Università degli Studi di Padova

Dipartimento di Scienze Statistiche
Corso di Laurea Triennale in Statistica per l'Economia e l'Impresa

Relazione Finale

Anomaly Detection: Metodi, Applicazioni e Sperimentazione su Dati Reali

Relatore Prof. Matteo Ceccarello
Dipartimento di Ingegneria dell'Informazione

Laureando: Gabriele Ansaldo
Matricola N2067194

Anno Accademico 2024/2025

Abstract

L'Anomaly Detection è un processo fondamentale nell'analisi dei dati, utile per distinguere comportamenti normali da errori, frodi o malfunzionamenti. Quando i dati sono organizzati in serie temporali, la presenza di trend, stagionalità e autocorrelazione introduce nuove sfide rendendo l'AD un problema ancora più complesso. Per questo motivo, l'identificazione di anomalie su serie storiche è diventata una disciplina a sé stante e in questa tesi viene esplorata attraverso il metodo delle sliding windows.

Questa tesi valuta l'efficacia di alcuni metodi di anomaly detection tramite una pipeline Python. In particolare vengono considerati k-Nearest Neighbors, AutoEncoder e Isolation Forest applicati con sliding windows su più dataset reali etichettati. L'analisi dei risultati dimostra che Isolation Forest fornisce il miglior compromesso tra accuratezza e velocità di esecuzione, rendendolo ideale in molte situazioni. In parallelo, sono stati confrontati diversi criteri per la definizione della soglia di anomalia, dimostrando che la classica $\mu + 3\sigma$ non garantisce prestazioni ottimali e che, tra le alternative analizzate, alcune soglie si sono dimostrate più robuste.

Indice

Capitolo 1

Introduzione

L'Anomaly Detection è un processo centrale nell'analisi dei dati e nel machine learning. Un'anomalia, o outlier, è un punto che si discosta in modo significativo dal resto dei dati, suggerendo che sia stato generato da un processo differente[1]. Distinguere i dati normali da quelli anomali è fondamentale in molti ambiti come il web, la diagnosi medica, il rilevamento delle frodi con carte di credito (tramite pattern anomali nelle transazioni), e molti altri.

Quando i dati sono organizzati come serie temporali, ossia sequenze ordinate di osservazioni nel tempo, il compito di individuare anomalie diventa ancora più critico. Le serie temporali descrivono l'andamento nel tempo di sistemi complessi (ad esempio, flussi di traffico di rete, andamenti finanziari, dati meteo), dove un'anomalia può indicare malfunzionamenti, eventi estremi o attacchi malevoli [2]. Inoltre, la correlazione temporale consente che un picco isolato possa essere naturale in un contesto, ma anomalo in un altro. Per questi motivi, l'Anomaly Detection su serie storiche è diventata una disciplina a sé stante nell'ambito dell'analisi dei dati, con metodologie dedicate a captare deviazioni sia puntuali sia strutturali nel tempo [3].

Sfide tecniche

Lo sviluppo di modelli efficaci di Anomaly Detection per serie temporali è ostacolato da diverse sfide tecniche [4]:

- **Variabilità intrinseca:** le serie temporali spesso presentano componenti di trend, stagionalità e rumore e quindi sono necessari modelli capaci di separarle.
- **Scarsità di dati etichettati:** in molti contesti reali, le anomalie sono eventi rari e costosi da etichettare, rendendo impossibile l'uso di tecniche supervisionate.
- **Dipendenze temporali:** le serie hanno relazioni sia a breve sia a lungo termine e il modello deve riuscire a cogliere entrambi senza diventare troppo lento o complesso.
- **Scalabilità e latenze:** con dataset di grandi dimensioni o in contesti di streaming, spesso si sacrifica un po' di accuratezza per ottenere latenze molto basse e aggiornamenti in tempo reale.
- **Valutazione delle performance:** quando sono disponibili etichette, lo sbilanciamento tra le poche istanze anomale e le moltissime normali rende difficoltoso scegliere metriche e soglie: una minima variazione della soglia può cambiare totalmente i risultati.

Struttura e contributo della tesi

Questa tesi si propone di approfondire le tecniche di Anomaly Detection su serie storiche, con un focus sul metodo delle *sliding windows*. In particolare, verranno applicati tre algoritmi (k-Nearest Neighbors, Isolation Forest e AutoEncoder) a più dataset reali etichettati, al fine di individuare il metodo che garantisce il miglior compromesso tra accuratezza e tempi di elaborazione, e di definire una soglia di anomalia che risulti robusta nella pratica.

La tesi è organizzata come segue:

- **Capitolo 3**: vengono introdotti i concetti fondamentali: panoramica sui metodi di Anomaly Detection, metriche di valutazione delle performance e approcci specifici per serie temporali.

- **Capitolo 4**: descrizione dettagliata dei tre algoritmi scelti, con analisi del funzionamento, dei punti di forza e di debolezza e delle principali caratteristiche.

- **Capitolo 5**: presentazione del set-up sperimentale (strumenti, pipeline Python e dataset utilizzati) e dei risultati: confronto degli AUC e dei tempi di esecuzione, selezione delle soglie ottimali e valutazione.

- **Capitolo 6**: sintesi dei risultati ottenuti, discussione e possibili sviluppi futuri.

Capitolo 2

Stato dell'Arte

L'Anomaly Detection (AD) è un'area di ricerca attiva da decenni, affrontata inizialmente con approcci statistici classici e, più recentemente, con tecniche di machine learning e deep learning. L'obiettivo principale è identificare pattern o osservazioni che si discostano significativamente dal comportamento ritenuto normale, indicando potenziali errori o guasti [4].

Nel contesto delle serie temporali, l'AD presenta sfide specifiche, dato che i dati sono ordinati temporalmente, possono mostrare autocorrelazione e spesso presentano stagionalità e trend. Queste caratteristiche rendono meno efficaci molti metodi generici, portando la ricerca verso approcci specializzati [1].

Un approccio comune per l'AD in serie temporali è l'utilizzo di finestre mobili (sliding windows), che trasformano i dati in matrici, rendendo possibile l'applicazione di tecniche tradizionali [3].

Con il deep learning, modelli come le Long Short-Term Memory (LSTM) [2], progettati appositamente per serie storiche, riescono a individuare le anomalie analizzando gli errori nella ricostruzione della sequenza.

TSB-UAD è un benchmark che, oltre a fornire dataset etichettati per l'AD, implementa strumenti per la valutazione sia di approcci basati su sliding windows sia di metodi specializzati per serie temporali come LSTM [3].

Per supportare la sperimentazione e la valutazione dei modelli di AD, sono state sviluppate librerie come PyOD [5], che integra oltre 40 algoritmi, dai

metodi classici a quelli basati su deep learning, all'interno di un framework unificato.

I modelli di anomaly detection, inclusi quelli classici come kNN e quelli basati su deep learning come gli autoencoder, sono soggetti a continue ottimizzazioni per migliorarne l'efficacia. Nel caso di kNN, ad esempio, si lavora su tecniche di riduzione della dimensionalità per aumentarne l'efficienza computazionale [6]. Tali ottimizzazioni sono fondamentali per adattare i modelli alle caratteristiche dei dati e per affrontare le nuove sfide nell'ambito dell'AD.

Capitolo 3

Concetti e Definizioni

In questo capitolo verranno fornite una definizione e una classificazione più approfondite delle anomalie e dell'anomaly detection, saranno analizzati i principali utilizzi dell'Anomaly Detection e le sfide che essa comporta.

3.1 Anomalia

Le anomalie sono osservazioni o comportamenti nei dati che si discostano da quella che è ritenuta la normalità. Le cause che provocano anomalie nei dati sono molte e il loro studio è fondamentale per rilevare dinamiche inconsuete, errori o eventi eccezionali. La definizione di anomalia non è univoca e può variare in base al contesto applicativo e alla struttura dei dati, ciò che appare anomalo in un contesto potrebbe essere perfettamente accettabile in un altro. Questo rende la rilevazione delle anomalie un problema complesso e multidimensionale che richiede l'utilizzo di metodi avanzati.

3.1.1 Tipi di anomalie

Per il rilevamento delle anomalie è fondamentale individuare che tipo di anomalia si vuole individuare. Si distinguono 3 tipi di anomalie:

- **Anomalie Puntuali:** Sono dati isolati che si discostano significativamente dai valori normali, indipendentemente dal contesto in cui si trovano.

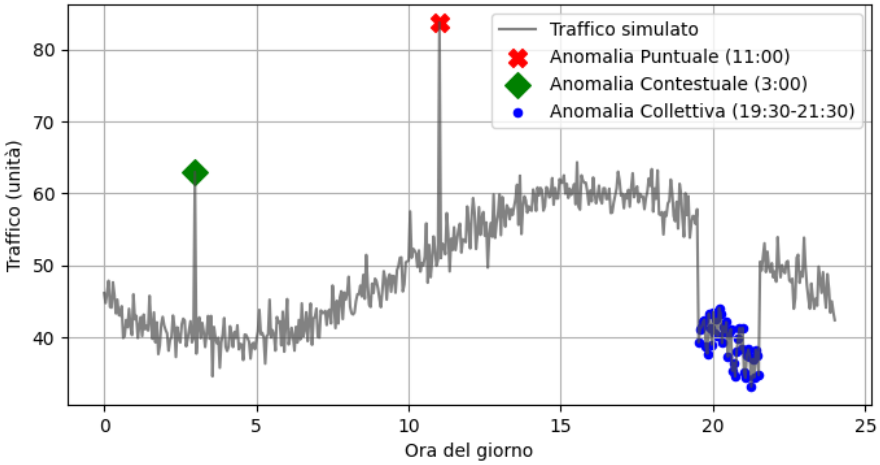


Figura 3.1: Esempi di anomalie in una serie temporale simulata di traffico.

- **Anomalie Contestuali:** Questo tipo di anomalie si distinguono dal precedente perché non si discostano complessivamente dagli altri dati, ma dal contesto in cui si trovano.

- **Anomalie Collettive:** Queste anomalie coinvolgono un gruppo di dati che, sebbene singolarmente potrebbero non risultare anomali, insieme costituiscono un pattern inusuale rispetto al comportamento atteso.

La Figura 3.1 mostra una simulazione del traffico giornaliero contenente esempi dei principali tipi di anomalie. Alle ore 3:00 è evidenziata un'anomalia contestuale: il valore si discosta dal comportamento atteso per quell'orario della notte, pur rientrando nei valori osservati durante la giornata. Alle 11:00 è presente un'anomalia puntuale, che si distingue nettamente da tutti gli altri valori della serie, risultando anomala in ogni contesto. Infine, nell'intervallo tra le 19:30 e le 21:30, si osserva un'anomalia collettiva. I valori in questo intervallo risultano bassi rispetto all'andamento tipico del traffico in quella fascia oraria. Singolarmente, questi valori potrebbero essere interpretati come anomalie contestuali, ma il loro susseguirsi e la coerenza dell'anomalia nel tempo indicano un comportamento anomalo di tipo collettivo.

3.2 Anomaly Detection

L'Anomaly Detection è un processo che punta ad identificare automaticamente gli outliers in un gruppo di dati, sfruttando modelli statistici, di deep learning o di machine learning. L'individuazione di outliers non si basa solo sull'algoritmo, ma include diverse fasi, dalla raccolta e preprocessing dei dati, alla costruzione di un modello di normalità, fino all'individuazione delle anomalie. In molti casi questa rilevazione non è sufficiente e deve essere accompagnata da un'ulteriore verifica da parte di esperti.

3.2.1 Sfide

Un'anomalia è un dato che si discosta da ciò che è considerato normale: un approccio classico consiste nel definire una regione di valori "normali" e considerare anomale tutte le osservazioni al di fuori di essa. Tuttavia, l'applicazione di questo approccio si scontra con diverse sfide operative che ne complicano l'impiego[4].

Le principali difficoltà si riducono a tre aspetti:

- **Dati scarsi e rumorosi:** le anomalie, per loro natura, si presentano come eventi rari e difficili da reperire, mentre il rumore e i valori mancanti rendono complessa la distinzione tra fluttuazioni normali e veri outlier.
- **Normalità dinamica o concept drift:** il comportamento considerato "normale" può variare nel tempo rendendo obsoleti i modelli statici e richiedendo l'impiego di tecniche di aggiornamento continuo.
- **Affidabilità e trasparenza:** nei contesti critici non basta individuare un'anomalia, occorre anche comprenderne le cause. In ambiti ostili gli attaccanti possono mascherare i propri comportamenti, richiedendo modelli resistenti e allo stesso tempo interpretabili.

L'Anomaly Detection richiede la scelta e l'adattamento di tecniche mirate alle caratteristiche dei dati e agli obiettivi applicativi, trovando il giusto equilibrio tra accuratezza, scalabilità e interpretabilità.

3.2.2 Livelli di supervisione

Per effettuare Anomaly Detection si può ricorrere a tecniche differenti, scegliendo il metodo più adatto innanzitutto in base alla natura dei dati. Una prima suddivisione di queste tecniche può essere fatta sulla base dalla presenza o meno di un dataset etichettato.

L'etichettatura dei dati, consiste nell'assegnare a ciascuna osservazione di un dataset un valore che la classifica come "normale" o "anomala". Questo processo risulta molto costoso e complesso perché richiede l'intervento manuale di esperti.

A partire da questa distinzione iniziale, i principali approcci sono:

- **Metodi supervisionati.** Questi metodi apprendono un modello da un dataset etichettato per poi classificare nuove osservazioni. Offrono buone prestazioni ma richiedono che il dataset etichettato riporti un esempio di ogni possibile tipo di anomalia che nella pratica è molto difficile da ottenere.

- **Metodi semi-supervisionati.** Partono da un dataset contenente solo osservazioni normali etichettate, così da costruire un modello del comportamento normale, per poi rilevare deviazioni significative che indicano la presenza di anomalie. Sono utili quando le anomalie sono rare e difficili da etichettare.

- **Metodi non supervisionati.** Non richiedono dati etichettati e si basano su assunzioni implicite. Questi metodi sono utili nei contesti reali ma possono essere più suscettibili a rumore o fluttuazioni dei dati.

3.2.3 Output dei metodi

Un aspetto importante per qualsiasi tecnica di Anomaly Detection è il modo in cui le anomalie vengono restituite. In generale, l'output di questi metodi si suddivide in due tipologie principali: score numerici o etichette binarie.

Score. Le tecniche basate sullo score assegnano ad ogni istanza un valore numerico che quantifica il "grado di sospetto" di anomalia: valori più elevati corrispondono a una probabilità maggiore che l'istanza sia anomala. Gli score permettono di individuare gli outlier secondo due modalità:

1. selezione dei primi k istanti con score più alto;

2. applicazione di una soglia τ (threshold): si considerano anomale tutte le istanze per cui $score > \tau$.

Etichette. Alcuni algoritmi forniscono direttamente una classificazione binaria: ogni punto viene marcato come "normale" o "anomalo". A differenza dello scoring, non è possibile applicare una threshold, ma il numero di istanze segnalate può comunque essere deciso tramite la scelta dei parametri del modello. Queste etichette si distinguono da quelle descritte nella Sezione 3.2.2, in quanto generate dal modello, mentre le altre sono fornite a priori da esperti del dominio.

Nonostante l'approccio basato su etichette permetta un utilizzo più semplice, gli score offrono maggiore granularità e flessibilità. Grazie alla misura continua del grado di anomalia, è possibile calibrare soglie in funzione di obiettivi specifici (ad esempio minimizzare falsi positivi o falsi negativi) o valutare le prestazioni complessive di un modello. Questo rende le tecniche basate sullo scoring particolarmente adatte sia all'esplorazione iniziale dei dati, sia in contesti che richiedono un'analisi più approfondita delle anomalie.

3.2.4 Categorie metodologiche

Un ulteriore suddivisione delle tecniche di Anomaly Detection può avvenire in base al meccanismo con cui si individuano le deviazioni dal comportamento "normale" dei dati.

Di seguito vengono introdotte le categorie principali.

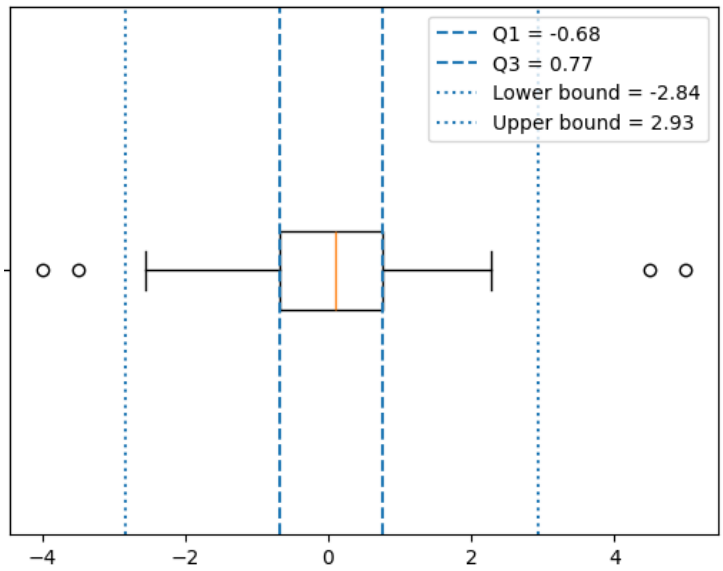


Figura 3.2: Boxplot che evidenzia Q1, Q3 e i due limiti IQR usati per individuare gli outlier.

Metodi basati sulla statistica

Questi metodi si fondano sull'assunzione che i dati normali seguano una certa distribuzione nota o stimabile. Per individuare le anomalie si analizzano le code di tale distribuzione, identificando come outlier i valori troppo distanti dalla media o dai quartili.

Una tecnica è lo *Z-score*, che considera i dati generati da una distribuzione Gaussiana e assegna gli score calcolando la distanza tra la media stimata e l'istanza stessa. Il valore soglia che separa i valori normali e quelli anomali o *threshold* viene spesso fissata a 3σ.

Altre tecniche di questo tipo sono basate sull'*Interquartile Range* (IQR) che definiscono l'intervallo sottostante e trattano come anomali i punti che ne ricadono al di fuori.

$$[Q_1 - 1.5 \cdot IQR,\ Q_3 + 1.5 \cdot IQR]$$

In Figura 3.2 è riportato un boxplot realizzato su un dataset di esempio, che mostra l'individuazione degli outlier mediante il metodo IQR.

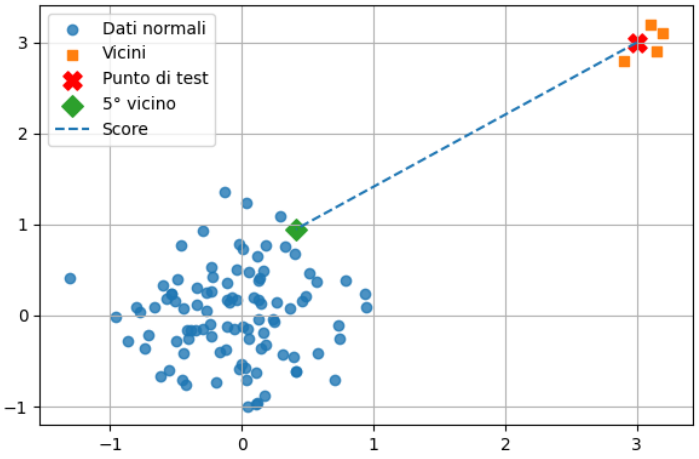


Figura 3.3: Esempio di applicazione di k-NN con $k = 5$.

Metodi basati sulla distanza

Questa tipologia di metodi si fonda sul calcolo delle distanze tra le istanze del dataset, partendo dall'assunto che i punti anomali siano caratterizzati da distanze maggiori rispetto ai normali.

In particolare, molte tecniche misurano per ogni istanza la distanza dai suoi k vicini più prossimi. Nel classico k-Nearest Neighbors (k-NN), lo score di un punto corrisponde esattamente alla distanza al k-esimo vicino. Altre varianti, invece, definiscono lo score come la media delle distanze ai primi k vicini, riducendo la sensibilità a singole distorsioni nei dati. Inoltre, la scelta della metrica di distanza può influire in modo determinante sull'efficacia del metodo.

Un vantaggio di questi metodi è che non fanno assunzioni sui dati, tuttavia il loro costo computazionale può essere elevato per dataset di grandi dimensioni.

Nella Figura 3.3 è illustrato un esempio di applicazione di k-NN con $k = 5$. Si nota il punto di test (in rosso) affiancato dai primi quattro vicini più prossimi (in arancione), mentre il quinto vicino (in verde) si trova a una distanza significativamente maggiore. La linea tratteggiata collega il punto di test proprio a questo quinto vicino, definendo così lo score di anomalia. Se fosse stato impostato $k = 4$, lo score sarebbe risultato molto più basso, facendo

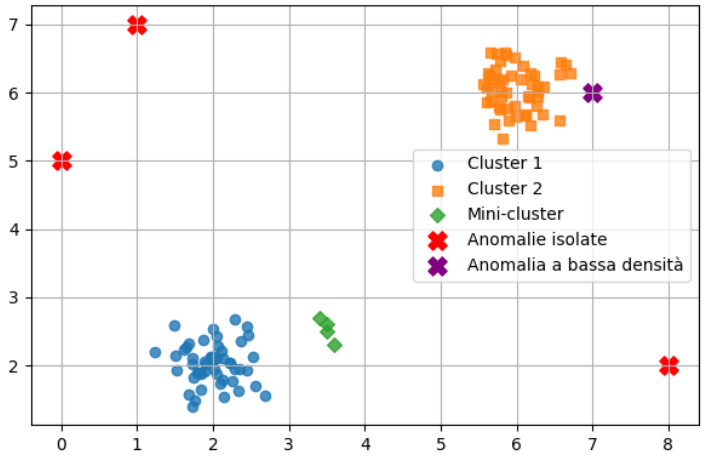


Figura 3.4: Esempio di applicazione di clustering con $k = 3$.

passare inosservate potenziali anomalie come quella evidenziata. Questo sottolinea l'importanza della scelta di k: un valore troppo basso o troppo alto può infatti far sfuggire anomalie reali o classificare come outlier punti che in realtà appartengono alla densità "normale" dei dati.

Metodi basati su densità e clustering

I metodi basati su densità e clustering condividono l'idea che i punti normali si trovino in regioni dense e ben definite, mentre le anomalie tendono ad essere isolate o appartenere a cluster piccoli o sparsi.

Uno dei metodi più noti è Local Outlier Factor (LOF), che misura quanto la densità locale di un punto differisca da quella dei suoi vicini, considerando anomali i punti significativamente più isolati.

Il *k-means* è un esempio di metodo basato sul clustering: in questo caso, i punti molto distanti dal centro del cluster più vicino, o appartenenti a cluster molto piccoli, possono essere considerati anomali. Tuttavia, questi metodi possono essere fortemente influenzati dalla scelta del numero di cluster e dalla forma delle distribuzioni.

Nella Figura 3.4 viene mostrato un esempio di clustering con $k = 3$, in cui alcune istanze (in rosso) sono considerate anomale perché isolate rispetto ai tre cluster principali. Un'altra istanza, evidenziata in viola, è anch'essa

classificata come anomalia a causa della bassa densità locale.

Questo esempio evidenzia alcuni aspetti fondamentali dei metodi basati su densità e clustering, tra cui la scelta di k (numero di cluster). In particolare, impostare $k = 3$ ha permesso di riconoscere come valido anche il mini-cluster composto dai punti verdi, invece con $k = 2$ quelle quattro istanze sarebbero state considerate anomale.

Metodi basati su machine learning e deep learning

I metodi basati su machine learning e deep learning utilizzano modelli flessibili per apprendere il comportamento normale dei dati e identificare le anomalie come deviazioni rispetto a ciò che è stato appreso. Possono essere supervisionati, semi-supervisionati o non supervisionati, a seconda della disponibilità di etichette nel dataset. Nei metodi supervisionati, il modello impara a distinguere tra dati normali e anomali, ma richiede un set bilanciato e rappresentativo. Per questo motivo, più spesso si usano approcci non supervisionati o semi-supervisionati.

Un esempio è Isolation Forest (IForest), che isola i punti suddividendo ripetutamente lo spazio lungo le variabili. Le anomalie, essendo meno simili alle altre istanze, tendono a essere isolate con meno suddivisioni, cioè con cammini più corti negli alberi che risultano in uno score più alto.

Questi metodi offrono una grande flessibilità e sono particolarmente utili quando si affrontano dati complessi o ad alta dimensionalità. Nonostante siano potenti e riescano a catturare strutture non lineari nei dati, richiedono un'attenta progettazione e validazione per evitare problemi.

3.2.5 Metriche di Valutazione delle Performance

La valutazione delle performance nei modelli di Anomaly Detection è fondamentale non solo per misurare l'efficacia di un metodo nell'identificare correttamente le anomalie, ma anche per confrontare in modo oggettivo le prestazioni tra modelli diversi. Quando si dispone delle etichette nel dataset, è possibile confrontare i risultati del modello con i valori reali tramite apposite metriche.

Queste metriche si basano sulla matrice di confusione 3.1, che classifica le istanze in quattro categorie: True Positives (TP), False Positives (FP), True Negatives (TN) e False Negatives (FN).

Valori reali \ Valori predetti	Anomalia	Normale
Anomalia	TP	FN
Normale	FP	TN

Tabella 3.1: Matrice di confusione

Le metriche più utilizzate sono:

- **Precision**:
$$\text{Precision} = \frac{TP}{TP + FP}$$
Indica la proporzione di elementi correttamente identificati come anomali sul totale delle predizioni anomale. Una precisione elevata implica pochi falsi positivi.

- **Recall**:
$$\text{Recall} = \frac{TP}{TP + FN}$$
Rappresenta la proporzione di anomalie effettivamente individuate rispetto al totale delle anomalie reali. Valori alti indicano pochi falsi negativi.

- **F1-score**:
$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$
È la media armonica tra precision e recall. È utile in scenari in cui è necessario bilanciare entrambi gli aspetti.

- **AUC-ROC (Area Under the ROC Curve)**:
$$\text{TPR} = \frac{TP}{TP + FN}, \quad \text{FPR} = \frac{FP}{FP + TN}$$
$$\text{AUC\_ROC} = \int_0^1 TPR(FPR)\, dFPR$$

L'AUC (Area Under the Curve) rappresenta l'area sotto la curva ROC, che mette in relazione il tasso di veri positivi (TPR) con il tasso di falsi positivi (FPR) al variare della soglia decisionale o threshold. Un valore di AUC pari a 1 indica una classificazione perfetta, mentre un valore vicino a 0.5 indica prestazioni casuali. È particolarmente utile nei problemi sbilanciati, come l'Anomaly Detection, perché valuta il modello indipendentemente dalla soglia scelta.

In sintesi, l'uso di più metriche consente di valutare i modelli a seconda degli obiettivi, in particolare in contesti sbilanciati come l'Anomaly Detection è importante utilizzare anche metriche come l'AUC-ROC, che forniscono misure più robuste e senza la necessità di impostare una soglia.

3.2.6 Anomaly Detection su Serie Storiche

Nel contesto delle serie temporali, l'Anomaly Detection presenta sfide specifiche, dato che i dati sono ordinati temporalmente, possono mostrare autocorrelazione e spesso presentano stagionalità e trend. Queste caratteristiche rendono meno efficaci o inutili molti metodi generici. L'Anomaly Detection su serie storiche può essere affrontata con due approcci differenti:

1. **Modelli basati su previsione:** si costruisce un modello che apprende trend, stagionalità e dipendenze temporali, e segnala come anomalie le istanze con residui significativamente maggiori dagli altri (ad esempio LSTM).

2. **Approcci a finestre mobili:** La serie viene suddivisa in finestre temporali e ciascuna finestra trattata come una singola istanza in uno spazio di dimensione $\mathbb{R}^w$, infine viene confrontata con le altre finestre per assegnarle uno score.

Modelli basati su previsione

I modelli ARIMA, SARIMA e ARIMAX apprendono trend, stagionalità e dipendenze autoregressive, e individuano anomalie quando l'errore di previsione supera soglie statistiche sui residui. Questi approcci sono interpretabili

e si basano su residui univariati, ma soffrono se la struttura della serie cambia frequentemente.

Sliding windows

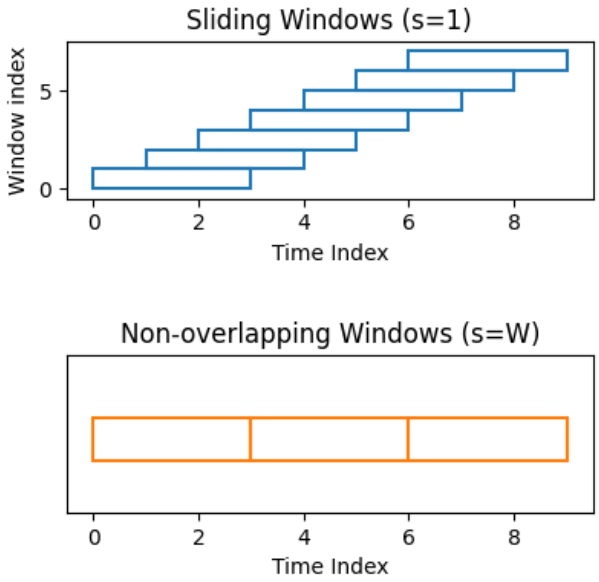


Figura 3.5: Esempio di finestre sovrapposte e non sovrapposte

Nell'approccio a finestre mobili si sceglie una lunghezza di finestra W e la si fa "scorrere" con passo $s = 1$ sull'intera serie di lunghezza N. In questo modo si ottengono $N - W + 1$ sotto-serie temporali, ciascuna rappresentata come un vettore di W osservazioni. Su questo nuovo dataset di vettori, che non rappresentano più una serie storica, si applica un metodo di Anomaly Detection che confronta ogni vettore con tutti gli altri e calcola uno score di anomalia. Lo score attribuito alla finestra viene poi associato all'elemento centrale di essa, garantendo che ogni punto della serie originale riceva un singolo valore di anomalia. Questo metodo conserva un'elevata granularità temporale ma comporta la generazione di $N - W + 1$ score e aumentando notevolmente la dimensionalità del problema, di conseguenza, anche il costo computazionale cresce significativamente.

Un caso particolare si verifica impostando il passo pari alla lunghezza della finestra $s = W$, dividendo il dataset in finestre non sovrapposte. In questo caso lo score viene assegnato a tutti i W punti al suo interno. Questo riduce drasticamente il numero di score prodotti (approssimativamente N/W), evidenziando anomalie collettive su intervalli fissi, a costo però di perdere l'elevata granularità del caso con $s = 1$.

Conclusioni: In sintesi, l'approccio tradizionale con passo $s = 1$ è ideale per attribuire a ciascun punto uno score basato sul contesto immediato e garantire un'alta granularità temporale, mentre il caso con $s = W$ consente di ottenere valutazioni collettive e ridurre il numero totale di score. Nelle analisi del Capitolo 5 sono state sfruttate entrambe le opzioni.

Capitolo 4

Metodi Selezionati

In questo capitolo vengono descritti in dettaglio i tre metodi scelti per le analisi presentate nel capitolo successivo. Tutti e tre gli approcci sono utilizzati in modalità non supervisionata, consentendo la rilevazione di anomalie senza bisogno di etichette predefinite.

- **k-Nearest Neighbors (k-NN):** rappresenta la categoria dei metodi basati sulla distanza, scelto per la sua semplicità implementativa e la versatilità nell'individuare sia anomalie puntuali sia collettive.
- **Isolation Forest (iForest):** metodo basato su machine learning, selezionato per l'efficienza computazionale e la capacità di scalare su dataset di grandi dimensioni
- **AutoEncoder:** metodo basato su deep learning, scelto per la capacità di catturare pattern non lineari e relazioni complesse tra le variabili.

Nelle sezioni successive verranno analizzati i metodi elencati, descrivendo i parametri principali e mettendo in evidenza le loro principali limitazioni nell'ambito dell'Anomaly Detection (Sez. 3.2.1).

4.1 k-Nearest Neighbors

K-NN è uno dei metodi classici di Anomaly Detection non supervisionata. Si basa sull'idea che le istanze anomale si trovino in regioni di bassa densità.

Score di anomalia

Con k-NN lo score viene quantificato dalla distanza tra il punto e il suo k-esimo vicino:

$$s_k(x) = d(x, \mathrm{NN}_k(x))$$

dove $\mathrm{NN}_k(x)$ indica il k-esimo vicino più prossimo di x secondo la distanza d. È possibile utilizzare diverse metriche di distanza, come la distanza euclidea, manhattan o altre, a seconda della struttura dei dati.

Scelta del parametro k

Come già citato nella Sezione 3.2.4 la scelta di k è fondamentale e può condizionare fortemente i risultati:

- k troppo piccolo $\rightarrow$ sensibilità a outlier puntuali e rumore.
- k troppo grande $\rightarrow$ perdita di sensibilità alle anomalie locali o contestuali.

Le strategie più comuni per scegliere k sono:

- **Curva k-dist:** per ogni valore di k in un intervallo prestabilito, si calcolano gli score $\{s_k(x_i)\}$ per tutti i punti, si ordinano in ordine decrescente e si traccia la curva risultante. Analizzando la seconda derivata della curva si individua il "gomito", che quantifica il punto di massima separazione tra potenziali anomalie (score elevati) e comportamento normale (score bassi). Il k corrispondente al gomito più pronunciato viene scelto come ottimale. Questo procedimento è però computazionalmente oneroso, poiché richiede di ripetere il calcolo delle distanze e l'ordinamento dei punteggi per ogni valore di k.
- **Cross-validation:** se si dispone delle etichette in almeno una parte del dataset, è possibile scegliere il valore di k che massimizza AUC-ROC o F1-score.
- **Adaptive** k: per ogni punto x_i si definisce un k_i in base alla densità locale.

Complessità computazionale

Il metodo k-NN presenta una complessità computazionale pari a $\mathcal{O}(N^2)$ in fase di scoring, dove N è il numero di istanze. Questo perché per ogni punto del dataset è necessario calcolare la distanza da tutti gli altri. Con dataset di grandi dimensioni, il costo computazionale cresce rapidamente, rendendo l'approccio poco scalabile senza l'adozione di tecniche di ottimizzazione.

Ottimizzazione tramite Partizionamento

Un'ottimizzazione al metodo k-NN è proposta in Ramaswamy et al. (2000) [6], dove viene introdotto un algoritmo che consente di evitare il calcolo completo di tutti gli score, selezionando direttamente gli n punti più anomali.

L'idea alla base di questo algoritmo partition-based è quella di suddividere il dataset in sottoinsiemi disgiunti, che vengono progressivamente scartati se si determina che non possono contenere outlier. La partizione è realizzata tramite clustering BIRCH, e i cluster vengono progressivamente esclusi calcolando un intervallo di valori possibili per lo score $s_k(x)$ dei punti al loro interno. Se il limite inferiore dello score di tutti i punti in un cluster è superiore al punteggio del peggiore tra i top-n outlier trovati finora, il cluster può essere scartato. Al termine, il metodo k-NN standard viene applicato solo ai punti contenuti nei cluster rimanenti, riducendo così significativamente il costo computazionale.

Varianti e Metodi Simili

Esistono diverse varianti che si basano su principi simili al k-NN, con l'obiettivo di migliorare la robustezza o la sensibilità del metodo. Una è il Mean k-NN, in cui lo score di anomalia di un punto viene calcolato come la media delle distanze ai suoi k vicini più prossimi. Questo approccio tende ad essere meno sensibile a variazioni locali e al rumore.

Altri approcci includono varianti che combinano statistiche diverse (media, mediana, massimo) o incorporano la densità locale, come nel caso del Local Outlier Factor (LOF).

4.2 Isolation Forest

Isolation Forest (iForest) è un metodo non supervisionato basato su alberi binari casuali, progettato per rilevare anomalie sfruttando il principio secondo cui i punti anomali sono più facili da isolare rispetto a quelli normali [7]. Questo perché gli outlier si trovano in regioni scarsamente popolate e lontane dal resto dei dati, rendendoli distinguibili con meno partizioni.

Score di anomalia

Il modello costruisce T alberi detti iTree, ognuno dei quali partiziona ricorsivamente i dati scegliendo casualmente una feature e un valore di split, fino a isolare ogni osservazione. I punti anomali tendono a essere isolati rapidamente, cioè ad essere foglie di profondità minori.

Per ogni istanza x, si calcola la profondità media $h(x)$ con cui viene isolata nei T alberi. Lo score di anomalia è definito come:

$$s(x, n) = 2^{-\frac{h(x)}{c(n)}},$$

dove n è la dimensione del sottoinsieme campionato per ciascun albero, e $c(n)$ rappresenta il valore atteso della lunghezza del percorso in un albero binario.

Complessità computazionale

Una delle principali caratteristiche di Isolation Forest è la sua efficienza. La costruzione di ogni albero avviene selezionando a caso una feature e un valore di split, di conseguenza, la complessità per costruire un singolo albero è $\mathcal{O}(n \log n)$. Costruendo T alberi, la complessità totale diventa $\mathcal{O}(Tn \log n)$. Il calcolo dello score di anomalia per ogni punto ha una complessità $\mathcal{O}(T \log n)$, rendendo iForest particolarmente adatto a dataset di grandi dimensioni.

Sintesi

Isolation Forest è un metodo efficace e scalabile per l'anomaly detection non supervisionata, grazie alla sua complessità lineare rispetto al numero di campioni n e al fatto che non richiede l'uso di metriche di densità o distanza.

4.3 AutoEncoder

Gli autoencoder sono reti neurali non supervisionate progettate per apprendere rappresentazioni compresse dei dati, con l'obiettivo di ricostruire l'input originale. Questa capacità di ricostruzione li rende strumenti efficaci per il rilevamento di anomalie: dopo l'addestramento su dati normali, gli autoencoder tendono a ricostruire accuratamente i dati simili a quelli visti durante il training, mentre mostrano errori di ricostruzione più elevati per dati anomali, che differiscono dalle caratteristiche apprese.

Poiché l'approccio è non supervisionato, il training può includere anche dati anomali, tuttavia, si assume che le anomalie siano rare e quindi che l'autoencoder impari a ricostruire bene solo le strutture normali.

È possibile addestrare l'autoencoder su un set di dati normali distinto da quello utilizzato per il calcolo degli score, migliorando ulteriormente la capacità di identificare anomalie.

Training del modello e calcolo degli score

Il training si svolge in E epoche e, dato l'input x, consiste in:

$$x \xrightarrow{\text{Encoder}} z \xrightarrow{\text{Decoder}} \hat{x}$$

Al termine di ogni epoca viene calcolata la perdita (loss) e aggiornati i pesi:

$$\mathcal{L} = \|x - \hat{x}\|^2$$

Una volta completato l'allenamento, lo score di anomalia di ciascun punto viene definito come:

$$s(x) = \|x - \hat{x}\|$$

Parametri principali

- **Dimensione del bottleneck (m)**: determina quanto i dati vengono compressi, impatta direttamente sulla qualità della rappresentazione e sulla discriminazione delle anomalie.
- **Learning rate**: regola la velocità di aggiornamento dei pesi, va scelto per bilanciare rapidità di convergenza e stabilità.
- **Numero di epoche (E)**: definisce quante volte viene ripetuto il training sui dati.

Complessità computazionale

La fase di training richiede un costo di ordine $\mathcal{O}\big(n \times d \times E\big)$ dove n è il numero di esempi, d la dimensione dell'input e E il numero di epoche. La fase di calcolo degli score è invece $\mathcal{O}(n \times d)$, risultando più veloce e adatta a scenari in cui i dati arrivano in tempo reale.

Innovazioni e varianti

- **MDL:** integra il principio di Minimum Description Length con l'energia libera di Helmholtz per bilanciare qualità della ricostruzione e semplicità della codifica [8].
- **Sparse AutoEncoders:** adotta un modello di energia che impone sparsità nelle unità latenti, attivandone poche per ogni input [9].
- **Stacked Denoising AutoEncoders:** propone un training che impiega autoencoder impilati che ricostruiscono dati corrotti, migliorando la robustezza delle rappresentazioni [10].

Sintesi

Gli autoencoder sono particolarmente adatti in scenari con dati complessi o non lineari dove modelli basati su distanze o statiche risultano inefficaci. I loro punti di forza includono la capacità di apprendere rappresentazioni compatte e non lineari e la flessibilità. Sono più efficaci quando si dispone di una buona quantità di dati normali per l'addestramento e le anomalie sono rare e diverse. Tuttavia, richiedono una fase di training più lunga rispetto ad altri metodi e una selezione dei parametri.

Capitolo 5

Analisi Comparative

In questo capitolo viene presentata la pipeline implementata per l'analisi dei dati, insieme ai risultati ottenuti su una selezione di dataset temporali etichettati. L'obiettivo è identificare il modello di Anomaly Detection più adatto a questi dati e, successivamente, determinare la soglia ottimale per la classificazione delle anomalie attraverso un confronto tra diverse metriche di valutazione.

5.1 Set-up sperimentale

L'intera sperimentazione è stata realizzata con Python, scelto per la sua flessibilità e per la disponibilità di librerie specializzate utili in tutte le fasi dell'analisi: dal caricamento e preprocessing dei dati fino alla valutazione delle prestazioni dei modelli di Anomaly Detection.

5.1.1 Ambiente e strumenti

Per garantire la riproducibilità del codice e la gestione delle dipendenze, l'analisi è stata eseguita all'interno di un ambiente virtuale. Sono state impiegate diverse librerie open-source, ciascuna selezionata per specifiche funzionalità utili per svolgere l'analisi.

TSB-UAD

TSB-UAD (Time Series Benchmark) è una suite di benchmark end-to-end dedicata alla valutazione di metodi di Anomaly Detection su serie temporali univariate. In questo lavoro sono state utilizzate solo delle sue funzioni principali: `find_length()`, utile per determinare automaticamente la lunghezza ottimale delle finestre, e `Window()`, che consente di segmentare le serie temporali. Per le altre fasi della pipeline sono state preferite funzioni più adatte agli obiettivi specifici dell'analisi, provenienti da altre librerie o sviluppate ad hoc.

PyOD

PyOD è una libreria Python specializzata nell'Anomaly Detection su dati multivariati, con oltre 40 algoritmi implementati, tra cui quelli utilizzati in questa tesi. È stata scelta per l'ampia disponibilità di metodi e per l'uniformità nel loro utilizzo.

Scikit-learn

Scikit-learn è uno dei principali framework per il machine learning in Python. In questo lavoro è stato impiegato principalmente per:

- `MinMaxScaler` e `StandardScaler`, utilizzati per scalare i dati.
- `roc_auc_score`, `precision_score`, `recall_score` e `f1_score`, impiegati per valutare le prestazioni dei modelli e delle soglie.

Altre librerie

Oltre alle librerie principali già menzionate, sono state utilizzate anche:

- **Pandas** e **NumPy**, fondamentali per la gestione, manipolazione ed elaborazione dei dati.
- **Matplotlib** e **Seaborn**, utilizzate per creare grafici utili all'analisi.

5.1.2 Pipeline dell'analisi

La pipeline dell'analisi è suddivisa in due file: il primo contiene l'implementazione delle funzioni principali per l'esecuzione dell'analisi, mentre il secondo gestisce i parametri di configurazione, permettendo all'utente di specificare le scelte relative ai modelli, alle soglie, ai dataset, alla visualizzazione dei risultati e ad altri aspetti.

Il programma gestisce i casi in cui non ci sono anomalie, sia quando il dataset non ha etichette, sia quando le etichette sono tutte zero. In questi casi, la pipeline viene eseguita comunque, ma senza calcolare le metriche di valutazione e adattando meglio i grafici ai risultati.

Di seguito sono descritte le fasi principali dell'analisi.

Lettura e preprocessing dei dati

La fase iniziale dell'analisi prevede la lettura dei dati grezzi da file, seguita da un'opportuna fase di preprocessing.

Nel caso di dataset univariati è necessario che i dati vengano trasformati mediante un'operazione di `reshape`, convertendo la lista di valori in una matrice con una singola colonna. Questa trasformazione è necessaria poiché i metodi di anomaly detection applicati successivamente richiedono che l'input sia strutturato in forma matriciale.

I dati vengono infine standardizzati per assicurare stabilità e prestazioni consistenti dei metodi di Anomaly Detection.

```
1 data=StandardScaler().fit_transform(data.reshape(-1,1))
```

Creazione delle finestre

Dopo la lettura e il preprocessing, i dati univariati vengono suddivisi in sequenze di lunghezza fissa tramite una procedura di windowing. Questa trasformazione consente di convertire una serie temporale in un insieme di vettori sovrapposti o disgiunti, a seconda del parametro `slide` impostato nel file di configurazione.

Se `slide = True`, le finestre vengono create con uno *step* pari a 1, producendo sequenze sovrapposte. Se invece `slide = False`, lo *step* è pari alla lunghezza della finestra w, generando così finestre disgiunte. Lunghezza w viene stimata automaticamente attraverso la funzione `find_length`, questa funzione, implementata dalla collezione TSB-UAD, sfrutta l'autocorrelazione per individuare un valore periodico ottimale nella serie temporale.

Il windowing viene effettuato mediante la classe `Window`, anch'essa proveniente da TSB-UAD. Questa classe costruisce, a partire da una serie unidimensionale, un DataFrame in cui ciascuna riga rappresenta una finestra di lunghezza w.

Un esempio del processo di creazione delle finestre è il seguente, in cui `W_data` è la matrice risultato del windowing:

```
len_Win = find_length(data)
step = 1 if slide else len_Win
W_data = Window(window=len_Win).convert(data)[::step].
    to_numpy()
```

Poiché la classe Window genera soltanto finestre complete di lunghezza w, qualsiasi campione residuo alla fine della serie nel caso con `slide=False`, viene automaticamente scartato, ciò non costituisce un problema in quanto il numero di osservazioni eliminate sarà probabilmente trascurabile.

Definizione del modello

Dopo il preprocessing, i dati vengono passati a una funzione che si occupa di definire il modello e calcolare gli score di anomalia. Come accennato in precedenza, i metodi offerti da PyOD condividono un'unica interfaccia, il che permette di riutilizzare lo stesso codice per l'addestramento, variando unicamente la definizione del modello.

La funzione riceve in input il nome del modello da applicare e calcola anche il tempo di addestramento, utile per confrontare l'efficienza dei diversi algoritmi. Tra i modelli supportati è previsto anche l'utilizzo dell'opzione *'Best_KNN'*, che richiama una funzione dedicata alla selezione automatica del parametro k, stimato tramite la tecnica della curva k-dist (descritta in 4.1). Tuttavia, questo approccio risulta computazionalmente costoso, poi-

ché richiede il calcolo degli score del modello KNN per un intervallo di valori di k, per questo motivo non è stato utilizzato per le analisi successive.

Di seguito un esempio con il modello AutoEncoder, in cui si definisce una struttura simmetrica con layer nascosti [64, 32], corrispondente a un bottleneck di dimensione 32, e si imposta il numero di epoche a 10. Il modello viene addestrato e vengono calcolati gli score:

```
clf = AutoEncoder(hidden_neuron_list=[64, 32], epoch_num=10)
clf.fit(W_data)
score = clf.decision_scores_
```

Postprocessing

Una volta definiti i modelli e calcolati gli score di anomalia, questi ultimi vengono scalati tramite `MinMaxScaler` in un intervallo $[0, 1]$, per renderli più interpretabili e facilitare l'applicazione delle soglie:

```
score = MinMaxScaler((0,1)).fit_transform(score.reshape(-1,1)
    ).ravel()
```

Se `slide=True` ($step = 1$), il windowing genera $n - w + 1$ finestre e quindi $n - w + 1$ score. Per riallineare gli score alla serie originale, vengono aggiunte $\lceil (w-1)/2 \rceil$ copie del primo valore all'inizio e $\lfloor (w-1)/2 \rfloor$ dell'ultimo alla fine. Questo permette di riallineare correttamente gli score con le istanze originali, considerando che il primo score prodotto dal modello non corrisponde alla prima osservazione del dataset originale, ma all'osservazione in posizione $\lfloor w/2 \rfloor$ cioè al centro della prima finestra.

```
if slide:
    pad_left = [score[0]] * math.ceil((len_Win-1)/2)
    pad_right = [score[-1]] * ((len_Win-1)//2)
    score = np.array(pad_left + list(score) + pad_right)
```

Se invece `slide=False` ($step = w$), sono le etichette ad essere modificate, creando un'unica etichetta per ciascuna finestra, pari a 1 se contiene almeno un'anomalia, 0 altrimenti:

```
label = W_label.any(axis=1).astype(int)
```

Calcolo delle metriche

Dopo il postprocessing, che riallinea gli score con le etichette, vengono calcolate le metriche di valutazione delle performance (descritte in 3.2.5) per i dataset etichettati. La funzione per il calcolo delle metriche è stata progettata in modo da consentire la scelta della soglia (`threshold`) da utilizzare per classificare le istanze.

Una volta selezionata la soglia, vengono generate le "predizioni": ogni istanza per cui $score > threshold$ viene classificata come anomala (1), altrimenti come normale (0). Sulla base di queste predizioni, vengono poi calcolate le quattro metriche principali utilizzando le funzioni di *Scikit-learn*: AUC, precision, recall e F1-score.

```
predictions = (score >= threshold).astype(int)
auc = roc_auc_score(labels, score)
prec = precision_score(labels, predictions, zero_division=0)
rec = recall_score(labels, predictions)
f1 = f1_score(labels, predictions)
```

Visualizzazione dei risultati

La visualizzazione dei risultati avviene tramite l'utilizzo delle funzioni che seguono:

- `plotFig_new` è una versione modificata della funzione `plotFig`, originariamente presente nella libreria TSB-UAD. La modifica è stata necessaria per adattare la visualizzazione alle esigenze dell'analisi, dato che la funzione originale utilizzava una soglia fissa pari a $\mu + 3\sigma$ e non ne permetteva la modifica. Inoltre, la nuova versione è utilizzabile anche con dati non etichettati: in questo caso nei grafici non vengono visualizzate le anomalie reali.

  Questa funzione genera un grafico composto da 3 subplot:

  1. **Dati originali:** Il primo subplot mostra la serie temporale dei dati. Se sono presenti etichette, le istanze corrispondenti alle anomalie reali vengono evidenziate in rosso.

2. **Anomaly scores:** il secondo subplot visualizza gli score di anomalia calcolati dal modello, con una linea rossa orizzontale che rappresenta la threshold selezionata.

3. **Predizioni:** il terzo subplot mostra le istanze classificate dal modello distinguendole in base al confronto tra predizione e realtà: True Positive (blu), False Positive (verde), False Negative (rosso) e True Negative (nero). Se i dati non sono etichettati vengono evidenziate le istanze rilevate come anomale in rosso o normali in nero.

In alto a destra del grafico vengono riportati i valori delle quattro metriche calcolate in precedenza e la soglia utilizzata. Questa visualizzazione è utile per valutare visivamente l'efficacia del modello e capire se la threshold selezionata separa correttamente le anomalie.

- `plot_auc_swarmplot` è una funzione basata su `swarmplot` di Seaborn, che permette di visualizzare e confrontare le performance dei diversi metodi con l'AUC calcolato su diversi dataset.

- Sono state implementate anche funzioni che generano tabelle con i risultati delle analisi, convertite automaticamente in codice LaTeX tramite il metodo `.to_latex` di pandas.

5.1.3 Dataset utilizzati

Per questa analisi comparativa è stata utilizzata la collezione TSB-UAD, un benchmark aperto per l'anomaly detection su serie temporali univariate. La collezione include dataset provenienti da 18 diversi domini, con anomalie di tipologie diverse.

Questi 18 dataset provengono da una varietà di domini reali, come per esempio la telemetria spaziale (NASA-SMAP e NASA-MSL), l'elettrocardiografia (ECG) e sensori ambientali (SensorScope). Sono stati scelti per rappresentare differenti contesti applicativi e includono serie temporali con anomalie eterogenee, rendendo il benchmark adatto a valutare la robustezza

dei metodi di anomaly detection in diversi casi. La difficoltà nel rilevare alcune anomalie in questi dati è causata dal fatto che sono state etichettate da esperti in contesti complessi. Queste anomalie possono essere sottili e rare, rendendo difficile la loro identificazione automatica. Uno di questi dataset (NAB) risulta non contenere anomalie etichettate, quindi non verrà incluso nelle analisi successive in quanto non è possibile calcolarne le metriche.

5.2 Confronto dei metodi

I metodi confrontati in questa analisi sono IForest, AutoEncoder e KNN, descritti nel Capitolo 4 di questa tesi. La loro implementazione è resa possibile grazie alla libreria PyOD. Di seguito vengono descritte alcune caratteristiche specifiche del loro utilizzo tramite PyOD, insieme ad alcune scelte necessarie per questa analisi.

Un parametro comune a IForest e KNN è `n_jobs`, che, se impostato ad un valore maggiore di 1, consente di sfruttare il parallelismo multicore, riducendo i tempi di calcolo. Tuttavia nell'implementazione in PyOD, la parallelizzazione non è effettivamente applicata all'intero processo e non produce effettivi miglioramenti nei tempi di esecuzione.

Isolation Forest

Il modello IForest di PyOD si basa sull'implementazione di IsolationForest di Scikit-learn e supporta diversi parametri che permettono l'ottimizzazione. Alcuni sono `n_estimators` e `max_samples`, che regolano rispettivamente il numero di alberi e la dimensione del campione per albero. Questi parametri possono essere adattati in base al dataset per ottenere un buon compromesso tra velocità e accuratezza, in questa analisi sono stati utilizzati i valori di default, ovvero: `n_estimators = 100` e `max_samples = min(256, n_samples)`

AutoEncoder

Il modello AutoEncoder di PyOD include ottimizzazioni predefinite come Batch Normalization, Dropout (0.2) e weight decay ($1e-5$). Questi accorgimenti migliorano stabilità e generalizzazione senza aumentare significativamente la complessità computazionale.

KNN

Il metodo k-Nearest Neighbors implementato in PyOD è ottimizzato tramite l'uso della struttura dati BallTree, che consente di eseguire le ricerche dei vicini in modo efficiente, evitando la complessità quadratica.

I parametri principali da impostare per il modello KNN sono k e `metric`, che rappresentano rispettivamente il numero di vicini da considerare e la metrica di distanza da utilizzare.

Poiché il valore ottimale di k può variare in base al dataset e il metodo della curva delle k-dist (Sezione 4.1), pur essendo stato implementato nella pipeline, risulta troppo oneroso dal punto di vista computazionale, si è deciso di applicare KNN tre volte a ciascun dataset, con $k = 5$, $k = 10$ e $k = 30$.

Per il secondo parametro `metric` si è deciso di utilizzare la distanza euclidea, considerata generalmente la più efficace per misurare la somiglianza tra serie temporali, poiché preserva la struttura geometrica dello spazio e consente una migliore discriminazione tra pattern simili e diversi [11].

5.2.1 AUC a confronto

Per un confronto delle performance dei modelli si è eseguita la pipeline per tutti i dataset e tutti i modelli, ottenendo le metriche per ciascuno.

Dallo swarmplot 5.1, in cui ogni pallino rappresenta l'AUC corrispondente ad un singolo dataset, si osserva che le performance dei modelli variano sensibilmente tra i diversi dataset. Questa distribuzione era prevedibile, considerando l'eterogeneità dei dataset, che includono anomalie di natura molto diversa e a volte difficili da riconoscere.

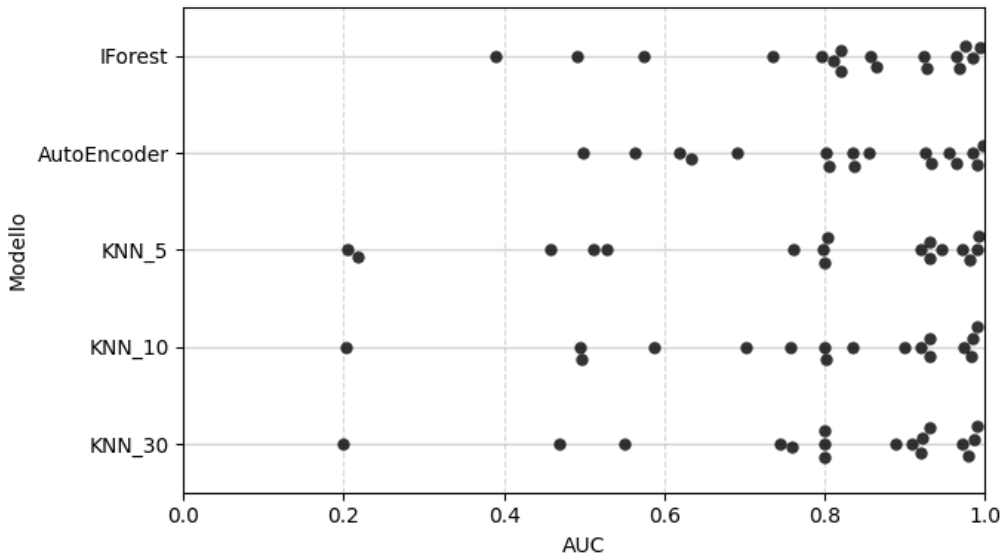


Figura 5.1: AUC per Dataset & Modello

	KNN_5	KNN_10	KNN_30
Genesis	0.529	0.702	0.909
KDD21	0.218	0.587	0.745

Tabella 5.1: Differenze più significative tra KNN

Tra KNN_5, KNN_10 e KNN_30 non emergono differenze significative nella maggior parte dei casi. Tuttavia, come mostra la tabella 5.2.1, in alcuni dataset la scelta del parametro k ha un impatto rilevante sulle prestazioni. Questo è coerente con la natura del metodo K-NN per cui valori piccoli di k rendono il modello più sensibile ad anomalie puntuali, mentre valori elevati ne riducono l'influenza. Ciò significa che i dataset in questione contengono osservazioni isolate che non dovrebbero essere classificate come anomalie e che quindi k maggiore porta a una classificazione più corretta.

La tabella 5.2.1 mostra che IForest e AutoEncoder ottengono un valore medio di AUC simile, ma AutoEncoder presenta una variabilità minore, che indica una maggiore stabilità. I modelli KNN hanno prestazioni mediamente inferiori e più instabili specialmente con $k = 5$. In sintesi, AutoEncoder risulta il modello più affidabile, combinando buona efficacia e consistenza.

model	Media	Std Err
IForest	0.818	0.179
AutoEncoder	0.817	0.161
KNN_5	0.750	0.266
KNN_10	0.782	0.222
KNN_30	0.802	0.215

Tabella 5.2: Media AUC per modello

5.2.2 Tempi di esecuzione a confronto

Durante il training dei modelli sono stati calcolati anche i tempi di esecuzione, riportati nella tabella 5.3. Per evitare ridondanze e poiché si sono dimostrati i peggiori in termini di performance, i modelli KNN_5 e KNN_10 sono stati esclusi. La tabella evidenzia chiaramente le differenze di efficienza computazionale, già presentate nel Capitolo 4 e riconducibili alle diverse complessità:

- IForest: $\mathcal{O}(T \times n \times \log n)$
- Autoencoder: $\mathcal{O}\big(n \times d \times E\big)$
- KNN: $\mathcal{O}(N^2)$

IForest risulta molto efficiente anche su dataset di grandi dimensioni, mentre AutoEncoder impiega tempi elevati anche con dataset più piccoli, a causa del suo lungo addestramento. KNN_30, nonostante l'ottimizzazione di PyOD, data la sua natura quadratica ha tempi di esecuzione che crescono rapidamente all'aumentare di n fino ad arrivare a 40 minuti per il dataset più grande.

Conclusioni

In conclusione, IForest rappresenta il miglior compromesso tra accuratezza e rapidità, offre performance elevate comparabili a quelle di AutoEncoder ma con tempi di esecuzione molto contenuti anche con dataset di grandi dimensioni.

Tabella 5.3: Tempi di esecuzione(s) per Dataset & Modello

	IForest	AutoEncoder	KNN_30	n
MITDB	5.285	1237.703	2393.638	650000
SensorScope	0.440	45.027	1.919	230400
ECG	3.093	262.172	151.792	227900
Genesis	0.219	16.064	0.709	200001
IOPS	1.357	197.256	50.768	149156
MGAB	0.678	119.327	12.910	100000
KDD21	0.565	122.705	26.031	79795
Occupancy	0.163	4.675	0.048	50670
Dodgers	0.582	55.024	16.411	50400
SMD	0.284	54.927	6.398	29553
Daphnet	0.262	34.862	1.608	28800
SVDB	2.306	457.093	160.208	28479
GHL	1.435	224.078	86.014	16220
NASA-SMAP	0.153	13.671	0.235	8640
OPPORTUNITY	0.466	85.437	3.820	2665
NASA-MSL	0.131	2.981	0.057	2264
YAHOO	0.092	1.261	0.029	1420

5.3 Scelta della soglia

Obiettivo

L'obiettivo di questa analisi è individuare una soglia (threshold) ottimale che sia in grado di adattarsi a tutti i dataset considerati. Un ulteriore scopo è verificare se le soglie comunemente utilizzate nella pratica, come IQR e $\mu+3\sigma$, siano effettivamente le più adatte. Un esempio rilevante è il benchmark TSB-UAD (Paparrizos et al., 2022) [3], in cui gli autori adottano proprio quest'ultima soglia.

Tipi di soglie

Le soglie utilizzate possono essere suddivise in due categorie principali:

- **Soglie basate sulla distribuzione:** Sono state testate diverse soglie del tipo $\mu + k\sigma$, dove μ è la media, σ la deviazione standard e k una costante. Questo approccio è semplice da implementare, ma si basa sull'assunzione di normalità dei dati, che raramente è verificata. Inoltre, risulta sensibile alla presenza di outlier, che possono influenzare significativamente i parametri, compromettendo l'efficacia della soglia.

- **Soglie basate su percentili:** Sono state testate diverse soglie basate su percentili. Questi metodi si basano sull'ipotesi implicita di conoscere in anticipo la proporzione di anomalie presenti nel dataset. Percentili troppo alti, come il 99.5%, tendono a dare cattivi risultati nei dataset che contengono numerose anomalie, mentre soglie troppo basse producono un numero eccessivo di falsi positivi.

  Sono stati considerati anche metodi robusti, meno sensibili agli outlier, tra cui:

  - **IQR:** basato sulla soglia $Q3 + 1.5 \cdot IQR$
  - **MAD:** $mediana + k \cdot MAD$, dove k è una costante e MAD è la mediana degli scarti assoluti rispetto alla mediana.

5.3.1 Risultati

Per confrontare le soglie sono state utilizzate le metriche principali (precision, recall e F1) calcolate per ogni soglia e per ogni dataset sugli anomaly scores ottenuti con IForest. È stato scelto IForest perché è risultato veloce e con buone performance in 5.2. Inoltre, si è verificato che le conclusioni tratte in questa sezione (5.3) si ottengono anche utilizzando AutoEncoder o KNN.

La pipeline è stata eseguita calcolando le metriche di valutazione per diverse soglie:

- $\mu + 3\sigma$, $\mu + 2\sigma$, $\mu + 1\sigma$, $\mu + 0.5\sigma$
- percentili (70%, 80%, 90%, 95%, 99%, 99.5%)
- IQR
- MAD3 e MAD5

I risultati completi con tutte le soglie utilizzate sono riportati nella Tabella 6.1 in appendice.

Nelle tabelle 5.4 e 5.5 sono mostrati rispettivamente i valori di precision, recall e F1 per ciascun dataset, limitatamente a un sottoinsieme di soglie ($\mu + 3\sigma$, $\mu + 1\sigma$, 90%, 99.5%, IQR e MAD3) considerate le più rilevanti per questa analisi.

Osservazioni

I risultati ottenuti mostrano una notevole variabilità sia al variare della soglia utilizzata, sia tra i diversi dataset. Questo comportamento può essere attribuito a diverse cause, di seguito vengono evidenziati alcuni dataset che presentano comportamenti particolari:

- **Daphnet, MGAB e MITDB**: come riportato nella Tabella 5.3.1, anche con una soglia (threshold) al 90%, che ha fornito risultati relativamente migliori rispetto ad altre, le metriche rimangono molto basse per tutti e tre i dataset. L'AUC risulta sempre basso, segnalando che il modello non ha saputo distinguere tra istanze normali e

Tabella 5.4: PRECISION per Dataset & Threshold

	$\mu + 3\sigma$	$\mu + 1\sigma$	90%	99.5%	IQR	MAD3
Daphnet	0.000	**0.007**	0.007	0.000	0.000	0.000
Dodgers	**0.822**	0.433	0.555	0.802	0.768	0.587
ECG	0.405	0.399	0.404	**0.406**	0.402	0.392
GHL	**0.021**	0.005	0.007	0.003	0.008	0.006
Genesis	0.000	0.020	0.028	**0.220**	0.000	0.043
IOPS	**0.022**	0.017	0.016	0.021	0.019	0.017
KDD21	0.215	0.067	0.059	**0.575**	0.077	0.052
MGAB	0.000	0.004	0.004	0.000	0.000	**0.005**
MITDB	0.072	0.036	0.038	**0.103**	0.062	0.046
NASA-MSL	**1.000**	0.576	0.758	**1.000**	**1.000**	0.951
NASA-SMAP	**0.747**	0.745	0.010	0.727	0.010	0.010
OPPORTUNITY	**0.607**	0.193	0.223	0.520	0.273	0.194
Occupancy	0.000	0.754	0.903	**1.000**	0.000	0.984
SMD	0.980	0.456	0.672	**1.000**	0.835	0.462
SVDB	0.825	0.591	0.647	**0.862**	0.710	0.558
SensorScope	0.368	0.317	0.340	**0.655**	0.448	0.254
YAHOO	**0.050**	0.010	0.014	0.000	0.010	0.008

Tabella 5.5: RECALL per Dataset & Threshold

	$\mu + 3\sigma$	$\mu + 1\sigma$	90%	99.5%	IQR	MAD3
Daphnet	0.000	**0.023**	0.012	0.000	0.000	0.000
Dodgers	0.058	**0.548**	0.498	0.036	0.359	0.483
ECG	0.040	0.531	0.319	0.016	0.180	**0.626**
GHL	0.299	**1.000**	**1.000**	0.021	0.528	**1.000**
Genesis	0.000	**1.000**	0.900	0.360	0.000	0.040
IOPS	0.093	0.603	0.389	0.025	0.278	**0.615**
KDD21	0.690	0.750	**0.755**	0.371	0.747	**0.755**
MGAB	0.000	**0.310**	0.185	0.000	0.005	0.270
MITDB	0.048	**0.223**	0.210	0.028	0.113	0.172
NASA-MSL	0.051	**0.635**	0.551	0.038	0.112	0.375
NASA-SMAP	0.729	0.859	**1.000**	0.376	**1.000**	**1.000**
OPPORTUNITY	**1.000**	**1.000**	**1.000**	0.117	**1.000**	**1.000**
Occupancy	0.000	**0.278**	0.248	0.014	0.000	0.131
SMD	0.074	**0.989**	0.710	0.053	0.122	0.986
SVDB	0.080	0.685	0.464	0.031	0.261	**0.762**
SensorScope	0.043	0.284	0.266	0.026	0.083	**0.845**
YAHOO	**1.000**	**1.000**	**1.000**	0.000	**1.000**	**1.000**

anomale, questo è probabilmente la causa delle metriche molto basse indipendentemente dalla soglia applicata.

	AUC	Precision	Recall
Daphnet	0.390	0.007	0.012
MGAB	0.575	0.004	0.185
MITDB	0.491	0.038	0.210

- **GHL, Genesis e YAHOO**: questi dataset si caratterizzano per valori di AUC elevati, ma la precision tende a essere molto bassa indipendentemente dalla soglia scelta, mentre il recall è estremamente alto o nullo. Un esempio evidente è dato dal dataset YAHOO (AUC = 0.984), riportato nella Tabella 5.3.1, dove la precision è sempre bassa e il recall risulta pari a 1 per quasi tutte le soglie, ad eccezione della soglia 99.5% dove è uguale a zero.

YAHOO	$\mu + 3\sigma$	$\mu + 1\sigma$	90%	99.5%	IQR	MAD3
Precision	0.050	0.010	0.014	0.000	0.010	0.008
Recall	1.000	1.000	1.000	0.000	1.000	1.000

  Questo comportamento può essere spiegato dalla presenza di un numero molto limitato di anomalie nel dataset. In questi casi, se il modello individua correttamente queste poche anomalie, allora recall = 1. Tuttavia, è probabile che vengano identificati anche diversi falsi positivi, facendo scendere di molto la precision e, di conseguenza, il valore di F1 (media armonica tra precision e recall).

  Per verificare questa ipotesi, è stato utilizzata la funzione `plotFig_new` per visualizzare la serie: dal grafico ottenuto 6.1 (riportato in appendice) risulta che nel dataset YAHOO sono presenti solo 2 anomalie, confermando l'ipotesi precedente.

5.3.2 Confronti tra soglie

Dalle tabelle 5.4 e 5.5 si osserva che alcune soglie producono valori di precision elevati a scapito di un recall molto basso. Questo comportamento può

Tabella 5.6: F1 per Dataset & Threshold

	$\mu + 3\sigma$	$\mu + 1\sigma$	90%	99.5%	IQR	MAD3	AUC
Daphnet	0.000	**0.011**	0.009	0.000	0.000	0.000	0.390
Dodgers	0.109	0.484	0.525	0.069	0.489	**0.530**	0.820
ECG	0.073	0.456	0.356	0.031	0.248	**0.482**	0.810
GHL	**0.039**	0.010	0.014	0.005	0.017	0.013	0.965
Genesis	0.000	0.039	0.054	**0.273**	0.000	0.042	0.976
IOPS	0.035	0.033	0.032	0.023	**0.036**	0.033	0.858
KDD21	0.327	0.122	0.109	**0.451**	0.140	0.097	0.820
MGAB	0.000	0.008	0.007	0.000	0.001	**0.009**	0.575
MITDB	0.058	0.062	0.065	0.044	**0.080**	0.073	0.491
NASA-MSL	0.098	0.604	**0.638**	0.074	0.202	0.538	0.736
NASA-SMAP	0.738	**0.798**	0.019	0.496	0.019	0.019	0.928
OPPORTUNITY	**0.756**	0.324	0.364	0.191	0.429	0.325	0.994
Occupancy	0.000	**0.406**	0.389	0.028	0.000	0.231	0.865
SMD	0.138	0.624	**0.690**	0.101	0.213	0.629	0.969
SVDB	0.147	0.635	0.540	0.060	0.382	**0.644**	0.924
SensorScope	0.077	0.299	0.299	0.049	0.140	**0.390**	0.796
YAHOO	**0.095**	0.020	0.028	0.000	0.020	0.016	0.984
Media	0.158	**0.290**	0.243	0.111	0.142	0.239	

essere dovuto a una threshold troppo elevata, che identifica correttamente solo poche anomalie, tralasciandone molte altre.

Al contrario, in presenza di valori bassi per precision e alti per recall, è possibile che la soglia applicata sia troppo bassa, portando il modello a classificare come anomali anche molti punti normali oltre a quelli effettivamente anomali.

Per questi motivi la soglia ottimale dovrebbe bilanciare precision e recall massimizzando il valore di F1. I valori di F1, calcolati da precision e recall, sono riportati nella Tabella 5.6, insieme all'AUC e alle medie per ciascuna soglia.

Dalla tabella si osserva che alcune soglie hanno, in media, prestazioni migliori rispetto ad altre. Questo è dovuto al fatto che si adattano meglio, rispetto alle altre soglie, alle caratteristiche dei dataset, come la percentuale di anomalie e la distribuzione dei dati.

Conclusioni

Con sorpresa, le soglie più comunemente utilizzate, come $\mu + 3\sigma$ e IQR, sono risultate tra le meno efficaci su questi dataset. Nel caso di $\mu + 3\sigma$, le prestazioni mediamente basse sono dovute al fatto che ha ottenuto una precision elevata ma una recall molto bassa, suggerendo che la soglia fosse troppo elevata in molti casi.

In conclusione, tra le soglie testate quella basata su $\mu + k\sigma$ con $k = 1$ ha mostrato le prestazioni migliori, con un valore di F1 mediamente più elevato senza richiedere assunzioni sulla percentuale di anomalie presenti.

Nonostante $\mu + 1\sigma$ si sia rivelata mediamente la soglia più efficace, se si vogliono ottenere buoni risultati su uno specifico dataset la scelta della soglia dovrebbe basarsi su un'analisi esplorativa, esaminando la distribuzione dei dati e stimando la percentuale di anomalie attese.

5.4 Analisi con finestre giustapposte

Come anticipato nella Sezione 3.2.6, le sliding windows vengono definite dallo step s tra le finestre. Finora è stato utilizzato $s = 1$ mentre in questa sezione verranno confrontati tali risultati con quelli ottenuti impostando $s = W$, ovvero con finestre giustapposte in cui ciascun punto della serie è incluso in una sola finestra.

Dal punto di vista implementativo, bisogna porre attenzione ai dataset di dimensioni ridotte dato che il numero totale di finestre può risultare troppo basso, provocando errori in fase di addestramento. Ad esempio, il modulo AutoEncoder di PyOD richiede una batch size minima di 32 istanze per funzionare correttamente, mentre metodi basati su vicinanze come KNN necessitano di un numero di finestre superiore al parametro k.

A causa di questo problema, sono stati esclusi alcuni dataset, in particolare: NASA-MSL, Occupancy e YAHOO. Nella Tabella 5.7 sono riportate le medie e le deviazioni standard degli AUC ottenuti per ciascun modello, sia nel caso $s = W$ che nel caso $s = 1$.

Tabella 5.7: Media AUC per modello

	$s = W$		$s = 1$	
	Media AUC	Std Dev	Media AUC	Std Dev
IForest	0.786	0.190	0.818	0.179
AutoEncoder	0.790	0.179	0.817	0.161
KNN_5	0.784	0.218	0.750	0.266
KNN_10	0.777	0.221	0.782	0.222
KNN_30	0.809	0.166	0.802	0.215

Rispetto a $s = 1$, con $s = W$ il metodo che ha ottenuto l'AUC medio maggiore e il minor standard error è risultato essere KNN_30, seguito da AutoEncoder. Nella Tabella 5.8 si vede inoltre che la soglia con il valore medio di F1 maggiore risulta MAD3, seguito da $\mu + 1\sigma$.

Questa analisi porta a conclusioni diverse rispetto a quelle con $s = 1$, perché utilizzando finestre si aggrega l'informazione temporale. Di conseguenza il dataset risulta più semplice, rendendo il metodo più adatto a rilevare anomalie collettive, ma meno efficace nel riconoscere anomalie puntuali.

Tabella 5.8: F1 per Dataset & Threshold $s = W$

	$\mu + 3\sigma$	$\mu + 1\sigma$	90%	99.5%	IQR	MAD3	AUC
Daphnet	0.000	0.020	**0.029**	0.000	0.000	0.000	0.408
Dodgers	0.076	0.271	0.286	0.020	0.212	**0.314**	0.790
ECG	0.091	0.606	0.437	0.038	0.305	**0.642**	0.847
GHL	**0.062**	0.020	0.030	0.000	0.022	0.023	0.943
Genesis	0.000	0.148	0.211	**0.667**	0.000	**0.667**	1.000
IOPS	0.130	0.280	0.263	0.056	0.201	**0.297**	0.832
KDD21	0.308	0.108	0.095	**0.800**	0.118	0.085	0.749
MGAB	0.000	**0.018**	0.009	0.000	0.000	0.009	0.576
MITDB	0.059	0.066	0.066	0.047	**0.093**	0.077	0.446
NASA-SMAP	**0.667**	**0.667**	0.045	**0.667**	0.045	0.045	0.750
OPPORTUNITY	**0.764**	0.337	0.389	0.233	0.449	0.345	0.990
SMD	0.111	**0.737**	0.643	0.111	0.435	**0.737**	0.931
SVDB	0.111	0.740	0.580	0.044	0.401	**0.798**	0.949
SensorScope	0.073	0.313	0.294	0.045	0.115	**0.398**	0.798
Media	0.175	0.309	0.241	0.195	0.171	**0.317**	

Inoltre, la percentuale complessiva di finestre anomale può variare in modo significativo rispetto al caso di punti singoli.

In conclusione, l'approccio $s = W$ può risultare utile quando l'obiettivo non è individuare il singolo istante dell'anomalia, ma la finestra in cui essa si verifica. Un vantaggio aggiuntivo è la notevole velocità di calcolo: il training su MITDB con $s = 1$ e KNN ha richiesto circa 2400s, mentre con $s = W$ è durato meno di 1 secondo grazie alla riduzione del numero di istanze.

Capitolo 6

Conclusioni

In questa tesi è stata approfondita l'Anomaly Detection su serie temporali utilizzando le sliding windows. Dopo aver introdotto l'argomento e le sfide da affrontare, sono stati presentati i tre metodi impiegati nell'analisi successiva: K-Nearest Neighbors, Isolation Forest e AutoEncoder. Infine, grazie alla pipeline Python, i tre algoritmi con sliding windows sono stati confrontati su dataset etichettati di serie storiche.

I risultati hanno mostrato come Isolation Forest garantisca il miglior compromesso tra accuratezza e tempi di elaborazione. Questo modello ha garantito AUC molto simili a quelli di AutoEncoder, ma con tempi dell'ordine di pochi secondi anche sui dataset più grandi, dove invece AutoEncoder e KNN, data la loro complessità, hanno richiesto decine di minuti.

Per quanto riguarda la soglia di anomalia, è risultato che la classica $\mu+3\sigma$, utilizzata anche nel benchmark di TSB-UAD, tende a fallire, generando molti falsi negativi. Una soglia che si è dimostrata più robusta con i dataset analizzati è $\mu+1\sigma$, anch'essa basata sulla distribuzione ma in grado di bilanciare meglio precision e recall, senza richiedere stime sulla percentuale di outlier come avviene per le soglie basate sui percentili.

In un possibile sviluppo futuro, si potrebbero esplorare ulteriori metodi di Anomaly Detection, tra cui LSTM che non richiede le sliding windows, in quanto progettato appositamente per le serie storiche.

Tra i limiti di questa tesi vi è l'utilizzo di sole soglie statiche, quindi in

futuro sarebbe utile definire le soglie considerando informazioni sulla distribuzione dei dati e, dove possibile, su una stima della percentuale di outlier. Un ulteriore sviluppo potrebbe essere quello di trovare un modo migliore per classificare le soglie, in quanto la media dei valori di F1 non rappresenta al meglio le performance delle soglie. Una nuova metrica comparativa per le soglie potrebbe essere definita come:

$$M_i = \frac{1}{N} \sum_{j=1}^{N} \frac{\mathrm{F1}_{i,j}}{\max \mathrm{F1}_j}$$

Dove N è il numero di dataset, $\mathrm{F1}_{i,j}$ il valore di F1 ottenuto con la soglia i sul dataset j e $\max \mathrm{F1}_j$ il valore massimo di F1 ottenuto sul dataset j.

Questa metrica renderebbe le soglie più confrontabili ed eviterebbe che dataset con F1 basso influenzino eccessivamente la performance delle soglie.

Infine, l'utilizzo di un solo benchmark, rappresenta una limitazione alla generalizzabilità dei risultati. Per verificare la robustezza dei metodi in scenari più eterogenei, l'analisi andrebbe estesa a ulteriori dataset.

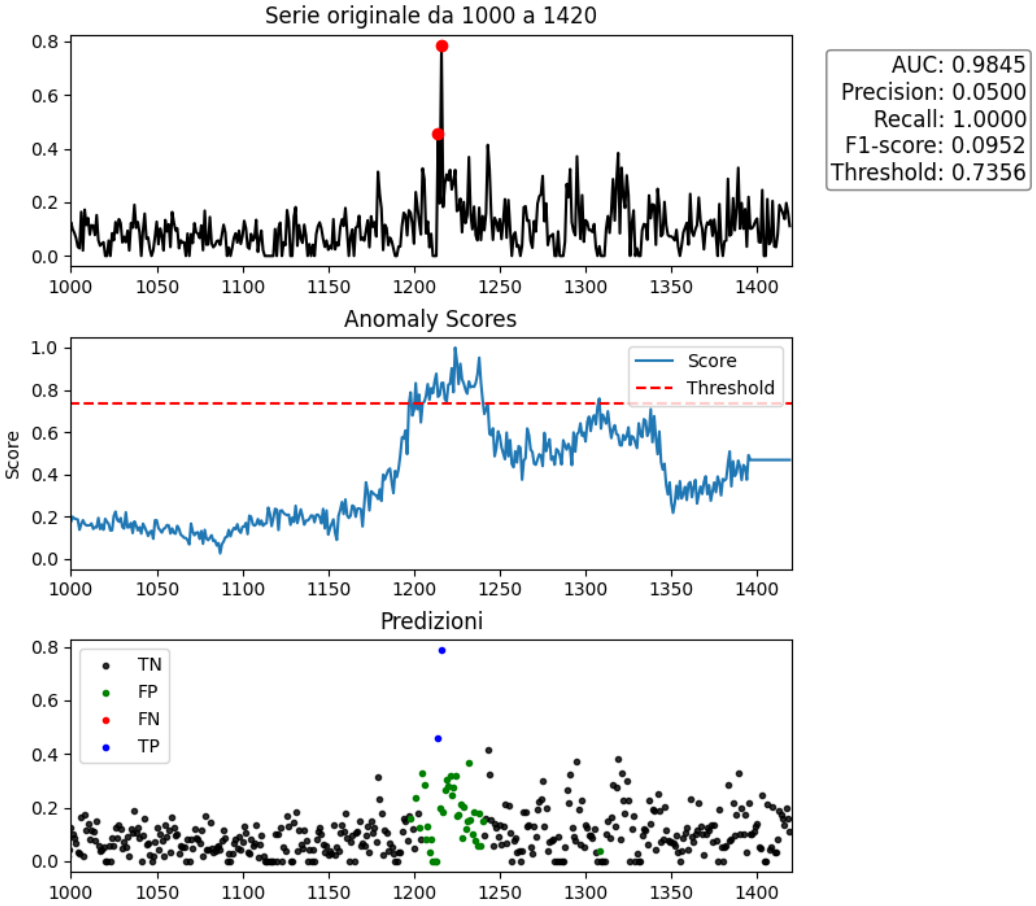
In sintesi, questa tesi ha fornito un quadro robusto da un punto di vista pratico e teorico per trattare il problema dell'anomaly detection su serie temporali, offrendo alcuni metodi efficaci e sottolineando l'importanza di un'accurata scelta della soglia. I risultati attuali possono essere considerati un punto di partenza per futuri sviluppi che potrebbero rendere l'individuazione delle anomalie più flessibile, interpretabile e adattabile a contesti reali.

Appendice

Tabella 6.1: F1 per Dataset & Threshold

	$\mu + 3\sigma$	$\mu + 2\sigma$	$\mu + 1\sigma$	$\mu + .5\sigma$	70%	80%	90%	95%	99%	99.5%	IQR	MAD3	MAD5
Daphnet	0.000	0.005	0.011	0.020	0.018	0.012	0.009	0.009	0.000	0.000	0.000	0.000	0.000
Dodgers	0.109	0.511	0.484	0.410	0.379	0.433	0.525	0.484	0.135	0.069	0.489	0.530	0.257
ECG	0.073	0.269	0.456	0.490	0.461	0.481	0.356	0.228	0.059	0.031	0.248	0.482	0.352
GHL	0.039	0.018	0.010	0.005	0.005	0.007	0.014	0.018	0.039	0.005	0.017	0.013	0.018
Genesis	0.000	0.315	0.039	0.015	0.020	0.030	0.054	0.093	0.282	0.273	0.000	0.042	0.000
IOPS	0.035	0.038	0.033	0.032	0.026	0.034	0.032	0.039	0.041	0.023	0.036	0.033	0.035
KDD21	0.327	0.216	0.122	0.077	0.039	0.057	0.109	0.197	0.518	0.451	0.140	0.097	0.157
MGAB	0.000	0.004	0.008	0.006	0.006	0.007	0.007	0.002	0.000	0.000	0.001	0.009	0.002
MITDB	0.058	0.080	0.062	0.048	0.043	0.050	0.065	0.078	0.051	0.044	0.080	0.073	0.074
NASA-MSL	0.098	0.485	0.604	0.390	0.406	0.525	0.638	0.516	0.137	0.074	0.202	0.538	0.032
NASA-SMAP	0.738	0.782	0.798	0.798	0.019	0.019	0.019	0.019	0.763	0.496	0.019	0.019	0.019
OPPORTUNITY	0.756	0.512	0.324	0.225	0.138	0.200	0.364	0.616	0.355	0.191	0.429	0.325	0.489
Occupancy	0.000	0.258	0.406	0.633	0.617	0.457	0.389	0.239	0.125	0.028	0.000	0.231	0.000
SMD	0.138	0.384	0.624	0.549	0.479	0.632	0.690	0.455	0.175	0.101	0.213	0.629	0.385
SVDB	0.147	0.427	0.635	0.633	0.593	0.644	0.540	0.376	0.112	0.060	0.382	0.644	0.512
SensorScope	0.077	0.219	0.299	0.358	0.355	0.354	0.299	0.237	0.063	0.049	0.140	0.390	0.370
YAHOO	0.095	0.038	0.020	0.016	0.009	0.014	0.028	0.055	0.118	0.000	0.020	0.016	0.021
Media	0.158	0.268	0.290	0.277	0.213	0.233	0.243	0.215	0.175	0.111	0.142	0.239	0.160

Figura 6.1: plotFig_new di YAHOO con $\mu + 3\sigma$



Bibliografia

[1] Charu C. Aggarwal. Outlier analysis. In *Data Mining*, pages 75–79. Springer, 2015.

[2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 11 1997.

[3] John Paparrizos, Yuhao Kang, Paul Boniol, Ruey S Tsay, Themis Palpanas, and Michael J Franklin. Tsb-uad: an end-to-end benchmark suite for univariate time-series anomaly detection. *Proceedings of the VLDB Endowment*, 15(8):1697–1711, 2022.

[4] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), July 2009.

[5] Sihan Chen, Zhuangzhuang Qian, Wingchun Siu, Xingcan Hu, Jiaqi Li, Shawn Li, Yuehan Qin, Tiankai Yang, Zhuo Xiao, Wanghao Ye, Yichi Zhang, Yushun Dong, and Yue Zhao. Pyod 2: A python library for outlier detection with llm-powered model selection. *arXiv preprint arXiv:2412.12154*, 2024.

[6] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD '00)*, pages 427–438. ACM, 2000.

[7] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the Eighth IEEE International Conference on Data Mining (ICDM)*, pages 413–422, Pisa, Italy, December 2008.

[8] Geoffrey E. Hinton and Richard S. Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Advances in Neural Information Processing Systems 6*, pages 3–10. Morgan Kaufmann, 1993.

[9] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems 19*, pages 1137–1144. MIT Press, 2006.

[10] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.

[11] John Paparrizos, Chunwei Liu, Aaron Elmore, and Michael Franklin. Debunking four long-standing misconceptions of time-series distance measures. In *SIGMOD/PODS '20: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1887–1905, June 2020.

[12] John Paparrizos, Paul Boniol, Themis Palpanas, Ruey S Tsay, Aaron Elmore, and Michael J Franklin. Volume Under the Surface: A New Accuracy Evaluation Measure for Time-Series Anomaly Detection. *Proceedings of the VLDB Endowment*, 15(11):2774–2787, 2022.

[13] Yue Zhao, Zain Nasrullah, and Zheng Li. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019.

[14] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. *SIGMOD Rec.*, 29(2):427–438, May 2000.