Università degli studi di Bari Aldo Moro

DIPARTIMENTO DI INFORMATICA
Corso di Laurea Triennale in Informatica

CASO DI STUDIO DEL CORSO IN INGEGNERIA DELLA CONOSCENZA

SISTEMA DI PREDIZIONE DELL'ENERGIA PRODOTTA DA IMPIANTI EOLICI

Repository GitHub: https://github.com/PalmDomenico/ICON24-25

Autore: Palmisano Domenico 778538 d.palmisano34@studenti.uniba.it

Indice

1. Introduzione

Obiettivo

L'obiettivo del sistema è quello di **prevedere la quantità di energia prodotta da un impianto eolico.**
La parte di predizione dell'energia prodotta viene supportata da 2 elementi fondamentali:

- **Una KB**
- **Apprendimento non supervisionato con clustering**

Il modello viene addestrato non solo per prevedere la produzione di energia a breve termine ma anche a lungo termine, consentendo una stima accurata della potenza che sarà prodotta in determinati intervalli di tempo. Questo strumento è fondamentale per **ottimizzare la gestione dell'impianto**, migliorare l'affidabilità delle operazioni e supportare decisioni strategiche legate alla pianificazione energetica.

Dataset di riferimento

Il dataset principale utilizzato è →
https://www.kaggle.com/datasets/mubashirrahim/wind-power-generation-data-forecasting

Il dataset è strutturato nel seguente modo:

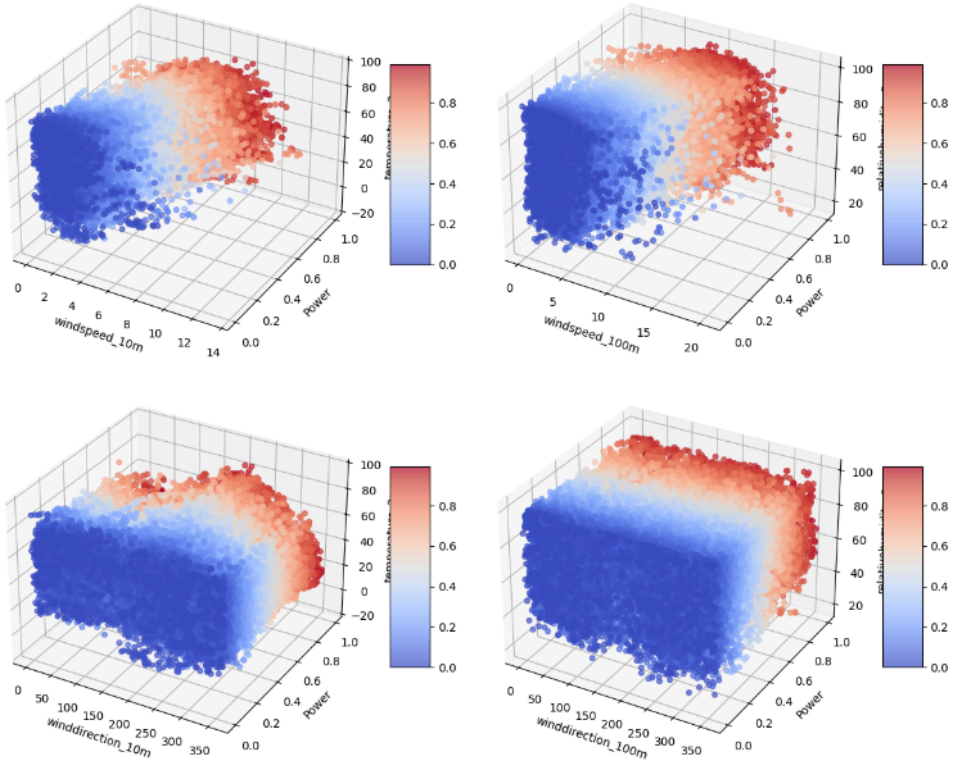
- **time:** Indica il momento della misurazione.
- **temperature_2m**: Temperatura dell'aria misurata a 2 metri dal suolo (°C).
- **relativehumidity_2m**: Umidità relativa dell'aria a 2 metri dal suolo (%).
- **dewpoint_2m**: Punto di rugiada a 2 metri dal suolo (°C), ovvero la temperatura alla quale l'aria diventa satura di umidità.
- **windspeed_10m**: Velocità del vento a 10 metri dal suolo (m/s).
- **windspeed_100m**: Velocità del vento a 100 metri dal suolo (m/s) (altezza delle turbine eoliche)
- **winddirection_10m**: Direzione del vento a 10 metri dal suolo (°), misurata in gradi rispetto al nord.
- **winddirection_100m**: Direzione del vento a 100 metri dal suolo (°).
- **windgusts_10m**: Raffiche di vento a 10 metri dal suolo (m/s).
- **power**: Potenza prodotta dall'impianto eolico

1.1. Grafico 3D a dispersione

Il grafico 3D a dispersione è progettato per visualizzare la **relazione tra tre variabili** contemporaneamente, facilitando la comprensione dell'**influenza di diverse condizioni atmosferiche e del vento sulla potenza prodotta** da un impianto eolico.
Per leggere il grafico, si devono osservare tre aspetti principali: l'asse X, l'asse Y e l'asse Z.

- **L'asse X** rappresenta una variabile legata al vento, come la velocità o la direzione del vento, e mostra come queste caratteristiche influenzano la potenza.
- **L'asse Y** rappresenta la potenza prodotta, che cambia in base ai valori delle altre due variabili.
- **L'asse Z**, mostra una variabile ambientale, come la temperatura o l'umidità, e il suo impatto sulla produzione di energia.

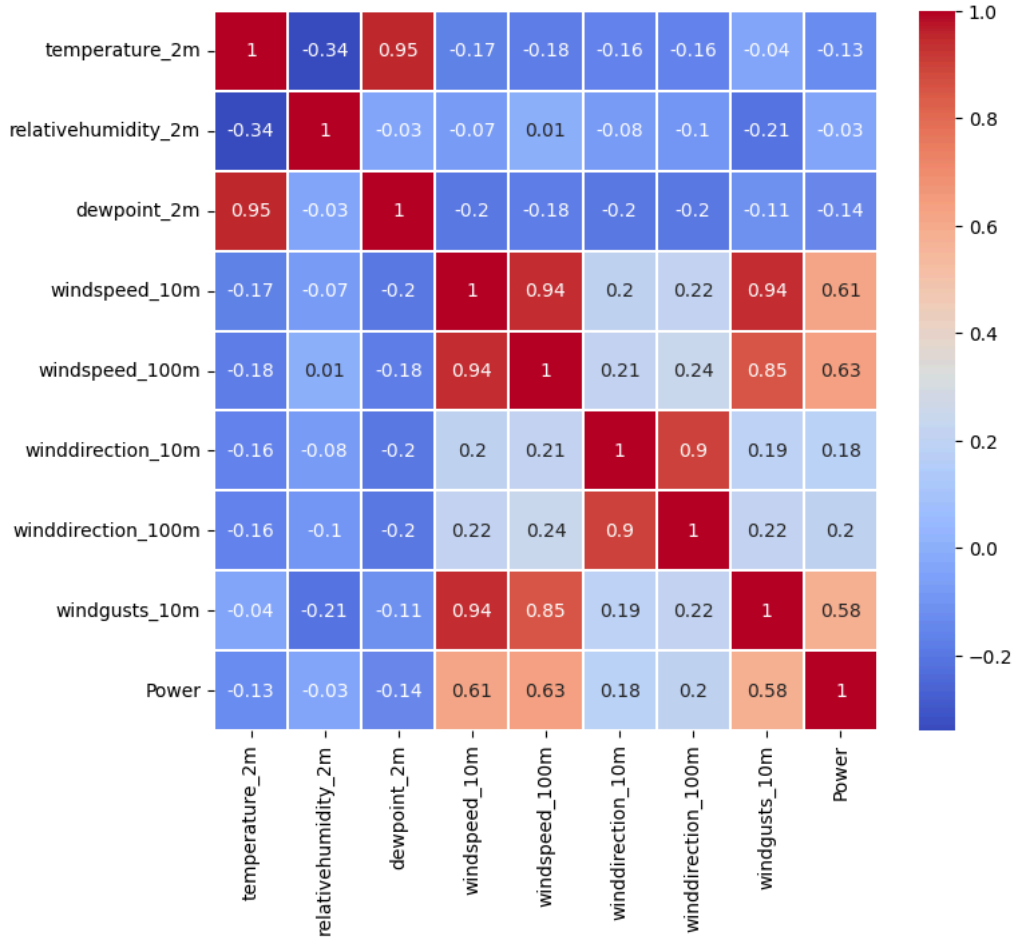
Ogni punto nel grafico rappresenta una combinazione di valori di queste variabili, e il colore dei punti indica **l'intensità della potenza prodotta**, con una scala che varia dal blu (bassa potenza) al rosso (alta potenza). Per interpretare il grafico, è importante osservare come i cambiamenti nella velocità del vento, nella direzione del vento o nelle condizioni atmosferiche influenzano la potenza, identificando eventuali tendenze o pattern che possono aiutare a ottimizzare la gestione dell'impianto eolico.



1.2 Matrice di Correlazione

La heatmap rappresenta la **matrice di correlazione** tra le variabili del dataset, fornendo una visione immediata delle relazioni tra di esse. La correlazione misura il grado di dipendenza lineare tra due variabili, con valori compresi tra -1 e +1:

- **+1: Correlazione positiva perfetta** → quando una variabile aumenta, anche l'altra aumenta.
- **0 Nessuna correlazione lineare.**
- **-1 Correlazione negativa perfetta** → quando una variabile aumenta, l'altra diminuisce.



Correlazione tra la Potenza e le altre variabili:

- **windspeed_100m (0.63)** → Forte correlazione positiva
    - La velocità del vento a 100 m ha un impatto significativo sulla potenza generata.
- **windspeed_10m (0.61)** → Forte correlazione positiva
    - Leggermente inferiore rispetto a windspeed_100m.
- **windgusts_10m (0.58)** → Correlazione positiva.

    - Le raffiche di vento influenzano la potenza, meno rispetto alla velocità media.
- **temperature_2m (-0.13)** → Correlazione trascurabile.
    - La temperatura non influisce molto sulla potenza prodotta.
- **dewpoint_2m (-0.14)** → Correlazione trascurabile.
- **relativehumidity_2m (-0.14)** → Correlazione trascurabile.

Analizzando il grafico, possiamo osservare non solo la correlazione tra la potenza prodotta e le altre variabili, ma anche le relazioni tra le variabili meteorologiche stesse.

2. Ragionamento logico e Prolog

Il ragionamento logico si differenzia da quello probabilistico per il fatto che opera sempre in un contesto deterministico. Il ragionamento logico sfrutta la **logica** matematica. Nel ragionamento logico si costruisce una **Knowledge Base (KB)**, ovvero una base di conoscenza composta da **assiomi**, che rappresentano affermazioni considerate sempre vere.

Gli assiomi si dividono in:

- **Fatti**: rappresentano la conoscenza primitiva.
- **Regole**: permettono di derivare conoscenza derivata, che si ottiene a partire da osservazioni o inferenze.

Nel progetto viene utilizzato Prolog, esso è un linguaggio di programmazione dichiarativo basato sul ragionamento logico.

Fatti

All'interno del progetto sono stati usati fatti con **forma modulare, atomica e predicativa**:

- temperature_2m
- relativehumidity_2m
- dewpoint_2m
- windspeed_10m
- windspeed_100m
- winddirection_10m
- winddirection_100m
- windgusts_10m
- power
- cluster

ESEMPIO

```
temperature_2m(r0, 22.7).
temperature_2m(r1, 22.0).
temperature_2m(r2, 21.7).
temperature_2m(r3, 21.7).
temperature_2m(r4, 22.4).
temperature_2m(r5, 22.8).
temperature_2m(r6, 23.8).
temperature_2m(r7, 26.8).
temperature_2m(r8, 28.5).
temperature_2m(r9, 29.8).
temperature_2m(r10, 31.1).
```

Regole

Lo scopo di usare questa KB è quella di **derivare nuove informazioni** a partire da fatti esistenti, questo è possibile attraverso l'uso di regole.
Dal momento che il nostro scopo è quello di predire la quantità di energia prodotta, sono state implementate **regole utili per fornirci nuove informazioni legate alla produzione di energia.**

Le regole implementate nella KB sono:

- **weak_wind(X)** → La velocità del vento a 100 metri in X è debole se è inferiore a 4 m/s.
- **optimal_wind(X)** → La velocità del vento a 100 metri in X è considerata ottimale se è compresa tra 8 e 14 m/s.
- **critical_humidity(X)** → L'umidità relativa a 2 metri in X è critica se supera il 90%.
- **strong_gusts(X)** → In X, ci sono raffiche di vento forti a 10 metri se superano i 10 m/s.
- **stable_conditions(X)** → Le condizioni sono stabili in X se:
  - il vento è ottimale,
  - non c'è umidità critica,
  - non ci sono raffiche forti.
- **low_expected_production(X)** → Possibile bassa produzione, dovuta ad un vento troppo debole o condizioni atmosferiche sfavorevoli

```
% Il vento è debole (< 4 m/s), spesso insufficiente per far girare le turbine in modo efficiente.
weak_wind(X) :-
    windspeed_100m(X, V),
    V < 4.

% Il vento è ottimale (8 e 14 m/s), ottimale per il funzionamento delle turbine con lo scopo di massimizzare la produzione.
optimal_wind(X) :-
    windspeed_100m(X, V),
    V >= 8,
    V =< 14.

% Umidità critica (> 90%), può indicare condizioni atmosferiche instabili (nebbia, condensa, pioggia) esse influenzano le prestazioni.
critical_humidity(X) :-
    relativehumidity_2m(X, H),
    H > 90.

% Raffiche forti (> 10 m/s).
strong_gusts(X) :-
    windgusts_10m(X, G),
    G > 10.


% Vento ottimale, assenza di raffiche e umidità elevata sono condizioni stabili che indicano una produzione continua e prevedibile.
stable_conditions(X) :-
    optimal_wind(X),
    \+ critical_humidity(X),
    \+ strong_gusts(X).
```

```
% Possibile bassa produzione, dovuta ad un vento troppo debole o condizioni atmosferiche sfavorevoli
low_expected_production(X) :-
    weak_wind(X);
    (strong_gusts(X), critical_humidity(X)).


meteo_info(RowId,
    Temp,
    RH,
    Dew,
    Wind10,
    Wind100,
    Dir10,
    Dir100,
    Gust,
    Power,
    Cluster) :-

    temperature_2m(RowId, Temp),
    relativehumidity_2m(RowId, RH),
    dewpoint_2m(RowId, Dew),
    windspeed_10m(RowId, Wind10),
    windspeed_100m(RowId, Wind100),
    winddirection_10m(RowId, Dir10),
    winddirection_100m(RowId, Dir100),
    windgusts_10m(RowId, Gust),
    power(RowId, Power),
    cluster(RowId, Cluster).
```

```
def integrate_logical_features(csv_path, prolog_file):
    df = pd.read_csv(csv_path)
    num_rows = df.shape[0]

    features = {
        'weak_wind': query_prolog(prolog_file, predicate: 'weak_wind'),
        'optimal_wind': query_prolog(prolog_file, predicate: 'optimal_wind'),
        'critical_humidity': query_prolog(prolog_file, predicate: 'critical_humidity'),
        'strong_gusts': query_prolog(prolog_file, predicate: 'strong_gusts'),
        'stable_conditions': query_prolog(prolog_file, predicate: 'stable_conditions'),
        'low_expected_production': query_prolog(prolog_file, predicate: 'low_expected_production'),
    }

    for feature_name, matching_ids in features.items():
        df[feature_name] = [1 if f"r{i}" in matching_ids else 0 for i in range(num_rows)]

    df.to_csv(csv_path, index=False)
```

Come possiamo vedere le regole non vengono lasciate a sé, ma si interagisce con la KB per trarre nuove informazioni dai dati e successivamente arricchire il dataset per ottimizzare i successivi train.

3. Apprendimento non supervisionato

Come abbiamo appreso nelle sezioni precedenti, determinate variabili influenzano **direttamente il livello di produzione dell'impianto.** Per questo ci viene in supporto l'apprendimento non supervisionato. Attraverso esso riusciremo a vedere come un **modello (KMeans) riesca a riconoscere condizioni meteo simili fra loro**, raggrupparle e associare ad essi diversi livelli di produzione.
L'apprendimento non supervisionato è una branca dell'apprendimento automatico in cui l'agente viene addestrato su un insieme di dati senza etichette oppure senza un target. Esistono due principali tipi di apprendimento non supervisionato con diversi obiettivi:

- **Clustering** → L'obiettivo è raggruppare gli elementi del dataset in base a delle somiglianze.
- **Riduzione della dimensionalità** → L'obiettivo è ridurre il numero di feature utilizzate mantenendo però le informazioni più significative. Un esempio è la PCA (Principal Component Analysis).

3.1. Clustering

Attraverso il clustering andremo a raggruppare in *n* cluster condizioni meteo simili fra loro.
Esistono due tipologie di apprendimento di clustering:

- **hard clustering** → Associa ogni esempio a un cluster specifico.
- **soft clustering** → Associa a ogni esempio la probabilità di appartenenza a ogni cluster.

Le due tipologie di clustering vengono messe a confronto. Esse saranno confrontate andando a confrontare i grafici che analizzano la distribuzione della potenza per cluster, ma anche tramite una riduzione dimensionale tramite PCA per valutare visivamente la separazione tra i cluster.

Scopo

Lo scopo principale di usare tecniche di clustering è quello di **trovare condizioni meteo simili fra loro** e successivamente scovare relazioni fra alcune condizioni meteo e la produzione di energia.
Andiamo a vedere come soft e hard clustering si comportano e andremo a trarre delle conclusioni su quella che è la tecnica migliore da usare in questo caso.
Sia per l'hard clustering che per il soft clustering sono stati seguiti i seguenti step:

- Normalizzazione dei dati.

```
scaler = StandardScaler()
features = ['temperature_2m', 'relativehumidity_2m', 'dewpoint_2m',
            'windspeed_10m', 'windspeed_100m',
            'winddirection_10m', 'winddirection_100m',
            'windgusts_10m', 'Power']
X = dataset[features]
X_scaled = scaler.fit_transform(X)
```

    - Per fare ciò viene usata la libreria sklearn.
- Ricerca del numero di cluster ottimale da utilizzare.
- Allenamento del modello.

3.1.1. Hard Clustering

L'hard clustering è una tecnica di clustering in cui ogni punto del dataset viene assegnato in **modo esclusivo a un solo cluster**. Ad esempio l'algoritmo KMeans assegna a ciascun punto il centroide più vicino. In questo caso appunto viene usato l'algoritmo KMeans.

Uno dei problemi principali del KMeans è **trovare il numero ideale di cluster** da fornire in input all'algoritmo. Per risolvere questo problema è stata usata una strategia nota come "curva del gomito". La curva del gomito mostra la **variazione dell'inertia al variare del numero di cluster**, dove l'inertia rappresenta la somma delle distanze quadrate tra ogni punto dati e il centro del cluster assegnato.
L'algoritmo prevede di eseguire KMeans per diversi valori del numero di cluster sulla quale viene calcolata l'**inertia**. L'obiettivo è identificare il cosiddetto **"gomito"** (*elbow point*), ovvero il punto a partire dal quale l'incremento del numero di cluster comporta una riduzione dell'inertia sempre meno significativa. Questo punto rappresenta una buona scelta per il numero ottimale di cluster.

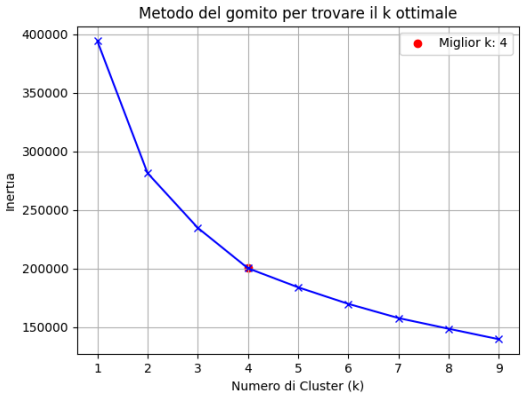
```
iterate = []
for i in range(1, maxK):
    kmeans = KMeans(n_clusters=i, n_init=10, init='random')
    kmeans.fit(dataSet)
    iterate.append(kmeans.inertia_)

kl = KneeLocator(range(1, maxK), iterate, curve="convex", direction="decreasing")
```

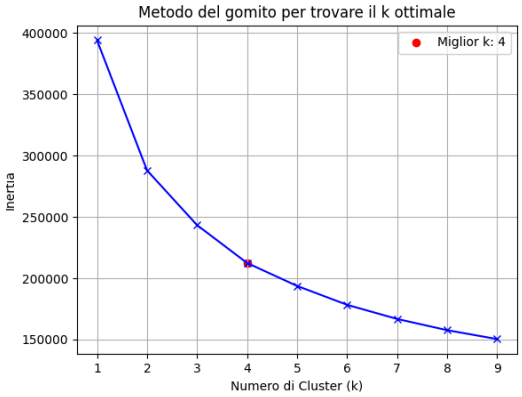
PARAMETRI:

- L'algoritmo viene eseguito con 10 valori di cluster, che variano da 1 a 10.
- Il parametro *n_init* indica quante volte verrà eseguito l'algoritmo sul cluster i, il quale una volta concluso restituirà il risultato migliore degli n_init allenamenti.
- Il parametro *init='random'* significa che i centroidi vengono inizializzati in maniera del tutto casuale.

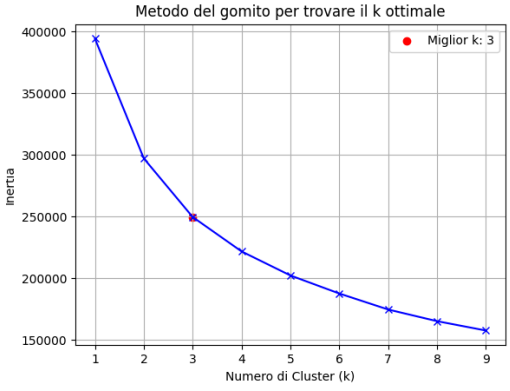
L'algoritmo è stato testato su diversi dataset simili nella forma ma contenente dati diversi.
Una volta eseguito l'algoritmo otteniamo un grafico che ci indica il numero ideale di cluster da utilizzare per quel specifico dataset.



Dataset 1.



Dataset 2.



Dataset 3.

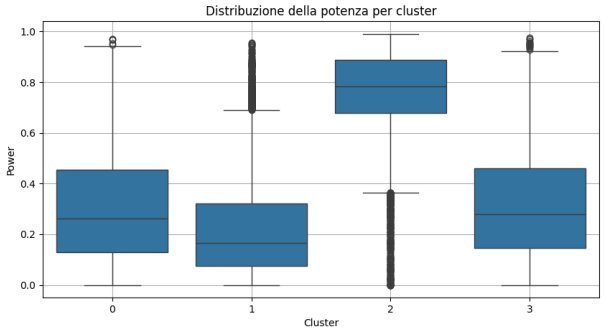
Conclusa la parte precedente alleniamo l'algoritmo KMeans sul relativo k trovato per i differenti dataset. Come possiamo notare sui diversi dataset viene trovato un numero di cluster abbastanza simile.

Per analizzare la **correlazione che c'è fra determinate condizioni meteo e potenza prodotta**, analizziamo i seguenti grafici che mostrano come i diversi cluster hanno range di potenza differenti.

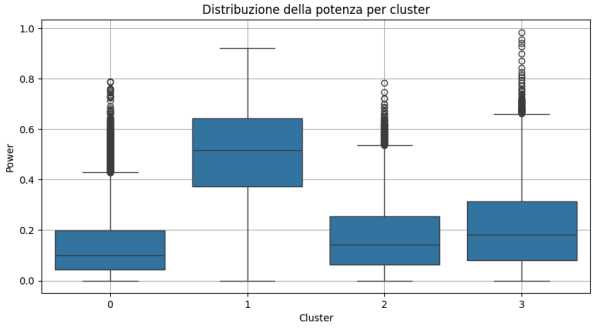
Esempio grafico 1:

- **Cluster 2** → ha la potenza mediamente più alta, con valori centrali attorno a 0.8–0.9.
- **Cluster 1** → presenta la potenza più bassa, con mediana intorno a 0.15.

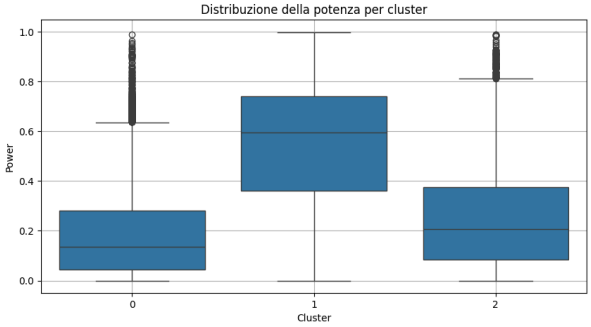
- **Cluster 0 e Cluster 3** → mostrano distribuzioni simili, con mediane vicine a 0.3 e una maggiore dispersione.



Dataset 1.



Dataset 2.



Dataset 3.

Possiamo notare come anche se ci sono cluster con range di potenza simile fra loro, c'è sempre un cluster che si differenzia molto dagli altri. In conclusione possiamo dire che ci sono determinate condizioni meteo che sono favorevoli alla produzione di energia e altre meno favorevoli, che non sempre è dovuta ad un assenza totale di vento ma potrebbe derivare anche da condizioni instabili o fattori simili.

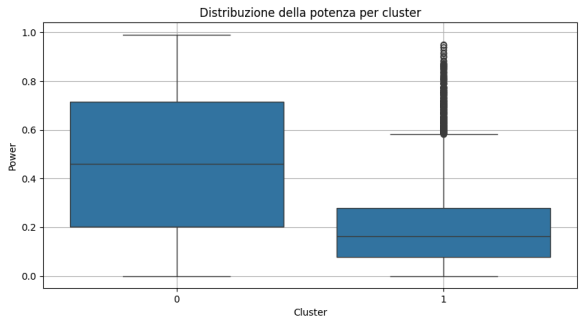
3.1.2 Soft clustering

Il soft clustering è una tecnica in cui ogni punto può appartenere a più cluster con un certo grado di probabilità. Invece di un'assegnazione netta, come nell'hard clustering, si lavora con appartenenze parziali. Un esempio tipico è il Gaussian Mixture Model (GMM), che calcola la probabilità che un punto appartenga a ciascun cluster in base alla distribuzione statistica.

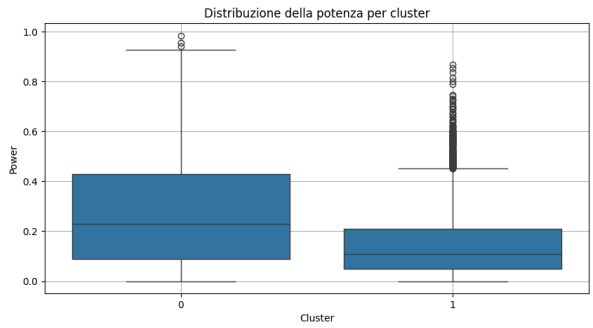
Come per l'hard clustering, anche qui è necessario ricercare il miglior numero di cluster da utilizzare.

```python
def research_n_clusters_gmm(X_scaled, max_k=10):
    best_k = 2
    best_score = -1
    for k in range(2, max_k + 1):
        gmm = GaussianMixture(n_components=k, n_init=10, random_state=0)
        labels = gmm.fit_predict(X_scaled)
        score = silhouette_score(X_scaled, labels)
        if score > best_score:
            best_k = k
            best_score = score
    return best_k
```

Questa funzione prova valori di k da 2 a max_k ed esegue il clustering per ciascuno, calcola il silhouette score e restituisce il k con il punteggio migliore. La **silhouette score** è una metrica che valuta la qualità di un clustering.
Conclusa la parte precedente alleniamo l'algoritmo Gaussian Mixture Model sul rispettivo numero di cluster trovati.



Dataset 1.



Dataset 2.

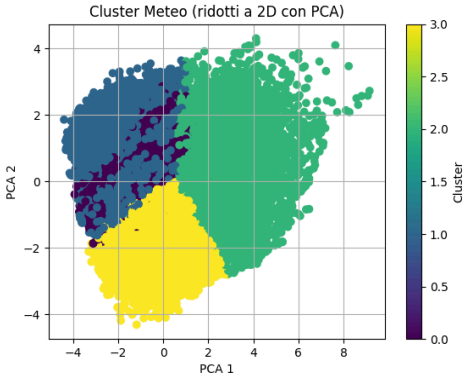
In questo caso vediamo come in entrambi i test i 2 cluster sono parzialmente sovrapposti e non c'è un cluster che si differenzia molto dagli altri, come succede nel hard clustering.

3.2. Riduzione della dimensionalità

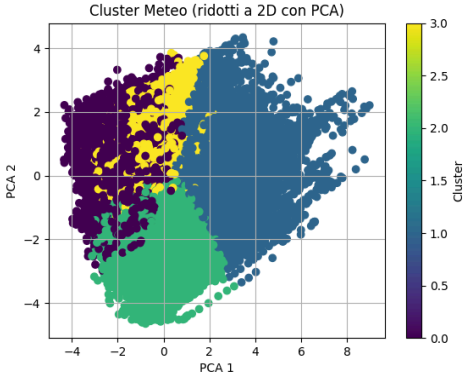
La **Principal Component Analysis (PCA)** è una tecnica di riduzione dimensionale che consente di trasformare un dataset ad alta dimensionalità in un nuovo spazio con meno variabili

(componenti principali), mantenendo la maggior parte della varianza informativa. Nel nostro caso, i dati meteorologici sono stati proiettati su due componenti principali per facilitarne la visualizzazione.

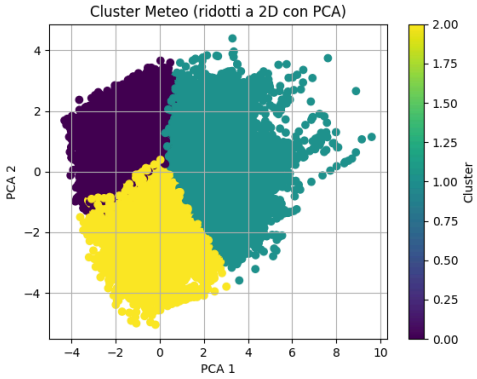
KMeans



Dataset 1.



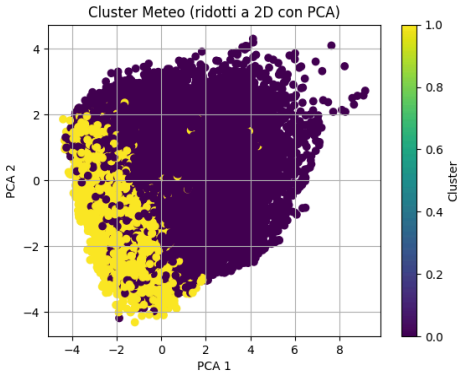
Dataset 2.



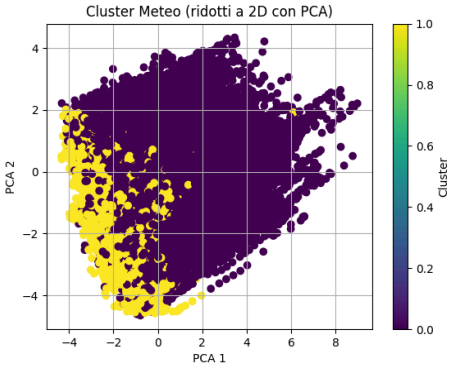
Dataset 3.

I grafici mostrano una chiara separazione tra i cluster nello spazio 2D delle componenti principali, indicando che i gruppi individuati rappresentano **condizioni meteorologiche ben distinte**. La distribuzione compatta e ordinata dei punti conferma l'efficacia del clustering.

GMM



Dataset 1.



Dataset 2.

I grafici mostrano una **separazione meno netta** tra i cluster nello spazio 2D delle componenti principali, con una significativa sovrapposizione tra i gruppi. Questo indica che i gruppi individuati rappresentano **condizioni meteorologiche che si sovrappongono.**.

3.3. Conclusione

Lo scopo principale per la quale è stato usato il clustering è quello di aggiungere informazioni utili al dataset. In conclusione viene scelto l'hard clustering dal momento che esso ha mostrato una migliore adesione ai dati ed un'individuazione più netta di condizioni meteo simili fra essi, ovvero una distinzione abbastanza definita fra i vari cluster. Il dataset originale viene quindi arricchito di informazioni, i quali saranno utili a raggiungere prestazioni migliori ed individuare più facilmente pattern all'interno dei dati.

4. Apprendimento supervisionato

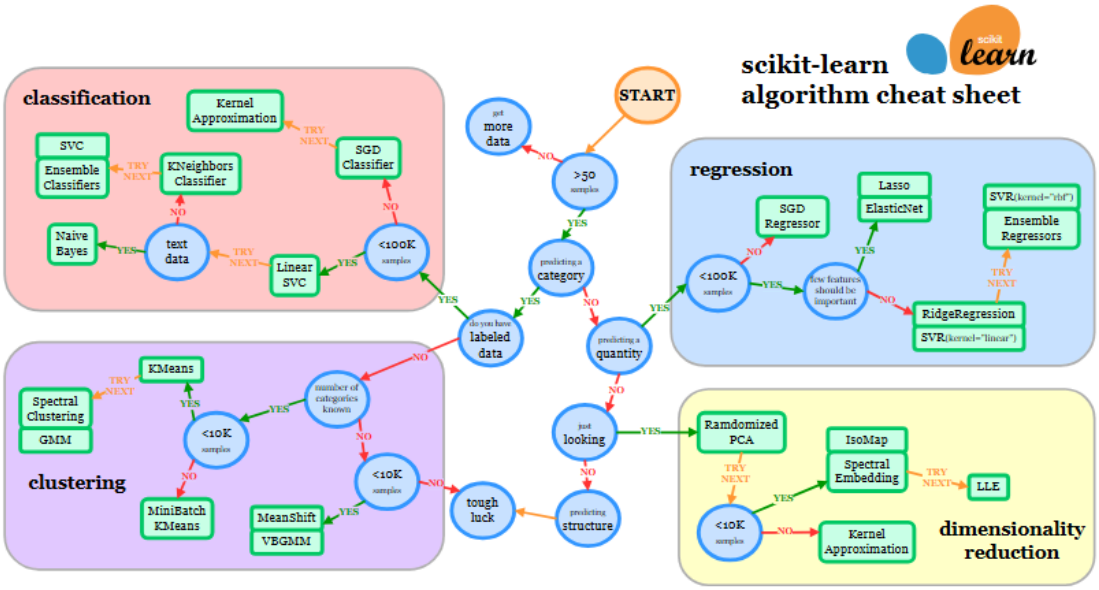
Una fase molto fondamentale del progetto è la fase di previsione dell'energia prodotta da un sistema in un dato momento. Se durante la fase precedente (apprendimento non supervisionato) lo scopo era quello di cercare condizioni meteo simili fra loro, raggrupparle e associare ad esse un range di produzione, adesso l'obiettivo è quello di andare a prevedere con esattezza l'energia prodotta.
Durante questa fase è stata usata la libreria Scikit-learn, che è un framework open-source di Machine Learning per Python, basato su librerie come NumPy, SciPy e Matplotlib.

Scikit-learn

Scikit-learn offre una vasta gamma di strumenti per l'apprendimento supervisionato e non supervisionato, tra cui diversi algoritmi di:

- **Classificazione**: Questi algoritmi assegnano etichette a un'osservazione in base ai dati di input. Il risultato è discreto.
    - Esempi sono Support Vector Machine (SVM), Random Forest e K-Nearest Neighbors (KNN).
- **Clustering**: Questi algoritmi raggruppano dati simili senza etichette predefinite, ideali per scoprire pattern nascosti nei dati.
    - Esempi sono K-Means, DBSCAN e Agglomerative Clustering.
- **Riduzione della dimensionalità**: Aiuta a ridurre il numero di variabili (feature) mantenendo le informazioni più rilevanti, migliorando l'efficienza e l'interpretabilità dei modelli.
    - Esempi sono PCA (Principal Component Analysis), t-SNE e LDA (Linear Discriminant Analysis)
- **Regressione**: Utilizzati per prevedere un valore numerico continuo in base alle variabili indipendenti.
    - Esempi sono Regressione Lineare, il Random Forest Regressor e il Support Vector Regression (SVR).

Oltre a questi algoritmi, Scikit-learn è fondamentale anche per la selezione dei modelli, la valutazione delle loro prestazioni e per il preprocessing dei dati, garantendo una gestione ottimale dell'intero flusso di lavoro di Machine Learning.



RIFERIMENTO → https://scikit-learn.org/stable/machine_learning_map.html

Quale tipologia di modelli si adatta meglio ai nostri scopi?
La tipologia di modelli che meglio si adatta al nostro scopo in questo caso, sono i modelli di regressione, dal momento che essi prevedono un valore continuo, e nel nostro caso dobbiamo appunto andare a prevedere un valore numerico che è la potenza prodotta dal sistema eolico.

Algoritmi da testare

In questa fase saranno confrontati differenti algoritmi, per un confronto più ottimale fra i diversi algoritmi, gli stessi saranno testati su diversi dataset con stessa struttura ma con dati differenti. Per fare ciò sono stati presi in considerazione diversi modelli:

- **Algoritmi di Regressione Lineare**
  - Regressione Lineare Semplice → Linear regression
  - Regressione Ridge
  - Regressione Lasso
  - Regressione Elastic Net
- **Algoritmi basati su Alberi di Decisione**
  - Decision Tree Regressor
  - Random Forest Regressor
- **Algoritmi basati su Support Vector Machine**
  - Support Vector Regressor
- **Reti neurali**
  - Long Short-Term Memory

L'obiettivo è creare uno script per confrontare tutti questi modelli, e valutare quello che è il modello migliore per il nostro caso di studio.

4.1. Workflow

Il workflow per la ricerca del miglior modello predittivo prevede le seguenti fasi:

- Preparazione dei dati.
- Selezione e ottimizzazione del modello.
- Train del modello
- Valutazione delle prestazioni.

Preparazione dei dati

Inizialmente, i dati verranno **normalizzati** per garantire che tutte le variabili siano sulla stessa scala, migliorando così l'efficacia degli algoritmi di Machine Learning. Successivamente, verrà utilizzato un **lookback** per considerare le variabili storiche al fine anche di prevedere la potenza futura, catturando così le tendenze temporali.

Selezione e l'ottimizzazione del modello

Per la selezione del miglior modello, verranno **testati diversi algoritmi**, come la regressione lineare, random forest e support vector regression (SVR), tra gli altri. Ogni modello (dove sarà possibile) sarà **ottimizzato attraverso la ricerca degli iperparametri** (come la profondità dell'albero, parametro alpha, il numero di estimatori, o i parametri di regolarizzazione), utilizzando il **Grid Search** per trovare la combinazione ottimale che massimizza le prestazioni del modello.

Valutazione delle prestazioni.

Per la valutazione del modello, verrà eseguita una valutazione delle prestazioni dei modelli, confrontando le metriche di errore tra le quali il Mean Absolute Error (MAE) e il Root Mean Squared Error (RMSE). L'obiettivo è selezionare il modello che fornisce la previsione più precisa e affidabile per la potenza prodotta.
Le metriche utilizzate per la valutazione sono:

- **MSE (Mean Squared Error)**: È la media degli errori al quadrato tra i valori previsti e quelli reali. Penalizza maggiormente gli errori più grandi ed è utile per evidenziare grosse discrepanze nei modelli.
- **RMSE (Root Mean Squared Error)**: È la radice quadrata dell'MSE e fornisce un valore con la stessa unità di misura della variabile target, rendendolo più interpretabile.
- **MAE (Mean Absolute Error)**: È la media del valore assoluto degli errori. A differenza dell'MSE, non penalizza in modo eccessivo gli errori più grandi, risultando più robusto in presenza di outlier.
- **Deviazione standard degli errori assoluti**: Misura quanto gli errori assoluti si discostano in media dal loro valore medio (MAE). Fornisce un'indicazione della variabilità degli errori nel modello: più è alta, più gli errori sono distribuiti in modo

irregolare; più è bassa, più gli errori sono omogenei. È utile per valutare la stabilità delle prestazioni del modello, oltre alla loro accuratezza.

5. Preparazione dei dati

Nella prima fase dello script i dati vengono organizzati in modo da essere adatti al train.

5.1. Organizzazione dei dati

Funzione di lookback

I modelli che andremo a testare, accettano dati con la seguente struttura (n_samples, n_features), quindi dati bidimensionali a differenza delle reti neurali (es. LSTM) che prendono in input dati a 3 dimensioni (n_samples, lookback, n_features).
Dal momento che andremo ad applicare una funzione di lookback dobbiamo cercare un modo per trasformarli da 3 dimensioni a 2.
Questo viene fatto con la seguente funzione.

```python
def create_lookback_features(df, target_column, lookback):
    """Aggiunge colonne di lookback al dataset."""
    for i in range(1, lookback + 1):
        df[f"{i}"] = df[target_column].shift(i)
    df.dropna(inplace=True)
    return df
```

La funzione aggiunge colonne che rappresentano i valori precedenti (lookback) della colonna di destinazione (target_column), che nel nostro caso è la potenza. Per ogni passo temporale da 1 fino a lookback, essa crea una nuova colonna contenente i valori della colonna di destinazione (potenza) spostati indietro nel tempo di i passi.

Cos'è il lookback?
Essa definisce il numero di passi temporali passati o osservazioni precedenti (nel nostro caso le rilevazioni precedenti) che il modello dovrebbe prendere in considerazione per fare previsioni. Quindi il lookback determina quante informazioni storiche devono essere incluse nel modello per catturare le dipendenze temporali.
Esempio → Nel nostro caso sono state usate anche le 50 precedenti rilevazioni per effettuare la predizione.

Divisione dei dati in train e test

```python
# split dataset
x_train, x_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=42)
```

Nella fase di preparazione dei dati, il dataset viene suddiviso in due insiemi distinti: uno per il training e uno per il testing. È fondamentale che i dati di training e di testing siano

completamente separati, in modo che i modelli non possano "vedere" i dati di test durante l'allenamento. Questo assicura che le valutazioni del modello siano accurate e riflettono realmente la sua capacità di generalizzare a nuovi dati, evitando il rischio di overfitting e garantendo che le metriche di performance siano valide.

5.2. Normalizzazione dei dati

```
def normalize_data(x_train,x_test):
    scaler = MinMaxScaler(feature_range=(-1, 1))
    x_train = scaler.fit_transform(x_train)
    x_test = scaler.transform(x_test)
    return x_train,x_test
```

La normalizzazione dei valori è un processo che consente di ridurre o eliminare le discrepanze tra le scale numeriche di diverse caratteristiche in un dataset. Viene applicata per garantire che tutte le caratteristiche abbiano lo stesso ordine di grandezza e, per trasformare i dati in un intervallo specifico.
Nel nostro caso è stato utilizzato il range da -1 a 1.

5. Ottimizzazione iperparametri

5.1. Algoritmi di Regressione Lineare

Per questa tipologia di algoritmi abbiamo confrontato 4 tipologie:

- Regressione Lineare Semplice
  - LinearRegression()
- Regressione Ridge
  - Ridge()
- Regressione Lasso
  - Lasso()
- Regressione Elastic Net
  - ElasticNet()

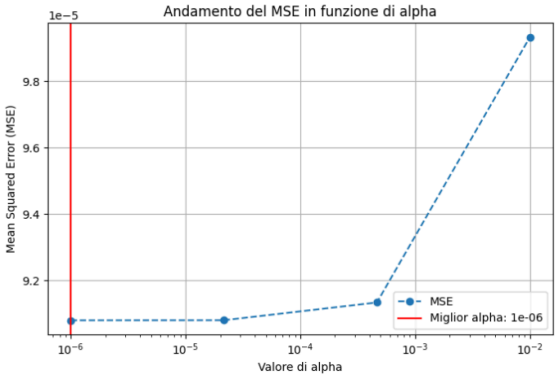
5.1.1. Parametro alpha

Alpha essa controlla la penalizzazione applicata ai coefficienti della regressione lineare.

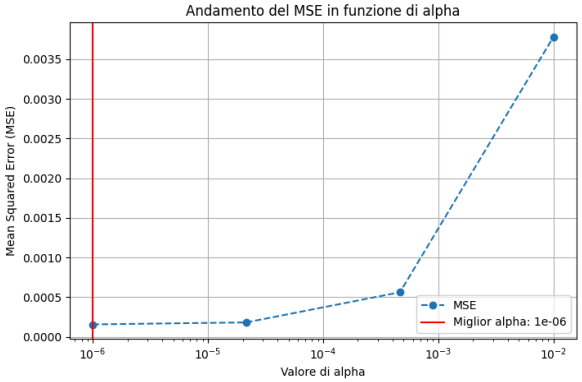
- **alpha alto** Maggiore penalizzazione → i coefficienti vengono ridotti di più, riducendo l'overfitting.
- **alpha basso** Minore penalizzazione → il modello si avvicina alla regressione lineare standard, adattandosi maggiormente ai dati.

Per ottimizzare il valore di alpha, abbiamo utilizzato la classe GridSearchCV, che permette di trovare il miglior parametro per l'algoritmo utilizzato. Gli algoritmi testati per migliorare il parametro Alpha sono stati:

- Ridge.
  - **CONCLUSIONI** → Secondo questo grafico il metodo parametro migliore di alfa e *10^(-6)*



- Lasso
    - **CONCLUSIONE** → Secondo questo grafico il metodo parametro migliore di alfa e *10^(-6)*



CONCLUSIONE → Il valore di alpha che offre le migliori performance per entrambi i modelli (Ridge e Lasso) è 10^-6. Man mano che alpha si avvicina a zero, i modelli tendono a comportarsi in modo simile alla regressione lineare standard.

5.1.2. Parametro l1 ratio

Il parametro l1_ratio in ElasticNet controlla il bilanciamento tra le **penalizzazioni L1 (Lasso) e L2 (Ridge):**

- **l1_ratio = 0** → ElasticNet equivale a una Ridge Regression pura, applicando solo la penalizzazione L2.
- **l1_ratio = 1** → ElasticNet si comporta come una Lasso Regression, applicando solo la penalizzazione L1.
- **0 < l1_ratio < 1** → ElasticNet combina entrambe le penalizzazioni, dove valori più vicini a 1 danno maggiore peso alla L1 (selezione delle feature), mentre valori più vicini a 0 favoriscono la L2 (riduzione della varianza).

L'uso di l1_ratio permette di **ottenere un compromesso tra la riduzione della dimensionalità (Lasso) e la stabilizzazione dei coefficienti (Ridge)**, rendendo ElasticNet adatto a problemi con dati correlati o con molte feature irrilevanti.

ElasticNet

Questo algoritmo ha subito l'ottimizzazione di 2 parametri attraverso la Grid Search:

- Apha
- l1_ratio

I risultati dei test mostrano che il **miglior valore di alpha è 10^-6**, poiché minimizza l'errore per quasi tutti i valori di l1_ratio. Inoltre, i risultati indicano che un **l1_ratio compreso tra 0.1 e 0.5 garantisce prestazioni ottimali**, bilanciando la penalizzazione L1 (selezione delle feature) e L2 (stabilizzazione dei coefficienti), rendendo il modello più robusto e generalizzabile.

5.2. Algoritmi basati su Alberi di Decisione

Per questa tipologia di algoritmi abbiamo confrontato 2 tipologie:

- Decision Tree Regressor
    - DecisionTreeRegressor()
- Random Forest Regressor
    - RandomForestRegressor()

5.2.1. Decision Tree Regressor

È un modello che costruisce un albero decisionale per fare previsioni basate su tutte le variabili del dataset.

5.2.2. Random Forest Regressor

È un insieme di molti alberi decisionali. Ogni albero prende decisioni su sottoinsiemi di dati e caratteristiche, migliorando la previsione della potenza dell'impianto eolico. La previsione finale è la media dei risultati ottenuti da tutti gli alberi.

5.3. Algoritmi basati su Support Vector Machine

Per questa tipologia di algoritmi abbiamo analizzato il Support Vector Regressor fornitoci dalla classe SVR().
L'obiettivo di SVR è trovare una funzione che mappi le variabili di input a valori continui, minimizzando l'errore tra i valori previsti e quelli reali.

Parametri

- **C (penalty parameter):** Controlla la penalizzazione degli errori. Un valore alto di C tende a dare un modello più complesso, mentre un valore basso consente maggiore flessibilità.
- **epsilon (ε):** Determina la larghezza del margine di tolleranza in cui l'errore non viene penalizzato. Un valore troppo grande può ridurre l'accuratezza, mentre uno troppo piccolo può portare a overfitting.
- **kernel**: Tipo di kernel da utilizzare per trasformare i dati. I principali sono linear, poly, rbf (Radial Basis Function), e sigmoid.

5.5. LSTM model

Le Long Short-Term Memory (LSTM) sono una classe di reti neurali ricorrenti (RNN) progettate per gestire serie temporali e dati sequenziali. Grazie alla loro architettura con gates (ingressi di controllo), le LSTM riescono a memorizzare informazioni a lungo termine e a mitigare il problema della scomparsa del gradiente, tipico delle RNN tradizionali.

Struttura del modello

Il modello testato è un modello molto semplice, esso prevede la seguente struttura.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 1, 128)            92672

 lstm_1 (LSTM)               (None, 64)                49408

 dense (Dense)               (None, 25)                1625

 dense_1 (Dense)             (None, 1)                 26

=================================================================
Total params: 143,731
Trainable params: 143,731
Non-trainable params: 0
```

Risultati

I risultati ottenuti sono molto scarsi in confronto a quelli degli algoritmi di regressione. Questi modelli hanno un grande potenziale, le reti neurali e gli LSTM possono offrire ottimi risultati, ma richiedono molta potenza di calcolo e tempo. Questi modelli necessitano di un'accurata ricerca per trovare la migliore architettura, come il numero di livelli e il tipo di funzioni di attivazione da usare. La fase di ottimizzazione può richiedere tempo, soprattutto nel caso si utilizzino grandi quantità di dati. In conclusione, mentre gli algoritmi di regressione sono più semplici e veloci, le reti neurali anche se più complesse, una volta ottimizzate, potrebbero fornire previsioni più accurate sulla potenza generata dagli impianti eolici.

6. Confronto modelli

Dal momento che il dataset prevedeva 4 diversi dataset i diversi modelli sono stati allenati e successivamente testati su differenti train e test set. Questo viene fatto per avere un confronto più veritiero delle prestazioni dei vari modelli.

Dataset 1.

Models	**MSE**	**RMSE**	**MAE**	**Deviazione standard**
Linear Regression	0.0001	0.0093	0.0051	0.0078
Ridge Regression	0.0001	0.0093	0.0051	0.0078
Lasso Regression	0.0002	0.0123	0.0078	0.0095
ElasticNet Regression	0.0002	0.0123	0.0078	0.0095
Random Forest Regressor	0.0002	0.0126	0.0072	0.0103
Decision Tree Regressor	0.0003	0.0173	0.0115	0.0129
Support Vector Regressor	0.0023	0.0476	0.0396	0.0265
LSTM	0.0271	0.1646	0.1352	0.0938

Dataset 2.

Models	**MSE**	**RMSE**	**MAE**	**Deviazione standard**
Linear Regression	0.0001	0.0076	0.004	0.0064
Ridge Regression	0.0001	0.0076	0.004	0.0064,
Lasso Regression	0.0001	0.0098	0.0059	0.0078
ElasticNet Regression	0.0001	0.0098	0.0059	0.0078,
Random Forest Regressor	0.0001	0.0107	0.0058	0.009
Decision Tree Regressor	0.0002	0.0143	0.0092	0.009
Support Vector Regressor	0.002	0.0452	, 0.0369	0.0261
LSTM	0.0336	0.1832	, 0.1472	0.1092

Dataset 3.

Models	MSE	RMSE	MAE	Deviazione standard
Linear Regression	0.0001	0.0088	0.0048	0.0074
Ridge Regression	0.0001	0.0088	0.0048	0.0074
Lasso Regression	0.0001	0.0117	0.0072	0.0092
ElasticNet Regression	0.0001	0.0117	0.0072	0.0092
Random Forest Regressor	0.0001	0.0121	0.0066	0.0101
Decision Tree Regressor	0.0003	0.0166	0.0106	0.0127
Support Vector Regressor	0.0022	0.0468	0.0383	0.027
LSTM	0.0713	0.2671	0.2193	0.1524

Dataset 4.

Models	MSE	RMSE	MAE	Deviazione standard
Linear Regression	0.0001	0.0077	0.0042	0.0065
Ridge Regression	0.0001	0.0077	0.0042	0.0065
Lasso Regression	0.0001	0.01	0.0062	0.0078
ElasticNet Regression	0.0001	0.01	0.0062	0.0078
Random Forest Regressor	0.0001	0.0108	0.006	0.009
Decision Tree Regressor	0.0002	0.0145	0.0094	0.011
Support Vector Regressor	0.0023	0.0476	0.0399	0.026
LSTM	0.0149	0.1219	0.0962	0.0749

Confronto finale

Models	MSE	RMSE	MAE	Deviazione standard
Linear Regression	0.0001	0.00835	0.004524	0.00802
Ridge Regression	0.0001	0.00835	0.0045249	0.008025
Lasso Regression	0.0001	0.01	0.0062	0.0078
ElasticNet Regression	0.000125	0.01095	0.006775	0.008575
Random Forest Regressor	0.000125	0.01155	0.00639	0.0096

Decision Tree Regressor	0.00025	0.015725	0.010175	0.0114
Support Vector Regressor	0.0022	0.0468	0.039175	0.0274
LSTM	0.036775	0.1942	0.149475	0.107575

Confronto modelli per previsioni a 5 passi nel futuro

Models	**MSE**	**RMSE**	**MAE**	**Deviazione standard**
Linear Regression	0.0156	0.1248	0.0893	0.0873
Ridge Regression	0.0156	0.1248	0.0893	0.0873
Lasso Regression	0.0166	0.1288	0.094	0.0881
ElasticNet Regression	0.0166	0.1288	0.094	0.0881
Random Forest Regressor	0.0128	0.1131	0.078	0.0819
Decision Tree Regressor	0.0201	0.1418	0.1013	0.0993
Support Vector Regressor	0.0142	0.1192	0.0896	0.0786
LSTM	0.0378	0.1944	0.1578	0.1135

7. Conclusioni

Dalla media delle metriche calcolate sui quattro dataset, emerge chiaramente che i modelli di regressione lineare (Linear, Ridge, Lasso, ElasticNet) sono più prestazionali stabili ed efficaci, con errori medi molto bassi. In particolare, **Linear Regression** si conferma il modello più performante. Il modello **LSTM**, pur essendo più complesso e adatto a dati sequenziali, mostra errori significativamente più alti in tutti gli indicatori. Il modello **LSTM** risulta più prestazionale e performante nel momento in cui si punta a prevedere dati nel futuro e non nell'immediato. Infatti come mostrato nella tabella precedente tutti i modelli che prima risultavano prestazionali, in questo caso la media di errore tende ad alzarsi andando a pareggiare quelli del LSTM che con un train più lungo e con una rete neurale più profonda porterebbe risultati molto più prestazionali.

Riferimenti Bibliografici

- www.kaggle.com
- https://scikit-learn.org/stable/
- https://scikit-learn.org/stable/machine_learning_map.html