimensionalità non contribuiva in maniera importanza a migliorare le performance del modello.

Per questi motivi, abbiamo deciso di non adottare la PCA nei modelli, così da evitare eventuali perdite di informazione.

3 Implementazione

Il progetto si divide in **tre principali funzionalità**, ognuna delle quali esplora diverse tecniche di Machine Learning supervisionato, con l'obiettivo di valutare e confrontare l'efficacia di vari approcci per l'analisi e la previsione sui dati forniti.

Queste funzionalità sono state sviluppate in differenti file:

- File `main.ipynb`: contiene la data acquisition e la data visualization.
- File `train_module.ipynb`: contiene il preprocessamento dei dati e il modeling dei modelli (LR, SVM, RF, KNN).
- File `FFNetwork.ipynb`: contiene lo studio della rete Feed Forward.

- File `TabNet.ipynb`: contiene la rete neurale TabNet.
- File `TabTransformer.ipynb`: contiene la rete neurale TabTransformer basata sui transformer.

La *prima funzionalità* del progetto si concentra sull'applicazione di **tecniche di Machine Learning supervisionato tradizionali**. In questa fase, abbiamo utilizzato quattro modelli: il random forest, la regressione lineare (LR), il support vector machines (SVM) e il k-nearest neighbors (KNN).

La *seconda funzionalità* esplora l'utilizzo di **tecniche di Machine Learning supervisionato basate su reti neurali**, in particolare con un'architettura Feed-Forward. In questa fase, abbiamo implementato e valutato reti neurali Feed-Forward per analizzare i dati e comparare le loro prestazioni rispetto ai modelli tradizionali.

Infine, la *terza funzionalità* si concentra sull'utilizzo di **tecniche di Machine Learning supervisionato avanzate per dati tabulari**, in particolare attraverso modelli deep learning come TabNet e TabTransformer.

3.1 Modelli Scikit-learn

Abbiamo testato diversi tecniche per fare preprocessing. In generale, il dataset è stato suddiviso in un **set di training** (80%) e set di test (20%).

Per la **ricerca della combinazione ottimale di iperparametri** che migliora le performance del modello, abbiamo fatto l'ottimizzazione degli iperparametri. Per farlo, abbiamo utilizzato la tecnica della **Grid Search** che crea una griglia contenenti differenti combinazioni di iperparametri: questa tecnica è molto utile per testare queste combinazioni di iperparametri e selezionare quella che produce le migliori prestazioni durante la validazione incrociata.

Le varie combinazioni sono state valutate tramite la tecnica della validazione incrociata (**cross-validation** con 5 fold): questa tecnica, che divide in diverse parti il dataset, ha permesso di testare il modello su vari sottoinsiemi di dati.

Tra tutte le possibile combinazioni di iperparametri, abbiamo sempre selezionato la **migliore combinazione di iperparametri** che contenesse i migliori risultati e quindi la miglior performance nella media. Questi risultati sono stati valutati dalle seguenti metriche di valutazione: Mean Squared Error (MSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE) e R-squared (R2).

Una volta identificati i migliori iperparametri, l'**addestramento finale** con essi è stato eseguito sull'intero dataset.

Sono stati creati dei file nominati con il nome del relativo modello e salvato con l'estensione `.save`. Questi file contengono l'addestramento del relativo modello che è stato eseguito sull'intero dataset preprocessato e hanno permesso di eseguire il salvataggio del modello.

3.1.1 Linear regression

In un primo momento, per testare l'efficacia del modello di regressione lineare, abbiamo preso in considerazione l'applicazione dell'analisi delle componenti principali (PCA) come tecnica di preprocessing. L'idea era quella di ridurre la dimensionalità dei dati con cui il modello deve lavorare per semplificarne la costruzione e potenzialmente migliorarlo in termini di performance, riducendo così il rischio di overfitting.

Tuttavia, una volta costruito il modello utilizzando la PCA e confrontato i risultati con quelli di un modello costruito con i dati solamente scalati, abbiamo notato migliori prestazioni con il modello sui dati scalati. Questo risultato suggerisce che, in questo caso specifico, mantenere tutte le feature originali, seppur scalate, ha fornito al modello maggiori informazioni utili per predire la variabile target.

Pertanto, abbiamo deciso di procedere con l'addestramento finale del modello di regressione lineare sui dati scalati, evitando la riduzione della dimensionalità tramite PCA. Sono state valutate le prestazioni del modello utilizzando le metriche Mean Squared Error (MSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE) e R-squared (R2).

Metrica	**MSE**	**MAE**	**MAPE**	**R2 Score**
Valore	81.79	6.70	0.00	0.25

Tabella 3: Risultati linear regression con PCA.

Metrica	**MSE**	**MAE**	**MAPE**	**R2 Score**
Valore	79.06	6.51	0.00	0.27

Tabella 4: Risultati linear regression con lo Standard Scaler.

3.1.2 Random forest

Nel processo di ottimizzazione e addestramento della Random Forest, sono stati definiti una serie di parametri iperparametrici da testare tramite un'operazione di Grid Search, con l'obiettivo di migliorare la performance del modello. I parametri considerati sono stati i seguenti:

- `n_estimators`: numero di alberi nella foresta.
- `max_depth`: profondità massima degli alberi.
- `min_samples_split`: numero minimo di campioni richiesti per suddividere un nodo.
- `min_samples_leaf`: numero minimo di campioni richiesti per essere considerati una foglia.

Il primo tentativo è stato fatto senza l'applicazione dello Standard Scaler, ma abbiamo riscontrato scarsi risultati e un tempo di esecuzione eccessivamente lungo. Di conseguenza, è stato deciso di scalare i dati prima di procedere con l'ottimizzazione dei parametri.

Utilizzando la tecnica di Grid Search con Cross-Validation, sono stati valutati diversi set di parametri per identificare la combinazione ottimale che minimizzasse l'errore quadratico medio negativo. Successivamente, il modello ottimizzato è stato utilizzato per fare previsioni sui dati di test e le prestazioni del modello sono state valutate utilizzando una funzione personalizzata.

Una volta determinati i migliori iperparametri, è stato creato e addestrato un modello finale di Random Forest utilizzando l'intero set di dati scalato. Questo modello finale è stato poi salvato su disco (`rf.save`).

I migliori iperparametri ottenuti sono stati:

```
n_estimators=450, max_depth=170, min_samples_split=8, min_samples_leaf=5,
```

Questo modello si è rivelato il più costoso in termini computazionali; abbiamo osservato che, aumentando il numero di alberi, la performance migliorava progressivamente, anche se l'incremento era meno significativo oltre i 400-450 alberi.

Metrica	**n_estimators = 400**	**n_estimators = 450**
MSE	65.3903	65.3022
MAE	5.6363	5.6318
MAPE	0.2829%	0.2826%
R-squared	0.4000	0.4008

Tabella 5: Performance della Random Forest con diversi valori di n_estimators.

3.1.3 SVM

Nel progetto, è stata eseguita un'ottimizzazione del modello Support Vector Regressor (SVR) utilizzando la GridSearch per identificare i migliori iperparametri. La griglia di parametri esplorati includeva diversi valori per ciascun parametro, con lo scopo di minimizzare l'errore quadratico medio negativo attraverso la Cross-Validation.

I parametri ottimali del modello identificati sono stati: `C = 0.4, kernel = 'rbf', gamma = 'scale', ed epsilon = 0.5`. Utilizzando questi parametri ottimizzati, il modello è stato addestrato sui dati di training scalati e successivamente valutato sui dati di test per determinare l'efficacia del modello.

Infine, il modello SVR con i migliori iperparametri è stato riaddestrato utilizzando l'intero dataset scalato e successivamente salvato su disco.

3.1.4 KNN

Infine, per l'ultimo modello è sempre stato svolto un processo di ottimizzazione utilizzando la Grid Search per selezionare i migliori iperparametri. La griglia di ricerca ha esplorato diverse combinazioni di questi parametri con l'obiettivo di minimizzare l'errore assoluto medio negativo attraverso la Cross-Validation.

A differenza della Random Forest, questo modello si basa sulle distanze tra i dati e queste distanze possono essere sensibili alla scalatura dei dati. Per garantire risultati accurati, abbiamo fatto la scalatura del dataset prima di addestrare il modello.

I parametri ottimali individuati sono stati: `n_neighbors = 17‘, ‘weights = 'distance'‘, e ‘metric = 'manhattan'`. Utilizzando questi parametri ottimizzati, il modello è stato addestrato sui dati di training scalati e successivamente testato sui dati di test per valutare le sue prestazioni.

Infine, il modello KNN con i migliori iperparametri è stato riaddestrato utilizzando l'intero dataset scalato e poi salvato su disco.

3.2 Rete Neurale Feed-Forward

In questa parte del progetto abbiamo la Rete Neurale Feed-Forward che risolve un problema di regressione. L'obiettivo è quello di approssimare una funzione che prendendo un input, lo mappa e apprende il valore dei parametri che approssimano meglio la funzione.

Architettura rete

La rete neurale Feedforward progettata è composta da tre livelli principali: un livello di input, due livelli nascosti e un livello di output. Ogni livello nascosto è seguito da una normalizzazione batch, una funzione di attivazione (ReLU) e un'operazione di dropout per ridurre l'overfitting.

Il **tasso di apprendimento** (`learning_rates`) controlla quanto i pesi del modello vengono aggiornati in ogni iterazione dell'ottimizzazione. Un valore troppo alto può causare un'apprendimento instabile, mentre un valore troppo basso può rallentare l'apprendimento. Inizialmente, avremmo voluto testare i valori 0.1 e 0.01: il primo consente aggiornamenti più rapidi dei pesi del modello così che il modello si adatti più velocemente ai dati; il secondo, invece, esegue aggiornamenti più piccoli dei pesi portando ad una progressione più graduale e controllata verso il minimo della funzione di perdita, riducendo il rischio di saltare oltre il minimo globale.

Per la dimensione dei due **layer nascosti** (`hidden layers`) della rete neurale sono stati testati due valori (64 e 128) per esplorare come la dimensione dei layer influisce sulle prestazioni del modello. Dimensioni più grandi possono aumentare la capacità del modello di apprendere rappresentazioni complesse, ma possono anche aumentare il rischio di overfitting.

La scelta delle **funzioni di attivazione** permette alla rete di apprendere nel migliore dei modi in quanto introducono non-linearità nel modello, permettendole di apprendere pattern complessi. Le due funzioni di attivazione più comuni sono la ReLu (Rectified Linear Unit) e la Tanh: la prima aiuta a mitigare il problema del vanishing gradient restituendo 0 per valori negativi e mantenendo i valori positivi invariati, mentre la seconda può avere problemi di vanishing gradient ed è più complessa computazionalmente.

Abbiamo pensato di ridurre il problema dell'overfitting nei seguenti modi.

La prima tecnica usata è quella del **dropout** (`dropout_rate`) che rappresenta la parte di neuroni che vengono casualmente spenti durante l'addestramento: questo aiuta a migliorare la generalizzazione del modello. Un valore di 0.5 significa che il 50% dei neuroni in un layer specifico verranno disattivati casualmente in ogni batch di addestramento.

Inolte, è stata introdotta la regolarizzazione L2 tramite il **weight decay** che penalizza i pesi grandi, stimolando il modello a preferire soluzioni con pesi più piccoli che solitamente tendono ad essere più generalizzabili. Abbiamo assegnato il valori di 0.01 che indica che i pesi vengono penalizzati con un termine di regolarizzazione pari all'1% dei valori dei pesi.

Abbiamo testato la rete con diverse **numero di epoche** e abbiamo notato che troppe poche epoche possono portare a underfitting, al contrario portano ad overfitting senza una regolarizzazione perciò abbiamo scelto come numero di epoche 50.

Come **dimensione del batch** abbiamo scelto 16, il che significa che il modello aggiornerà i pesi dopo aver processato 16 campioni alla volta; questo migliora la capacità di generalizzazione, ma rallenta l'addestramento.

Per prevenire l'addestramento eccessivo del modello e overfitting, abbiamo scelto 10 come valore per la **patience**, che consente di fermare il modello quando non si osservano miglioramenti per 10 epoche consecutive.

Procedura di addestramento

La progettazione della rete comprende la **cross validation a 5 fold** (buon equilibrio tra costo computazionale e l'affidabilità della stima della performance) e la ricerca degli iperparametri per testare diverse configurazioni di iperparametri e di selezionare la combinazione che minimizza l'errore quadratico medio (MSE) sui dati di validazione.

Per evitare l'overfitting, durante l'addestramento è stato introdotto l'**early stopping** che consente al processo di interrompersi in maniera automatica se il modello non migliora dopo un numero specificato di epoche (definito dal parametro `patience`). Abbiamo adottato questa tecnica poiché nelle prove precedenti abbiamo ottenuti dei tempi eccessivamente lunghi e questa tecnica ci ha permesso di migliorarli. Inoltre, per ottimizzare il processo di addestramento, è stato utilizzato uno **scheduler** che riduce il tasso di apprendimento quando la perdita di validazione smette di migliorare, permettendo al modello di convergere più precisamente.

Abbiamo, poi, utilizzato l'**ottimizzatore Adam** (Adaptive Moment Estimation) poichè è conosciuto per essere efficace nel gestire i problemi derivati dai gradienti rumorosi.

Per valutare le performance del modello abbiamo utilizzato la funzione di perdita Mean Squared Error Loss (**MSELoss**) che è adatta ai problemi di regressione e serve per misurare la differenza tra i valori previsti dal modello e i valori reali.

Durante l'addestramento, il modello è stato valutato utilizzando le **metriche di regressione** MAE (Mean Absolute Error), MSE (Mean Squared Error) e R^2 (coefficiente di determinazione) per comprendere la precisione delle previsioni del modello.

Annotazioni

Il nostro obiettivo sarebbe stato quello di effettuare la ricerca degli iperparametri testando più valori ed è quello che abbiamo provato a fare, ma dopo cinque giorni che il modello si stava addestrando sui nostri pc c'è stato un problema tecnico al computer e abbiamo perso tutto l'addestramento. Per il poco tempo rimasto a disposizione abbiamo dovuto procedere con l'addestramento su pochi valori per i parametri.

Abbiamo addestrato due modelli uno con 50 epoche e uno con 23 ma abbiamo notato che il modello migliore è quello con 50 epoche in quanto possiede un Average Validation MSE inferiore anche se entrambi i modelli raggiungevano la loro miglior performance con 19 epoche. Infine abbiamo scelto come miglior numero di epoche 50 per massimizzare la precisione del modello.

I risultati ottenuti sono: `Average Validation MSE: 141.2285`
`Best Hyperparameters: 'learning_rate': 0.1, 'hidden_size1': 64, 'hidden_size2': 128, 'activation_function': 'ReLU', 'dropout_rate': 0.5, 'weight_decay': 0.01, 'best_epoch': 19`

3.3 Tabular

Quest'ultima parte del progetto prevede l'uso di tecniche di Machine Learning supervisionate con modelli deep per dati tabulari, per cui abbiamo sviluppato i modelli TabNet e TabTransform.

Abbiamo utilizzato questi due modelli per affrontare il problema della previsione della variabile `Year` in un dataset complesso.

Siamo partite sviluppando una versione base per entrambi i modelli, dando ai parametri dei valori fissi. Una volta capito il funzionamento, abbiamo proceduto con l'implementazione della Grid Search per svolgere la ricerca dei migliori iperparametri.

Per sviluppare entrambi questi modelli di Tabular Data, abbiamo fatto riferimento a una risorsa disponibile su github, sia per la TabNet [1] che per la TabTransformer [2].

3.3.1 TabNet

TabNet è una rete neurale che migliora le prestazioni grazie alla sua capacità di selezionare quali caratteristiche dei dati sono più rilevanti.

La libreria `pytorch_tabular` ci ha permesso di eseguire la configurazione del modello TabNet, permettendo di definirne facilmente i parametri. L'obiettivo per il compito di regressione era prevedere gli anni, per cui abbiamo fissato la colonna `Year` come target.

Per quanto riguarda la configurazione, il modello è stato impostato utilizzando i seguenti valori: la dimensione del layer di predizione (`n_d`) e la dimensione del layer di attenzione (`n_a`) sono stati fissati a 32 , il numero di steps è stato impostato a 4, gamma a 1.0, il batch size a 256 e il massimo di epoche a 70.

- `Migliori Parametri: 'batch_size': 256, 'gamma': 1.0, 'max_epochs': 70, 'n_a': 32, 'n_d': 32, 'n_steps': 4`

- `Miglior Score: 81.61441676153096`

L'addestramento è stato eseguito su CPU, poiché abbiamo scoperto dopo diversi tentativi che la GPU non era compatibile. Questo probabilmente ha causato un rallenamento nel processo e ci ha impedito di poter portare a termine la ricerca degli iperparametri tramite Grid Search che avevamo implementato in maniera più approfondita.

3.3.2 TabTransformer

TabTransformer, invece, è un modello che combina architetture di transformer con reti neurali per gestire dati tabulari, utilizzando meccanismi di attenzione multi-head. Queste peculiarità è utile per catturare relazioni complesse, sfruttando la capacità del transformer di prestare attenzione a diverse parti dei dati in maniera parallela.

Anche in questo caso, la libreria `pytorch_tabular` ci ha permesso di configurare il modello TabTransformer in modo semplice.

Per la configurazione, il modello è stato costruito impostando il batch size di 128 e un massimo di 100 epoche.

- `Migliori Parametri: 'batch_size': 128, 'max_epochs': 100`
- `Miglior Score: 79.17603057206712`

Anche qui, l'addestramento è stato eseguito su CPU per lo stesso motivo di incompatibilità con GPU.

4 Risultati

I seguenti risultati di prova sono stati calcolati splittando il file `train.csv` in fase di allenamento (80%), e di convalida e test (20%), in modo tale da verificare la bontà dei modelli.

Linear regression

Metrica	**MSE**	**MAE**	**MAPE**	**R2 Score**
Valore	79.06	6.51	0.00	0.27

Tabella 6: Risultati linear regression.

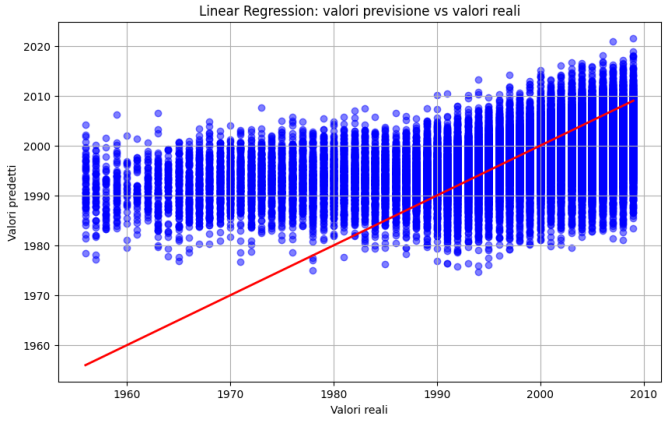

Figura 5: Valori previsione vs valori reali LR.

Random Forest

Metrica	MSE	MAE	MAPE	R-squared
Valore	65.3022	5.6318	0.2826%	0.4008

Tabella 7: Risultati random forest.

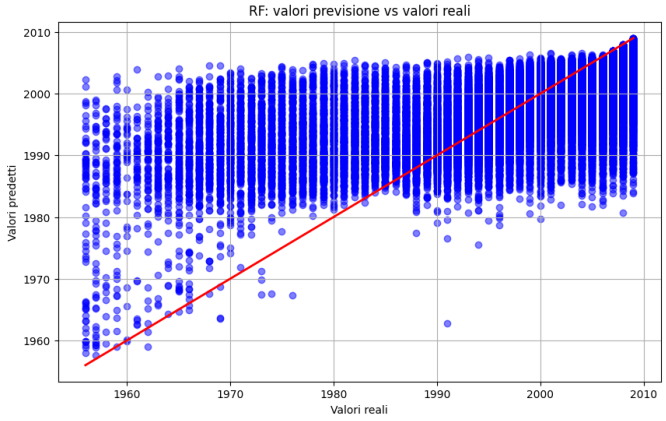

Figura 6: Valori previsione vs valori reali RF.

SVM

Metrica	MSE	MAE	MAPE	R2 Score
Valore	75.1262	5.6833	0.2856	0.3107

Tabella 8: Risultati support vector machine.

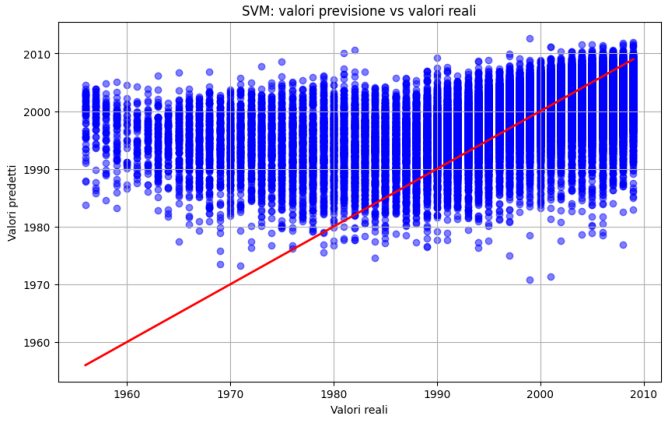

Figura 7: Valori previsione vs valori reali SVM.

KNN

Metrica	MSE	MAE	MAPE	R2 Score
Valore	80.6958	6.7374	0.3379	0.2596

Tabella 9: Risultati k-nearest neighbors.

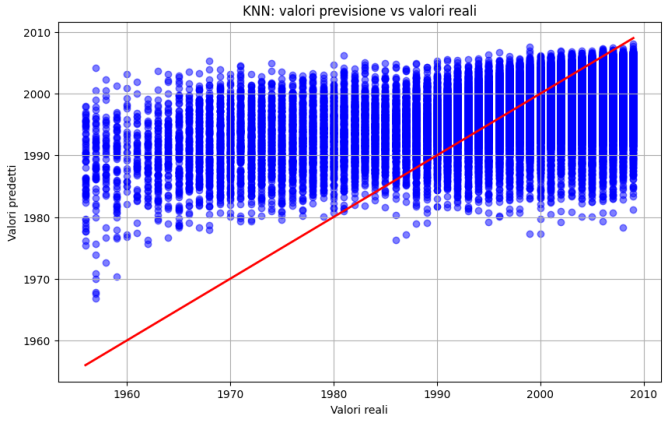

Figura 8: Valori previsione vs valori reali KNN.

Rete Neurale Feed-Forward

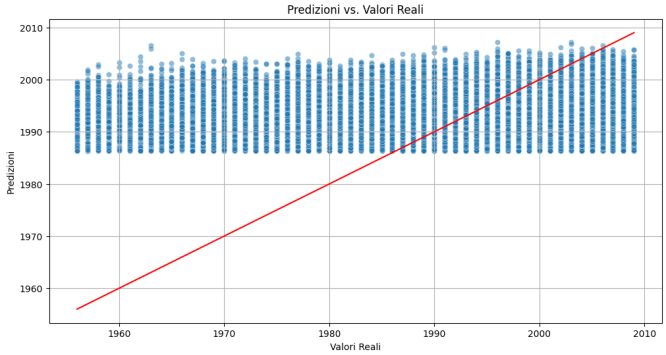

Figura 9: Valori predizione e valori reali della rete neurale.

In questo grafico di dispersione è possibile vedere il confronto tra valori previsti e valori reali. La linea rossa diagonale rappresenta la perfetta corrispondenza tra le previsioni e i valori reali, fornendo un'indicazione visiva della precisione del modello.

TabNet

Test metric	DataLoader 0
test_loss	97.62861633300781
test_mean_absolute_error	7.508788108825684
test_mean_absolute_percentage_error	0.0037667506840080023
test_mean_squared_error	97.62861633300781
test_r2_score	0.10504426807165146

Figura 10: Risultati metriche TabNet.

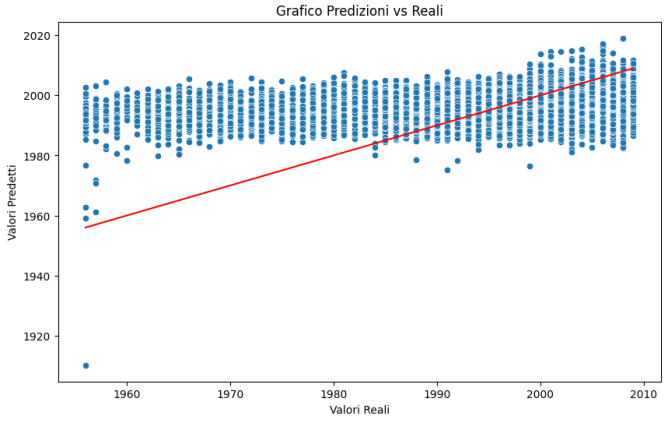

Figura 11: Previsione vs reali TabNet.

TabTrasformer

Test metric	DataLoader 0
test_loss	79.19596099853516
test_mean_absolute_error	6.5241169929504395
test_mean_absolute_percentage_error	0.0032740216702222824
test_mean_squared_error	79.19596099853516
test_r2_score	0.2634124457836151

Figura 12: Risultati metriche TabTrasformer.

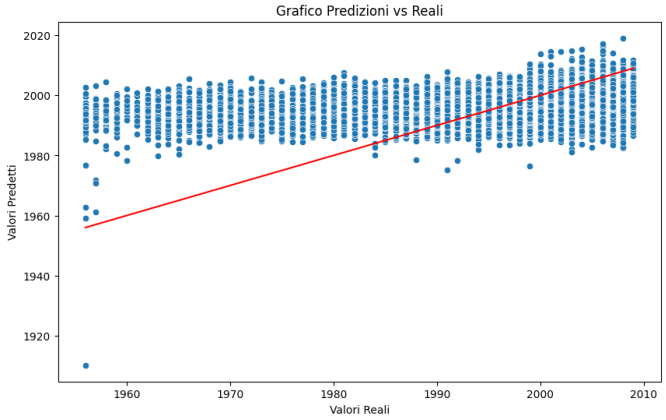

Figura 13: Previsione vs reali TabTrasformer.

5 Limitazioni

Nel corso dello sviluppo del progetto, si sono presentate diverse sfide che hanno influenzato le scelte progettuali finali. In particolare, la complessità e i tempi richiesti dall'esecuzione delle tre principali funzionalità descritte nell'assignment sono stati superiori alle aspettative.

Le limitazioni dei nostri hardware e il tempo ci hanno impedito di eseguire analisi più approfondite e di costruire modelli migliori come previsto nel progetto. Le problematiche legate ai tempi di esecuzione e alla gestione delle risorse ci hanno costretto ad adottare un approccio semplificato, non avendo tempo sufficiente per approfondire.

Annotazioni

Il nostro lavoro è disponibile nella repository pubblica di **GitHub** [3].

Bibliografia

[1] Tabnet. https://github.com/manujosephv/pytorch_tabular/tree/main/src/pytorch_tabular/models/tabnet.

[2] Tabtransformer. https://github.com/manujosephv/pytorch_tabular/tree/main/src/pytorch_tabular/models/tab_transformer.

[3] Repository del progetto su github, https://github.com/martinadaghia/Project_DataAnalytics, 2024.