RdS
RICERCA DI SISTEMA

ENEA
AGENZIA NAZIONALE PER LE NUOVE TECNOLOGIE, L'ENERGIA E LO SVILUPPO ECONOMICO SOSTENIBILE

MINISTERO DELL'AMBIENTE E DELLA SICUREZZA ENERGETICA

Ricerca di Sistema elettrico

Sviluppo e confronto prestazionale di algoritmi di rilevamento di cyber-attacchi nel contesto delle microreti elettriche

(LA3.12)

Rds_PTR 22-24_PR 2.1_LA3.12_148

SVILUPPO E CONFRONTO PRESTAZIONALE DI ALGORITMI DI RILEVAMENTO DI CYBER-ATTACCHI NEL CONTESTO DELLE MICRORETI ELETTRICHE

Corrado Aaron Visaggio e gli altri ricercatori dell'Unità di ricerca (Unisannio)

Dicembre 2024

Report Ricerca di Sistema Elettrico

Accordo di Programma Ministero dell'Ambiente e della Sicurezza Energetica -ENEA Piano Triennale di Realizzazione 2022-2024

Obiettivo 2: Digitalizzazione ed evoluzione delle reti

Progetto 2.1: Cybersecurity dei Sistemi Energetici

Linea di attività: LA 3.12

Responsabile del Progetto: Maria Valenti

Responsabile Linea di Attività: Corrado Aaron Visaggio

Mese inizio previsto: luglio 2023

Mese inizio effettivo: luglio 2023

Mese fine previsto: dicembre 2024

Mese fine effettivo: dicembre 2024

Indice

Indice delle figure

1 Risultati attesi

Questo rapporto è orientato a descrivere lo sviluppo gli algoritmi identificati per la rilevazione di attacchi informatici nel contesto delle micro-reti elettriche.

Saranno quindi valutate le prestazioni degli algoritmi in oggetto.

Il rapporto prevede, quindi, un lavoro in tre fasi.

Nella prima fase sarà realizzata un'analisi della letteratura di settore, ovvero la letteratura riguardante le vulnerabilità e gli attacchi effettuati alle microreti elettriche. Questi attacchi e queste vulnerabilità saranno analizzati in termini in termini di procedimenti utilizzati, strumenti utilizzati e finalità.

Successivamente saranno analizzati gli studi relativi all'efficacia di questi attacchi ed alla loro realizzabilità.

In una seconda fase saranno selezionati gli attacchi ritenuti più adeguati allo scopo del progetto. L'adeguatezza degli attacchi sarà valutata tenendo in conto due fattori principali: la efficacia degli stessi – è necessario individuare quegli attacchi che consentono di ottenere risultati significativi-, la loro realizzabilità – bisogna individuare attacchi che siano riproducibili in modo agevole, considerando che attacchi puramente teorici o eccessivamente costosi non sono riscontrati nella realtà-.

Quindi dovrà essere identificato un dataset rappresentativo degli attacchi selezionati e che sarà necessario per la valutazione dell'algoritmo di rilevamento degli attacchi.

Il dataset dovrà essere costruito utilizzando sia le risorse endogene che le risorse esogene, ricercando cioè i dataset disponibili nella comunità scientifica ed industriale.

La scelta del dataset dovrà essere orientata a soddisfare criteri di:

- Verosimiglianza, ovverosia non dovrà essere costituito da dati sintetici
- Rappresentatività, ovverosia dovrà essere sufficientemente rappresentativo della realtà che si intende rappresentare
- Correttezza e completezza, ovverosia i dati utilizzati dovranno essere corretti e completi, ma soprattutto presentati in un formato che può essere sottoposto ad elaborazione
- Etichettati, ovverosia i dati devono essere categorizzati per poter consentire una efficace verifica.

Quindi si realizzerà un algoritmo per il rilevamento di tali attacchi. Sarà definito un banco di metriche utilizzate per la validazione ed un processo di validazione ed infine eseguita la validazione.

L'efficacia di tali attacchi sarà valutata utilizzando il dataset costruito.

2 Risultati ottenuti

L'obiettivo principale del progetto è la selezione, implementazione e confronto prestazionale di algoritmi di rilevamento di attacchi nelle microreti elettriche, con particolare attenzione agli attacchi di False Data Injection (FDI), che mirano a compromettere l'affidabilità delle misurazioni e delle comunicazioni, con potenziali impatti significativi sull'operatività e sulla sicurezza dei sistemi elettrici. L'obiettivo principale è stato individuare un algoritmo efficace per rilevare tali attacchi garantendo prestazioni elevate e tempi di rilevamento rapidi. Dopo un'analisi comparativa tra diverse metodologie di rilevamento delle anomalie (supervisionate, semi-supervisionate e non supervisionate), è stato scelto l'algoritmo Density-Based Spatial Clustering of Applications with Noise (DBSCAN), per la sua robustezza e flessibilità, in particolare per la capacità di identificare cluster di forma arbitraria e distinguere le potenziali anomalie senza richiedere la conoscenza preventiva del numero di cluster.

L'implementazione è stata sviluppata in tre fasi principali:

- Simulazione attacchi FDI
- ottimizzazione dei parametri `eps' e `min_samples` (attraverso l'analisi della curva k-distance e validazioni empiriche)
- rilevamento delle anomalie.

Per la valutazione del modello, sono stati simulati attacchi FDI manipolando dal 1% al 10% del dataset attraverso l'iniezione di rumore gaussiano e modifiche sistematiche a tensioni, correnti e potenze, con la creazione di scenari realistici. I test hanno mostrato un' Accuracy dell' 87%, una Precision dell'95%, una Recall del 87%, un F1-score dell'89%, un Silhouette score di 0.57 e un Davies-Bouldin index di 1.32, evidenziando una buona capacità di separazione tra cluster normali e anomali, sebbene con qualche sovrapposizione dovuta alla variabilità dei dati. La validazione visiva tramite grafici ha permesso di osservare la distribuzione dei dati e identificare le aree di confine tra cluster, utili per l'analisi di falsi positivi e negativi.

In conclusione, l'utilizzo di DBSCAN si è dimostrato efficace per il rilevamento di attacchi FDI grazie alla sua capacità di operare senza dati pre-etichettati e di adattarsi alle peculiarità delle microreti. Gli sviluppi futuri includono l'integrazione di altri algoritmi, l'applicazione di tecniche di ottimizzazione automatica dei parametri e l'estensione dello studio a scenari più complessi, al fine di rafforzare ulteriormente la sicurezza e l'affidabilità delle microreti elettriche in un contesto di crescente digitalizzazione.

2.1 Posizionamento della ricerca rispetto allo stato dell'arte internazionale

L'utilizzo di algoritmi di clustering per il rilevamento delle anomalie è diventato uno strumento fondamentale per la sicurezza delle Smart Grid e, più in generale, per i sistemi energetici moderni. Il clustering, come tecnica di apprendimento non supervisionato, permette di raggruppare i dati in insiemi omogenei senza la necessità di label predefinite. Questo approccio è particolarmente adatto per rilevare anomalie e attacchi come le False Data Injection (FDI), che mirano a manipolare le misurazioni dei sensori e compromettere la stima

dello stato del sistema. In letteratura, diversi algoritmi di clustering sono stati proposti e testati per la rilevazione delle anomalie nelle microreti e nelle reti di distribuzione. Tra questi, il Density-Based Spatial Clustering of Applications with Noise (DBSCAN) emerge per la sua capacità di individuare cluster di forma arbitraria e isolare i punti anomali senza richiedere la specifica del numero di cluster a priori [1][2]. Questo lo rende particolarmente efficace per i sistemi caratterizzati da dati complessi e non lineari, come quelli generati dagli smart meters e dai dispositivi IoT nelle microreti. Nel lavoro di Mohammadpourfard et al. [3], è stato proposto un approccio basato su metodi statistici non supervisionati per distinguere tra modelli normali e di attacco, utilizzando il clustering fuzzy C-means per localizzare le anomalie. Un altro studio di Khond et al. [1] ha confrontato vari algoritmi di clustering (tra cui DBSCAN, K-means e Gaussian Mixture Models) per la rilevazione di dati corrotti nelle microreti, evidenziando come DBSCAN offra una migliore separazione tra dati normali e anomali grazie alla sua capacità di gestire dati rumorosi e outlier. Inoltre, Bhattacharjee et al. [4] hanno introdotto un approccio basato sul deep clustering, combinando autoencoder e tecniche di clustering per rilevare attacchi stealthy FDI in sistemi di stima dello stato. Questo metodo ha dimostrato prestazioni comparabili, se non superiori, ai modelli supervisionati, mantenendo la capacità di operare senza dati etichettati. Infine, lo studio di Sharma et al. [2] ha proposto un sistema ibrido che combina la Principal Component Analysis (PCA) con DBSCAN per rilevare le FDI nei dati di consumo degli smart meters. Questa combinazione ha permesso di ridurre la dimensionalità dei dati e migliorare la rilevazione delle anomalie in scenari ad alta dimensionalità [5]. Nel complesso, gli algoritmi di clustering rappresentano una soluzione versatile e robusta per il rilevamento delle anomalie nelle Smart Grid, offrendo un equilibrio tra efficacia, flessibilità e capacità di adattarsi a scenari operativi dinamici e non prevedibili.

3 Prodotti attesi

I prodotti attesi consistono nei seguenti elementi:

- Algoritmo di rilevamento degli attacchi alle microreti elettriche e relativi confronti prestazionali nei differenti scenari.
- Casi di test per la valutazione dell'algoritmo
- Definizione di un banco di metriche per la sua valutazione.

Essendo l'algoritmo basato su meccanismi di apprendimento automatico, sarà necessario addestrare con un opportuno processo il sistema di machine learning e validarlo, secondo i processi tradizionalmente adottati in letteratura.

Sarà quindi definito un insieme di casi di test necessari a testare l'algoritmo in questione che siano sufficientemente rappresentativi degli scenari reali in cui gli attacchi vengono effettuati.

Infine dovrà essere realizzato un banco di metriche per valutare l'efficacia di tale algortimo.

4 Prodotti ottenuti

Il progetto ha portato allo sviluppo di una serie di prodotti finalizzati al rilevamento di attacchi False Data Injection (FDI) nelle microreti elettriche, garantendo un sistema completo e facilmente integrabile in contesti reali.

I prodotti ottenuti si articolano in tre componenti principali:

- **Generazione del dataset di attacchi - `fdi.py`:** Questo script permette di simulare attacchi FDI sui dati delle microreti attraverso l'iniezione di anomalie controllate. L'algoritmo seleziona casualmente una percentuale configurabile di righe (`**fault_ratio**`) e introduce rumore gaussiano sui parametri numerici (come tensioni, correnti e potenze), con un'intensità definita dal parametro `**noise_scale**`. Ogni riga del dataset viene etichettata tramite la colonna `**label**` (0 per dati normali, 1 per dati manipolati), facilitando l'addestramento e la valutazione degli algoritmi di rilevamento. L'output è un file CSV contenente esclusivamente le righe modificate e contrassegnate come anomalie.
- **Ottimizzazione dei parametri DBSCAN - `Parameter_tuning.py`:** Lo script è dedicato alla selezione ottimale dei parametri `**eps**` e `**min_samples**` dell'algoritmo DBSCAN, fondamentali per una corretta individuazione delle anomalie. Utilizzando la tecnica della k-distance, lo script calcola le distanze medie dei punti rispetto ai loro vicini più prossimi e genera un grafico per identificare visivamente il "**gomito**", corrispondente al valore ottimale di `**eps**`. Nel progetto, il valore selezionato è stato 0.25 con `**min_samples**` impostato a 10, garantendo un equilibrio tra sensibilità alle anomalie e limitazione dei falsi positivi.
- **Rilevamento e valutazione delle anomalie - `DBSCAN.py`:** Questo script rappresenta la fase finale del processo. Dopo aver caricato e pre-elaborato i dati (rimozione dei valori mancanti, scaling e selezione delle variabili numeriche), applica l'algoritmo DBSCAN per rilevare le anomalie. I punti classificati come anomali vengono etichettati come "Anomalous". Lo script fornisce anche una visualizzazione grafica delle anomalie rilevate tramite scatter plot, evidenziando la separazione tra dati normali e anomali. Infine, genera report quantitativi sulle prestazioni del modello, calcolando metriche come Accuracy (87%), Precision (95%), Recall (87%), F1-Score (89%), Silhouette Score (0.57) e Davies-Bouldin Index (1.32). Gli output comprendono file CSV contenenti i dati etichettati e report sulle prestazioni del modello.

Nel complesso, i prodotti sviluppati costituiscono una suite completa per la simulazione, rilevamento e analisi di attacchi FDI nelle microreti elettriche. In prospettiva futura, è previsto l'ampliamento della suite con l'integrazione di algoritmi alternativi, l'automatizzazione dell'ottimizzazione dei parametri e la creazione di un sistema di allerta in tempo reale, al fine di incrementare ulteriormente la capacità di risposta e la sicurezza delle infrastrutture energetiche.

5 Analisi degli scostamenti su attività e risultati

Non ci sono stati scostamenti significativi rispetto ai risultati attesi.
Questi sono stati tutti ottenuti con successo.
I costi di esercizio non sono stati sostenuti perché per l'attività sono state utilizzate le strumentazioni di laboratorio abitualmente utilizzate nella usuale attività di ricerca.

6 Sintesi delle attività svolte

Il progetto è orientato alla rilevazione di attacchi alle microreti elettriche, concentrandosi in particolare sugli attacchi di False Data Injection (FDI). Questi attacchi rappresentano una minaccia significativa per la sicurezza e l'affidabilità delle microreti, in quanto mirano a manipolare i dati di monitoraggio e controllo, compromettendo il funzionamento dei sistemi energetici. Il progetto si è posto l'obiettivo di individuare soluzioni in grado di rilevare tempestivamente tali intrusioni, garantendo la stabilità operativa delle microreti.

La prima fase ha visto lo svolgimento di un'analisi comparativa delle principali tecniche di rilevamento delle anomalie, considerando approcci supervisionati, semi-supervisionati e non supervisionati. In questo contesto, si è cercato di valutare non solo l'efficacia degli algoritmi disponibili, ma anche la loro capacità di adattarsi alle caratteristiche specifiche dei dati provenienti dalle microreti. La scelta si è infine orientata verso l'algoritmo DBSCAN, selezionato in base a fattori come la possibilità di individuare cluster di forma arbitraria e la robustezza nella gestione di dati rumorosi e, soprattutto, la capacità di operare in assenza di conoscenze preliminari sul numero di cluster presenti. Questa caratteristica ha reso DBSCAN particolarmente adatto a contesti reali, dove la distribuzione dei dati può essere irregolare e mutevole.

Dopo aver identificato la metodologia più idonea, si è passati alla fase operativa, dedicata alla sviluppo e implementazione di una pipeline modulare e completa. L'obiettivo principale di questa fase è stato creare un flusso di lavoro chiaro e riproducibile, capace di simulare attacchi FDI, rilevare le anomalie introdotte e valutarne le prestazioni in modo sistematico. La pipeline è stata suddivisa in tre componenti principali, ciascuna progettata per rispondere a specifiche esigenze del processo:

Generazione degli attacchi FDI:

Il primo passo è stato sviluppare uno script dedicato alla simulazione degli attacchi. Questo strumento consente di iniettare anomalie controllate nei dati delle microreti, riproducendo così scenari realistici di attacco. La simulazione avviene attraverso l'aggiunta di rumore gaussiano a una percentuale definita dei dati, con parametri configurabili per modulare l'intensità e la frequenza delle anomalie. Grazie a questa flessibilità, è stato possibile creare dataset personalizzati, utili per testare l'algoritmo di rilevamento in condizioni variabili. L'iniezione controllata permette inoltre di distinguere chiaramente tra dati normali e dati alterati, facilitando così la successiva fase di valutazione. Questo passaggio è stato fondamentale per garantire che i modelli di rilevamento sviluppati potessero essere testati non solo su dati teorici, ma anche su scenari verosimili e rappresentativi delle sfide reali che le microreti potrebbero affrontare.

Ottimizzazione dei parametri dell'algoritmo DBSCAN:

Una volta creato il dataset con le anomalie iniettate, è stato definito uno script orientato all'ottimizzazione dei parametri di DBSCAN, elemento cruciale per garantire un'efficace rilevazione delle anomalie. L'algoritmo si basa su due parametri chiave: **eps**, che definisce la distanza massima tra due punti affinché siano considerati parte dello stesso cluster, e **min_samples**, che stabilisce il numero minimo di punti necessari per formare un cluster. La definizione di questi parametri ha richiesto un'attenta analisi, in quanto valori non ottimali avrebbero potuto portare a un aumento dei falsi positivi o, al contrario, alla mancata individuazione di anomalie. È stata pertanto implementata una procedura che, attraverso l'utilizzo della curva k-distance, ha permesso di individuare in modo visivo il "gomito" della curva, indicativo del valore ottimale per eps. Questa fase ha richiesto diverse iterazioni e valutazioni empiriche per bilanciare sensibilità e specificità del modello. La scelta finale dei parametri ha garantito un buon compromesso tra la capacità di individuare cluster significativi e la riduzione del rumore indesiderato nei dati.

Rilevamento e valutazione delle anomalie:

Dopo aver identificato i parametri ottimali per garantire una corretta rilevazione delle anomalie da parte dell'algoritmo sviluppato si è passati alla fase di rilevamento vera e propria. Il processo inizia con il preprocessing dei dati, durante il quale i dataset vengono puliti e normalizzati per garantire coerenza tra le variabili. Successivamente, l'algoritmo DBSCAN viene applicato per individuare i punti anomali, che vengono automaticamente etichettati e isolati per ulteriori analisi. Per facilitare l'interpretazione dei risultati, sono state generate visualizzazioni grafiche che mostrano la distribuzione dei dati e la posizione delle anomalie rilevate. Questi grafici, che evidenziano chiaramente i confini tra cluster e punti considerati rumore, si sono rivelati strumenti preziosi per comprendere visivamente le dinamiche dei dati e le performance dell'algoritmo. Infine, è stata effettuata una valutazione quantitativa delle prestazioni, utilizzando metriche standard come Accuracy, Precision, Recall e F1-score. Queste misure hanno permesso di quantificare l'efficacia del modello nel rilevare le anomalie introdotte, fornendo un quadro completo della sua capacità di generalizzare anche su dati non visti.

7 Dettaglio delle attività svolte

Il progetto è stato sviluppato attraverso una serie di attività, ciascuna delle quali finalizzata alla rilevazione efficace degli attacchi di False Data Injection (FDI) nelle microreti elettriche. Le attività si sono concentrate sulla selezione, implementazione, ottimizzazione e validazione di un algoritmo di anomaly detection in grado di operare in contesti reali e caratterizzati da dati complessi. Gli aspetti centrali del lavoro hanno riguardato la scelta dell'algoritmo più adatto, la creazione di scenari di attacco realistici, la preparazione e gestione del dataset e, infine, l'analisi approfondita delle prestazioni ottenute.

7.1 Tipologie di algoritmi

La prima fase del progetto ha previsto un'analisi approfondita delle tecniche di rilevamento delle anomalie basate su algoritmi di apprendimento automatico (Machine Learning). Tale analisi è stata essenziale per comprendere quali approcci potessero meglio adattarsi alle peculiarità delle microreti elettriche, tenendo conto della natura dei dati e delle caratteristiche operative delle Smart Grid. Gli algoritmi presi in considerazione sono stati suddivisi in tre categorie principali: supervisionati, non supervisionati e semi-supervisionati.

Gli algoritmi supervisionati richiedono dataset etichettati, in cui ogni istanza dei dati è associata a una classe nota. Questo approccio è molto efficace per compiti come la classificazione e la regressione e trova largo impiego in applicazioni come gli Intrusion Detection Systems (IDS) basati su firme note o nella previsione del consumo energetico. Tuttavia, in contesti complessi e dinamici come quelli delle Smart Grid, le anomalie possono presentarsi in maniera imprevedibile e non essere note a priori. La necessità di disporre di dataset etichettati completi e aggiornati rappresenta quindi una limitazione importante, che può compromettere l'efficacia degli algoritmi supervisionati in scenari reali.

Al contrario, gli algoritmi non supervisionati operano senza la necessità di etichette associate ai dati e cercano autonomamente pattern e strutture nascoste all'interno del dataset. Questa caratteristica li rende particolarmente indicati per l'anomaly detection nelle Smart Grid, dove il comportamento normale della rete può variare nel tempo e i dati relativi agli attacchi potrebbero essere assenti o non facilmente identificabili. Tra le tecniche non supervisionate, il clustering si è rivelato particolarmente utile per suddividere i dati in gruppi omogenei e individuare così deviazioni significative, potenzialmente indicatrici di anomalie o malfunzionamenti. In questo contesto, l'algoritmo DBSCAN si è distinto per la sua capacità di rilevare pattern complessi e gestire efficacemente i dati rumorosi e gli outlier, aspetti cruciali per il monitoraggio delle microreti elettriche.

Gli algoritmi semi-supervisionati rappresentano un compromesso tra i due approcci precedenti. Essi utilizzano un numero limitato di dati etichettati per guidare l'apprendimento e l'identificazione di anomalie nei dati non etichettati. Nonostante questo approccio possa

risultare efficace in determinati scenari, nelle Smart Grid la disponibilità di dati etichettati affidabili è spesso scarsa, specialmente quando si tratta di rilevare nuove tipologie di attacchi o anomalie emergenti.

Dopo un'attenta valutazione delle tre categorie, si è deciso di adottare un approccio non supervisionato per la rilevazione delle anomalie. La scelta dell'algoritmo DBSCAN è stata guidata da diverse considerazioni pratiche e metodologiche. In particolare, DBSCAN si è dimostrato in grado di:

- Operare efficacemente senza necessità di conoscere a priori il numero di cluster presenti nel dataset.
- Rilevare cluster di forma arbitraria, caratteristica fondamentale in contesti con dati non lineari.
- Isolare automaticamente i dati rumorosi e trattarli come outlier, consentendo di individuare potenziali anomalie.
- Adattarsi alle variazioni dinamiche tipiche delle microreti, che possono manifestare comportamenti normali estremamente variabili nel tempo.

7.2 Casi di test e simulazione degli attacchi

Per valutare le prestazioni dell'algoritmo selezionato, è stato sviluppato un caso di test realistico basato su un dataset contenente rilevazioni provenienti dagli smart meters. Le misurazioni sono state effettuate a intervalli regolari di 15 minuti, includendo dati relativi a potenza attiva e reattiva, legata ai consumi degli utenti finali ed alle variazioni di carico, strettamente connesse alle fasce orarie e ai diversi profili di utilizzo dell'energia.

7.2.1 Generazione delle anomalie (attacchi FDI)

Per simulare gli attacchi FDI, è stato sviluppato uno script dedicato che ha consentito di introdurre anomalie realistiche all'interno del dataset. In particolare:

- È stato iniettato rumore gaussiano sui dati, riproducendo così le fluttuazioni che un attaccante potrebbe introdurre per manipolare le decisioni di controllo della rete.
- È stata data la possibilità di personalizzare la percentuale di dati alterati tramite un parametro denominato fault_ratio, regolando così la frequenza delle anomalie in funzione degli scenari considerati.
- L'intensità delle modifiche è stata controllata dal parametro noise_scale, che ha permesso di modulare la deviazione standard del rumore aggiunto.

7.2.2 Preprocessing e divisione del dataset

Dopo la generazione delle anomalie, il dataset è stato sottoposto a una fase di preprocessing al fine di rimuovere eventuali valori mancanti, normalizzare il dataset e filtrare le colonne non rilevanti. Successivamente, il dataset è stato suddiviso nel modo seguente:

- **Train set:** 80% dei dati non anomali, utilizzato per addestrare il modello.
- **Test set:** 10% dei dati non anomali e 10% dei dati anomali, utilizzato per valutare le capacità di generalizzazione e rilevamento dell'algoritmo.

7.3 Selezione e funzionamento dell'algoritmo DBSCAN

L'algoritmo DBSCAN è stato selezionato in virtù della sua robustezza e della capacità di gestire dati complessi e rumorosi senza richiedere la specifica del numero di cluster a priori, a differenza di altri metodi più comuni come K-Means o Agglomerative Clustering. In ambito Smart Grid, i dati provenienti da sensori, smart meters e dispositivi IoT presentano spesso strutture non lineari, la presenza di outlier e variazioni significative legate alle condizioni operative. In questi scenari, la flessibilità offerta da DBSCAN risulta un vantaggio determinante.

Il funzionamento di DBSCAN si basa sull'identificazione di aree ad alta densità di punti. Due sono i parametri principali che regolano il comportamento dell'algoritmo:

- **eps:** Definisce il raggio massimo entro il quale due punti possono essere considerati parte dello stesso cluster.
- **min_sample:** Rappresenta il numero minimo di punti richiesto per formare un cluster denso.

7.4 Ottimizzazione dei parametri di DBSCAN

L'efficacia dell'algoritmo DBSCAN dipende in modo cruciale dalla corretta definizione dei parametri **eps** e **min_sample**. Una scelta inappropriata può compromettere significativamente le prestazioni del modello:

- Un valore troppo grande di eps può generare cluster eccessivamente estesi, riducendo la sensibilità e portando a falsi negativi.
- Un valore troppo piccolo di eps può provocare la formazione di cluster troppo frammentati, aumentando i falsi positivi.

Per identificare il valore ottimale di eps, è stata utilizzata la curva k-distance, che rappresenta graficamente la distanza media dei punti rispetto ai loro k-esimi vicini. Il punto di flessione

della curva, noto come "gomito", indica la soglia oltre la quale le distanze iniziano ad aumentare rapidamente. Attraverso questa analisi visiva, si sono individuati i seguenti valori ottimali:

- **eps** = 0.25: Ha garantito un equilibrio tra la capacità di rilevare le anomalie e la riduzione dei falsi allarmi.
- **min_sample** = 10: È stato scelto per assicurare la formazione di cluster densi e rappresentativi delle normali condizioni operative della microrete.

7.5 Valutazione delle prestazioni

Per quantificare le prestazioni dell'algoritmo nel rilevamento delle anomalie, sono state adottate metriche standard come **Silhouette Score** (utilizzato per valutare la qualità dei cluster ottenuti e **Davies-Bouldin Index** (utilizzato per valutare la separazione dei cluster ottenuti). Sono state inoltre considerate metriche tipiche dell'anomaly detection come Accuracy, Precision, Recall ed F1-Score.

8 Contributo delle eventuali consulenze alle attività sopra descritte

N/A

9 Pubblicazioni scientifiche

Elenco delle pubblicazioni scientifiche eventualmente risultanti dall'attività svolta.

A. Sgueglia, C. A. Visaggio, S. De Vito, G. Di Francia. FALSE DATA INJECTION IDENTIFICATION IN ENERGY MICROGRIDS THROUGH AN ANOMALY DETECTION APPROACH, in Lectures Notes in Electrical Engineering (edt. S. Conosci, C. Di Natale, L. Prodi, G. Valenti) proc. Of AISEM 2025, XXIII Conference on Sensors and Microsystems, Springer Verlag, 2025.

10 Eventi di disseminazione

Lista degli eventi di disseminazione eventualmente scaturiti dall'attività svolta.

N/A

11 Descrizione dei risultati ottenuti

Le attività svolte nel corso del progetto hanno portato allo sviluppo e alla validazione di un sistema per il rilevamento degli attacchi di False Data Injection (FDI) all'interno delle microreti elettriche. Per raggiungere questo obiettivo, il lavoro è stato articolato in più fasi, che hanno compreso la simulazione di scenari di attacco realistici, l'implementazione e l'ottimizzazione di un algoritmo di anomaly detection basato sull'approccio non supervisionato di DBSCAN e, infine, la valutazione delle prestazioni ottenute. Questa sezione descrive nel dettaglio i risultati emersi da tali attività e illustra il processo seguito per raggiungere le configurazioni ottimali dell'algoritmo.

11.1 Casi di test

Per testare la robustezza e l'efficacia dell'algoritmo sviluppato nella rilevazione di attacchi FDI, è stato generato un secondo dataset contenente anomalie iniettate artificialmente. La creazione di questo dataset è partita dai dati originali rilevati dagli smart meter installati nella Smart Grid, che registravano misurazioni relative alla potenza attiva e reattiva dei consumi degli utenti, con rilevazioni effettuate a intervalli regolari di 15 minuti.

Generazione delle anomalie: lo script fdi.py
La generazione delle anomalie è stata realizzata tramite lo script fdi.py, appositamente sviluppato per simulare scenari di attacco realistici e riproducibili. Il funzionamento dello script si basa sull'utilizzo di una funzione che, partendo dal dataset originale, seleziona un sottoinsieme di righe da alterare e applica modifiche alle variabili numeriche, seguendo una logica controllata e parametrizzabile. Il codice completo è riportato nella Sezione Fdi.py.

Gli step principali dello script sono stati i seguenti:

- **Selezione delle righe da modificare:** Lo script utilizza il parametro **fault_ratio** per determinare la percentuale di righe del dataset da alterare. Ad esempio, un valore di fault_ratio = 0.1 indica che il 10% delle righe verrà selezionato in maniera casuale per la simulazione degli attacchi.
- **Iniezione di rumore gaussiano:** Una volta selezionate le righe, lo script procede con l'iniezione delle anomalie. Questo viene fatto aggiungendo un valore casuale, generato tramite una distribuzione gaussiana, ai dati numerici delle righe selezionate. Il parametro **noise_scale** permette di controllare l'intensità del rumore iniettato: valori più alti comportano modifiche più marcate, mentre valori più bassi simulano anomalie più sottili e difficili da rilevare.
- **Labelling dei dati:** Per facilitare la successiva fase di valutazione dell'algoritmo, ogni riga alterata viene etichettata con la variabile label, che assume valore 1 per i dati

modificati e 0 per i dati rimasti invariati. Questa etichettatura è fondamentale per confrontare le anomalie rilevate dall'algoritmo con quelle effettivamente presenti.

L'esecuzione dello script produce un file CSV (fdi_data.csv) contenente i dati alterati e le rispettive label.

11.2 Ottimizzazione dei parametri dell'algoritmo

Uno degli aspetti più delicati nell'utilizzo di DBSCAN riguarda la scelta dei parametri **eps** (raggio massimo per includere un punto in un cluster) e **min_samples** (numero minimo di punti per formare un cluster denso). Valori non appropriati possono portare a risultati poco affidabili:

- Un **eps** troppo grande rischia di aggregare dati anomali all'interno dei cluster, riducendo la sensibilità e aumentando i falsi negativi.
- Un **eps** troppo piccolo può portare a un'eccessiva frammentazione, con un conseguente aumento dei falsi positivi.

Ottimizzazione tramite lo script Parameter_tuning.py

Al fine di individuare i parametri ottimali, è stato sviluppato lo script Parameter_tuning.py, che utilizza la curva k-distance. Il codice completo è riportato nella Sezione Parameter_tuning.py. Questa curva rappresenta la distanza media dei punti rispetto ai loro k-esimi vicini e permette di individuare visivamente il "gomito" della curva, indicativo del valore ottimale di eps. Come è possibile osservare in Figura 1, il parametro **eps** stimato è 0.25, il quale garantisce un identificazione ottimale dei punti afferenti allo stesso cluster.

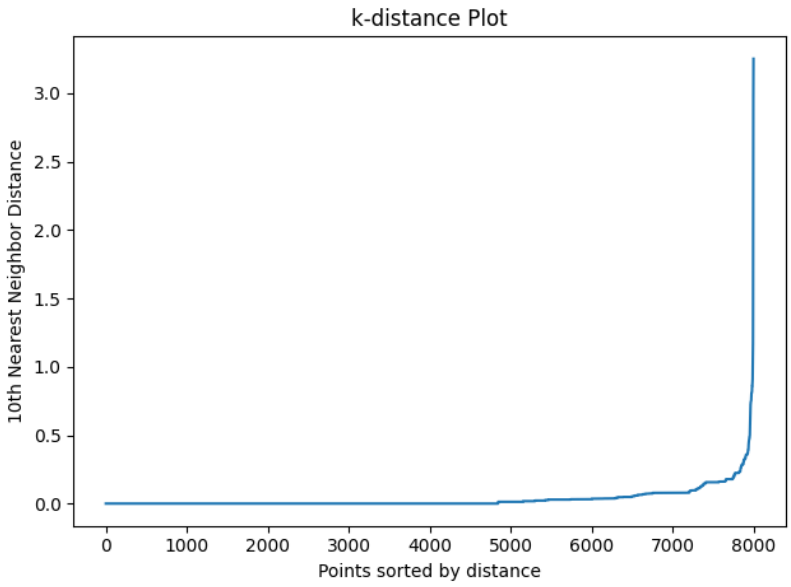

Figura 1 - eps tuning

11.3 Testing dell'algoritmo sviluppato

Dopo aver creato i casi di test e generato il dataset contenente le anomalie, la fase successiva si è concentrata sull'implementazione e sul testing dell'algoritmo di rilevamento. A tal fine, è stato sviluppato lo script DBSCAN.py, che rappresenta il nucleo operativo del sistema. Questo script gestisce l'intero processo di preparazione dei dati, applicazione dell'algoritmo DBSCAN, visualizzazione dei risultati e valutazione delle prestazioni. Il codice completo è riportato nella Sezione DBSCAN.py.

Struttura e funzionalità dello script DBSCAN.py

Lo script è stato strutturato in diverse fasi per garantire un flusso di lavoro chiaro, modulare e facilmente eseguibile.

Fase 1: Preprocessing dei dati

La prima fase consiste nel caricamento e nella preparazione dei dati. In questa fase, vengono letti due dataset:

- Il dataset di addestramento (train set), contenente dati normali e privo di anomalie, utilizzato per addestrare l'algoritmo a riconoscere i pattern di comportamento regolare.
- Il dataset di test, ottenuto combinando i dati non anomali con quelli alterati generati nella fase precedente.

Il preprocessing prevede le seguenti operazioni:

- **Pulizia dei dati:** Viene effettuata la rimozione di colonne non rilevanti (come i timestamp, se non necessari) e la gestione dei valori mancanti, sostituiti con zeri per evitare distorsioni nell'analisi.
- **Normalizzazione delle variabili:** Poiché le variabili numeriche possono avere scale diverse, viene applicata la tecnica dello StandardScaler, che porta tutte le feature ad avere media zero e deviazione standard unitaria. Questo passaggio è essenziale per evitare che variabili con valori numerici più grandi influenzino eccessivamente l'algoritmo.

Fase 2: Rilevamento delle anomalie

Conclusa la fase di preprocessing, lo script passa all'applicazione dell'algoritmo DBSCAN. La funzione principale, detect_anomalies, riceve in input:

- I dati normalizzati.

- I parametri **eps** e **min_samples**, che determinano rispettivamente la dimensione massima del raggio di ricerca e il numero minimo di punti richiesto per formare un cluster denso.

DBSCAN classifica i punti in:

- **Punti normali:** Appartenenti a cluster identificati come comportamenti regolari.
- **Anomalie:** Punti non assegnati a nessun cluster e considerati outlier.

Fase 3: Visualizzazione dei risultati

Per facilitare la comprensione e l'interpretazione dei risultati, lo script genera grafici scatter che mostrano la distribuzione dei dati e l'individuazione delle anomalie. In questi grafici:

- I punti blu rappresentano i dati normali.
- I punti rossi evidenziano le anomalie rilevate.

Queste visualizzazioni consentono di osservare a colpo d'occhio come l'algoritmo abbia individuato i dati anomali e la loro distribuzione rispetto ai dati normali.

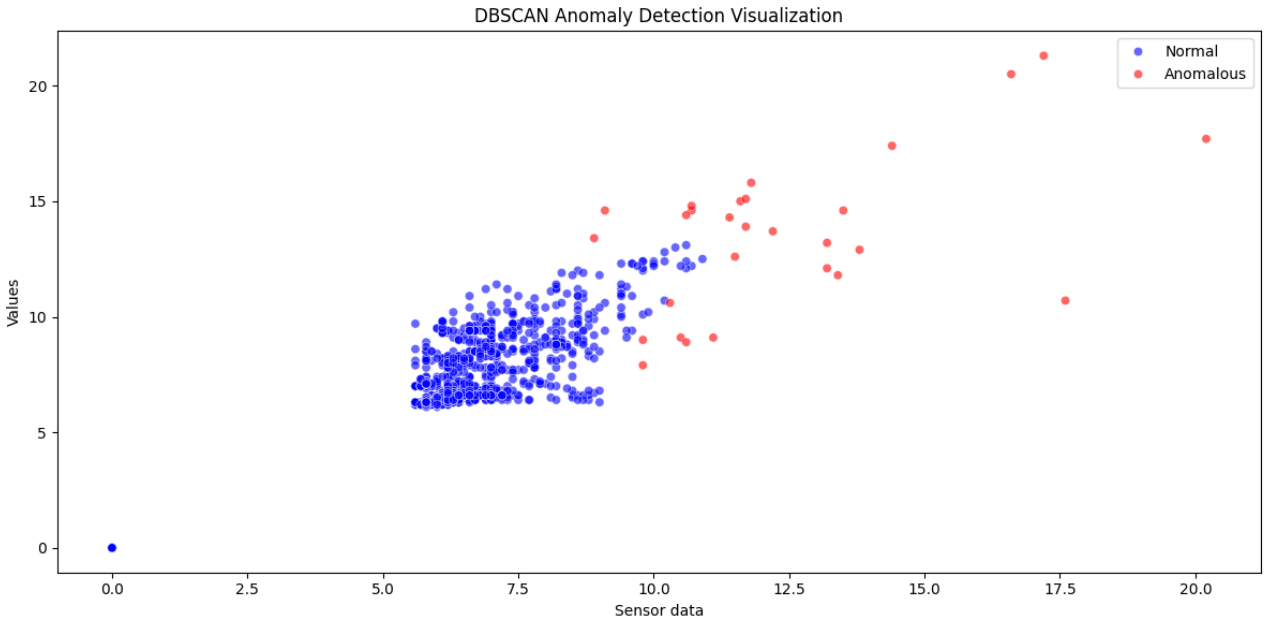

Figura 2 - Anomalie train set

Come è possibile osservare dalla Figura 2, è possibile osservare la presenza di alcune anomalie anche all'interno del training set. Tuttavia, è importante sottolineare che, in questo specifico contesto, il modello è stato addestrato esclusivamente sui dati considerati normali, escludendo deliberatamente quelli etichettati come anomali. Le anomalie rilevate nel training set non sono quindi riconducibili alle simulazioni di attacchi False Data Injection (FDI)

effettuate durante questa sperimentazione, ma risultano piuttosto correlate a possibili malfunzionamenti del sistema di acquisizione dei dati o a irregolarità operative intrinseche. Queste anomalie "naturali" evidenziano la complessità dei dati reali e sottolineano l'importanza di considerare anche le imperfezioni dei sistemi di rilevamento nella fase di analisi e addestramento dei modelli.

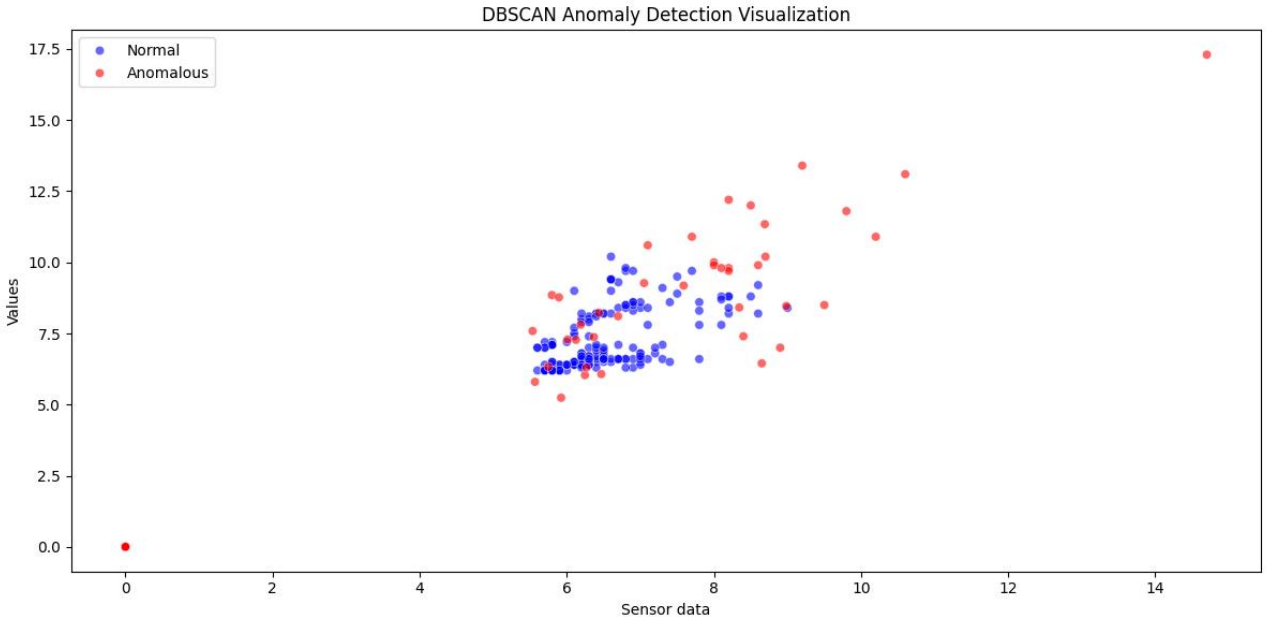

Figura 3 - Anomalie Test set

Come mostrato nella Figura 3, è possibile osservare la presenza di diverse anomalie all'interno del test set. In questo caso specifico, a differenza del training set, le anomalie rilevate sono effettivamente riconducibili alle simulazioni di attacchi False Data Injection (FDI) introdotte durante questa sperimentazione. Queste anomalie sono state intenzionalmente iniettate nel dataset per valutare la capacità dell'algoritmo di rilevare manipolazioni dei dati che potrebbero compromettere la sicurezza e l'affidabilità del sistema di monitoraggio della rete. La presenza di tali anomalie nel test set ha quindi permesso di testare la robustezza del modello in scenari realistici, evidenziando la sua efficacia nel distinguere i dati normali da quelli compromessi da attacchi mirati.

Fase 4: Valutazione delle prestazioni

L'ultima fase dello script è dedicata alla valutazione quantitativa delle prestazioni dell'algoritmo. Per questa analisi sono state utilizzate sia metriche specifiche per il clustering, sia metriche tipiche della classificazione delle anomalie:

- **Silhouette Score:** Indica quanto i cluster sono ben separati tra loro.

- **Davies-Bouldin Index:** Misura la compattezza e la separazione tra i cluster.
- **Accuracy, Precision, Recall e F1-Score:** Offrono una visione completa della capacità dell'algoritmo di identificare correttamente le anomalie.

11.4 Risultati finali

Con i parametri ottimizzati, l'algoritmo è stato applicato al dataset contenente le anomalie. La Figura 3 mostra la distribuzione dei dati e la rilevazione delle anomalie da parte dell'algoritmo. Dal grafico emerge chiaramente come il modello sia in grado di identificare correttamente le righe modificate durante la simulazione dell'attacco FDI.

Metrica	Valore
Accuracy	87%
Precision	95%
Recall	87%
F1-Score	89%
Silhouette Score	0.57
Davies-Bouldin Index	1.32

Tabella 1 - Performance algoritmo sviluppato

Le metriche evidenziate in Tabella 1 ottime prestazioni complessive:

- L'accuracy del 87% conferma l'elevata affidabilità del sistema.
- La precision dell'95% indica che la maggior parte delle anomalie rilevate era effettivamente corretta, con pochi falsi positivi.
- Un recall del 87% mostra la capacità dell'algoritmo di individuare la maggior parte delle anomalie presenti.
- L'F1-Score dell'89% rappresenta un buon compromesso tra precision e recall.

11.5 Conclusioni

Il sistema sviluppato si è dimostrato efficace nel rilevare gli attacchi FDI simulati sui dati delle microreti elettriche. L'utilizzo dell'algoritmo DBSCAN, combinato con un'accurata ottimizzazione dei parametri, ha permesso di ottenere prestazioni soddisfacenti sia in termini

di rilevamento delle anomalie che di qualità del clustering. La pipeline proposta si è rivelata versatile, modulare e facilmente adattabile a diversi scenari operativi, rappresentando un valido strumento per la protezione delle infrastrutture energetiche da potenziali minacce informatiche. Questi risultati costituiscono una base solida per futuri sviluppi, come l'integrazione di sistemi di allerta in tempo reale e l'adattamento a dati provenienti da fonti eterogenee e dinamiche.

Sulla base dei risultati riportati nella tabella precedente, è stato scelto di utilizzare un valore di **eps** pari a 0.25 e di **min_sample** pari a 10 poiché questa combinazione di parametri garantiva un buon compromesso tra le due metriche calcolate.

Infine, l'algoritmo sviluppato nello script precedente ha portato ai risultati evidenziati nella seguente figura (Figura 3), nella quale è possibile osservare come questo algoritmo risulta essere in grado di rilevare le anomalie artificiali iniettate attraverso la simulazione di un attacco di FDI.

12 Appendice

12.1 Fdi.py

```
import os
import numpy as np
import pandas as pd

def inject_faults_with_labels(df, fault_ratio=0.1, noise_scale=0.5):
    """
    Injects faults into the DataFrame and adds a 'label' column:
    - 0 indicates an unaltered row.
    - 1 indicates a tampered (faulty) row.

    Parameters:
    - df (pd.DataFrame): Input dataset.
    - fault_ratio (float): Ratio of rows to inject faults into.
    - noise_scale (float): Scale of the noise relative to the column's standard deviation.

    Returns:
    - pd.DataFrame: Dataset containing only the tampered rows with a 'label' column.
    """
    if not 0 < fault_ratio <= 1:
        raise ValueError("fault_ratio must be between 0 and 1.")

    tampered_df = df.copy()

    # Ensure there are numeric columns to inject faults into
```

```
    numeric_columns = tampered_df.select_dtypes(include=['float64', 'int64']).columns
    if numeric_columns.empty:
        raise ValueError("No numeric columns found for fault injection.")

    # Initialize the label column with zeros (unaltered rows)
    tampered_df['label'] = 0

    total_rows = tampered_df.shape[0]
    num_faults = max(1, int(total_rows * fault_ratio))  # Ensure at least one fault is injected

    np.random.seed(42)  # For reproducibility
    fault_indices = np.random.choice(total_rows, num_faults, replace=False)

    # Update 'label' to 1 for the tampered rows
    tampered_df.loc[fault_indices, 'label'] = 1

    # Inject noise into the selected numeric columns for the tampered rows
    for col in numeric_columns:
        if tampered_df[col].std() == 0:
            continue  # Skip columns with zero standard deviation
        noise = np.random.normal(loc=0, scale=tampered_df[col].std() * noise_scale,
size=num_faults)
        tampered_df.loc[fault_indices, col] += noise

    # Return only the tampered rows (label == 1)
    anomalies_df = tampered_df[tampered_df['label'] == 1].reset_index(drop=True)
    return anomalies_df

if __name__ == "__main__":
```

```
# Paths to the input and output CSV files
dataset_path = os.path.join(os.getcwd(), "all_merged_data.csv")
out_path = os.path.join(os.getcwd(), "fdi_data.csv")

# Check if the dataset exists
if not os.path.exists(dataset_path):
    raise FileNotFoundError(f"Dataset not found at {dataset_path}")

# Load dataset
dataset = pd.read_csv(dataset_path)

if dataset.empty:
    raise ValueError("The input dataset is empty.")

print(f"Dataset loaded successfully with shape: {dataset.shape}")

# Inject faults with labels and retrieve only anomalies
fault_ratio = 0.1  # Adjust as needed
noise_scale = 0.5  # Adjust the intensity of noise if required
anomalies_with_labels = inject_faults_with_labels(dataset, fault_ratio=fault_ratio,
noise_scale=noise_scale)

if anomalies_with_labels.empty:
    print("Warning: No anomalies were injected.")
else:
    # Save only the tampered rows (anomalies) with labels to a CSV file
    anomalies_with_labels.to_csv(out_path, index=False)
    print(f"Anomalies dataset saved to {out_path} with 'label' column included.")
    print(f"Total anomalies injected: {anomalies_with_labels.shape[0]}")
```

```
print(anomalies_with_labels.head())
```

12.2 Parameter_tuning.py

```
import os

from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
import numpy as np
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# Load datasets
train_set = pd.read_csv('all_merged_data_with_label.csv')
test_set = pd.read_csv('fdi_data.csv')

# Split the merged dataset: 80% for training, 20% for testing
train_df, temp_df = train_test_split(train_set, test_size=0.2, random_state=42)

# From the remaining 20%: 10% from merged and 10% from fdi for the test set
merged_test_sample = temp_df.sample(frac=0.5, random_state=42) # 10% of all_merged_data
fdi_test_sample = test_set.sample(frac=0.1, random_state=42) # 10% of fdi_data

# Combine samples to form the final test set
test_df = pd.concat([merged_test_sample, fdi_test_sample]).reset_index(drop=True)
```

```
# Data Cleaning: Remove 'Timestamp' and drop columns with all NaN values
train_df_clean = train_df.drop(columns=["Timestamp"], errors='ignore').dropna(axis=1, how='all')
test_df_clean = test_df.drop(columns=["Timestamp"], errors='ignore').dropna(axis=1, how='all')

# Fill remaining NaN values with column means
train_df_clean = train_df_clean.fillna(0)
test_df_clean = test_df_clean.fillna(0)

#print(train_df_clean.shape)
#print(test_df_clean.shape)



# Check if dataframes are empty after cleaning
if train_df_clean.empty or test_df_clean.empty:
   raise ValueError("One of the datasets is empty after cleaning. Check the input files.")

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(train_df_clean)
X_test_scaled = scaler.transform(test_df_clean)

neigh = NearestNeighbors(n_neighbors=10)
nbrs = neigh.fit(X_train_scaled)
distances, _ = nbrs.kneighbors(X_train_scaled)
distances = np.sort(distances[:, 9])  # 4 for 5th nearest neighbor
plt.plot(distances)
plt.title('k-distance Plot')
plt.xlabel('Points sorted by distance')
plt.ylabel('10th Nearest Neighbor Distance')
#plt.legend()
plt.tight_layout()
output_plot = os.path.join(os.getcwd(), f"DBSCAN_parameter_tuning.png")
```

```
os.makedirs(os.path.dirname(output_plot), exist_ok=True)
plt.savefig(output_plot)
plt.show()
print(f"Plot saved to: {output_plot}")
```

12.3 DBSCAN.py

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, classification_report, silhouette_score, davies_bouldin_score
)
from sklearn.model_selection import train_test_split


# ————————————————-
# Step 1: Data Preprocessing
# ————————————————-
def preprocess_data(df):
    """
    Preprocesses the input DataFrame:
    - Fills NaN values with zeros
    - Scales numeric columns
```

```
    Returns:
    - Scaled data
    - Original DataFrame (with filled NaNs)
    - List of numeric columns
    """
    df.fillna(0, inplace=True)
    numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(df[numeric_columns])
    return scaled_data, df, numeric_columns


# ——————————————-
# Step 2: DBSCAN Anomaly Detection
# ——————————————-
def detect_anomalies(scaled_data, original_df, eps=0.25, min_samples=10):
    """
    Detects anomalies using DBSCAN and appends 'Anomaly' labels to the DataFrame.

    Parameters:
    - scaled_data: Scaled feature matrix
    - original_df: Original DataFrame
    - eps: DBSCAN epsilon parameter
    - min_samples: DBSCAN min_samples parameter

    Returns:
    - DataFrame with 'Anomaly' column
    - DBSCAN model
    - Cluster labels
    """
    dbscan = DBSCAN(eps=eps, min_samples=min_samples)
    labels = dbscan.fit_predict(scaled_data)
    original_df['Anomaly'] = np.where(labels == -1, 'Anomalous', 'Normal')
    return original_df, dbscan, labels
```

```
# ————————————————-
# Step 3: Visualization
# ————————————————-
def visualize_anomalies(df, numeric_columns, filename):
    """
    Visualizes anomalies over the first two numeric features.

    Parameters:
    - df: DataFrame with 'Anomaly' column
    - numeric_columns: List of numeric columns
    - filename: Name for saving the plot
    """
    plt.figure(figsize=(12, 6))
    if len(numeric_columns) < 2:
        print("Warning: Not enough numeric columns for scatter plot visualization.")
        return

    sns.scatterplot(x=df[numeric_columns[0]], y=df[numeric_columns[1]], hue=df['Anomaly'],
                    palette={'Normal': 'blue', 'Anomalous': 'red'}, alpha=0.6)

    plt.xlabel("Sensor data")
    plt.ylabel("Values")
    plt.title('DBSCAN Anomaly Detection Visualization')
    plt.legend()
    plt.tight_layout()
    output_plot = os.path.join(os.getcwd(), f"{filename}_DBSCAN_anomalies.png")
    os.makedirs(os.path.dirname(output_plot), exist_ok=True)
    plt.savefig(output_plot)
    plt.show()
    print(f"Plot saved to: {output_plot}")
```

```
# ——————————————-
# Step 4: Evaluation Metrics
# ——————————————-
def evaluate_metrics(y_true, y_pred, dataset_name, scaled_data, labels):
    """
    Calculates evaluation metrics for anomaly detection, including silhouette score and Davies-Bouldin index.

    Parameters:
    - y_true: Ground truth labels
    - y_pred: Predicted anomaly labels
    - dataset_name: Name of the dataset (Train/Test)
    - scaled_data: Scaled features used for clustering
    - labels: Cluster labels from DBSCAN

    Returns:
    - Dictionary of evaluation metrics
    """
    silhouette = silhouette_score(scaled_data, labels) if len(set(labels)) > 1 else -1
    davies_bouldin = davies_bouldin_score(scaled_data, labels) if len(set(labels)) > 1 else -1

    metrics = {
        "Dataset": dataset_name,
        "Accuracy": accuracy_score(y_true, y_pred),
        "Precision": precision_score(y_true, y_pred, pos_label=1, zero_division=0),
        "Recall": recall_score(y_true, y_pred, pos_label=1, zero_division=0),
        "F1 Score": f1_score(y_true, y_pred, pos_label=1, zero_division=0),
        "Silhouette Score": silhouette,
        "Davies-Bouldin Index": davies_bouldin
    }
    return metrics

# ——————————————-
```

```
# Step 5: Main Workflow
# ——————————————-
if __name__ == "__main__":
    # Load datasets
    train_path = os.path.join(os.getcwd(), "all_merged_data_with_label.csv")
    test_path = os.path.join(os.getcwd(), "fdi_data.csv")

    train_set = pd.read_csv(train_path)
    test_set = pd.read_csv(test_path)

    # Train-Test Split
    train_df, temp_df = train_test_split(train_set, test_size=0.2, random_state=42)
    merged_test_sample = temp_df.sample(frac=0.5, random_state=42)  # 10% of all_merged_data
    fdi_test_sample = test_set.sample(frac=0.1, random_state=42)  # 10% of fdi_data
    test_df = pd.concat([merged_test_sample, fdi_test_sample]).reset_index(drop=True)

    # Preprocessing
    X_train_scaled, train_df_clean, train_numeric = preprocess_data(train_df)
    X_test_scaled, test_df_clean, test_numeric = preprocess_data(test_df)

    # DBSCAN Anomaly Detection
    eps, min_samples = 0.25, 10
    train_anomalies, train_dbscan, train_labels = detect_anomalies(X_train_scaled, train_df_clean, eps=eps, min_samples=min_samples)
    test_anomalies, test_dbscan, test_labels = detect_anomalies(X_test_scaled, test_df_clean, eps=eps, min_samples=min_samples)

    # Visualization
    visualize_anomalies(train_anomalies, train_numeric, "train")
    visualize_anomalies(test_anomalies, test_numeric, "test")

    # Evaluation
    if 'label' not in train_df or 'label' not in test_df:
```

```
        raise KeyError("Both train and test datasets must contain a 'label' column.")

    y_train_true = train_df['label']
    y_test_true = test_df['label']
    y_train_pred = np.where(train_anomalies['Anomaly'] == 'Anomalous', 1, 0)
    y_test_pred = np.where(test_anomalies['Anomaly'] == 'Anomalous', 1, 0)

    #train_metrics = evaluate_metrics(y_train_true, y_train_pred, "Train", X_train_scaled, train_labels)
    test_metrics = evaluate_metrics(y_test_true, y_test_pred, "Test", X_test_scaled, test_labels)

    #metrics_df = pd.DataFrame([train_metrics, test_metrics])
    metrics_df = pd.DataFrame([test_metrics])
    print("\nEvaluation Metrics:")
    print(metrics_df)
    metrics_df.to_csv("test_metrics.csv")

    #print("\nTrain Classification Report:")
    #print(classification_report(y_train_true, y_train_pred))

    print("\nTest Classification Report:")
    print(classification_report(y_test_true, y_test_pred))

    #print("\nTrain Confusion Matrix:")
    #print(confusion_matrix(y_train_true, y_train_pred))

    print("\nTest Confusion Matrix:")
    print(confusion_matrix(y_test_true, y_test_pred))

    # Save results
    train_output_path = os.path.join(os.getcwd(), "train_with_anomalies.csv")
    test_output_path = os.path.join(os.getcwd(), "test_with_anomalies.csv")

    train_anomalies.to_csv(train_output_path, index=False)
```

```
test_anomalies.to_csv(test_output_path, index=False)

print(f"\nTrain anomalies saved to: {train_output_path}")
print(f"Test anomalies saved to: {test_output_path}")
```

13 Riferimenti

[1] Khond, S., Kale, V., & Ballal, M. S. (2022, January). Data mining methods for bad data detection and event data acquisition in microgrids. In 2022 IEEE International Conference on Power Electronics, Smart Grid, and Renewable Energy (PESGRE) (pp. 1-6). IEEE.

[2] Sharma, R., Joshi, A. M., Sahu, C., & Nanda, S. J. (2023). Detection of false data injection in smart grid using PCA based unsupervised learning. *Electrical Engineering*, *105*(4), 2383-2396.

[3] Mohammadpourfard, M., Sami, A., & Seifi, A. R. (2017). A statistical unsupervised method against false data injection attacks: A visualization-based approach. Expert Systems with Applications, 84, 242-261.

[4] Bhattacharjee, A., Mondal, A. K., Verma, A., Mishra, S., & Saha, T. K. (2022). Deep latent space clustering for detection of stealthy false data injection attacks against AC state estimation in power systems. IEEE Transactions on Smart Grid, 14(3), 2338-2351.

[5] Chen, C., Wang, Y., Cui, M., Zhao, J., Bi, W., Chen, Y., & Zhang, X. (2022). Data-driven detection of stealthy false data injection attack against power system state estimation. IEEE Transactions on Industrial Informatics, 18(12), 8467-8476.