Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

Time Series Anomaly Detection presso INFN-CNAF Data Center: un approccio basato sul Machine Learning

Relatore:
Chiar.mo Prof.
Andrea Asperti

Presentata da:
Gabriele Raciti

Correlatrice:
Chiar.ma Prof.ssa
Elisabetta Ronchieri

Sessione II
Anno Accademico 2023/2024

Abstract

Il rilevamento di anomalie in serie temporali è fondamentale per garantire la stabilità e la sicurezza dei data center, in particolare in contesti scientifici come l'INFN-CNAF, il Centro Nazionale per la ricerca e lo sviluppo nelle tecnologie informatiche e telematiche dell'Istituto Nazionale di Fisica Nucleare, dove viene gestita un'enorme quantità di dati eterogenei significativi per esperimenti internazionali attraverso l'adozione di sistemi di monitoraggio dedicati. Dovendo garantire una disponibilità continua, si sono iniziate a valutare soluzioni di intelligenza artificiale per la rilevazione di anomalie, volte a prevedere possibili guasti. Questo lavoro presenta un approccio basato sul Machine Learning per la rilevazione automatica di anomalie nel servizio WebDAV dell'INFN-CNAF. Tra i metodi testati figurano Long Short-Term Memory (LSTM), Random Forest e reti neurali di vario tipo, con l'obiettivo di valutare la precisione e la sensibilità dei modelli nel distinguere tra comportamenti normali e anomali, oltre a offrire una maggiore comprensione delle aree del sistema in cui si verificano problematiche. I risultati mostrano l'efficacia relativa dei metodi sperimentati, evidenziando l'utilità dei modelli non solo nel rilevare anomalie, ma anche nel localizzare i punti critici all'interno delle metriche monitorate, contribuendo così al perfezionamento del monitoraggio proattivo delle risorse IT e alla gestione efficiente del data center.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione all'INFN-CNAF e al Concetto di Anomalia

Il **Centro Nazionale per la ricerca e lo sviluppo nelle tecnologie informatiche e telematiche** dell'**Istituto Nazionale di Fisica Nucleare (INFN-CNAF)** è situato a Bologna e rappresenta una delle infrastrutture informatiche più avanzate in Italia e a livello internazionale, svolgendo un ruolo chiave nel supporto alla ricerca scientifica ad alta intensità di calcolo. Fondato per gestire i bisogni computazionali dei grandi esperimenti di fisica delle particelle, il CNAF è cresciuto nel tempo fino a diventare un punto di riferimento per la comunità scientifica internazionale. Dal 2003, ospita il **Tier-1** italiano, una delle componenti centrali del Worldwide LHC Computing Grid (WLCG), un'infrastruttura distribuita che gestisce e analizza una quantità enorme di dati dati prodotti dagli esperimenti presso l'acceleratore Large Hadron Collider (LHC) del CERN. Oltre a supportare il LHC, che utilizza circa il 70% delle risorse del CNAF, il centro fornisce servizi essenziali per molti altri progetti di ricerca di livello mondiale. Tra questi figurano esperimenti di astroparticelle, come AMS e DarkSide, esperimenti legati allo studio dei neutrini, come ICARUS e JUNO, oltre a progetti come Virgo, dedicati alla rilevazione delle onde gravitazionali. Il CNAF è quindi un'infrastruttura essenziale per la comunità scientifica italiana, essendo uno dei principali centri di calcolo distribuito nel paese, e gioca un ruolo cruciale nei progetti di trasferimento tecnologico, soprattutto in ambito medico e nella conservazione del patrimonio culturale. Il ruolo del CNAF non si limita però al calcolo scientifico. Il centro è infatti profondamente coinvolto nella gestione e nello sviluppo delle tecnologie di **Grid Computing**, che permette di distribuire risorse computazionali su scala globale. Grazie all'integrazione di

sistemi di storage distribuito e tecnologie come l'*IBM General Parallel File System (GPFS)* e il *Tivoli Storage Manager (TSM)*, il CNAF garantisce la massima affidabilità nella gestione e nella conservazione a lungo termine dei dati. Questi sistemi sono supportati da avanzate tecnologie di monitoraggio basate su *Grafana* e *Sensu*, che assicurano un controllo in tempo reale delle prestazioni e della sicurezza delle infrastrutture IT, permettendo di rilevare tempestivamente eventuali anomalie e di mantenere elevati livelli di operatività. Nel corso degli anni, il CNAF si è quindi evoluto in un'infrastruttura strategica non solo per la fisica delle particelle e l'astrofisica, ma anche per l'intero ecosistema della ricerca scientifica globale. Il continuo sviluppo delle sue risorse e l'adozione di nuove tecnologie lo rendono un centro di eccellenza che garantisce alla comunità globale una capacità di calcolo e archiviazione senza pari. Questo impegno costante nella gestione delle risorse IT e nella loro ottimizzazione rende il CNAF non solo un centro all'avanguardia, ma anche un ambiente in cui il concetto di **anomalia** riveste un ruolo cruciale. In generale, un'anomalia può essere definita come un comportamento o un'osservazione che si discosta significativamente da ciò che è considerato normale o atteso all'interno di un sistema, facendo sorgere il sospetto che sia stata generata da un meccanismo diverso. Le anomalie possono quindi indicare disfunzioni o comportamenti imprevisti che richiedono attenzione immediata per evitare interruzioni del servizio, perdita di dati o vulnerabilità alla sicurezza. Il rilevamento di esse è quindi fondamentale per mantenere l'integrità e la continuità operativa di un sistema. Le anomalie possono essere suddivise in diverse categorie a seconda delle loro caratteristiche e del contesto in cui si manifestano e possono essere indicative di diversi fenomeni, alcuni dei quali possono avere implicazioni critiche. Nella letteratura sono identificate quattro tipologie di anomalia differenti:

1. **Anomalie Puntuali**: questa tipologia di anomalie si verifica quando un singolo evento o osservazione differisce in modo evidente dagli altri valori. Nel contesto dei sistemi informatici, un'anomalia puntuale può rappresentare un errore isolato o un problema che necessita di attenzione immediata, come un picco improvviso nell'utilizzo della CPU. Sono le anomalie più facili da identificare poiché si basano su confronti diretti con i dati attesi.

2. **Anomalie Contestuali**: questa tipologia di anomalie si verifica quando un evento o un'osservazione viene considerato anomalo solo nel contesto specifico in cui si verifica. Ad esempio, un picco nell'utilizzo della CPU durante un'operazione di backup potrebbe essere normale, mentre lo stesso picco durante un'operazione a basso carico potrebbe

indicare un problema. In questi casi il contesto temporale o spaziale risulta essere fondamentale per identificare questo tipo di anomalie, poiché l'evento in sé potrebbe non sembrare anomalo se analizzato in maniera isolata. L'identificazione di queste anomalie rappresenta una sfida più complessa, poiché richiede un'analisi del contesto operativo complessivo.

3. **Anomalie Collettive**: si verificano quando un insieme di dati, preso collettivamente, appare anomalo rispetto al resto dei dati. Il singolo dato all'interno del gruppo potrebbe quindi non essere anomalo se considerato individualmente, ma l'insieme di questi dati risulta essere anomalo. Un esempio potrebbe riguardare i file di log di accesso a un sistema. Un singolo file di accesso potrebbe sembrare perfettamente normale e rientrare nei parametri di utilizzo atteso, ma se in più file di log si verifica un accesso ripetuto a un servizio da diverse località geografiche in un breve lasso di tempo, emerge un comportamento che non rispecchia più un'attività normale. Il rilevamento di queste anomalie richiede tecniche avanzate, poiché si basa sul riconoscimento di pattern complessi piuttosto che sulla semplice identificazione di eventi isolati.

4. **Anomalie Emergenti Lentamente (drifts)**: sono anomalie che si sviluppano gradualmente nel tempo, rendendole più difficili da rilevare rispetto alle anomalie precedenti. Esse possono manifestarsi come cambiamenti lenti nel comportamento dei dati, che inizialmente possono sembrare normali ma che, con il passare del tempo, si discostano sempre di più dai valori attesi portando a possibili malfunzionamenti. Questo tipo di anomalie è molto complesso da rilevare proprio perché non si manifesta in modo immediato ma evolve lentamente, richiedendo un monitoraggio continuo per essere identificato prima che provochi gravi conseguenze nel sistema.

Il rilevamento delle anomalie è quindi un aspetto cruciale per la gestione dei sistemi informatici moderni, specialmente in ambienti distribuiti. In questi contesti la presenza di anomalie può avere conseguenze significative, tra cui la riduzione delle prestazioni, l'interruzione dei servizi o, nei casi peggiori, la perdita di dati critici. Questo è particolarmente importante in sistemi complessi e su larga scala, come i data center distribuiti, dove le risorse computazionali e di archiviazione devono essere gestite in modo efficiente e con elevata affidabilità. Inoltre il rilevamento delle anomalie non riguarda solo l'identificazione di problemi operativi, ma anche la prevenzione di eventi potenzialmente dannosi, come attacchi informatici o violazioni di sicurezza. Infatti, al giorno d'oggi, comportamenti anomali nei sistemi

informatici possono indicare tentativi di intrusione, accessi non autorizzati o la presenza di malware che potrebbero compromettere la sicurezza dei dati. Pertanto, l'integrazione di sistemi di monitoraggio e rilevamento delle anomalie diventa essenziale per garantire la sicurezza e l'affidabilità delle infrastrutture IT. Il rilevamento tempestivo delle anomalie consente non solo di intervenire rapidamente per ripristinare le condizioni normali di operatività, ma rappresenta anche una strategia proattiva per prevenire futuri problemi e ottimizzare l'efficienza dell'intero sistema. I metodi per il rilevamento delle anomalie possono variare a seconda del tipo di sistema e del contesto in cui vengono applicati. Tecniche statistiche, metodi basati su regole, e algoritmi di Machine Learning sono solo alcune delle soluzioni utilizzate per identificare comportamenti anomali.

Il lavoro presentato in questa tesi si concentra sull'adozione di un approccio basato sul Machine Learning per l'individuazione di anomalie all'interno del servizio **WebDAV** dell'INFN-CNAF, una piattaforma che consente l'accesso e la gestione remota di file tramite il protocollo HTTP, rendendo i dati accessibili e modificabili come se fossero ospitati su una cartella locale. Utilizzando un dataset di **Time Series** che rappresenta diverse metriche di sistema, come l'utilizzo di CPU, della memoria e del traffico di rete, è stato adottato un approccio basato su diversi modelli di Machine Learning per rilevare e analizzare anomalie, con l'obiettivo di identificare disfunzioni o malfunzionamenti del servizio. Il lavoro è suddiviso come segue: il Capitolo 2 approfondisce lo stato dell'arte nell'ambito dell'Anomaly Detection, presentando le principali tecniche e gli approcci utilizzati per il rilevamento delle anomalie. Il Capitolo 3 descrive il metodo proposto e l'implementazione, percorrendo l'intera pipeline che include la raccolta dei dati, l'analisi delle metriche e delle loro relazioni, il pre-processing, l'organizzazione dei dati per il training, la descrizione dei modelli di Machine Learning utilizzati e la loro applicazione. Nel Capitolo 4 vengono presentati e confrontati i risultati ottenuti, insieme a un'analisi tramite Saliency Map di alcuni casi di esempio. Infine, nel Capitolo 5, sono presentate le conclusioni e una panoramica dei possibili lavori futuri.

Capitolo 2

Stato dell'arte

Le soluzioni per l'Anomaly Detection presenti in letteratura si suddividono principalmente in due categorie: il monitoraggio dei dati organizzati in **serie temporali** e l'analisi dei **Log**. Le soluzioni basate sulle serie temporali si focalizzano sull'elaborazione di informazioni raccolte a intervalli regolari, come l'utilizzo della CPU, della memoria o del disco, che vengono comunemente rappresentate come **Time Series**. Questo approccio mira a rilevare anomalie o variazioni significative rispetto ai comportamenti attesi all'interno dei flussi temporali. Al contrario, le soluzioni basate sui Log si focalizzano sull'analisi delle sequenze di messaggi o eventi registrati dal sistema, che non sempre seguono intervalli temporali costanti. In questo caso, l'obiettivo è individuare pattern o sequenze anomale che possano segnalare malfunzionamenti o comportamenti inattesi del sistema. Entrambi gli approcci presentano quindi delle sfide specifiche e distinte. Nel contesto di questo lavoro, ci si focalizza sull'Anomaly Detection tramite l'analisi delle serie temporali, proponendo metodologie per distinguere tra comportamenti normali e anomali nei flussi di dati temporali. In questo capitolo verrà comunque fornita una panoramica dei principali approcci per l'Anomaly Detection, basati sia sul monitoraggio delle serie temporali che sull'analisi dei log di sistema, con l'obiettivo di offrire una visione completa delle metodologie, delle tecniche e dei risultati raggiunti nel campo della rilevazione delle anomalie. A tal fine, nel paragrafo 2.1 verranno descritte le soluzioni basate sul monitoraggio dei log, sebbene non rappresentino il focus centrale di questo studio. Saranno comunque analizzati i principali metodi utilizzati per l'analisi dei log, comprendendo sia i modelli consolidati sia gli approcci più recenti per la rilevazione di anomalie all'interno delle sequenze di eventi registrati. Successivamente, nel paragrafo 2.2, verrà fornita un'analisi approfondita delle soluzioni per l'Anomaly Detection basate su Time Series, che costituiscono

il nucleo centrale di questo studio. Questa sezione illustrerà in dettaglio i metodi, le architetture e le tecniche utilizzate per individuare anomalie in dati raccolti come serie temporali e valutati a intervalli regolari. L'obiettivo è identificare gli approcci più efficaci e innovativi per l'Anomaly Detection su dati strutturati in sequenze temporali, offrendo una panoramica completa delle tecniche più adatte all'identificazione di comportamenti anomali in contesti di monitoraggio continuo.

2.1 Soluzioni per l'Anomaly Detection basate sull'analisi dei Log

Le soluzioni per l'Anomaly Detection basate sull'analisi dei Log necessitano di tecniche preparatorie specifiche per estrarre informazioni utili, a causa della natura testuale e non strutturata dei Log. Essi infatti si presentano come sequenze di eventi o messaggi registrati da diverse fonti, che spesso includono dettagli tecnici, Timestamp e descrizioni testuali non standardizzate. Questo formato dei messaggi rende necessaria, nella maggior parte dei casi, una fase di parsing e pre-processing iniziale per estrarre le informazioni rilevanti e organizzarle in modo strutturato. Una volta organizzati i dati in un formato strutturato, è possibile applicare metodi di analisi che spaziano da semplici tecniche di estrazione di pattern a tecniche avanzate di machine learning. Un approccio interessante presente in letteratura è quello proposto da Viola, Ronchieri e Cavallaro con l'articolo *Combining Log Files and Monitoring Data to Detect Anomaly Patterns in a Data Center* [1]. Nell'articolo viene proposto un approccio di Anomaly Detection per il Data Center dell'INFN-CNAF che combina un approccio di analisi dei file di log con l'analisi delle serie temporali dei dati di monitoraggio raccolti da sensori, con l'obiettivo di identificare schemi di anomalie a livello di servizio. Per quanto riguarda la parte di analisi sui file di log, il lavoro è iniziato con una fase di parsing che ha trasformato i log in formato csv, consentendo di estrarre informazioni strutturate come timestamp, nome del servizio e messaggi di errore. Successivamente i log sono stati puliti e pre-elaborati rimuovendo il testo non informativo, come caratteri non alfanumerici, punteggiatura e stopwords, e mantenendo invece i termini rilevanti per l'identificazione delle anomalie. Si è poi proceduto con la fase principale dell'approccio, ovvero la costruzione di un dizionario specifico per ciascun registro. Questo dizionario non è altro che un insieme di parole o sequenze di parole strettamente legate all'area semantica delle anomalie specifiche del singolo servizio. Per ogni log è stato quindi creato un dizionario differente, poiché la struttura del messaggio e il

tipo di anomalie sono diverse tra i servizi, per poi mettere insieme i risultati e ottenere un dizionario globale dei termini identificativi di un'anomalia. Il dizionario è stato creato tramite una fase di esplorazione dei messaggi di log che si è concentrata sull'analisi dei singoli termini, in particolare su quelli meno frequenti, poiché in genere le anomalie hanno una bassa frequenza rispetto ai messaggi normali. Per arricchire ulteriormente il dizionario sono state utilizzate tecniche di NLP come l'analisi degli n-grammi e la costruzione di wordcloud, che hanno evidenziato le parole o le sequenze di parole più rilevanti per le anomalie. È stato inoltre utilizzato il topic modeling, in particolare con la tecnica Latend Dirichlet Allocation (LDA), per individuare i temi latenti all'interno dei messaggi di log e identificare possibili temi anomali e le parole associate ad essi. Una volta costruito il dizionario è stata creata una matrice di feature per ogni servizio, rappresentativa del numero di volte in cui ogni n-gramma incluso nel dizionario appare nei messaggi di log. Essa ha quindi come righe i messaggi di log e come colonne gli n-grammi presenti nel dizionario, e i valori al suo interno rappresentano il numero di volte che un dato n-gramma è presente all'interno di un determinato messaggio di log. In questo modo si riesce quindi a correlare i messaggi di log alle anomalie individuate nel dizionario, creando un vettore rappresentativo delle anomalie per ciascun messaggio. Infine è stato applicato un algoritmo di clustering ai diversi vettori ottenuti, che ha permesso di raggruppare i messaggi di log con caratteristiche simili, individuando così dei gruppi di messaggi anomali. I nuovi messaggi di log vengono quindi trasformati in vettori utilizzando lo stesso dizionario e la matrice delle feature e vengono poi confrontati con i cluster ricavati, venendo classificati come anomali o normali in base al cluster di appartenenza. Un altro articolo affine è quello di Cavallaro e Ronchieri intitolato *Identifying Anomaly Detection Patterns from Log Files: A Dynamic Approach* [2].

Un altro approccio interessante è stato proposto da Li et al. nel lavoro *Natural Language Processing-based Model for Log Anomaly Detection* [3]. Lo studio riguarda lo sviluppo di un modello di Anomaly Detection basato su tecniche di Natural Language Processing, che consente di ridurre l'intervento manuale tramite l'uso di tecniche avanzate di NLP come l'analisi delle parti del discorso (PoS, Parts of Speech) e la Named Entity Recognition (NER). L'approccio ha inizio con una fase di Template Parsing con FT-Tree, un algoritmo utilizzato per il log parsing che non richiede la definizione di regole manuali per l'estrazione di template dai messaggi di log e l'eliminazione di informazioni irrilevanti e non informative. Si prosegue con una fase di *PoS Analysis*, ovvero un'analisi delle parti del discorso dei messaggi di log. Ogni parola all'interno di un messaggio di log ha delle proprietà PoS che la identificano come una determinata parte del discorso (nome, verbo, aggettivo e così

via). Alcuni token possiedono delle proprietà PoS importanti per comprendere il significato del messaggio, mentre altri, come ad esempio le congiunzioni, sono considerate irrilevanti per il rilevamento delle anomalie e vengono quindi scartati. Questa fase permette di ridurre la complessità del modello e migliora l'accuratezza dell'analisi, perché permette di concentrarsi solo sui token significativi. I token rimanenti vengono poi trasformati in vettori numerici utilizzando la tecnica **word2vec**, una tecnica che mappa ogni parola in un vettore a dimensione fissa sfruttando le relazioni semantiche tra le parole. Non tutti i token rimanenti hanno però la stessa importanza ai fini del rilevamento delle anomalie. Per poter individuare i token più rilevanti viene quindi utilizzata la tecnica della *Named Entity Recognition*, che permette di costruire un vettore di pesi che assegnano maggiore o minore importanza a un singolo token. In questo modo i token più rilevanti hanno un peso maggiore e sono considerati maggiormente nella classificazione di un log come anomalo o normale, mentre quelli meno rilevanti hanno un peso inferiore. Si ottiene quindi un vettore composto moltiplicando il vettore dei pesi per il vettore ottenuto tramite *word2vec*. Infine, il vettore composto viene utilizzato come input per una rete neurale addestrata con i vettori risultanti di log normali e anomali, al fine di classificare ciascun esempio come normale o anomalo in base alla rappresentazione vettoriale ottenuta. Altri lavori interessanti che utilizzano in modo esteso tecniche di Natural Language Processing nella fase di pre-elaborazione dei log, trasformandoli in un formato strutturato che può essere successivamente utilizzato da modelli di machine learning per l'anomaly detection, includono quello di Li et al. intitolato *LogPS: A Robust Log Sequential Anomaly Detection Approach Based on Natural Language Processing* [4] e quello di Bertero et al. intitolato *Experience Report: Log Mining using Natural Language Processing and Application to Anomaly Detection* [5].

Ci sono inoltre ulteriori approcci che utilizzano i **Transformer**, un'architettura introdotta da Vaswani et al. nel 2017 nell'articolo *Attention is All You Need* [6] che ha rivoluzionato il campo del Natural Language Processing. Il modello si basa sul meccanismo della **self-attention**, che consente di attribuire pesi differenti alle parole a seconda della loro importanza nel contesto della frase, aprendo la strada a modelli avanzati come *BERT* e *GPT*, che sono oggi diventati lo stato dell'arte in numerosi task di NLP. Un approccio interessante è quello proposto da Guo et al. nel lavoro *LogBERT: Log Anomaly Detection via BERT* [7], in cui viene introdotto l'uso di BERT per la rilevazione di anomalie nei log. Il modello proposto utilizza due task per apprendere i pattern delle sequenze di log normali. Il primo task prevede la predizione delle *log keys*, che rappresentano template di stringhe catturanti la struttura comune di un messaggio di log, estratti tramite un log parser. Con l'approc-

cio proposto, i messaggi di log vengono quindi trasformati in una sequenza ordinata di log keys. Successivamente, ogni log key viene trasformata in una rappresentazione vettoriale, ottenuta dalla somma dell'embedding della log key e dell'embedding della sua posizione nella sequenza. L'embedding della log key è ottenuto tramite una matrice inizialmente generata casualmente, mentre l'embedding della posizione è calcolato utilizzando una funzione sinusoidale. Viene poi utilizzata l'architettura Transformer come encoder per apprendere le relazioni tra le varie log keys in una sequenza. Il modello Transformer produce così una rappresentazione contestuale per ciascuna log key, catturando informazioni che derivano dal contesto completo della sequenza, che non racchiude solo le caratteristiche della singola log key, ma anche le relazioni che essa ha con le altre log keys della sequenza. Questo risulta particolarmente utile, poiché una sequenza anomala potrebbe contenere log keys che prese singolarmente sembrano normali, ma che, se messe in relazione tra loro, formano un pattern anomalo. Una volta apprese le relazioni contestuali tra le log keys, esse vengono utilizzate per il primo task, ovvero la predizione delle log keys mascherate. In questo task, alcune log keys all'interno delle sequenze vengono mascherate casualmente tramite un token speciale, e il modello, utilizzando le informazioni contestuali apprese precedentemente, viene allenato a predire le log keys mascherate. Questo permette al modello di apprendere la struttura tipica delle sequenze normali. Successivamente viene eseguito il secondo task, ovvero la *minimizzazione del volume dell'ipersfera*. L'obiettivo di questo task è ridurre il volume della sfera che racchiude tutte le rappresentazioni delle sequenze di log normali nello spazio di embedding. L'idea è di fare in modo che tutte le rappresentazioni delle sequenze normali siano quanto più vicine possibile tra loro, minimizzando una funzione di Loss che misura la distanza tra ogni rappresentazione e il centro della sfera. Questo approccio si basa sull'ipotesi che le sequenze normali condividano pattern simili, per cui le loro rappresentazioni dovrebbero essere vicine nello spazio di embedding, mentre le sequenze anomale, che presentano deviazioni dai pattern normali, dovrebbero trovarsi a una maggiore distanza dal centro della sfera. L'addestramento finale del modello LogBERT si basa quindi su una funzione di Loss ottenuta dalla combinazione dei due task, cioè tramite una somma ponderata della funzione di Loss per la predizione delle log keys mascherate e della funzione di Loss per la minimizzazione del volume dell'ipersfera. Le sequenze anomale vengono quindi identificate come tali se il modello fallisce nel predirre correttamente un numero significativo di log keys oppure se la loro rappresentazione risulta essere troppo distante dal centro della sfera che racchiude le sequenze normali nello spazio di embedding. Ulteriori lavori interessanti che utilizzano l'architettura Transformer sono quello di Han et al. intitolato *LogGPT: Log Anomaly Detection via*

GPT [8] e quello di Yan et al. intitolato *Log-based Anomaly Detection with Transformers Pre-trained on Large-scale Unlabeled Data* [9]. Altri approcci degni di nota sono stati adottati da Guo et al. nel lavoro *TRANSLOG: A Unified Transformer-based Framework for Log Anomaly Detection* [10] e da Wang et al. nel lavoro *LogEvent2vec: LogEvent-to-Vector Based Anomaly Detection for Large-Scale Logs in Internet of Things* [11].

2.2 Soluzioni per l'Anomaly Detection su Time Series

Le soluzioni per l'Anomaly Detection su Time Series si basano sull'analisi continua di metriche specifiche raccolte a intervalli regolari, con l'obiettivo di rilevare variazioni significative dal loro normale comportamento. Le principali soluzioni si basano su metodi statistici e tecniche di Machine Learning che sfruttano i dati acquisiti dai sensori per costruire modelli capaci di rilevare delle situazioni anomale. Per quanto riguarda i metodi basati sul Machine Learning, un approccio interessante è quello proposto da Wei et al. nell'articolo *LSTM-Autoencoder based Anomaly Detection for Indoor Air Quality Time Series Data* [12], che propone un approccio di rilevazione delle anomalie utilizzando un modello ibrido composto da una **Long Short-Term Memory (LSTM)** e un Autoencoder. L'architettura complessiva è composta da due componenti principali: un LSTM Encoder e un LSTM Decoder. L'encoder LSTM opera come un livello di compressione della sequenza temporale, elaborando ciascun passo temporale e aggiornando lo stato nascosto in ogni fase. Alla fine del processo, il vettore nascosto prodotto dall'ultimo layer LSTM rappresenta una codifica compressa della sequenza temporale, sintetizzando le dipendenze a lungo termine apprese dalla sequenza di input. Successivamente il Decoder LSTM utilizza il vettore codificato prodotto dall'Encoder come input per ricostruire la sequenza temporale originale. Questa fase di decodifica prevede la duplicazione del vettore precedentemente codificato attraverso un layer chiamato *RepeatVector*, che crea un numero di copie del vettore codificato pari al numero di passi temporali della sequenza originale. In questo modo il decoder LSTM può elaborare la sequenza temporale in modo simile all'encoder cercando di ricostruire il più fedelmente possibile la sequenza di input originale. Durante l'addestramento del modello, l'obiettivo principale è quello di minimizzare la differenza tra la sequenza temporale originale e quella ricostruita. Questa differenza è misurata tramite la funzione di Loss *Mean Absolute Error*, che quantifica l'errore per ogni campione della sequenza. L'addestramento avviene soltanto sui dati normali,

in modo da consentire al modello di apprendere una rappresentazione latente che descriva i pattern regolari presenti nei dati. Successivamente, dopo l'addestramento, viene calcolato l'errore massimo di ricostruzione osservato e lo si utilizza per fissare una soglia, che servirà come punto di riferimento per la rilevazione delle anomalie. Durante la fase di test quindi, la sequenza temporale viene elaborata dall'encoder e ricostruita dal decoder e viene calcolato l'errore di ricostruzione: se supera la soglia fissata precedentemente, il campione viene classificato come anomalo; se è inferiore o uguale alla soglia, il campione viene considerato normale. Altri articoli con approcci simili, basati sull'uso di un'architettura Autoencoder costruita su LSTM, includono il lavoro di Githinji e Maina intitolato *Anomaly Detection on Time Series Sensor Data Using Deep LSTM-Autoencoder* [13] e quello di Ho et al. intitolato *A Multi-Input Bi-LSTM Autoencoder Model with Wavelet Transform for Air Quality Prediction* [14].

Un ulteriore approccio interessante che combina le Long Short-Term Memory, gli Autoencoder e gli approcci statistici è quello di Frehner et al., intitolato *Detecting Anomalies in Time Series Using Kernel Density Approaches* [15]. Per quanto riguarda le soluzioni basate su metodi statistici, uno degli approcci più utilizzati è il **Kernel Density Estimation (KDE)**, che stima la distribuzione di probabilità di un insieme di dati senza assumere una forma specifica per la distribuzione, etichettando come anomalie i punti con bassa densità di probabilità rispetto alla distribuzione stimata. Tuttavia, gli approcci puramente statistici, come la KDE, presentano alcune limitazioni quando applicati all'anomaly detection nelle serie temporali. In primo luogo, questi metodi richiedono che i dati seguano una distribuzione specifica o, quantomeno, assumono che i pattern anomali si manifestino come punti che si discostano in modo netto dalla distribuzione stimata. Questo si rivela spesso inefficace nelle serie temporali, dove le anomalie possono assumere forme complesse e multivariate, non sempre caratterizzate da singoli valori estremi, ma anche da pattern sequenziali e relazioni non lineari. Inoltre, i metodi statistici tradizionali tendono a ignorare le dipendenze temporali tra osservazioni successive, rendendo difficile la rilevazione di anomalie che emergono solo osservando le correlazioni nel tempo. Il lavoro di Frehner et al. propone quindi un metodo ibrido che combina tecniche di Machine Learning, come l'architettura autoencoder basata su LSTM, con l'approccio statistico della KDE. L'approccio proposto utilizza inizialmente un modello autoencoder per individuare le anomalie nelle serie temporali. Il modello è addestrato esclusivamente su eventi normali, senza alcun dato anomalo, e viene quindi utilizzato per apprendere una rappresentazione latente della normalità. Durante l'addestramento, il modello cerca di minimizzare l'errore di ricostruzione tra i dati originali e quelli ricostruiti, in modo che i dati

normali possano essere rappresentati in modo accurato dallo spazio latente appreso. Le osservazioni anomale, ovvero che non seguono il pattern di normalità addestrato, tendono a mostrare errori di ricostruzione maggiori. La differenza tra l'input e la ricostruzione rappresenta quindi un indicatore di anomalia. In questo lavoro però si evita l'utilizzo di una soglia fissa sugli errori di ricostruzione, che spesso si rivela inadeguata a causa della variabilità delle anomalie, introducendo una stima della densità non parametrica tramite Kernel Density Estimation. Con KDE, viene stimata una distribuzione empirica degli errori di ricostruzione generati durante l'addestramento, fornendo una misura probabilistica della normalità. In questo modo, i nuovi eventi possono essere valutati in termini di probabilità di appartenenza alla distribuzione stimata: gli eventi con bassa densità rispetto alla distribuzione stimata vengono classificati come anomalie. L'approccio KDE permette dunque di ottenere una classificazione più flessibile e accurata, superando i limiti dei metodi tradizionali che si basano su semplici soglie di errore, spesso inadeguate per gestire serie temporali complesse. Un elemento interessante nel metodo proposto è anche l'utilizzo di tecniche di data augmentation per migliorare la robustezza del modello. Viene infatti utilizzato un variational autoencoder (VAE) addestrato sui dati normali per generare nuove rappresentazioni simili a quelle normali. In questo processo, il VAE mappa ciascun dato normale su uno spazio latente, introducendo piccole perturbazioni casuali sotto forma di rumore gaussiano; i dati così modificati vengono quindi decodificati per generare nuove istanze che somigliano ai dati normali originali. Questa tecnica di augmentation incrementa la diversità dei dati normali durante l'addestramento, migliorando la capacità del modello di generalizzare e quindi di distinguere più efficacemente i pattern di normalità da quelli anomali. Al termine dell'addestramento, i dati di test vengono elaborati in sequenza. Ciascuna finestra temporale viene trasformata dall'autoencoder e l'errore di ricostruzione viene valutato in base alla distribuzione stimata tramite KDE. Questo approccio, quindi, non si limita a etichettare un evento in base all'errore di ricostruzione individuale, ma considera il contesto generale del pattern di normalità appreso, migliorando la sensibilità e l'accuratezza nel rilevamento di anomalie. Un ulteriore lavoro interessante basato sull'approccio del KDE è quello di Lindstrom et al. intitolato *Functional Kernel Density Estimation: Point and Fourier Approaches to Time Series Anomaly Detection* [16].

Altri approcci sono basati su un particolare modello chiamato **Isolation Forest**, introdotto da Liu et al. nel loro lavoro del 2008 [17]. Esso si basa sul concetto di isolamento, affermando che le anomalie, essendo rare e diverse dai dati normali, sono più facilmente isolabili. Infatti, in un albero generato casualmente, ovvero selezionando casualmente un attributo e un valore di so-

glia casuale per quell'attributo, i punti anomali tendono ad avere percorsi più brevi perché richiedono meno partizioni per essere separati rispetto ai punti normali, dato che le anomalie sono poche e i loro valori di attributo differiscono molto dagli altri punti. Un Isolation Tree viene costruito fino a raggiungere un determinato limite di altezza o fino a quando tutti i punti non sono isolati. Il cammino di un'osservazione dall'inizio dell'albero fino a un nodo foglia rappresenta la lunghezza del percorso. Le anomalie tendono ad avere quindi percorsi più brevi rispetto ai punti normali. Nell'articolo vengono anche trattati i problemi dello *Swamping*, che si verifica quando alcuni punti normali sono troppo vicini alle anomalie e vengono erroneamente identificati come anomali, e del *Masking*, che si manifesta quando le anomalie sono troppo vicine tra loro, rendendo difficile la loro identificazione attraverso l'isolamento. Questi problemi vengono affrontati attraverso il sotto-campionamento. Infatti l'Isolation Forest utilizza un sottoinsieme ridotto del dataset per costruire gli Isolation Trees e questo permette di migliorare la capacità dell'algoritmo di isolare le anomalie e controllare la dimensione del campione. Grazie a questo approccio il modello riesce quindi a isolare le anomalie e valutare nuovi campioni come anomali o normali in base alla lunghezza del percorso dalla radice fino al nodo foglia. Un lavoro interessante che utilizza questo approccio è quello presentato da Petkovski e Shehu nel loro articolo *Anomaly Detection on Univariate Sensing Time Series Data for Smart Aquaculture Using K-Means, Isolation Forest, and Local Outlier Factor* [18], in cui viene utilizzato il modello dell'Isolation Forest per la rilevazione di anomalie su serie temporali univariate per acquacoltura intelligente, confrontandolo con altri metodi di apprendimento non supervisionato, in particolare il clustering tramite K-Means. Il K-Means è un algoritmo di clustering che suddivide i dati in gruppi o cluster, associando ciascun punto al cluster con il centroide più vicino. Nell'anomaly detection, il K-Means può rivelarsi utile perché permette di identificare come anomalie i punti che risultano lontani dai centroidi dei cluster principali, ossia quelli che si trovano fuori dai gruppi più densi di punti normali. L'approccio del clustering è particolarmente vantaggioso in contesti dove le anomalie non sono necessariamente isolate in termini di valori unici o percorsi brevi, ma dove esistono pattern distintivi o insiemi di valori che deviano dalla struttura dei cluster principali. Di conseguenza, tecniche di clustering come il K-Means sono utilizzate per l'Anomaly Detection in modo complementare agli approcci basati sull'Isolation Forest, poiché consentono di rilevare anomalie che presentano caratteristiche distintive rispetto ai gruppi di dati normali e che potrebbero essere difficili da identificare con modelli basati sull'isolamento puro. Il lavoro di Petkovski e Shehu dimostra quindi come anche le tecniche di clustering possano rappresentare un valido approccio per l'anomaly detection, affiancandosi a tecniche

basate sull'Isolation Forest. Altri approcci avanzati che utilizzano l'Isolation Forest includono il lavoro di Xu et al. intitolato *Deep Isolation Forest for Anomaly Detection* [19] e quello di AbuAlghanam et al. intitolato *Fusion-based anomaly detection system using modified isolation forest for internet of things* [20], i quali evidenziano ulteriormente l'efficacia di questo modello in contesti di rilevamento delle anomalie. Articoli aggiuntivi che si concentrano invece su tecniche basate sul clustering sono quello di Enayati et al. intitolato *Time series anomaly detection via clustering-based representation* [21] e quello di Dong et al. intitolato *Subsequence Time Series Clustering-Based Unsupervised Approach for Anomaly Detection of Axial Piston Pumps* [22].

Esistono infine approcci di Anomaly Detection basati sull'apprendimento supervisionato che utilizzano dataset bilanciati e rappresentativi, ovvero con un numero sufficiente di esempi anomali, per addestrare modelli in grado di distinguere tra esempi normali e anomali. In questi casi il modello impara a riconoscere schemi e caratteristiche distintive associate agli esempi anomali. Le architetture più utilizzate in questo contesto includono le reti neurali profonde, che possono apprendere rappresentazioni complesse dai dati, le reti Long Short-Term Memory (LSTM) per l'analisi di serie temporali, le Random Forest e i Support Vector Machines (SVM), in particolare con la sua versione a una classe, la One-Class SVM. Negli ultimi anni, oltre agli approcci classici, ha guadagnato una crescente attenzione nel campo dell'Anomaly Detection un'architettura particolare: le **Graph Neural Networks (GNN)**. Esse rappresentano una classe avanzata di reti neurali progettate per lavorare su dati organizzati sotto forma di grafi, dove i nodi rappresentano delle entità e gli archi definiscono le relazioni tra queste entità. Il loro funzionamento si basa sul concetto del *messagge passing*, che consente ai nodi di aggregare informazioni dai nodi vicini e aggiornare le loro rappresentazioni in maniera progressiva in base a queste interazioni. Il processo di aggregazione delle informazioni è iterativo: ogni nodo aggiorna la propria rappresentazione aggregando i dati dai nodi vicini e da se stesso. Le operazioni fondamentali sono quindi due: l'aggregazione, dove ogni nodo riceve informazioni dai suoi vicini e le combina tramite funzioni differenziabili (come ad esempio somma o media), e l'aggiornamento, dove ogni nodo aggiorna il proprio stato in base all'aggregazione delle informazioni ricevute. L'aggiornamento dello stato di ogni nodo avviene tramite una funzione di attivazione seguita dalla moltiplicazione per un set di pesi appresi durante il processo di addestramento della rete. I pesi sono condivisi tra tutti i nodi e determinano quanto ogni nodo debba essere influenzato dai suoi vicini. Le GNN hanno visto un ampio uso in applicazioni di anomaly detection in contesti come le reti di comunicazione e le reti sociali, grazie alla loro capacità di gestire dati complessi e non strutturati in modo efficiente. Un lavoro interessante è quello di Chen et al.

intitolato *DGNN: Dynamic Graph Neural Networks for Anomaly Detection in Multivariate Time Series* [23]. Nel lavoro viene proposto un approccio avanzato per rilevarie anomalie in serie temporali multivariate, il Dynamic Graph Neural Network (DGNN), che si distingue per la gestione dinamica delle relazioni tra le metriche di sensori, anziché trattare le serie temporali come un blocco unico. A differenza dei modelli di GNN tradizionali che utilizzano grafi completamente connessi, DGNN introduce una struttura di grafi dinamici ottimizzata, riducendo così il numero di connessioni non necessarie, oltre che i costi computazionali. In questo approccio i nodi rappresentano singole metriche. Il modello DGNN crea dei sottografi dinamici raggruppando i nodi in base alla correlazione tra le metriche che rappresentano. Solo se due metriche hanno un elevato livello di correlazione, determinato tramite un valore soglia, viene creata una connessione tra i loro nodi. In questo modo ci si concentra solo sulle relazioni statisticamente rilevanti, mantenendo il grafo leggero e focalizzato solo sulle connessioni significative, migliorando l'efficienza del modello e riducendo il rischio di includere informazioni irrilevanti. Successivamente viene introdotto un meccanismo di attenzione adattiva, che rafforza l'aggregazione delle informazioni dai nodi correlati. Questo processo assegna un peso variabile ai nodi vicini, determinato in base al grado di correlazione, consentendo a ciascun nodo di attribuire maggiore rilevanza alle metriche strettamente correlate e minore rilevanza a quelle meno significative. In questo modo si migliora la capacità del sistema di rappresentare le caratteristiche locali di ogni nodo, aiutandolo a focalizzarsi soltanto sui nodi realmente rilevanti. Infine l'approccio utilizza un modello predittivo per la rilevazione delle anomalie, sfruttando le Gated Recurrent Units (GRU), che hanno il compito di modellare la dipendenza temporale e prevedere i valori futuri di ciascun nodo sulla base delle osservazioni precedenti. Il sistema confronta quindi i valori previsti con quelli osservati per ciascuna metrica, e utilizza l'errore di previsione per determinare se ci sia un'anomalia. Per farlo in modo accurato, il modello DGNN applica il metodo *Peak Over Threshold* (POT), che sfrutta la teoria dei valori estremi per stabilire una soglia ottimale che identifica le anomalie significative. In questo modo, è possibile classificare come anomala solo la previsione che superi la soglia, minimizzando i falsi positivi e migliorando l'affidabilità del rilevamento. Ulteriori approcci interessanti basati sulle Graph Neural Network includono il lavoro di Hongtai Guo et al. intitolato *EGNN: Energy-efficient anomaly detection for IoT multivariate time series data using graph neural network* [24] e quello di Ziyu Guo et al. intitolato *Grouped Graph Neural Networks for Anomaly Detection in Time Series* [25]. Infine, lavori come quello di Schmidl et al., intitolato *Anomaly Detection in Time Series: A Comprehensive Evaluation* [26], o quello di Zhang et al., intitolato *An Experimental Evaluation of*

Anomaly Detection in Time Series [27], offrono una valutazione dettagliata dei metodi di Anomaly Detection in dati di tipo Time Series, confrontando diverse tecniche in termini di efficacia e robustezza.

Nonostante in letteratura siano presenti numerosi approcci per la rilevazione di anomalie ed esistano molteplici varianti che ampliano ulteriormente le possibilità di applicazione di ciascuno di essi, non esiste un metodo universalmente superiore agli altri. L'efficacia di ogni approccio varia infatti in base al contesto applicativo e alle caratteristiche specifiche dei dati. La scelta del metodo più adatto per l'Anomaly Detection deve quindi essere valutata caso per caso, tenendo conto delle esigenze specifiche del problema.

Capitolo 3

Metodo proposto e implementazione

Nel seguente capitolo verrà descritto lo studio effettuato sul servizio **WebDAV** dell'INFN-CNAF, per i quali sono stati resi disponibili i dati temporali delle metriche del servizio che misurano l'utilizzo delle risorse del sistema. All'interno del capitolo verrà inizialmente trattata la metodologia di raccolta dei dati, seguita da una descrizione del servizio e dall'analisi delle metriche disponibili e delle loro relazioni. Successivamente, verrà illustrato il pre-processing effettuato e l'organizzazione dei dati per il training. Infine, verranno descritti i modelli di Machine Learning implementati e la loro applicazione.

3.1 Data Collection

I dati relativi al servizio WebDAV sono mantenuti tramite un database *InfluxDB v2*. InfluxDB è una piattaforma creata appositamente per raccogliere, archiviare, elaborare e visualizzare dati di serie temporali. I dati di serie temporali rappresentano una sequenza di osservazioni organizzate in ordine cronologico. Queste osservazioni, generalmente misurazioni consecutive provenienti dalla stessa fonte, vengono utilizzate per monitorare e analizzare i cambiamenti nel tempo. InfluxDB organizza i dati delle serie temporali in Bucket e Misurazioni. Un Bucket può contenere più misurazioni, e le misurazioni contengono più tag e campi. Definiamo quindi un Bucket come una posizione denominata in cui vengono archiviati i dati delle serie temporali. Una misurazione è invece un raggruppamento logico per i dati delle serie temporali. Tutti i punti di una determinata misurazione devono avere gli stessi

Tag. Definiamo un Tag come delle coppie chiave-valore con valori diversi, pensati per archiviare i metadati per ogni punto (ad esempio qualcosa per identificare l'origine dei dati come host, esperimento, e così via). Per ogni dato all'interno del database è inoltre associato un **Timestamp**, che identifica il dato temporalmente. InfluxDB permette di modificare la frequenza di conservazione dei dati attraverso un processo di downsampling. Questo consente di ridurre la **granularità dei dati** archiviati nel tempo per risparmiare spazio su disco e mantenere solo le informazioni più rilevanti a lungo termine. È possibile definire una **Retention Policy**, ovvero delle regole che specificano per quanto tempo i dati devono essere mantenuti e con quale granularità. La Retention Policy riguardante il servizio WebDAV analizzato è mostrata nella Tabella 3.1.

Nome Bucket	Frequenza raccolta dati	Tempo Conservazione
one_week	5 minuti	Una settimana
one_month	15 minuti	Un mese
six_month	2 ore	Sei mesi
all_data	3 ore	Infinito

Tabella 3.1: Retention Policy servizio WebDAV

I dati sono stati resi disponibili tramite dei dump del database effettuati in specifici istanti temporali. Dopo un processo di restore dei dump forniti, i dati sono stati estratti tramite query al database che hanno permesso di selezionare solo i dati pertinenti con lo studio, applicando filtri sui vari campi disponibili. In particolare, le query applicate sono servite a ottenere i dati con granularità differente, filtrare le finestre temporali di interesse e gli esperimenti oggetto di studio, e selezionare gli host di riferimento appartenenti al singolo esperimento. Infine, i dati estratti sono stati riorganizzarti in formato csv per facilitare l'analisi successiva.

3.2 Analisi metriche tramite Dashboard

La prima attività svolta dopo aver ottenuto i dati è stata un'analisi delle metriche disponibili. Esse coprono diverse componenti del sistema, includendo misurazioni relative alla CPU, i dischi, le interfacce di rete, input/output del sistema, carico medio del sistema, memoria e performance del servizio

WebDAV, insieme a molte altre. L'INFN-CNAF ha messo a disposizione una dashboard interattiva che mostra l'andamento delle varie metriche attraverso grafici e visualizzazioni dettagliate, facilitando l'analisi dei dati disponibili e permettendo di identificare eventuali pattern o anomalie nel comportamento delle metriche. La dashboard in questione non mostra tutte le metriche disponibili ottenute precedentemente tramite dump del database, ma solo le più rilevanti e rappresentative. Di seguito verranno mostrati i grafici delle metriche presenti nella Dashboard, con una descrizione di ciascuna e del relativo utilizzo. Le metriche sono suddivise in tre gruppi distinti: *General*, *Detailed Graph* e *StoRM WebDAV Metrics*

3.2.1 Metriche 'General'

In questa sezione verranno descritte le metriche presenti nel gruppo *'General'*, il quale raccoglie quelle misurazioni che forniscono una visione complessiva delle principali componenti del sistema monitorato. Come suggerisce il nome, queste metriche sono pensate per offrire un quadro generale, permettendo di osservare tendenze e variazioni significative che possono influenzare il comportamento complessivo dell'infrastruttura, fornendo così informazioni utili per un monitoraggio di base.

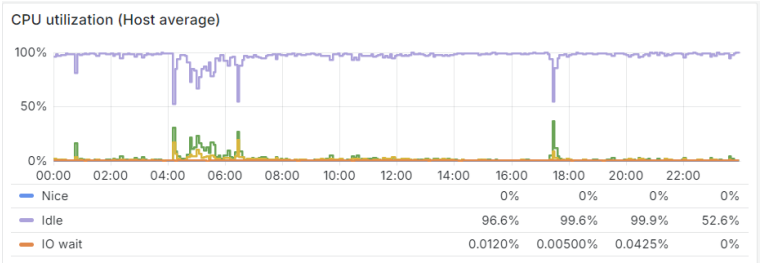

Figura 3.1: Sezione metriche CPU in dashboard INFN-CNAF

In Figura 3.1 si può notare la sezione della Dashboard che mantiene le metriche riguardanti l'utilizzo della CPU. Per le varie metriche di utilizzo disponibili, è possibile osservare il valore medio, l'ultimo valore rilevato, il valore massimo e il valore minimo. Le metriche disponibili riguardano il tempo di utilizzo della CPU da parte di processi utente, il tempo di utilizzo da parte di processi di sistema, il tempo che la CPU ha trascorso eseguendo processi con priorità "nice" (ovvero processi che hanno una bassa priorità rispetto ad altri), il tempo cui la CPU non sta eseguendo alcun processo (ossia è inattiva) e il tempo che la CPU è in attesa di operazioni di input/output.

L'insieme di queste metriche permette di avere una panoramica abbastanza completa sull'utilizzo della CPU, permettendo di comprendere in dettaglio il suo stato operativo e l'effettiva distribuzione del carico di lavoro.

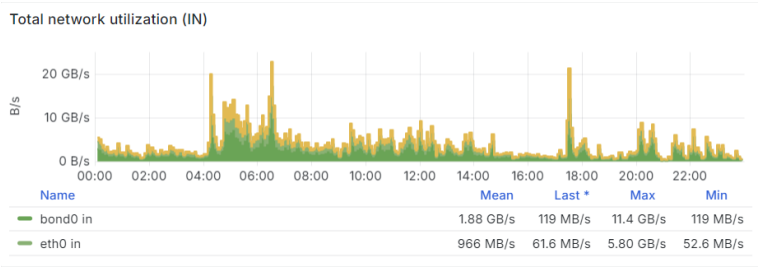

Figura 3.2: Sezione metriche Dati in ingresso in dashboard INFN-CNAF

In Figura 3.2 si può osservare la sezione della Dashboard relativa alla quantità totale di dati in ingresso che vengono trasferiti attraverso le interfacce di rete. Nel grafico è inoltre possibile osservare la metrica in relazione alle singole interfacce (*bond0* e *eth0* in questo caso).

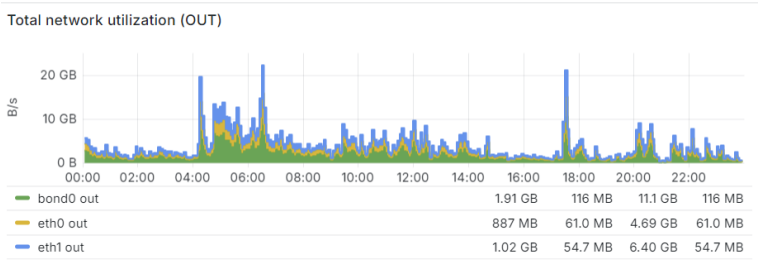

Figura 3.3: Sezione metriche Dati in uscita in dashboard INFN-CNAF

La parte relativa alla quantità totale di dati in uscita, invece che in ingresso, è mostrata nella Figura 3.3, con le relative interfacce singole visionabili.

3.2.2 Metriche 'Detailed'

In questa sezione verranno descritte le metriche presenti nel gruppo *'Detailed Graph'*. Questa sezione mostra dei grafici dettagliati di metriche riguardanti la memoria e il carico di sistema, permettendo di avere una panoramica chiara del carico del sistema in un determinato istante temporale. Per ognuna

di queste metriche è possibile visualizzare, oltre al valore puntuale in un dato istante temporale, il valore medio, l'ultimo valore rilevato, il valore massimo e il valore minimo.

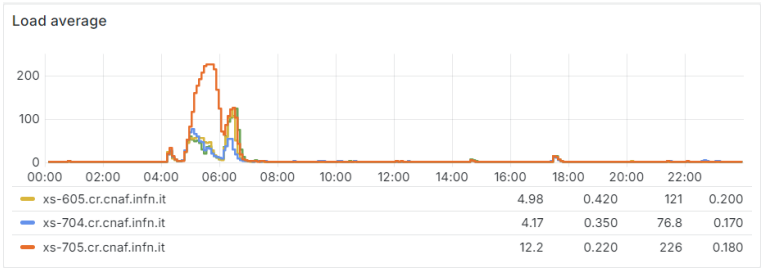

Figura 3.4: Sezione Load Average in dashboard INFN-CNAF

Nella Figura 3.4 si può osservare la sezione della Dashboard riguardante il *Load Average*, ovvero una metrica che rappresenta il carico medio all'interno di un sistema. Esso indica il numero medio di processi in esecuzione o in attesa di essere eseguiti dalla CPU, suddiviso per i vari host disponibili. Nei dati ottenuti in precedenza tramite dump del database si hanno tre diverse metriche riguardanti il Load Average, ognuna associata a intervalli temporali differenti: la metrica *'load_avg.one'*, che rappresenta il carico medio del sistema nell'ultimo minuto, la metrica *'load_avg.five'*, che rappresenta il carico medio del sistema negli ultimi 5 minuti, e la metrica *'load_avg.fifteen'*, che rappresenta il carico medio del sistema negli ultimi 15 minuti.

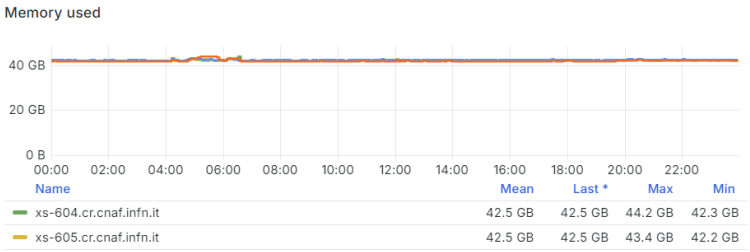

Figura 3.5: Sezione Memory Used in dashboard INFN-CNAF

Verranno ora mostrate una serie di metriche che offrono una panoramica dettagliata e completa sull'uso della memoria all'interno del sistema in ogni istante temporale. La prima metrica è mostrata in Figura 3.5. La metrica, chiamata *'Memory Used'*, rappresenta la quantità di memoria RAM

attualmente utilizzata nel sistema. Essa misura la memoria totale che è stata allocata e usata per eseguire processi attivi, ed è cruciale per monitorare l'uso della RAM nel sistema. Anche qui è possibile osservare la metrica suddivisa in base ai vari host.

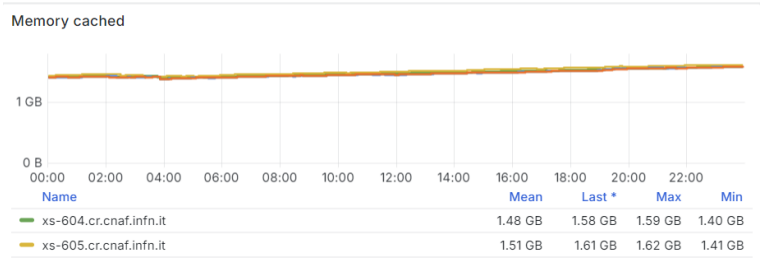

Figura 3.6: Sezione Memory Cached in dashboard INFN-CNAF

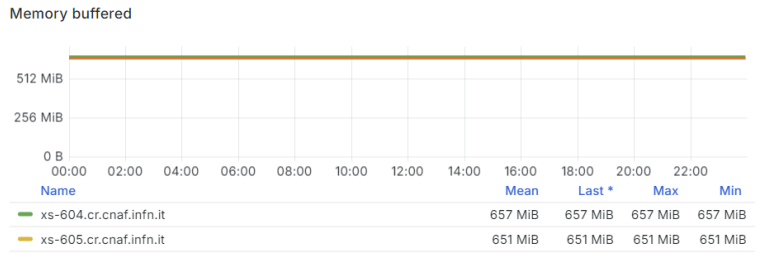

Figura 3.7: Sezione Memory Buffered in dashboard INFN-CNAF

Nelle Figure 3.6 e 3.7 è possibile osservare altre due metriche indispensabili per il monitoraggio della memoria. La prima metrica, ovvero *'Memory Cached'*, rappresenta la quantità di memoria cache utilizzata, ovvero la quantità dei dati conservati in cache per rendere più veloci le successive operazioni di lettura. Quando un file viene letto dal disco, il sistema operativo lo memorizza momentaneamente in cache nella RAM. Se in un futuro prossimo lo stesso file viene richiesto di nuovo, si sfrutta la cache per accedere ai dati più velocemente, evitando di dover così accedere nuovamente al disco. La seconda metrica invece, ovvero *'Memory Buffered'*, rappresenta la quantità di memoria utilizzata per memorizzare dati in buffer, ovvero dati che sono temporaneamente in attesa di essere scritti su disco. I dati vengono memorizzati in buffer per ottimizzare le prestazioni riguardo le operazioni di scrittura, raggruppandoli prima di scriverli su disco. In questo modo si scri-

vono blocchi di dati più grandi, riducendo di conseguenza la frequenza delle operazioni di scrittura.

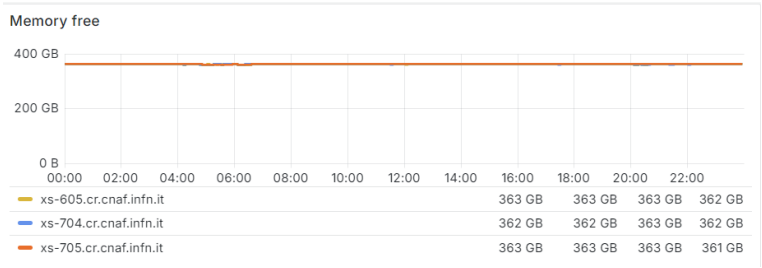

Figura 3.8: Sezione Memory Free in dashboard INFN-CNAF

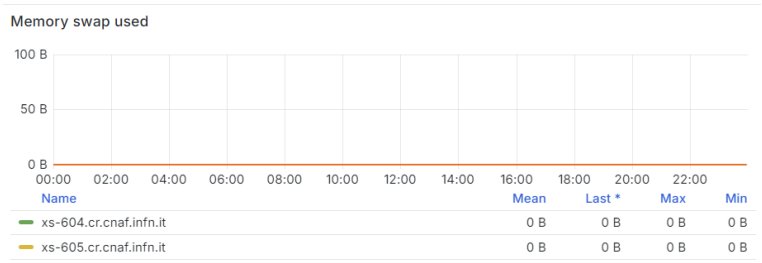

Figura 3.9: Sezione Memory Swap Used in dashboard INFN-CNAF

Infine, nelle Figure 3.8 e 3.9 è possibile osservare le ultime due metriche descrittive della memoria. La prima metrica, ovvero la metrica *'Memory Free'*, rappresenta la quantità di memoria RAM non utilizzata dal sistema operativo o dai processi. Si riferisce quindi alla porzione immediatamente disponibile per l'uso. La seconda metrica invece, ovvero la metrica *'Memory Swap Used'*, rappresenta la quantità di memoria swap utilizzata dal sistema, ovvero una porzione di spazio sul disco utilizzata come estensione della RAM. Queste metriche sono quindi utili per monitorare la disponibilità delle risorse, valutare l'efficienza complessiva del sistema e identificare eventuali problemi di memoria.

3.2.3 Metriche 'StoRM WebDAV'

In questa sezione verranno descritte le metriche appartenenti al gruppo *StoRM WebDAV metrics*. In questo gruppo sono presenti le metriche

specifiche del servizio WebDAV, che forniscono informazioni dettagliate sulle risorse del servizio e monitorano diversi parametri come il numero di richieste, il tasso di completamento e altri ancora.

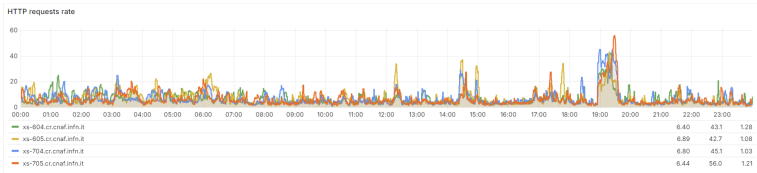

Figura 3.10: Sezione HTTP Request Rate in dashboard INFN-CNAF

Nella Figura 3.10 è possibile osservare la sezione della dashboard relativa alla *'HTTP Request Rate'*. Essa misura il numero di richieste HTTP ricevute in un determinato intervallo di tempo, riflettendo quindi il tasso con cui i client fanno richieste al server tramite il protocollo HTTP. Un alto valore della metrica indica che il server sta gestendo un numero elevato di richieste. Il monitoraggio di questa metrica è utile per comprendere il carico del server e monitorare i picchi, prevenendo potenziali sovraccarichi del sistema.

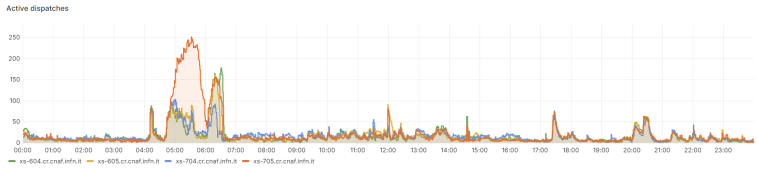

Figura 3.11: Sezione Active Dispatches in dashboard INFN-CNAF

Un'altra metrica molto utile è la metrica *'Active Dispatches'*, mostrata in Figura 3.11. La metrica descrive il numero di task attualmente in elaborazione da parte del sistema. Il monitoraggio di questa metrica fornisce una panoramica riguardo la quantità di richieste attive che il sistema sta gestendo, offrendo così informazioni utili sul carico del sistema e prevenendo potenziali sovraccarichi. Questa metrica, in combinazione con le metriche presenti in Figura 3.12 e in Figura 3.13, consente una visione completa dello stato del sistema e del carico di lavoro. Infatti, la metrica in Figura 3.12, *'Thread Pool Size'*, rappresenta il numero di thread disponibili per eseguire i task. Essi sono disponibili all'interno di una 'pool', ovvero un insieme predefinito di thread pronti a gestire le richieste, che consente di ridurre il tempo necessario per creare e distruggere i thread su richiesta. La metrica in

Figura 3.13, *'Thread Pool Utilization'*, misura la percentuale di utilizzo della pool di thread, ovvero il numero di thread attualmente utilizzati rispetto al totale disponibile. L'analisi congiunta di queste metriche consente quindi di comprendere come vengono gestite le richieste nel sistema e identificare potenziali sovraccarichi che possono condurre a una eventuale interruzione del servizio o crash del sistema.

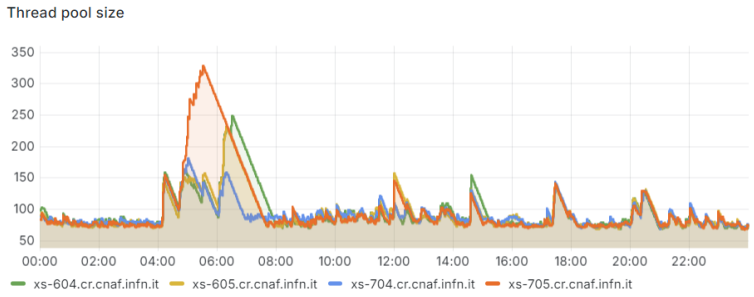

Figura 3.12: Sezione Thread Pool Size in dashboard INFN-CNAF

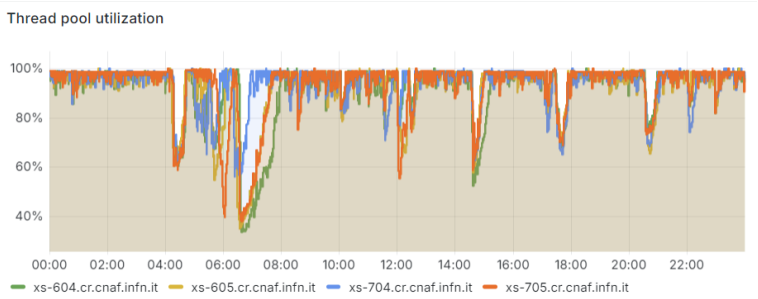

Figura 3.13: Sezione Thread Pool Utilization in dashboard INFN-CNAF

Ulteriori metriche particolarmente importanti sono mostrate in Figura 3.14. Le metriche sono *'4xx Error Rate'* e *'5xx Error Rate'* e rappresentano rispettivamente il rate di risposte HTTP che restituiscono un codice di stato di tipo 4xx e quelle che restituiscono un codice di stato di tipo 5xx. Gli errori di tipo 4xx indicano problemi nella richiesta effettuata dal client, che o non è valida o non può essere soddisfatta. Un esempio di errore 4xx è il famoso errore '404 Not Found', presente quando la risorsa richiesta non esiste sul server. Un alto Error Rate 4xx indica quindi che i client stanno inviando molte richieste sintatticamente scorrette o che non possono essere soddisfatte.

Gli errori di tipo 5xx invece indicano problemi nella gestione della richiesta da parte del server. In questo caso la richiesta del client è valida, ma il server non è stato in grado di completarla correttamente. Un esempio di errore 5xx è '500 Internal Server Error', un errore generico che indica che il server ha incontrato un problema imprevisto e non è riuscito a completare la richiesta. Monitorare la metrica *'5xx Error Rate'* permette quindi di rilevare potenziali problemi con il server come ad esempio sovraccarichi.

Figura 3.14: Sezione Error rate 4xx e 5xx in dashboard INFN-CNAF

Si possono osservare infine tre metriche molto utili per il monitoraggio del servizio WebDAV, mostrate per ogni singolo host in maniera separata. La prima è mostrata in Figura 3.15.

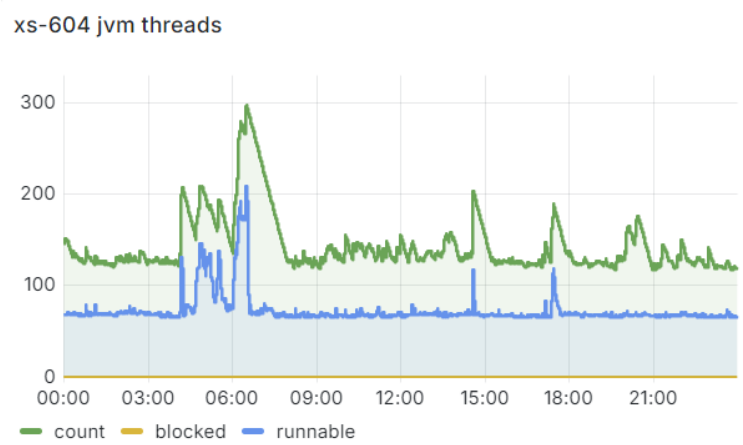

Figura 3.15: Sezione JVM Threads in dashboard INFN-CNAF

La metrica in questione è *'JVM Threads'*, relativa in questo caso all'host xs-604, e fornisce una serie di informazioni riguardo ai thread JVM (Java Virtual Machine) del servizio. In particolare, si hanno a disposizione tre campi differenti: count, blocked e runnable. Il campo count indica il numero totale di thread attivi nella JVM, ovvero in esecuzione, bloccati o in attesa. Il campo blocked indica il numero di thread attualmente bloccati, ovvero che stanno provando ad accedere a una risorsa che è già utilizzata e che attendono il rilascio di essa. Infine, il campo runnable indica il numero di thread pronti per essere eseguiti dalla CPU, anche se potrebbero non essere eseguiti immediatamente. L'insieme di queste metriche ci permette di avere una visione completa dello stato dei thread all'interno della JVM e prevenire problemi legati al sovraccarico della CPU.

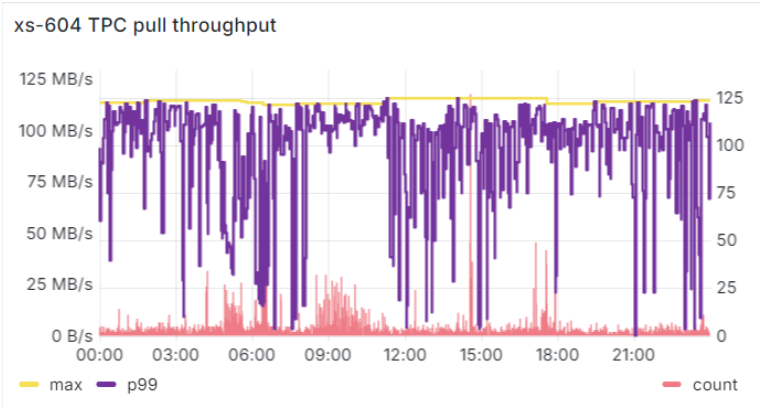

Figura 3.16: Sezione TPC Pull Throughput in dashboard INFN-CNAF

La metrica successiva è mostrata in Figura 3.16. La metrica, *'TPC Pull Throughput'*, rappresenta la quantità di dati trasferiti tra server e la velocità di questo trasferimento. In particolare, si hanno a disposizione tre campi differenti: max, p99 e count. Il campo max indica il valore massimo di throughput registrato durante un'operazione di TPC pull. Questo valore indica la massima velocità di trasferimento raggiunta durante il periodo di osservazione. Il campo p99 si riferisce al 99° percentile di throughput, ovvero il valore al di sotto del quale si trovano il 99% delle misurazioni di throughput registrate. Esso fornisce una rappresentazione di come si comporta il throughput nella maggior parte dei casi, escludendo valori outlier particolarmente alti o bassi. Infine, il campo count indica il numero totale di operazione di TPC pull. Questo campo evidenzia quante operazioni di trasferimento sono state completate, fornendo una misura quantitativa delle operazioni di TPC

pull effettuate e un'idea del carico del sistema in base al numero di richieste che sta gestendo. Combinando queste metriche si può valutare la qualità e la quantità delle operazioni di TPC pull.

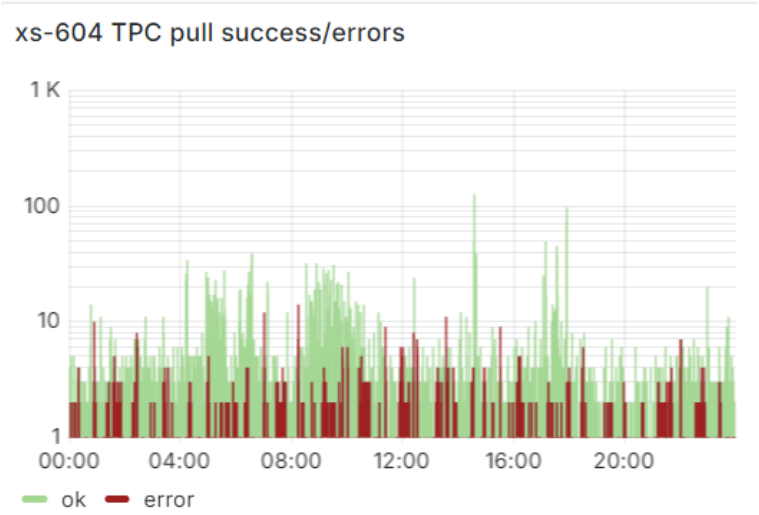

Figura 3.17: Sezione TPC Pull success/error in dashboard INFN-CNAF

L'ultima metrica considerata è la metrica mostrata in Figura 3.17. La metrica in questione è *'TPC Pull success/error'*, una metrica che rappresenta il rapporto tra il numero di operazioni di TPC pull completate con successo e quelle che hanno prodotto degli errori. In un servizio WebDAV, una richiesta TPC pull avviene quando un server richiede direttamente a un altro server di trasferire dei dati al proprio storage, senza che il client debba svolgere alcuna azione. Questa metrica fornisce quindi una panoramica delle operazioni di trasferimento da server a server, evidenziando il numero di operazioni andate a buon fine rispetto a quelle che hanno prodotto errori. Questa metrica risulta essere fondamentale per monitorare le operazioni di trasferimento tra server e permette di identificare eventuali problemi di rete o risorse che possono compromettere il corretto flusso di dati.

3.3 Analisi Relazione tra metriche

Dopo aver ottenuto i dati disponibili in formato CSV e aver studiato e analizzato le metriche tramite la Dashboard dell'INFN-CNAF, è iniziata una fase di analisi delle relazioni tra le diverse metriche. Le metriche sono state

inizialmente suddivise per tipologia e per componente di sistema di riferimento, per poi procedere con un'ulteriore fase di analisi volta a esplorare le loro interdipendenze. L'obiettivo è stato quello di comprendere la correlazione tra le metriche di una stessa componente e successivamente estendere l'analisi per individuare possibili correlazioni tra le metriche di componenti diverse. L'analisi è stata avviata sulle metriche relative alla stessa componente, considerando finestre temporali prive di anomalie, ed è stata condotta tramite script Python utilizzando le seguenti librerie:

- **Pandas**: uno strumento di analisi e manipolazione dei dati strutturati che permette di caricare, filtrare, trasformare e analizzare dataset in modo efficiente grazie alle particolari strutture dati che utilizza, quali DataFrame e Series.
- **Matplotlib**: una libreria che consente di creare visualizzazioni grafiche dei dati, facilitando l'interpretazione dei risultati dell'analisi.
- **Seaborn**: una libreria che offre strumenti avanzati per la visualizzazione statistica dei dati.

Lo script filtra i dati in base alla componente scelta e calcola la matrice di correlazione dei dati. La matrice di correlazione è una struttura che mostra le correlazioni tra tutte le coppie di metriche. La misura di correlazione scelta per la costruzione della matrice è la **Correlazione per Ranghi di Spearman**. Essa è una misura non parametrica di dipendenza tra due variabili che quantifica il grado di associazione monotona, ossia la tendenza di una variabile a variare in una direzione consistente quando un'altra variabile varia. Per ciascuna variabile, si assegnano ranghi ai valori in base all'ordine crescente e successivamente si calcola il coefficiente di Pearson tra i ranghi delle due variabili, ottenendo un valore compreso tra -1 e +1:

- Un valore di **+1** indica una perfetta relazione monotona crescente, in cui i ranghi delle due variabili aumentano insieme.
- Un valore di **-1** indica una perfetta relazione monotona decrescente, in cui i ranghi di una variabile aumentano mentre i ranghi dell'altra diminuiscono
- Un valore di **0** indica assenza di relazione monotona, ovvero i cambiamenti nei ranghi delle variabili non sono coordinati.

La formula per il calcolo del Coefficiente di Correlazione di Spearman ρ tra due variabili X e Y è la seguente:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

dove d_i rappresenta la differenza tra i ranghi delle due variabili per ogni coppia di osservazioni i, e n è il numero totale di osservazioni. La scelta di utilizzare la correlazione di Spearman rispetto ad altre misure di correlazione classiche, come quella di Pearson, è motivata principalmente dalla sua robustezza nella gestione di valori outlier e nella capacità di catturare relazioni monotone tra variabili che non seguono necessariamente un andamento lineare. Infatti, a differenza della correlazione di Pearson che misura esclusivamente la relazione lineare tra le variabili e può essere influenzata da picchi estremi, la correlazione di Spearman calcola il grado di associazione basandosi sui ranghi, risultando quindi meno sensibile alle fluttuazioni improvvise e ai valori outlier. Questo approccio permette di ottenere una misura della relazione tra variabili in contesti in cui i dati non sono sempre lineari o continuativi e possono avere dei picchi non rappresentativi del comportamente generale, come nelle metriche di un sistema informatico. Nell'analisi svolta, non tutte le metriche a disposizione nel periodo analizzato hanno dei valori validi. Alcune hanno infatti dei valori *nan*, che verranno rappresentati all'interno della Matrice di Correlazione da una riga vuota.

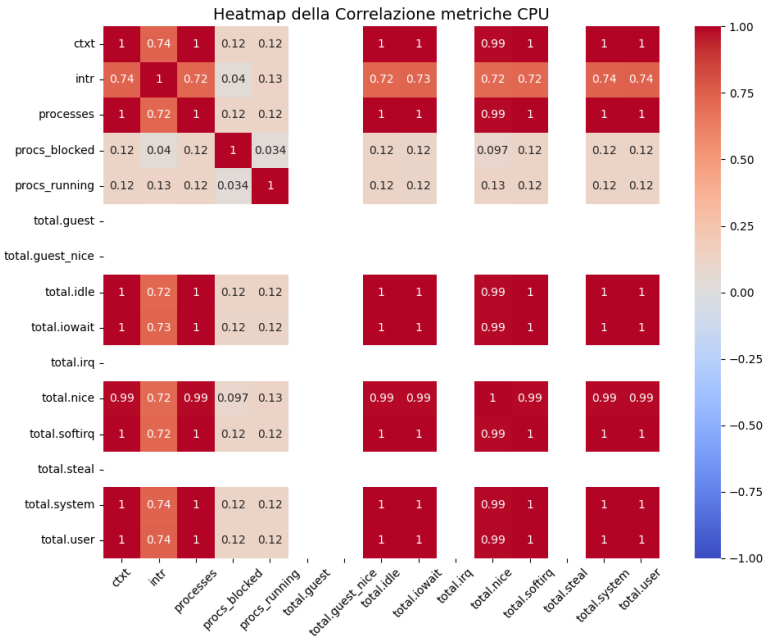

Figura 3.18: Matrice di correlazione per metriche relative alla CPU

Nella Figura 3.18 si può osservare la matrice di correlazione di un periodo privo di anomalie che si riferisce alle metriche relative alla CPU. Dalla matrice di correlazione risultante è possibile osservare il comportamento delle metriche e le loro interazioni. Analizzando la matrice possiamo individuare e scoprire le relazioni tra le diverse metriche e comprendere come esse influenzino e siano influenzate l'una dall'altra. Queste informazioni ci permettono di evidenziare correlazioni significative e identificare eventuali metriche che potrebbero avere un impatto rilevante sul funzionamento complessivo del sistema. Inoltre, ci permette di capire quali metriche siano ridondanti e espressive di informazioni simili o sovrapponibili. Questo permette di semplificare i dati da utilizzare successivamente, eliminando metriche meno utili senza perdere informazioni significative. Nella Figura 3.19 si può osservare un'ulteriore matrice di correlazione, riguardante lo stesso periodo privo di anomalie ma che si riferisce alle metriche relative alle percentuali di memoria.

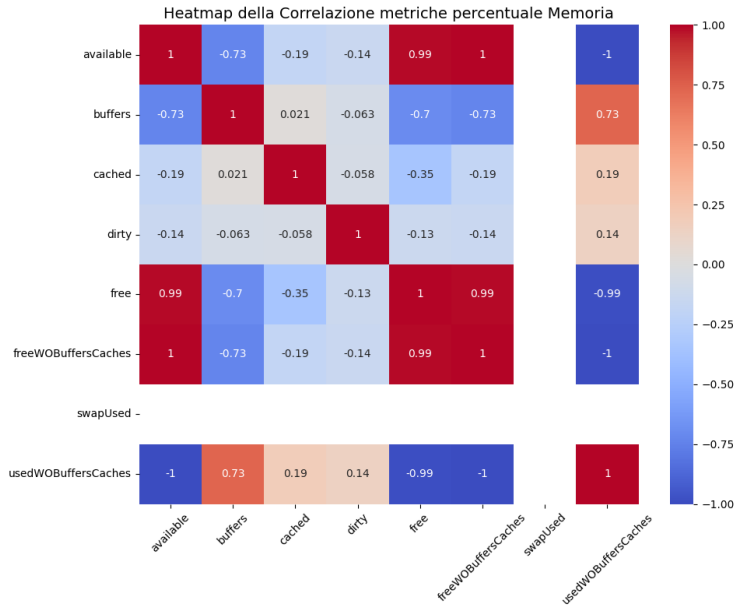

Figura 3.19: Matrice di correlazione per metriche relative alle percentuali di memoria

Anche in questo caso vengono messe in evidenza le influenze delle varie metriche l'una sull'altra. Ad esempio, si nota subito l'altissima correlazione positiva tra la metrica *available*, che rappresenta la percentuale della memoria totale che è attualmente disponibile per l'uso, e la metrica *free*, che rappresenta la percentuale della memoria fisica totale che non è attualmente in uso

da alcun processo. Le metriche sono infatti molto simili tra loro e esprimono un concetto sovrapponibile, quindi la loro correlazione è molto alta. Al contrario, si osserva una situazione opposta analizzando la metrica *freeWOBuffersCaches*, che rappresenta la percentuale di memoria libera escludendo la memoria occupata da buffer e cache, e la metrica *usedWOBuffersCaches*, che rappresenta la percentuale di memoria utilizzata escludendo la memoria usata da buffer e cache. Come previsto si può quindi osservare una perfetta correlazione negativa, dovuta alla diminuizione di memoria utilizzata quando la memoria libera aumenta. Questo processo è stato ripetuto per tutte le componenti del sistema, analizzando e comprendendo le relazioni tra le metriche per ciascuna componente. L'obiettivo di questa fase è stato quello di identificare delle correlazioni significative tra le metriche e possibili ridondanze. Si è successivamente proceduto alla generazione e l'analisi di Matrici di Correlazione miste, nelle quali sono state combinate metriche relative a componenti diverse all'interno del sistema. L'obiettivo di questa fase è stato quello di esplorare le interazioni tra metriche appartenenti a componenti diverse e comprendere le relazioni esistenti all'interno dell'intero sistema.

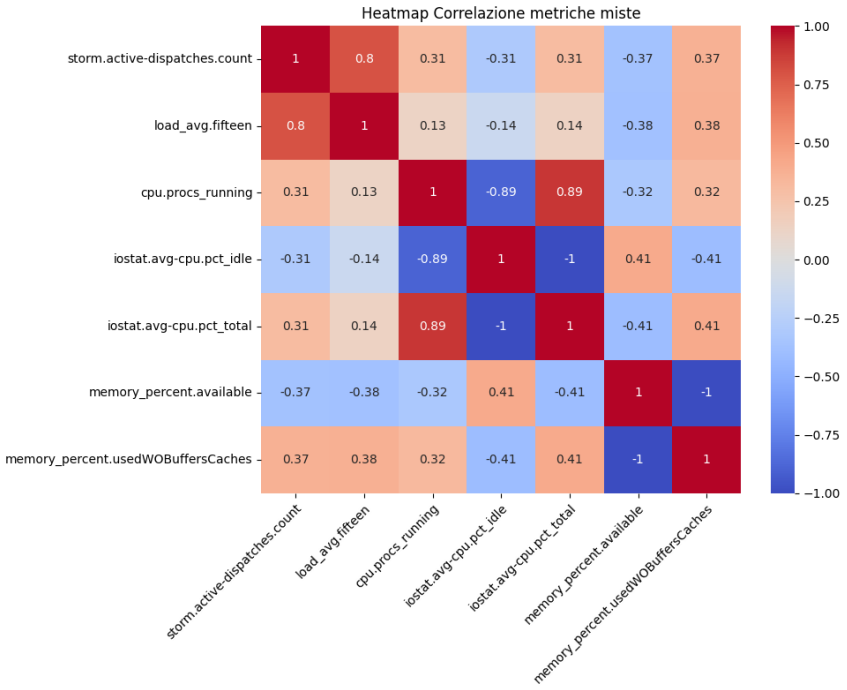

Figura 3.20: Matrice di correlazione per metriche relative a componenti miste 1

Analizzando la Figura 3.20 è possibile identificare e comprendere relazioni appartenenti a metriche relative a componenti differenti all'interno del sistema. Osservando ad esempio la riga della metrica *storm.active-dispatches.count*, una metrica che misura il numero di richieste attualmente in elaborazione da parte del servizio WebDAV, si può subito intuire la correlazione con le altre metriche. Si può infatti notare una correlazione positiva molto alta con la metrica *load_avg.fifteen*, una metrica che rappresenta la media del carico di lavoro del sistema negli utltimi quindici minuti, come era prevedibile osservare data la natura delle metriche coinvolte. Infatti, questa correlazione è attesa, perché all'aumento di richieste attualmente in elaborazione ci si aspetta un aumento del carico medio del sistema. Si può osservare una correlazione positiva anche con le metriche *cpu.procs_running* e *iostat.avg-cpu.pct_total*, che rappresentano rispettivamente il numero di processi attualmente in esecuzione sul processore e la percentuale media totale di utilizzo della CPU. Anche questa correlazione positiva è una correlazione prevedibile, dato che all'aumento delle richieste attualmente in elaborazione ci si aspetta un aumento del numero di processi pronti a gestire queste richieste e un conseguente aumento dell'utilizzo della CPU. Si nota invece una correlazione negativa con le metriche *iostat.avg-cpu.pct_idle* e *memory_percent.available*, che rappresentano rispettivamente la percentuale di tempo in cui la CPU è rimasta inattiva (ovvero non ha eseguito nessun processo utente, di sistema, o è stata in attesa di operazioni di I/O) e la percentuale di memoria totale del sistema che è attualmente disponibile per l'uso. Questa correlazione negativa è presente perché, con l'incremento del numero di richieste attualmente in elaborazione, la CPU verrà utilizzata maggiormente e di conseguenza rimarrà meno tempo inattiva. Inoltre, la percentuale di memoria del sistema disponibile per l'uso diminuirà a causa delle maggiori risorse utilizzate dai processi in esecuzione. Infine, come previsto, e in modo esattamente speculare rispetto alla metrica *memory_percent.available*, si osserva una correlazione positiva con la metrica *memory_percent.usedWOBuffersCache*, che rappresenta la percentuale di memoria utilizzata escludendo quella impiegata per buffer e cache. Infatti, con l'aumento del numero di richieste in elaborazione, ci si aspetta un incremento della memoria utilizzata nel sistema per supportare il carico aggiuntivo delle operazioni. Un'ulteriore Matrice di Correlazione è mostrata in Figura 3.21.

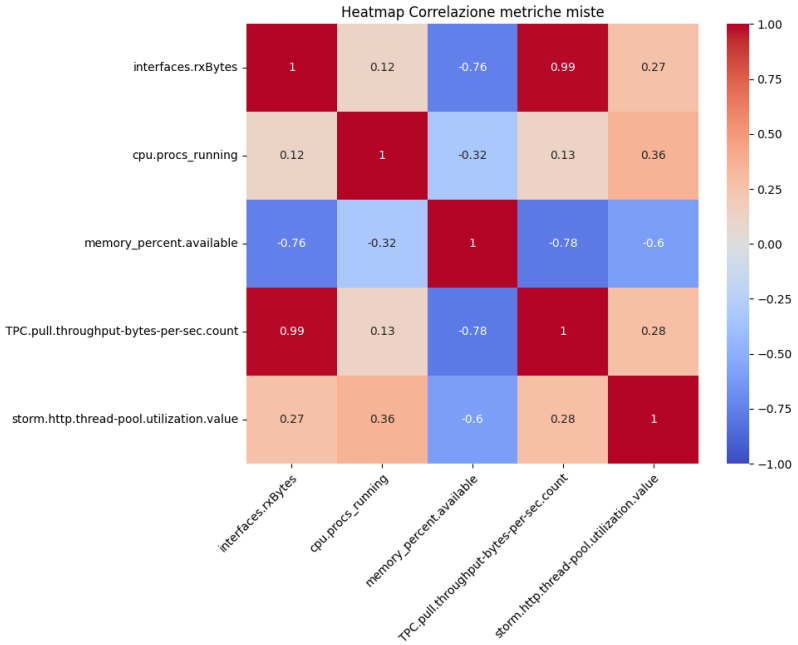

Figura 3.21: Matrice di correlazione per metriche relative a componenti miste 2

Nella prima riga della matrice è possibile osservare la metrica *interfaces.rxBytes*. Questa è una metrica composta, creata tramite script Python e aggiunta all'insieme delle metriche già disponibili. La metrica non è altro che il risultato della somma del campo *rxBytes* delle metriche riguardanti le varie interfacce di rete disponibili, che rappresenta la quantità di byte ricevuti. La metrica risultante rappresenta quindi il traffico di rete in ingresso, misurando la quantità di dati, in byte, ricevuti tramite tutte le interfacce di rete. Essa è stata messa in relazione con varie metriche provenienti da diverse componenti del sistema. La prima correlazione risultante è quella con *cpu.procs_running*, una metrica che, come descritto in precedenza, indica il numero di processi attualmente in esecuzione sulla CPU. Comprendere la correlazione con questa metrica consente di analizzare il legame tra l'incremento del traffico di rete e il numero di processi in esecuzione, mettendo in relazione le interfacce di rete e la CPU. La correlazione risultante è una correlazione moderatamente positiva. Ciò indica che, in alcuni casi, un maggior traffico di rete in ingresso potrebbe richiedere un numero maggiore di processi in esecuzione per gestirlo, anche se molti di essi potrebbero non essere direttamente collegati all'attività di rete. Un incremento dei byte ricevuti corrisponde quindi

a un modesto incremento dei processi in esecuzione. La seconda correlazione risultante è quella con la metrica *memory_percent.available*, che rappresenta la percentuale di memoria disponibile nel sistema. Analizzare la correlazione tra queste due metriche permette di approfondire il legame tra le interfacce di rete e la memoria, evidenziando come l'attività di rete possa influenzare l'utilizzo delle risorse di memoria. La correlazione risultante è una correlazione fortemente negativa, indicando che l'aumento del traffico di rete in ingresso comporta un maggiore consumo di memoria, probabilmente dovuto all'elaborazione e alla temporanea memorizzazione dei dati. La terza correlazione risultante è quella con la metrica *TPC.pull.throughput-bytes-per-sec.count*, che misura il numero di operazioni TPC pull completate in un determinato intervallo di tempo. Come previsto, la correlazione risultante è fortemente positiva. Infatti, se il traffico di rete in ingresso è elevato, ci si aspetta che anche il numero di operazioni di TPC pull aumenti, in quanto più dati ricevuti dalle interfacce di rete, soprattutto in un servizio WebDAV, indicano più operazioni di trasferimento in corso. Questa correlazione permette quindi di comprendere a fondo il servizio WebDAV, evidenziando come la maggior parte del traffico di rete in ingresso sia direttamente legato alle operazioni di TPC pull. Infine, si osserva una correlazione con la metrica *storm.http.thread-pool.utilization.value*, che misura la percentuale di utilizzo della pool di thread. La correlazione risultante è moderatamente positiva, poiché un aumento del traffico di rete in ingresso tende a incrementare il numero di richieste gestite dal server. Ciò si traduce in un maggiore utilizzo della pool di thread, in quanto un numero maggiore di thread viene impiegato per gestire le richieste in arrivo.

La creazione delle Matrici di Correlazione Miste è proceduta mettendo a confronto diverse metriche e ha permesso di comprendere le relazioni tra le varie componenti del sistema e di individuare le metriche più rappresentative di ciascuna componente. Combinando queste analisi con uno studio approfondito del comportamento delle componenti e delle relative metriche, è stato possibile ottenere una visione completa delle dinamiche interne del sistema, identificando metriche chiave e ridondanti e selezionando accuratamente quelle più significative per analisi future.

3.4 Separazione osservazioni normali e anomale

Dopo aver compreso il significato delle metriche disponibili e aver analizzato le loro relazioni, si è proceduto con una ulteriore fase di analisi dei

dati sfruttando la dashboard dell'INFN-CNAF. Lo scopo di questa fase di analisi è stato quello di suddividere i dati in osservazioni normali e osservazioni anomale. Infatti, i dati disponibili erano non etichettati. È stato quindi necessario fissare un criterio per classificare un'osservazione come normale o anomala. L'identificazione del criterio corretto è avvenuta attraverso una fase di analisi approfondita delle varie metriche presenti nella dashboard, supportata dai consigli di esperti del dominio dell'INFN-CNAF riguardo a potenziali situazioni anomale in specifici intervalli temporali. Si è quindi proceduto all'analisi degli intervalli segnalati come anomali per poter comprendere le cause delle anomalie e ricercare potenziali pattern utili a definire una situazione come anomala. Dopo aver analizzato molteplici situazioni anomale, è stato possibile individuare un pattern nelle anomalie. Si è infatti individuata una metrica che, in modo chiaro e facilmente riconoscibile, segnalava un'anomalia nel sistema. La metrica in questione è *'TPC Pull success/error'*. È stato infatti osservato che, nei momenti in cui il sistema risultava inattivo a causa di probabili crash o malfunzionamenti temporanei, la metrica in questione mostrava un'interruzione nella ricezione dei dati, facilmente identificabile e oggettiva grazie alla rappresentazione grafica utilizzata per visualizzarla.

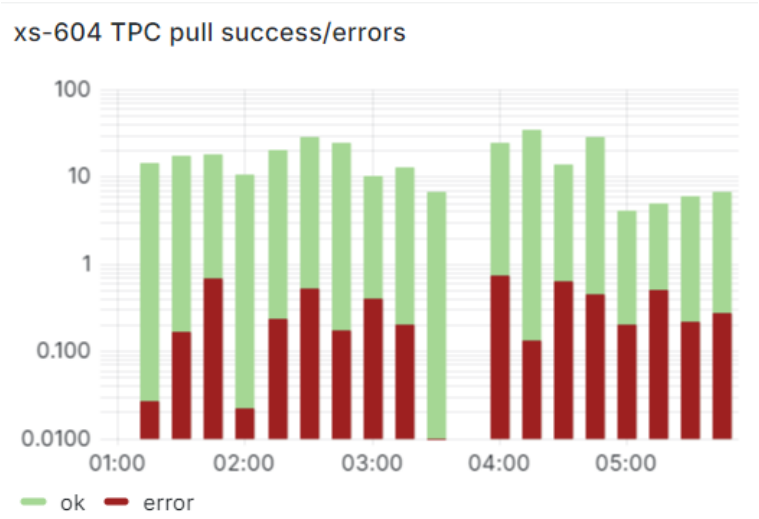

Figura 3.22: Esempio di anomalia individuabile tramite metrica TPC pull success/error

Come mostrato nella Figura 3.22, l'anomalia risulta essere facilmente individuabile grazie alla mancanza di dati in un determinato istante temporale.

È importante notare che l'assenza di dati non indica l'assenza di richieste di TPC pull, ma rappresenta un intervallo di tempo per il quale non sono disponibili informazioni. Questo accade probabilmente perché, durante un crash del sistema o un malfunzionamento, i processi responsabili del monitoraggio delle operazioni TPC pull vengono interrotti e il sistema non è in grado di registrare correttamente i dati riguardanti tali operazioni. La diretta conseguenza è l'osservazione di un "buco" di informazioni nella rappresentazione grafica della metrica, che riflette la mancanza di attività del sistema, evidenziandone un'anomalia. Questo comportamento può essere utilizzato come criterio per classificare le osservazioni come normali o anomale. Si è quindi deciso di utilizzare questa metrica per tracciare tutti i casi in cui si verificasse una mancanza di dati, al fine di identificare le anomalie nel sistema e separare i periodi di normale attività da quelli caratterizzati da malfunzionamenti o crash. Un ulteriore esempio è mostrato nella Figura 3.23, che rappresenta un'anomalia individuata correttamente tramite la metrica *'TPC Pull success/error'*. L'osservazione, analizzata con Retention Policy *one_month*, ovvero con una frequenza di raccolta dati ogni 15 minuti, presenta una mancanza di dati alle ore 07:00. In linea con quanto osservato dal buco di informazioni presente nella metrica, il servizio risulta essere inattivo in corrispondenza di quell'orario, con un'assenza di valori anche per le altre metriche presenti nella dashboard.

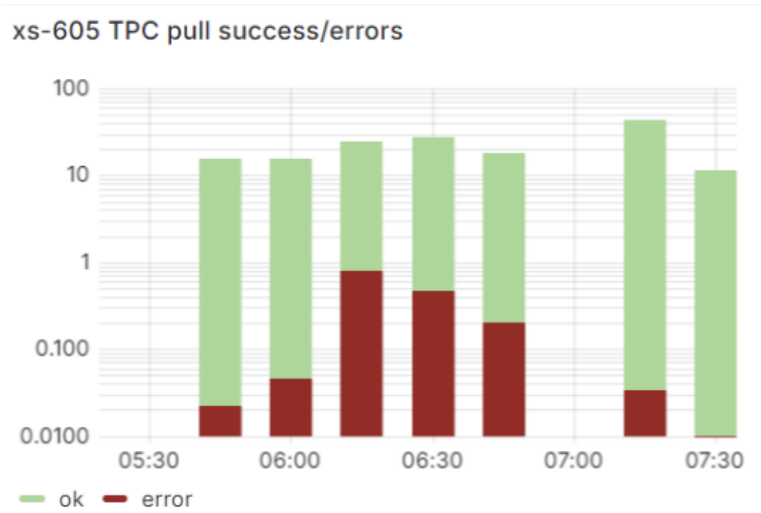

Figura 3.23: Caso d'esempio 1 - Anomalia visibile tramite metrica TPC pull success/error

Si è quindi proceduto a individuare il motivo del malfunzionamento o crash del sistema. Dopo un'attenta analisi sistematica delle metriche presenti nella dashboard, si è identificato il problema principale che ha portato il sistema al crash. In particolare, come mostrato nella Figura 3.24, il problema ha avuto origine a causa di un incremento incontrollato dei dispatch attivi, ovvero il numero di attività o processi che sono attualmente in fase di elaborazione all'interno del sistema. Questo comportamento ha determinato un progressivo sovraccarico, con i dispatch che sono aumentati costantemente, fino a saturare completamente le risorse disponibili. Il sistema, non riuscendo più a gestire il carico di lavoro, ha infine subito un malfunzionamento. Il picco massimo dei dispatch attivi, evidenziato con un cerchio rosso alle 06:45, ha causato il crash del sistema, rendendolo irraggiungibile fino alle 07:15, come indicato dal cerchio blu.

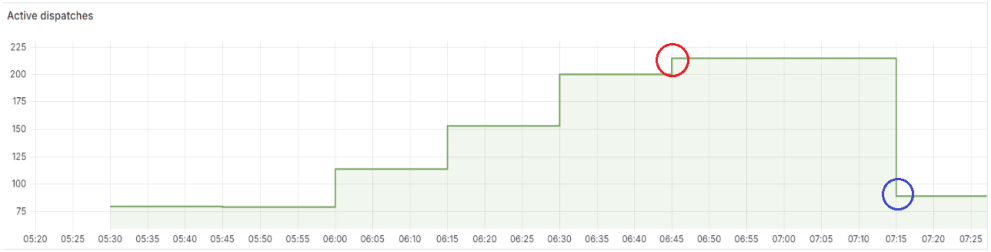

Figura 3.24: Caso d'esempio 1 - Active Dispatches

Inoltre, nella Figura 3.25, è mostrata la metrica *'Thread Pool Utilization'*, che mostra l'incremento dell'utilizzo della pool di thread fino a saturazione, lasciando il sistema senza thread disponibili per gestire le continue richieste in arrivo. Il picco massimo della metrica, raggiunto alle 06:45, è evidenziato tramite un cerchio rosso e indica un utilizzo della Pool di thread pari al 100%. Da quel momento in poi il sistema risulta essere irraggiungibile. Successivamente, un cerchio blu evidenzia il momento in cui il sistema torna disponibile, alle 07:15, mostrando un carico di lavoro ristabilito e un utilizzo della pool di thread rientrato a livelli regolari.

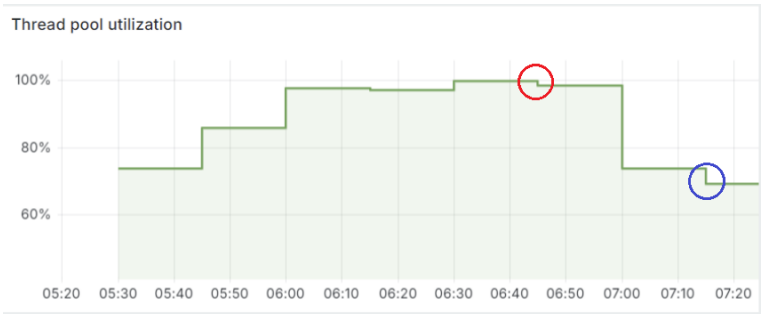

Figura 3.25: Caso d'esempio 1 - Thread Pool Utilization

Il comportamento anomalo del sistema è confermato anche dalle altre metriche presenti nella Dashboard. In particolare, nella Figura 3.26, è possibile osservare le metriche relative al Throughput e la loro variazione temporale, che confermano l'analisi effettuata precedentemente sulle metriche di *Active Dispatches* e *Thread Pool Utilization*. Si osservano: in giallo, la metrica relativa al valore massimo del throughput in MB/s raggiunto in ogni intervallo temporale, in viola, la metrica relativa al 99° percentile del throughput e, in rosa, la metrica relativa al numero di operazioni TPC pull effettuate in un determinato periodo di tempo. Si osserva che il valore massimo di throughput inizia a diminuire progressivamente man mano che il sistema si saturara. Infatti, più richieste vengono elaborate contemporaneamente, maggiore è il grado di saturazione del sistema e minore risulta la velocità massima di throughput. Esso raggiunge il suo minimo alle 06:45, momento in cui si registra l'ultima misurazione del sistema prima del crash, per poi aumentare nuovamente dopo il ripristino del sistema. Il valore relativo al 99° percentile, che rappresenta il valore sotto il quale si trovano il 99% delle misurazioni, rimane invece più stabile rispetto al massimo. Questo indica che, nonostante i picchi, il throughput tende a mantenersi su valori relativamente costanti. Tuttavia, questo comportamento è osservabile fino alle 06:15. Successivamente, anche il 99° percentile inizia a diminuire a causa del sovraccarico del sistema, raggiungendo il minimo alle 06:45, prima del crash. Come per il valore massimo, anche il 99° percentile torna a crescere quando il sistema viene ripristinato e il carico di lavoro torna normale. Infine, prevedibilmente, anche il valore relativo al numero di operazioni TPC pull effettuate mostra un buco di informazioni alle ore 07:00, dovuto al crash del sistema e alla conseguente sospensione dei processi responsabili del monitoraggio delle operazioni.

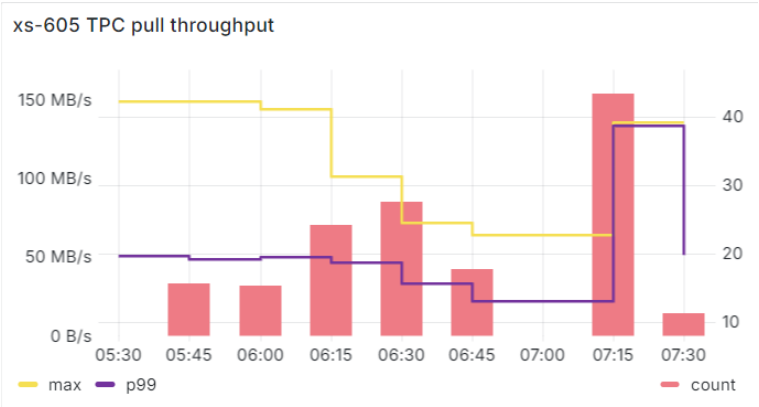

Figura 3.26: Caso d'esempio 1 - Misure relative al Throughput

Questo processo di analisi è stato ripetuto per tutte le anomalie individuate grazie alla metrica *TPC pull success/error*. L'obiettivo è stato quello di individuare dei pattern anomali ricorrenti ancora prima di utilizzare un modello di Machine Learning, in modo da avere una comprensione migliore delle anomalie del sistema e ottenere un riferimento con cui confrontare i risultati ottenuti da futuri modelli. In alcuni casi si è riusciti a comprendere perfettamente la causa dell'anomalia, ma in molti altri l'osservazione e l'analisi delle metriche presenti nella dashboard non sono state sufficienti per identificare la radice del problema. L'obiettivo di questa fase è stato comunque raggiunto, consentendo di annotare accuratamente tutti gli intervalli temporali in cui si sono verificate anomalie e di comprendere la causa e la struttura di alcune di esse.

3.5 Pre-processing dei dati e scelta metriche

Una volta terminata la fase di identificazione dei momenti anomali e lo studio di essi, si è passati ad una fase di pre-processing dei dati. Infatti, nonostante si siano individuati i momenti anomali, è necessario comprendere come organizzare i dati a disposizione per poter permettere a un modello di Machine Learning di riuscire a classificare correttamente non soltanto le anomalie puntuali, ma anche quelle collettive, contestuali e lentamente emergenti. Questa fase di pre-processing è quindi necessaria per preparare i dati per l'addestramento dei modelli, garantendo che essi siano strutturati correttamente e includano le informazioni rilevanti per permettere al modello di distinguere comportamenti normali e anomali.

La prima decisione presa ha riguardato il tipo di dati da utilizzare. Infatti, come descritto nel Paragrafo 3.1, si hanno a disposizione dati con diversa granularità, definita in base alla Retention Policy applicata. Le Retention Policy disponibili sono quattro, come descritto precedentemente:

- **one_week**, con dati ogni 5 minuti e tempo di conservazione di una settimana;
- **one_month**, con dati ogni 15 minuti e tempo di conservazione di un mese;
- **six_month**, con dati ogni 2 ore e tempo di conservazione di sei mesi;
- **all_data**, con dati ogni 3 ore e tempo di conservazione infinito.

La scelta migliore sarebbe adottare la Retention Policy *one_week*, che consentirebbe di ottenere una granularità più elevata. Con questa scelta si potrebbe disporre di dati più dettagliati, raccolti con maggior frequenza e privi di elaborazioni intermedie, come operazioni di media o simili, provenendo direttamente dai sensori. Al contrario, con le altre Retention Policy, i dati con frequenza temporale minore (ad esempio ogni 2 ore) sono ottenuti con operazioni di media o simili su quelli con granularità più elevata, riducendo la precisione delle informazioni fornite dai sensori. Tuttavia, i dati a disposizione sono limitati. Di conseguenza, si è cercato di trovare un compromesso tra la precisione dei dati e la loro quantità. Si è quindi deciso di utilizzare la Retention Policy *one_month*, che ha consentito di poter usufruire di un numero consistente di osservazioni e di mantenere una discreta precisione nelle misurazioni, seppur mediate. Questa scelta si è rivelata la più sensata per garantire la disponibilità di un numero significativo di dati, sia anomali che normali, con cui lavorare. In alternativa, il campione sarebbe stato troppo ridotto per consentire un'analisi efficace e adeguata. Si è poi proceduto a svolgere un primo pre-processing dei dati per organizzare al meglio le informazioni sensoriali provenienti dai vari file CSV ottenuti in precedenza. Utilizzando uno script Python basato sulla libreria **Pandas**, i file CSV sono stati riorganizzati in un formato più strutturato e leggibile, permettendo una visione chiara delle misure associate a ciascun istante temporale. I file originali, infatti, erano organizzati in modo disordinato: per ogni istante temporale, venivano prima elencate tutte le misurazioni di una determinata metrica associate a informazioni di durata, seguite poi dalle misurazioni della stessa metrica relative ai valori. Questo schema si ripeteva per tutte le metriche disponibili, rendendo i file eccessivamente lunghi e difficili da interpretare. Lo script Python ha permesso di associare ogni metrica a un preciso

istante temporale, separando le informazioni di durata da quelle di valore. Nel nuovo formato, ogni colonna rappresenta una misurazione di una specifica metrica, mentre ogni riga corrisponde a un determinato istante temporale. Il risultato è un file più leggibile, ordinato cronologicamente, in cui tutte le misure di un determinato istante sono raggruppate insieme. Questo processo di riorganizzazione è stato essenziale per preparare i dati alle fasi successive, in quanto era fondamentale avere un dataset organizzato temporalmente e con tutte le metriche disponibili associate a ciascun istante. Questo approccio ha semplificato il trattamento dei dati nelle analisi successive, offrendo una visione più chiara e intuitiva, facilitando così la comprensione complessiva del dataset. Si è affrontata, successivamente, una delle decisioni più rilevanti dell'intero processo di preparazione dei dati, indispensabile per la fase successiva di addestramento dei modelli di Machine Learning. La scelta in questione riguarda l'organizzazione delle finestre temporali da considerare in una singola osservazione. Come già accennato, l'obiettivo è fare in modo che il modello sia in grado di classificare correttamente non solo le anomalie puntuali, ma anche quelle collettive, contestuali e lentamente emergenti. Per raggiungere questo scopo, una singola osservazione non può essere trattata come un unico istante temporale isolato, ma deve essere considerata come una finestra temporale composta da più istanti. Questo approccio consente di catturare l'evoluzione delle metriche nel tempo, piuttosto che limitarsi a una rappresentazione statica di un singolo momento. Si è quindi deciso di trattare la grandezza della finestra temporale come un iperparametro, considerando tre diverse alternative. I dati sono stati successivamente organizzati di conseguenza, in modo da riflettere questa struttura e consentire un'analisi coerente.

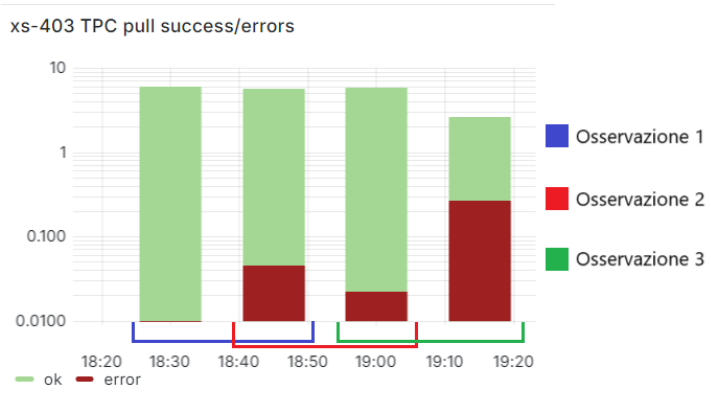

Figura 3.27: Suddivisione osservazioni normali - Finestra temporale grandezza 2

La prima alternativa, come mostrato in Figura 3.27, pone la grandezza della finestra temporale uguale a 2. Questo significa che ogni osservazione è composta da 2 istanti temporali consecutivi. Nella figura vengono mostrate le osservazioni individuate in questo caso, ciascuna rappresentata con un colore diverso. Allo stesso modo, una osservazione anomala sarà composta dai due istanti temporali precedenti all'anomalia, ovvero al "buco" di informazione individuato tramite la metrica 'TPC Pull success/error', come mostrato in Figura 3.28. Si nota che con la Retention Policy scelta precedentemente, ovvero un dato ogni 15 minuti, osservare i due istanti temporali precedenti all'anomalia equivale a osservare i dati dei 30 minuti precedenti all'istante anomalo.

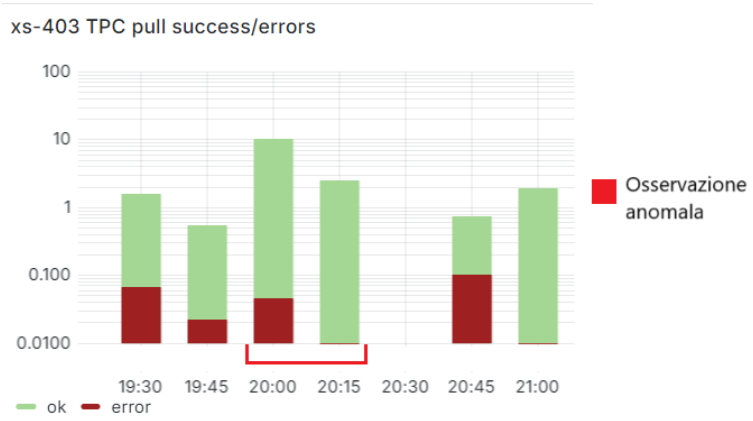

Figura 3.28: Suddivisione osservazione anomala - Finestra temporale grandezza 2

La seconda alternativa riguardo la dimensione della finestra temporale da considerare, stabilisce una grandezza pari a 3. In questo caso, come mostrato in Figura 3.29, ogni osservazione sarà composta da 3 istanti temporali successivi. Nella figura, come nel caso precedente, è possibile visualizzare le osservazioni individuate, ciascuna rappresentata con un colore differente. Allo stesso modo, una osservazione anomala sarà composta dai tre istanti temporali precedenti all'anomalia, ovvero al "buco" di informazione. In Figura 3.30 è mostrato un esempio di osservazione anomala. Si nota che osservare i tre istanti temporali precedenti all'anomalia equivale a osservare i dati dei 45 minuti precedenti all'istante anomalo. Con una finestra più ampia rispetto a quella precedente, il risultato atteso è quello di rilevare anche le anomalie che hanno uno sviluppo più lento.

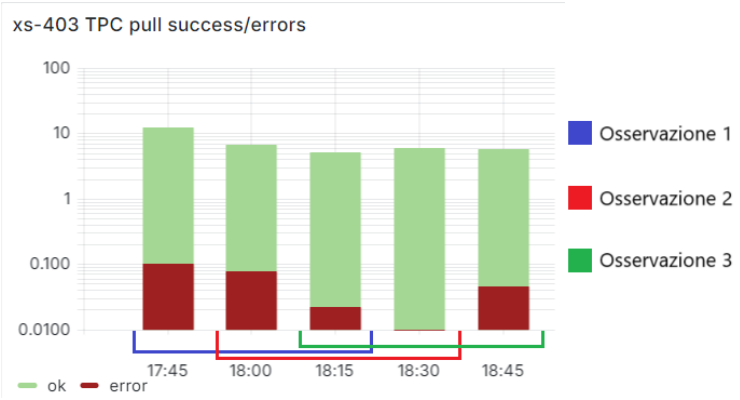

Figura 3.29: Suddivisione osservazioni normali - Finestra temporale grandezza 3

Figura 3.30: Suddivisione osservazione anomala - Finestra temporale grandezza 3

Infine, l'ultimo valore considerato relativo alla dimensione della finestra temporale di ogni osservazione è pari a 5. Nella Figura 3.31 è mostrato un esempio di suddivisione delle osservazioni, ciascuna rappresentata da un colore differente e composta da 5 istanti temporali consecutivi. Si ha quindi un ulteriore aumento della grandezza della finestra temporale rispetto ai casi precedenti, con l'obiettivo di individuare un numero maggiore di anomalie, incluse le anomalie di tipo drift, ovvero quelle che emergono gradualmente in un periodo prolungato.

Figura 3.31: Suddivisione osservazioni normali - Finestra temporale grandezza 5

Come nei casi precedenti, viene mostrata in Figura 3.32 un esempio di osservazione anomala, composta dai cinque istanti temporali precedenti all'anomalia. In questo caso osservare i cinque istanti temporali precedenti all'anomalia equivale a osservare i dati dei 75 minuti precedenti all'istante anomalo. Poiché questo valore è sufficientemente ampio da includere tutte le tipologie di anomalie, si è deciso di non incrementarlo ulteriormente e di mantenerlo come valore massimo considerato.

Figura 3.32: Suddivisione osservazione anomala - Finestra temporale grandezza 5

Dopo aver definito i valori della dimensione della finestra temporale che compone ogni osservazione, si è proceduto alla realizzazione di uno script

Python per filtrare i dati disponibili in normali e anomali. Lo script utilizza la libreria Pandas per caricare il contenuto del file CSV in un DataFrame e definire due Timestamp che rappresentino l'intervallo di interesse, ovvero l'inizio e la fine. Si è utilizzato lo script per filtrare il DataFrame, selezionando solo le righe che rientravano nell'intervallo temporale specificato, e si è proceduto al salvataggio dei vari intervalli temporali in cartelle differenti, a seconda che l'intervallo riguardasse osservazioni normali o anomale, facilitando così la gestione e l'analisi successiva dei dati. Questo processo è stato ripetuto per tutti gli intervalli temporali disponibili, distinguendo in modo chiaro i momenti normali da quelli anomali. Lo script procede anche alla creazione di nuove metriche, ottenute mediante combinazione e aggregazione di metriche già presenti. Le nuove metriche introdotte sono quattro:

- **'iostat.avg-cpu.pct_total'**: metrica che rappresenta il totale dell'utilizzo della CPU in termini percentuali, fornendo una visione complessiva dell'utilizzo della CPU. Viene ottenuta combinando la percentuale di utilizzo della CPU da parte del sistema operativo e la percentuale di utilizzo della CPU da parte delle applicazioni utente;
- **'cpu.procs_total'**: metrica che rappresenta il numero totale di processi attivi in uno specifico momento, offrendo una panoramica sul carico di lavoro del sistema. Viene ottenuta combinando il numero di processi attualmente in esecuzione e il numero di processi che sono bloccati;
- **'interfaces.total.rxBytes'**: metrica che rappresenta il totale dei byte ricevuti attraverso tutte le interfacce di rete disponibili, fornendo un'idea del traffico di rete in ingresso in uno specifico momento. Viene ottenuta sommando i dati ricevuti dalle singole interfacce;
- **'interfaces.total.txBytes'**: metrica che rappresenta il totale dei byte trasmessi attraverso tutte le interfacce di rete disponibili, indicando il traffico di rete in uscita. Viene ottenuta sommando i dati trasmessi dalle singole interfacce.

Si è quindi passati a una fase cruciale del processo: la selezione delle metriche da utilizzare. Questa fase è stata fondamentale per garantire che i dati utilizzati fossero rilevanti e rappresentativi, migliorando l'efficacia delle analisi successive. In seguito alle approfondite analisi descritte nei paragrafi precedenti, si è deciso di utilizzare un sottoinsieme delle metriche disponibili, eliminando quelle non informative, a causa della mancanza di valori significativi, e quelle ridondanti. Dopo un'attenta valutazione, sono state selezionate 32 metriche ritenute essenziali per descrivere i fenomeni di interesse. Nella

Tabella 3.2 sono riportate le metriche scelte, accompagnate da una breve descrizione del loro significato e della funzione che svolgono.

Metrica	Descrizione
_time	Timestamp in cui la misurazione è stata effettuata.
cpu.ctxt	Numero di context switches, ovvero il numero di volte che la CPU ha cambiato contesto tra processi.
cpu.total.system	Tempo durante il quale la CPU è stata utilizzata da processi del sistema operativo.
cpu.total.user	Tempo durante il quale la CPU è stata utilizzata da processi utente.
cpu.total.nice	Tempo durante il quale la CPU è stata utilizzata per eseguire processi a priorità bassa.
cpu.total.idle	Tempo durante il quale la CPU è rimasta inattiva.
cpu.total.iowait	Tempo durante il quale la CPU è rimasta in attesa di operazioni di input/output.
iostat.avg-cpu.pct_idle	Percentuale di tempo in cui la CPU è stata inattiva.
iostat.avg-cpu.pct_iowait	Percentuale di tempo in cui la CPU ha atteso operazioni di input/output.
iostat.avg-cpu.pct_system	Percentuale di tempo in cui la CPU è stata occupata da processi di sistema.
iostat.avg-cpu.pct_user	Percentuale di tempo in cui la CPU è stata occupata da processi utente.
iostat.avg-cpu.pct_total	Percentuale complessiva di tempo di utilizzo della CPU.
load_avg.fifteen	Media del carico di lavoro del sistema negli ultimi quindici minuti.
memory.used	Quantità di memoria utilizzata dal sistema.
memory.cached	Quantità di memoria utilizzata per la cache di sistema.

Metrica	Descrizione
memory.buffers	Quantità di memoria utilizzata per i buffer di sistema.
memory.free	Quantità di memoria libera disponibile nel sistema.
memory.swapUsed	Quantità di memoria di swap utilizzata dal sistema.
memory.used WOBuffersCaches	Quantità di memoria utilizzata escludendo buffer e cache.
storm.http.handler. active-dispatches.count	Numero di dispatch attivi nel servizio WebDAV.
storm.http.thread-pool. size.value	Dimensione della pool di thread nel servizio WebDAV.
storm.http.thread-pool. utilization.value	Percentuale di utilizzo della pool di thread nel servizio WebDAV.
storm.http.handler. 4xx-responses.m1_rate	Tasso di risposte HTTP con errore 4xx nel servizio WebDAV.
storm.http.handler. 5xx-responses.m1_rate	Tasso di risposte HTTP con errore 5xx nel servizio WebDAV.
jvm.threads. blocked.count.value	Numero di thread JVM bloccati nel servizio WebDAV.
jvm.threads. count.value	Numero totale di thread JVM nel servizio WebDAV.
jvm.threads. runnable.count.value	Numero di thread JVM in esecuzione nel servizio WebDAV.
TPC.pull.error-count.count	Numero di errori nelle operazioni di TPC pull.
TPC.pull.ok-count.count	Numero di operazioni di TPC pull completate con successo.
storm.http.handler. dispatches.m1_rate	Tasso di dispatch nel servizio WebDAV.
interfaces.total. rxBytes	Byte totali ricevuti attraverso le interfacce di rete.

Metrica	Descrizione
interfaces.total.txBytes	Byte totali trasmessi attraverso le interfacce di rete.

Tabella 3.2: Metriche utilizzate per classificazione

Dopo aver fissato le metriche da utilizzare e le dimensioni della finestra temporale da considerare per ogni osservazione, si è proceduto con l'organizzazione dei dati in finestre temporali. In questa fase, il codice suddivide il DataFrame in finestre, che saranno trattate come singole osservazioni durante l'addestramento del modello. Lo script permette di specificare la dimensione della finestra temporale e genera tante finestre quanti sono i blocchi consecutivi di dati disponibili. Ogni dato viene etichettato con un indice di finestra e un indice di istante temporale, informazioni essenziali per identificare la finestra temporale di appartenenza e la posizione relativa dell'istante all'interno di essa (primo, secondo, e così via). I dati così riorganizzati e suddivisi in finestre vengono infine salvati in nuovi file CSV, i cui nomi indicano la dimensione della finestra temporale utilizzata. I file CSV risultanti vengono archiviati in cartelle separate, a seconda che contengano dati di momenti normali o anomali, per agevolare le fasi successive. Questo procedimento viene ripetuto più volte, variando il parametro della dimensione della finestra temporale, così da ottenere tutti i dati necessari a una successiva fase di addestramento dei modelli. Conclusa questa fase, i dati sono organizzati in finestre temporali in modo chiaro e strutturato, suddivisi tra normali e anomali e pronti per essere utilizzati nella successiva fase di addestramento.

3.6 Organizzazione dei Dati per il Training

Dopo aver completato le operazioni relative al pre-processing e alla scelta delle metriche interessate, sono state effettuate ulteriori operazioni preliminari per preparare i dati all'utilizzo con diversi modelli di Machine Learning. Queste operazioni hanno riguardato il corretto caricamento e la preelaborazione dei dati, oltre alla gestione dello sbilanciamento del dataset. La fase è stata gestita tramite script Python e ha avuto inizio con il caricamento dei dati precedentemente organizzati in finestre temporali e la loro gestione tramite DataFrame. In particolare, il processo è iniziato selezionando esclusivamente le colonne contenenti le metriche necessarie per l'addestramento dei modelli, escludendo quelle non rilevanti. Le colonne escluse sono:

- **'window_index'**, la colonna che indica la specifica finestra temporale considerata.
- **'temporal instant_index'**, la colonna che indica lo specifico istante temporale considerato all'interno della finestra temporale.
- **'_time'**, la colonna che indica il timestamp associato all'istante temporale

I dati vengono riorganizzati in una struttura tridimensionale, realizzata tramite un array della libreria **Numpy**. In questa struttura, la prima dimensione rappresenta il numero di finestre temporali, la seconda il numero di istanti temporali all'interno di ciascuna finestra, e la terza il numero di metriche associate a ogni istante. Dopo aver riorganizzato i dati, questi vengono suddivisi in Training-Set, Validation-Set e Test-Set utilizzando la libreria *scikit-learn*. Data la disponibilità limitata del dataset, in particolare per le osservazioni anomale, la suddivisione dei dati è gestita in modo diverso a seconda della natura delle osservazioni. Per i dati normali, il processo inizia calcolando il numero di dati da destinare al Test-Set, utilizzando il parametro *'1 - split_train_normal'*. Una volta isolati i dati per il Test-Set, il parametro *'1 - split_train_normal'* viene applicato ai dati rimanenti per determinare la dimensione del Validation-Set. I dati normali residui, dopo questa seconda suddivisione, vengono utilizzati per l'addestramento. Il valore di *'split_train_normal'* è stato fissato a 0.997, garantendo così che solo una piccola frazione dei dati normali sia destinata a Validation-Set e Test-Set, massimizzando i dati disponibili per l'addestramento. Per quanto riguarda i dati anomali, a causa della loro quantità limitata, il Test-Set viene ottenuto calcolando il 10% dei dati totali. Una volta isolati i dati per il Test-Set, lo stesso criterio viene applicato ai dati rimanenti, calcolando un ulteriore 10% per costruire il Validation-Set. I dati residui, dopo questa seconda suddivisione, vengono utilizzati per il processo di addestramento. Per garantire la riproducibilità, si utilizza un seme fisso nella funzione di split, assicurando che la suddivisione dei dati sia ogni volta uguale. Successivamente, una fase essenziale per ottenere buoni risultati dai modelli addestrati è stata la gestione del dataset sbilanciato. Si hanno infatti a disposizione solo 200 osservazioni anomale, rispetto alle circa 39000 relative a situazioni normali. Questo potrebbe impedire ai modelli di apprendere correttamente le caratteristiche delle anomalie. Il codice affronta questo problema tramite la tecnica dell'**Oversampling**, una tecnica che consiste nel replicare gli esempi della classe minoritaria (in questo caso le anomalie) per aumentarne la presenza nel dataset. I dati anomali vengono quindi replicati fino a raggiungere un

numero complessivo di esempi simile a quello della classe normale. Successivamente, vengono combinati con quelli normali e mescolati casualmente, per evitare che il modello apprenda una sequenza specifica di dati durante l'addestramento. Questa operazione viene effettuata esclusivamente sui dati di addestramento, in modo da consentire al modello di osservare esempi normali e anomali in modo bilanciato, migliorando così la sua capacità di apprendere le caratteristiche delle anomalie. Infine, vengono utilizzati i **DataLoader** della libreria **PyTorch** per gestire il caricamento dei dati durante l'addestramento del modello. Questi strumenti facilitano il caricamento e la suddivisione dei dati in batch, che possono essere processati dal modello in modo progressivo. La suddivisione del dataset in mini-batch presenta diversi vantaggi, che riguardano principalmente l'efficienza computazionale e l'aggiornamento graduale dei pesi. Infatti, addestrare un modello su un intero dataset in una singola passata può richiedere molte risorse computazionali, invece suddividere i dati in batch permette di processare un sottoinsieme di dati alla volta. Inoltre, i pesi del modello vengono aggiornati in modo progressivo dopo ogni batch, consentendo al modello di apprendere in maniera più efficace e convergere più rapidamente. Si ha quindi la creazione di tre diversi DataLoader: uno per il Train-Set, uno per il Validation-Set e uno per il Test-Set. In particolare, il DataLoader relativo al Train-Set, utilizzato per l'addestramento, viene configurato con il parametro *'shuffle=True'*, che mescola i dati a ogni epoca. Questo impedisce al modello di apprendere un ordine fisso, favorendo una migliore generalizzazione. La grandezza del batch è stata trattata come un iperparametro e si sono esplorate varie configurazioni per trovare il valore ottimale. I dati sono stati quindi pre-processati e bilanciati, assicurandone un formato adeguato per una gestione corretta. Conclusa questa fase, la preparazione dei dati è ufficialmente terminata e si può procedere con l'addestramento dei vari modelli.

3.7 Panoramica Architetture utilizzate

Nella fase di addestramento sono state utilizzate diverse architetture. In questo paragrafo verranno descritte e analizzate le architetture principali che hanno mostrato le migliori prestazioni. Il codice dell'implementazioni dei modelli è stato sviluppato utilizzando **PyTorch**, una delle librerie di deep learning più diffuse e flessibili. Essa è stata scelta per merito del suo alto grado di personalizzazione dell'addestramento e la disponibilità di numerosi strumenti per il monitoraggio dell'addestramento. Si è infatti utilizzata in combinazione con PyTorch la libreria **Tensorboard**, una libreria per il monitoraggio che permette un controllo visivo delle metriche di performance,

facilitando l'analisi comparativa tra le diverse architetture sviluppate. Di seguito vengono descritti i modelli utilizzati.

3.7.1 Single Layer Perceptron

Il primo modello utilizzato è stato un **Single Layer Perceptron** [28]. Esso è uno dei modelli più basilari di rete neurale artificiale e rappresenta un punto di partenza ideale per comprendere l'entità del problema e sviluppare successive reti neurali più complesse. È costituito da un singolo strato di neuroni, ovvero lo strato di output, che riceve input direttamente dai dati iniziali e fornisce un risultato in uscita. Non ci sono quindi strati nascosti, il che rende l'architettura relativamente semplice rispetto ad altre reti neurali. Un Single Layer Perceptron riceve in input un vettore $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, dove ogni x_i rappresenta una feature dei dati in ingresso. Ogni input viene moltiplicato per un peso w_i, e la somma ponderata di questi prodotti viene calcolata insieme a un termine di bias b. L'espressione risultante è quindi la seguente:

$$z = \sum_{i=1}^{n} w_i x_i + b$$

Il valore calcolato z viene poi passato a una funzione di attivazione che determina l'output finale. In questo caso si tratta di classificazione binaria e la funzione di attivazione utilizzata è la funzione *sigmoide*, che restituisce valori tra 0 e 1. Essa è definita come:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Il risultato della sigmoide rappresenta la probabilità che un'osservazione appartenga alla classe delle osservazioni normali (classe 0) o delle osservazioni anomale (classe 1). Se l'output della sigmoide è maggiore di 0.5 il risultato sarà 1, ovvero si classifica l'osservazione come anomala, mentre se è minore di 0.5 il risultato sarà 0, ovvero si classifica come osservazione normale. Per quanto riguarda i dati che la rete prende in input, essendo organizzati in finestre temporali di grandezza fissata, è stato necessario applicare un'operazione di **flattening** per trasformarli in un formato compatibile con la rete neurale. I dati originali sono infatti organizzati in un formato tridimensionale che rappresenta il numero di finestre, la dimensione della singola finestra e il numero di feature, ma il Single Layer Perceptron richiede un input bidimensionale. I dati sono stati quindi trasformati in una struttura bidimensionale in cui la

prima dimensione rappresenta il totale degli esempi e la seconda dimensione viene ottenuta moltiplicando la lunghezza della finestra temporale per il numero di caratteristiche. Questo processo ha l'effetto di "appiattire" i dati temporali preservando tutte le informazioni di ciascun istante temporale, ma in un formato compatibile con la rete.

Il Single Layer Perceptron, a causa della sua mancanza di strati nascosti, è quindi in grado di compiere solo previsioni lineari. Esso può classificare correttamente i dati solo se le classi sono separabili linearmente, ovvero se esiste un iperpiano che divide gli esempi appartenenti a classi diverse. Nonostante non sia possibile gestire problemi non linearmente separabili con il Single Layer Perceptron, l'utilizzo di questa architettura rimane comunque un punto di partenza molto utile che permette di stabilire una base delle prestazioni ottenibili e identificare la difficoltà del problema.

3.7.2 Shallow Network

Il successivo modello utilizzato è una **Shallow Network con un solo layer nascosto**. Si tratta di un'architettura di rete neurale più avanzata rispetto al Single-Layer Perceptron e rappresenta un passo in avanti nell'affrontare problemi più complessi. Infatti, l'aggiunta dello strato nascosto consente alla rete di apprendere relazioni non lineari tra le caratteristiche degli input, migliorando significativamente la capacità di classificazione rispetto a un Single-Layer Perceptron. L'architettura utilizzata ha tre componenti principali:

1. **Layer di input**: riceve i dati in ingresso, ottenuti tramite un'operazione di flattening per trasformarli in un formato compatibile con la rete neurale.

2. **Layer nascosto**: costituito da uno o più neuroni che elaborano i dati provenienti dal layer di input. Ogni neurone del layer nascosto calcola una somma ponderata degli input ricevuti, moltiplicando ogni input per un peso e aggiungendo un termine di bias. Successivamente viene applicata una funzione di attivazione non lineare all'output della somma ponderata calcolata, per introdurre non linearità nel modello. Questo passaggio è cruciale per permettere alla rete di apprendere pattern complessi nei dati.

3. **Layer di output**: riceve l'output dal layer nascosto e produce il risultato finale della rete. Anche in questo caso, viene calcolata una somma ponderata degli output del layer nascosto utilizzando pesi e bias specifici. La combinazione lineare risultante viene passata a una funzione

di attivazione sigmoide, che determina la probabilità di appartenenza alle varie classi. La soglia decisionale è stata fissata a 0.5. Se il valore calcolato supera questa soglia, l'osservazione viene classificata come anomala. Al contrario, se il valore è inferiore o uguale, l'osservazione viene classificata come normale.

Per quanto riguarda il layer nascosto, il numero di neuroni è stato trattato come un iperparametro, cercando il valore che garantisse i risultati migliori. La scelta di questo valore è cruciale, in quanto un numero troppo basso potrebbe limitare la capacità del modello, mentre un numero troppo alto aumenterebbe il rischio di overfitting. Per quanto riguarda la funzione di attivazione, invece, si è adottata la funzione **ReLU**, definita come:

$$\mathrm{ReLU}(z) = \max(0, z)$$

Essa restituisce il valore z se è positivo, altrimenti restituisce zero. È ampiamente utilizzata nelle reti neurali moderne perché a differenza delle funzioni di attivazione sigmoide o tangente iperbolica, che tendono a saturare i valori in ingresso per valori estremamente alti o bassi, la ReLU mantiene i gradienti elevati per gli input positivi. Le funzioni di attivazione sigmoide e tangente iperbolica, infatti, mappano i valori di input in un intervallo limitato, rispettivamente tra 0 e 1 o tra -1 e 1. Questo comportamento può causare problemi di saturazione, poiché, quando i valori degli input sono estremamente alti o bassi, il gradiente della funzione diventa molto piccolo, rallentando l'aggiornamento dei pesi durante la fase di backpropagation. Al contrario, la funzione ReLU non satura per valori positivi, contribuendo a mitigare il problema descritto e a velocizzare il processo di addestramento. L'aggiunta di uno strato intermedio e l'utilizzo della funzione di attivazione ReLU consente quindi alla rete di apprendere relazioni non lineari nei dati, superando i limiti del Single-Layer Perceptron. La capacità di apprendere dei pattern più complessi tra i dati permette alla rete di migliorare le performance sui compiti di classificazione, aumentando la probabilità di separare correttamente le diverse classi.

3.7.3 Random Forest

Un altro modello di Machine Learning utilizzato è la **Random Forest**, introdotto ufficialmente in un articolo pubblicato da Leo Breiman nel 2001 [29]. Si tratta di un modello di apprendimento supervisionato che utilizza un gran numero di alberi decisionali indipendenti per effettuare delle previsioni o classificazioni. Ogni albero viene costruito tramite il metodo del *bootstrap*,

ossia utilizzando un sottinsieme casuale dei dati del Training-Set. Inoltre, a ogni nodo dell'albero, viene selezionato casualmente un sottoinsieme delle feature disponibili per determinare la migliore suddivisione del nodo. Questo meccanismo di selezione casuale delle feature è fondamentale per introdurre diversità tra gli alberi. In particolare, permette di evitare il problema che potrebbe insorgere in presenza di un predittore molto forte nel dataset, ovvero una feature che ha una grande influenza nel determinare l'output del modello. In assenza di una selezione casuale delle feature, la maggior parte degli alberi addestrati tenderebbe a utilizzare il predittore forte nella prima suddivisione, causando una forte somiglianza tra gli alberi e riducendo la diversità complessiva del modello. La mancanza di diversità tra gli alberi limiterebbe l'efficacia della Random Forest, poiché porterebbe a una forte correlazione tra gli alberi stessi, riducendo la capacità di generalizzazione del modello. Una volta addestrati tutti gli alberi, la Random Forest combina le loro previsioni per ottenere il risultato finale. In particolare, nei task di classificazione, il risultato è ottenuto tramite una votazione a maggioranza, in cui ogni albero "vota" per una classe e la classe con il maggior numero di voti è restituita come output finale. Questo processo di aggregazione è noto come *bagging* e contribuisce a migliorare la robustezza del modello e ridurre il rischio di overfitting. Gli iperparametri esplorati nel modello sono stati: *n_estimators*, *max_depth*, *min_samples_split*, *min_samples_leaf* e *max_features*. Questi iperparametri servono a regolare la robustezza del modello, ridurre l'overfitting, controllare la crescita dell'albero e regolare la diversità tra gli alberi. Una descrizione più accurata dei singoli iperparametri verrà fornita in seguito. La Random Forest rappresenta quindi un approccio potente e flessibile per la classificazione, grazie alla combinazione di molteplici alberi decisionali indipendenti. La selezione casuale delle feature insieme al processo di bagging rende il modello più robusto e aumenta la capacità di generalizzare su dati non visti, evitando la tendenza all'overfitting tipica dei singoli alberi decisionali. Essa si dimostra quindi un modello versatile e affidabile, capace di fornire risultati accurati anche in presenza di correlazioni non lineari tra le feature.

3.7.4 Long Short-term memory con classificatore finale

L'ultima architettura utilizzata è una **Long Short-Term Memory** con un classificatore finale. Essa è un'architettura adatta per il trattamento di dati sequenziali o temporali ed è una particolare tipologia di rete neurale ricorrente introdotta per la prima volta da Hochreiter e Schmidhuber nel 1997 [30]. Il funzionamento delle LSTM è basato su una complessa struttura

interna che permette alla rete di imparare dipendenze a lungo termine e a breve termine, tramite l'uso di celle di memoria e meccanismi di gating. Le variabili principali che le LSTM utilizzano per mantenere le informazioni nel tempo sono:

1. C_t: rappresenta la memoria a lungo termine delle unità LSTM.

2. h_t: rappresenta la memoria a breve termine delle unità LSTM.

Questi due elementi lavorano insieme per consentire alle unità LSTM di ricordare e dimenticare informazioni specifiche. Il comportamento delle LSTM è regolato da tre meccanismi di gating, che regolano il flusso delle informazioni. Il primo è il **Forget Gate Layer**, mostrato in Figura 3.33. Esso controlla quali informazioni della cella di memoria devono essere dimenticate o conservate. È implementato tramite una funzione sigmoide che prende in input lo stato nascosto dell'epoca precedente h_{t-1} e l'attuale input x_t, producendo un valore compreso tra 0 e 1 per ciascun componente della cella di memoria C_{t-1}. Più il valore prodotto si avvicina a zero, maggiore è la quantità di informazione dimenticata. Al contrario, più il valore si avvicina a uno, maggiore è la quantità di informazione conservata. L'equazione che descrive il Forget Gate Layer è la seguente:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

dove W_f è la matrice dei pesi associata al forget gate, b_f è il bias e σ è la funzione sigmoide.

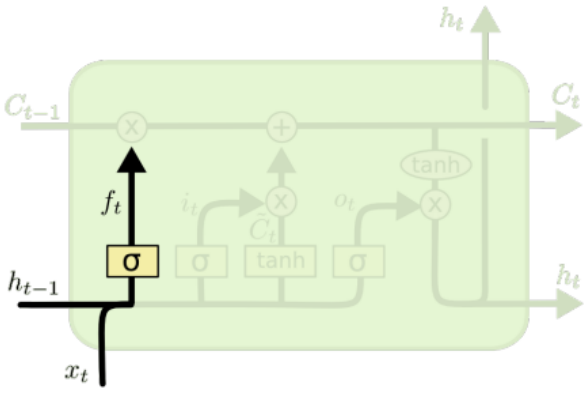

Figura 3.33: Long Short-Term Memory - Forget Gate Layer

Il secondo meccanismo di gating è l'**Input Gate Layer**, mostrato in Figura 3.34. Questo gate determina quali nuove informazioni devono essere aggiunte alla cella di memoria. Il calcolo avviene in due parti. Inizialmente, viene utilizzata una funzione sigmoide per stabilire quali valori della nuova informazione devono essere aggiornati, producendo un vettore i_t:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

Successivamente, viene calcolato un vettore di nuove informazioni candidate tramite una tangente iperbolica, che determinerà i valori effettivi che verranno aggiunti alla cella di memoria:

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

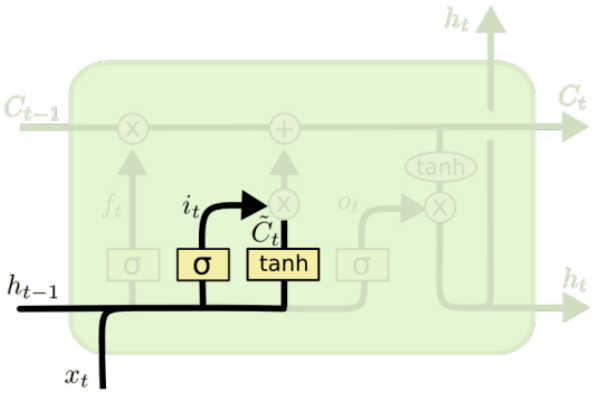

Figura 3.34: Long Short-Term Memory - Input Gate Layer

Le informazioni finali da aggiornare nella cella di memoria vengono determinate moltiplicando i valori candidati per l'output del gate di input, come mostrato in Figura 3.35, secondo la seguente formula:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

Questa combinazione permette alla rete di aggiornare la memoria con nuove informazioni, mantenendo al contempo una parte della memoria precedente.

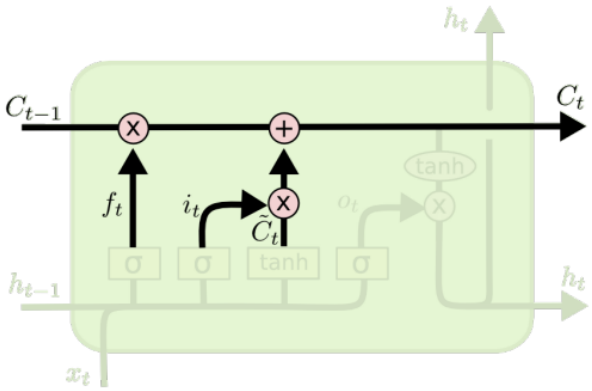

Figura 3.35: Long Short-Term Memory - Calcolo dei nuovi valori candidati

L'ultimo meccanismo di gating è l'**Output Gate Layer**, mostrato in Figura 3.36. Questo layer decide quale parte dello stato della cella C_t deve essere utilizzata per calcolare l'output al tempo corrente. È implementato tramite una funzione sigmoide che determina quali informazioni verranno considerate per il calcolo dell'output:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

Infine, l'output finale viene calcolato moltiplicando il valore del gate di output appena determinato per lo stato della cella, dopo aver applicato la funzione di attivazione tangete iperbolica:

$$h_t = o_t \cdot \tanh(C_t)$$

L'utilizzo di questi meccanismi di gating consente alla rete di "ricordare" informazioni precedenti e utilizzarle per fare previsioni sui dati successivi. Inoltre, la rete seleziona dinamicamente quali informazioni mantenere o dimenticare e come utilizzare le informazioni memorizzate per generare l'output corrente, come mostrato precedentemente.

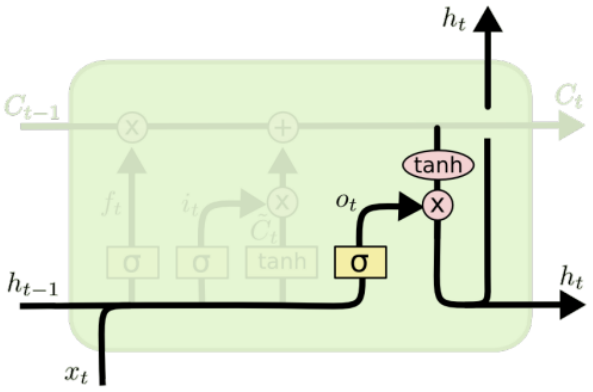

Figura 3.36: Long Short-Term Memory - Output Gate Layer

Questa architettura permette quindi di lavorare con i dati organizzati in finestre temporali, estraendo e analizzando le dipendenze tra istanti temporali successivi. Trattandosi di un task di classificazione, si è utilizzato uno strato completamente connesso finale, seguito da una funzione di attivazione sigmoide per produrre le probabilità di appartenenza alle classi. L'architettura può quindi essere suddivisa in tre parti principali:

1. **Layer di Input**: la rete riceve una sequenza di dati tridimensionali, dove la prima dimensione indica il numero di campioni nel batch, la seconda rappresenta la lunghezza della sequenza temporale e la terza il numero di feature associate a ciascun istante temporale.

2. **Layers LSTM**: la parte centrale dell'architettura è costituita da una serie di layer LSTM in sequenza, che elaborano la sequenza temporale. Il primo layer LSTM riceve in input la sequenza iniziale, mentre quelli successivi elaborano l'output del layer precedente. Come descritto in precedenza, h_t e C_t vengono aggiornati a ogni passo temporale per consentire alla rete di mantenere informazioni passate. Inoltre, tra i layer LSTM viene applicata un'operazione di dropout, con l'obiettivo di ridurre il rischio di overfitting.

3. **Classificatore finale**: la parte finale dell'architettura è costituita da un layer completamente connesso che svolge il ruolo di classificatore. L'output dell'ultimo layer LSTM viene trasmesso al layer finale per generare la predizione definitiva. Il layer completamente connesso mappa quindi l'output dei layer LSTM a un unico neurone, il cui valore viene passato attraverso una funzione di attivazione sigmoide per ottenere la probabilità di appartenenza a una delle due classi.

Nell'architettura, il numero di layer LSTM, il numero di unità LSTM per layer e il dropout sono stati trattati come iperparametri e sono stati ottimizzati per garantire le migliori prestazioni del modello. La scelta del numero di layer e di unità LSTM influenza la capacità della rete di apprendere rappresentazioni complesse e relazioni a lungo termine nei dati, mentre il dropout migliora la generalizzazione del modello e riduce il rischio di overfitting. L'architettura descritta si rivela una scelta ottimale per il trattamento dei dati disponibili, poiché sfrutta le capacità delle LSTM di apprendere relazioni temporali e mantenere informazioni rilevanti lungo le sequenze. Questo tipo di architettura risulta quindi essere la più adatta a compiti in cui le dinamiche temporali rivestono un ruolo fondamentale, come in questo caso.

3.8 Panoramica delle Funzioni di Loss utilizzate

Durante la fase di addestramento sono state utilizzate diverse funzioni di Loss, con l'obiettivo di migliorare i risultati ottenuti. Alcune di queste funzioni sono fornite nativamente da PyTorch, altre sono state modificate per adattarsi meglio al task specifico, mentre alcune sono state implementate manualmente poiché non disponibili tra quelle predefinite. Di seguito vengono descritte e analizzate le principali funzioni di Loss utilizzate durante la fase di addestramento.

3.8.1 Binary Cross-Entropy

La prima funzione di Loss utilizzata è la **Bynary Cross-Entropy** (BCE), una delle funzioni di Loss più comuni nei problemi di classificazione binaria. Essa misura la dissimilarità tra le probabilità previste dal modello e i valori target reali, assegnando una penalità maggiore agli errori più significativi. Si basa sul concetto di entropia e viene calcolata su ogni campione indipendentemente. La BCE è espressa come segue:

$$\text{BCE}(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

dove:

- y_i è il valore target per il campone i

- $\hat{y}_i$ è la probabilità predetta dal modello che il campione appartenga alla classe 1

- N è il numero totale di campioni nel batch

La Binary Cross-Entropy penalizza fortemente le previsioni errate, in particolare quando la previsione si discosta significativamente dal valore target reale. Infatti, grazie alla natura logaritmica della funzione, essa penalizza in modo non lineare gli errori, amplificando la penalità per errori sostanziali e riducendola per errori più piccoli. Ad esempio, se un modello assegna una probabilità molto alta a una classe errata, la funzione restituirà un valore elevato, segnalando un errore significativo che il modello deve correggere. Viceversa, se il modello predice correttamente con un'alta probabilità, la perdita sarà molto bassa, indicando che il modello sta imparando correttamente. Uno degli svantaggi della BCE emerge però nei casi in cui è presente uno sbilanciamento delle classi nel dataset, in quanto il modello tende a dare più peso alla classe predominante, trascurando le istanze della classe meno frequente. Per affrontare questo problema sono stati utilizzati due approcci differenti. Il primo riguarda l'utilizzo di una variante comune della Binary-Cross Entropy, ovvero la **Weighted Binary Cross-Entropy**, che assegna pesi diversi alle classi in funzione della loro frequenza nel dataset. Essa è definita nel seguente modo:

$$\text{Weighted BCE}(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\left(w_1 y_i \log(\hat{y}_i) + w_0(1-y_i)\log(1-\hat{y}_i)\right)$$

dove w_1 e w_0 sono i pesi assegnati alle classi 1 e 0, rispettivamente, per bilanciare il contributo delle classi più rare rispetto a quelle maggioritarie. Il secondo approccio, che è quello effettivamente utilizzato in fase di addestramento, si basa sulla tecnica dell'**Oversampling** descritta precedentemente. Questa tecnica consiste consiste nel replicare gli esempi della classe minoritaria (ovvero le anomalie) per aumentare la loro presenza nel dataset. In questo modo, la Binary Cross-Entropy Loss può funzionare in maniera più efficace, poiché le classi hanno un numero equivalente di campioni.

La Binary Cross-Entropy rappresenta quindi una scelta ottimale per i problemi di classificazione binaria, grazie alla sua capacità di guidare il modello verso una separazione efficace tra le classi. Attraverso un'adeguata gestione degli errori, questa funzione consente un apprendimento progressivo e mirato, favorendo il miglioramento delle performance del modello nel distinguere correttamente le osservazioni appartenenti a classi diverse.

3.8.2 Binary Cross-Entropy in combinazione con F1-Score

Un'altra funzione di loss utilizzata durante l'addestramento del modello è una combinazione tra la **Binary Cross-Entropy** e l'**F1-Score**. Questa combinazione è stata pensata per concentrare l'ottimizzazione non solo sulla minimizzazione della Loss, ma anche sulla massimizzazione della F1-Score, una metrica molto importante nei contesti di Anomaly Detection, definita come segue:

$$F1\text{-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

dove la Precision è definita nel seguente modo:

$$\text{Precision} = \frac{TP}{TP + FP}$$

e la Recall nel seguente modo:

$$\text{Recall} = \frac{TP}{TP + FN}$$

L'F1-Score rappresenta la metrica più adeguata per dare priorità alla corretta identificazione delle anomalie. Tuttavia, è importante sottolineare che la funzione di Loss effettivamente utilizzata per aggiornare i pesi del modello durante il processo di ottimizzazione è la Binary Cross-Entropy. La combinazione con l'F1-Score, invece, viene impiegata come criterio per confrontare e selezionare la configurazione ottimale del modello durante la scelta dell'epoca migliore, valutata sulla base delle performance ottenute sul Validation-Set. Questa combinazione consente di focalizzarsi su configurazioni che non solo minimizzano la Loss della Binary Cross-Entropy, ma ottengono anche un valore più elevato di F1-Score. In questo modo, vengono privilegiate le configurazioni in cui il modello è più efficace nell'individuare le anomalie. La combinazione tra Binary Cross-Entropy e F1-Score viene definita dalla seguente formula:

$$\text{Composite Score} = \alpha \cdot \text{F1-Score} - \beta \cdot \text{BCE}$$

dove α è il parametro che bilancia l'importanza dell'F1-Score e β è il parametro che bilancia l'importanza della Binary Cross-Entropy. Entrambi i parametri sono stati trattati come iperparametri, al fine di individuare la configurazione migliore che massimizzi le prestazioni del modello.

Questa metrica composta rappresenta un'ottima soluzione nei contesti di Anomaly Detection, poiché permette di bilanciare la minimizzazione degli errori con la massimizzazione della capacità del modello di identificare correttamente le anomalie, garantendo così una maggiore attenzione alle osservazioni più critiche.

3.8.3 Soft-F1 Loss

Una ulteriore funzione di Loss utilizzata durante l'addestramento è la **Soft-F1 Loss**, una funzione introdotta da Joan Pastor-Pellicer et al. nel 2013 nel lavoro *F-Measure as the Error Function to Train Neural Networks* [31]. Come descritto nell'articolo, questa funzione di Loss è stata pensata specificatamente per i dataset sbilanciati, in cui molto spesso le classiche funzioni di Loss non ottengono dei buoni risultati. Questo la rende particolarmente adatta per compiti come l'anomaly detection, in cui l'F1-Score risulta essere una metrica più rilevante rispetto all'accuracy. Tuttavia, essendo non differenziabile, l'F1-Score non può essere utilizzata direttamente per addestrare modelli tramite backpropagation. La Soft-F1 Loss risolve questo problema rendendo l'F1-Score continuo e differenziabile, trattando le predizioni come probabilità, così da poter essere utilizzata direttamente come funzione di Loss all'interno del processo di addestramento. L'idea alla base della Soft-F1 Loss è quella di sostituire i True Positive (TP), False Positive (FP) e False Negative (FN) con versioni probabilistiche definite di seguito:

$$TP = \sum_{i=1}^{m} o^{(i)} \cdot t^{(i)}$$

$$FP = \sum_{i=1}^{m} o^{(i)} \cdot (1 - t^{(i)})$$

$$FN = \sum_{i=1}^{m} (1 - o^{(i)}) \cdot t^{(i)}$$

dove:

- $o^{(i)}$ è l'output del modello, ovvero la probabilità predetta.
- $t^{(i)}$ è il valore targer reale, quindi 0 o 1.

Il calcolo della Soft-F1 segue quindi la formula standard dell'F1-Score, ma con queste versioni probabilistiche:

$$\text{Soft-F1} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

Questa funzione consente quindi di ottimizzare direttamente l'F1-Score durante il processo di addestramento, massimizzando così le capacità del modello di bilanciare precision e recall. Una ulteriore estensione della Soft-F1 è rappresentata dalla **Soft-F$_\beta$**, una versione parametrizzata dell'F1-Score che consente di dare un peso maggiore alla precision o al recall. Questo è particolarmente utile nei casi in cui è preferibile massimizzare una delle due metriche rispetto all'altra. In dei task di anomaly detection, ad esempio, potrebbe essere preferibile ottimizzare maggiormente il *recall* per assicurarsi che la maggior parte delle anomalie venga correttamente identificata, anche a costo di un piccolo aumento dei falsi positivi. La formula che definisce la Soft-F$_\beta$ è la seguente:

$$\text{Soft-F}_\beta = \frac{(1 + \beta^2) \cdot \text{TP}}{(1 + \beta^2) \cdot \text{TP} + \beta^2 \cdot \text{FN} + \text{FP} + \epsilon}$$

dove β è il parametro che regola il comportamento del modello a seconda delle esigenze specifiche del task, ovvero predilige la precision o il recall. Si hanno quindi tre possibili casi:

- $\beta = 1$: la funzione si riduce alla classica Soft-F1 Loss
- $\beta > 1$: la funzione pone maggiore enfasi sul recall
- $\beta < 1$: la funzione pone maggiore enfasi sulla precision

Queste versioni differenziabili dell'F1-Score e dell'Fβ-Score sono particolarmente utili in contesti come l'anomaly detection, dove l'obiettivo principale è l'identificazione delle anomalie. Esse consentono infatti di concentrare l'ottimizzazione del modello sulla capacità di rilevare eventi rari come le anomalie. L'integrazione di queste funzioni direttamente nei processi di backpropagation permette quindi, in questi particolari contesti, di migliorare le prestazioni del modello rispetto all'utilizzo di funzioni di Loss classiche.

3.9 Modelli migliori e iperparametri

In questo paragrafo sono descritte le configurazioni migliori dei modelli utilizzati, in termini di architetture e iperparametri. Ogni modello è stato sot-

toposto a un processo di ottimizzazione degli iperparametri tramite l'utilizzo di random search e grid search, con l'obiettivo di massimizzare le prestazioni complessive e migliorare le capacità di generalizzazione del modello, riducendo al minimo il rischio di overfitting. Per ogni modello verranno descritti gli iperparametri esplorati e verranno mostrate le configurazioni migliori trovate.

3.9.1 Single-Layer Perceptron

Nel caso di un Single-Layer Perceptron, gli iperparametri da esplorare sono meno numerosi rispetto a quelli dei modelli più complessi. Si è comunque svolto un processo di esplorazione per determinare i seguenti iperparametri:

- **Metodo di Bilanciamento**: determina come gestire lo squilibrio tra le classi nel dataset durante l'addestramento del modello.
- **Funzione di Loss**: misura quanto le predizioni del modello si discostano dai valori reali durante l'addestramento, risultando essenziale per garantire un apprendimento corretto del modello
- **Ottimizzatore**: meccanismo che aggiorna i pesi del modello in base al gradiente della funzione di loss. È fondamentale, poiché determina come i pesi vengono modificati durante l'addestramento
- **Batch size**: definisce il numero di campioni utilizzati per calcolare il gradiente a ogni iterazione di aggiornamento dei pesi. Influisce direttamente sulla velocità di convergenza e sull'efficienza del processo di addestramento
- **Normalizzazione**: tecnica usata per trasformare i dati in modo che siano scalati entro un intervallo specifico. Aiuta a garantire che tutte le feature abbiano un contributo simile e che il modello non sia influenzato da scale differenti tra le variabili
- **Learning Rate**: tasso di apprendimento, un parametro cruciale, poiché un valore non ottimale può compromettere l'efficacia del processo di apprendimento.
- **Finestra temporale**: definisce la dimensione della finestra temporale considerata, ossia il numero di istanti temporali che costituiscono un'osservazione.

L'esplorazione di questi iperparametri è avvenuta tramite una grid search, valutando quindi il modello su tutte le combinazioni possibili di iperparametri per individuare la configurazione ottimale che garantisce le migliori prestazioni del modello. I valori ottimali degli iperparametri utilizzati per l'addestramento del modello sono stati:

- **Metodo di Bilanciamento** = Oversampling tramite replicazione delle osservazioni anomale
- **Funzione di Loss** = Binary Cross-Entropy in combinazione con F1-Score, con $\alpha = 0.7$ e $\beta = 0.3$
- **Ottimizzatore** = Adam con valori di default per β_1 e β_2
- **Batch size** = 256
- **Normalizzazione** = Standard Scaling
- **Learning Rate** = 0.0001
- **Finestra temporale** = 5

Il processo descritto ha permesso di individuare la combinazione ottimale di iperparametri, consentendo al modello di migliorare i risultati ottenuti nonostante la semplicità dell'architettura. I risultati verranno approfonditi nel prossimo capitolo.

3.9.2 Shallow Network

Per quanto riguarda l'architettura della Shallow Network, anche in questo caso gli iperparametri su cui soffermarsi non sono numerosi, sebbene siano più numerosi rispetto al caso precedente. È stato comunque condotto un processo di esplorazione per determinare i seguenti iperparametri:

- **Metodo di Bilanciamento**: determina come gestire lo squilibrio tra le classi nel dataset durante l'addestramento del modello.
- **Funzione di Loss**: misura quanto le predizioni del modello si discostano dai valori reali durante l'addestramento, risultando essenziale per garantire un apprendimento corretto del modello
- **Ottimizzatore**: meccanismo che aggiorna i pesi del modello in base al gradiente della funzione di loss. È fondamentale, poiché determina come i pesi vengono modificati durante l'addestramento

- **Batch size**: definisce il numero di campioni utilizzati per calcolare il gradiente a ogni iterazione di aggiornamento dei pesi. Influisce direttamente sulla velocità di convergenza e sull'efficienza del processo di addestramento

- **Normalizzazione**: tecnica usata per trasformare i dati in modo che siano scalati entro un intervallo specifico. Aiuta a garantire che tutte le feature abbiano un contributo simile e che il modello non sia influenzato da scale differenti tra le variabili

- **Learning Rate**: tasso di apprendimento, un parametro cruciale, poiché un valore non ottimale può compromettere l'efficacia del processo di apprendimento.

- **Numero di neuroni dello strato intermedio**: determina la capacità espressiva della rete. Un numero maggiore di neuroni consente al modello di apprendere pattern più complessi dai dati, ma aumenta anche il rischio di overfitting.

- **Funzione di attivazione**: introduce non linearità nel modello, permettendo alla rete di apprendere pattern complessi. La scelta della funzione di attivazione influisce significativamente sulle prestazioni e la velocità di convergenza del modello

- **Finestra temporale**: definisce la dimensione della finestra temporale considerata, ossia il numero di istanti temporali che costituiscono un'osservazione.

Anche in questo caso l'esplorazione degli iperparametri è avvenuta tramite una grid search, valutando quindi il modello con tutte le combinazioni di iperparametri possibili e individuando la configurazione ottimale che garantisce le migliori prestazioni. I valori ottimali degli iperparametri utilizzati per l'addestramento del modello sono stati:

- **Metodo di Bilanciamento** = Oversampling tramite replicazione delle osservazioni anomale

- **Funzione di Loss** = Binary Cross-Entropy in combinazione con F1-Score, con $\alpha = 0.7$ e $\beta = 0.3$

- **Ottimizzatore** = Adam con valori di default per β_1 e β_2

- **Batch size** = 256

- **Normalizzazione** = Standard Scaling
- **Learning Rate** = 0.0001
- **Numero di neuroni dello strato intermedio** = 32
- **Funzione di attivazione** = ReLU
- **Finestra temporale** = 5

Tramite l'esplorazione degli iperparametri si è quindi riusciti a trovare la combinazione migliore per permettere al modello di ottenere dei risultati robusti. I risultati ottenuti verranno approfonditi nel prossimo capitolo.

3.9.3 Random Forest

Per quanto riguarda il modello Random Forest, anche in questo caso si è proceduto a un'esplorazione accurata degli iperparametri disponibili. In particolare, gli iperparametri esplorati sono:

- **Metodo di Bilanciamento**: determina come gestire lo squilibrio tra le classi nel dataset durante l'addestramento del modello.
- **n_estimators**: rappresenta il numero di alberi decisionali nella foresta. Un numero più alto di alberi migliora la robustezza del modello, ma può anche aumentare i tempi di addestramento e l'uso di risorse quali la memoria.
- **max_depth**: specifica la profondità massima degli alberi nella foresta. Limitare la profondità degli alberi previene l'overfitting e migliora la capacità di generalizzazione su nuovi dati.
- **min_samples_split**: definisce il numero minimo di campioni richiesti per suddividere un nodo. Aumentare questo valore aiuta a prevenire la creazione di nodi troppo specifici, riducendo di fatto l'overfitting.
- **min_samples_leaf**: definisce il numero minimo di campioni che un nodo foglia deve avere. Un valore più alto riduce il rischio che il modello apprenda rumore dai dati, limitando così l'overfitting.
- **max_features**: determina il numero massimo di feature da considerare per ogni suddivisione del nodo. La scelta di un buon valore per questo iperparametro migliora la robustezza del modello e favorisce la diversità tra gli alberi.

- **Finestra temporale**: definisce la dimensione della finestra temporale considerata, ossia il numero di istanti temporali che costituiscono un'osservazione.

L'esplorazione di questi iperparametri è avvenuta tramite una random search iniziale per avvicinarsi alle configurazioni che fornivano risultati migliori, seguita da una grid search per poter identificare la configurazione migliore tra tutte. I valori migliori per gli iperparametri utilizzati per l'addestramento del modello sono stati:

- **Metodo di Bilanciamento** = Oversampling tramite replicazione delle osservazioni anomale
- **n_estimators** = 100
- **max_depth** = 10
- **min_samples_split** = 10
- **min_samples_leaf** = 4
- **max_features** = 'log2', ovvero a ogni nodo vengono considerate $\log_2(n)$ feature (dove n è il numero totale di feature)
- **Finestra temporale** = 5

Tramite l'ottimizzazione degli iperparametri si è riusciti a ottenere dei risultati robusti e una buona capacità di generalizzazione da parte del modello. I risultati ottenuti verranno approfonditi nel prossimo capitolo.

3.9.4 Long Short-term memory con classificatore finale

Per quanto riguarda il modello Long Short-term memory con classificatore finale gli iperparametri da esplorare erano diversi. Si è quindi proceduto come nei casi precedenti a una attenta esplorazione per identificare la configurazione migliore. In particolare, gli iperparametri esplorati sono:

- **Metodo di Bilanciamento**: determina come gestire lo squilibrio tra le classi nel dataset durante l'addestramento del modello.
- **Funzione di Loss**: misura quanto le predizioni del modello si discostano dai valori reali durante l'addestramento, risultando essenziale per garantire un apprendimento corretto del modello

- **Ottimizzatore**: meccanismo che aggiorna i pesi del modello in base al gradiente della funzione di loss. È fondamentale, poiché determina come i pesi vengono modificati durante l'addestramento

- **Batch size**: definisce il numero di campioni utilizzati per calcolare il gradiente a ogni iterazione di aggiornamento dei pesi. Influisce direttamente sulla velocità di convergenza e sull'efficienza del processo di addestramento

- **Normalizzazione**: tecnica usata per trasformare i dati in modo che siano scalati entro un intervallo specifico. Aiuta a garantire che tutte le feature abbiano un contributo simile e che il modello non sia influenzato da scale differenti tra le variabili

- **Learning Rate**: tasso di apprendimento, un parametro cruciale, poiché un valore non ottimale può compromettere l'efficacia del processo di apprendimento.

- **Numero di layer LSTM**: rappresenta il numero di livelli impilati di unità LSTM nel modello. Un numero maggiore di layer consente al modello di apprendere relazioni più complesse

- **Numero di unità LSTM per layer**: definisce il numero di unità LSTM (neuroni) per ciascun layer. Un numero maggiore di unità consente alla rete di apprendere rappresentazioni più dettagliate delle sequenze, ma aumenta anche il rischio di overfitting

- **Funzione di attivazione**: introduce non linearità nel modello, permettendo alla rete di apprendere pattern complessi. La scelta della funzione di attivazione influisce significativamente sulle prestazioni e la velocità di convergenza del modello

- **Tasso di dropout**: rappresenta la probabilità con cui le unità LSTM vengono temporaneamente disattivate durante l'addestramento. Migliora la capacità di generalizzazione del modello e combatte l'overfitting, poiché impedisce alla rete di dipendere eccessivamente da singoli unità neurali, consentendo un apprendimento più robusto e distribuito.

- **Finestra temporale**: definisce la dimensione della finestra temporale considerata, ossia il numero di istanti temporali che costituiscono un'osservazione.

Anche l'esplorazione di questi iperparametri, come nel caso precedente, è avvenuta tramite una random search iniziale per avvicinarsi alle configurazioni che fornivano risultati migliori, seguita da una grid search per poter identificare la configurazione migliore tra tutte. I valori migliori per gli iperparametri utilizzati per l'addestramento del modello sono stati:

- **Metodo di Bilanciamento** = Oversampling tramite replicazione delle osservazioni anomale
- **Funzione di Loss** = Binary Cross-Entropy in combinazione con F1-Score, con $\alpha = 0.7$ e $\beta = 0.3$
- **Ottimizzatore** = Adam con valori di default per β_1 e β_2
- **Batch size** = 256
- **Normalizzazione** = Standard Scaling
- **Learning Rate** = 0.0001
- **Numero di layer LSTM** = 2
- **Numero di unità LSTM per layer** = 128
- **Funzione di attivazione** = ReLU
- **Tasso di dropout** = 0.5
- **Finestra temporale** = 5

L'ottimizzazione degli iperparametri descritti ha permesso di ottenere dei risultati robusti, una buona capacità di generalizzazione da parte del modello e delle performance positive sui dati non visti. I risultati ottenuti verranno approfonditi nel prossimo capitolo.

Capitolo 4

Risultati ottenuti e analisi

Nel seguente capitolo verranno descritti e analizzati i risultati ottenuti con i diversi modelli impiegati nello studio. Il capitolo sarà suddiviso in più sezioni: si inizierà con una descrizione delle metriche utilizzate per valutare le performance dei modelli e della loro importanza in un contesto di Anomaly Detection, per poi presentare i risultati del Single-Layer Perceptron, seguiti da quelli della Shallow Network, della Random Forest e, infine, della Long Short-Term Memory con classificatore finale. A conclusione del capitolo, sarà presentata un'analisi di alcuni casi d'esempio attraverso l'utilizzo della tecnica della **Saliency Map**, la quale consentirà di mettere in risalto le caratteristiche più significative per l'individuazione delle anomalie.

4.1 Metriche utilizzate

Le performance di ciascun modello verranno valutate in termini di Accuracy, Precision, Recall e F1-Score. In particolare, l'**Accuracy** misura la percentuale di previsioni corrette rispetto al totale dei campioni. La formula che la rappresenta è:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

dove TP indica i True Positive, TN i True Negative, FP i False Positive e FN i False Negative. Tuttavia, nei contesti di dataset sbilanciati e Anomaly Detection, l'accuratezza può risultare poco informativa. Infatti, un modello che predice sempre la classe dominante (in questo caso le osservazioni normali) potrebbe ottenere un'alta accuracy, senza però essere in grado di rilevare efficacemente le anomalie. In questi casi quindi ci si concentra su metriche

quali Precision, Recall e F1-score per valutare le capacità del modello di individuare correttamente le anomalie. In particolare, la **Precision** è una metrica che misura la proporzione di campioni classificati come positivi (ovvero come anomali) che sono effettivamente positivi. Rappresenta, quindi, quanto possiamo fidarci delle previsioni positive del modello, ossia quanto un'anomalia predetta sia realmente un'anomalia. La formula della Precision è la seguente:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Una Precision elevata indica che il modello commette pochi errori nel classificare un campione come anomalo. Un'altra metrica importante nell'ambito dell'Anomaly Detection è la **Recall**. Essa misura la capacità del modello di identificare correttamente tutte le osservazioni positive (ovvero le anomalie). Una Recall elevata significa che il modello è in grado di rilevare la maggior parte delle anomalie presenti. Nei contesti di Anomaly Detection, massimizzare la Recall è spesso prioritario, poiché l'obiettivo principale è non lasciare inosservate le anomalie, garantendo che la maggior parte delle irregolarità vengano rilevate in modo accurato e tempestivo. La formula della Recall è la seguente:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Tuttavia una Recall elevata può talvolta significare un numero maggiore di falsi positivi. Quindi è necessario trovare un equilibrio tra Recall e Precision per ottenere un modello efficiente. Si utilizza quindi la metrica dell'**F1-Score**, che rappresenta la media armonica tra Precision e Recall, fornendo una misura unica che bilancia entrambe le metriche. Essa è preferita rispetto alla semplice media aritmetica perché penalizza fortemente gli estremi, richiedendo che sia la Precision che la Recall siano alti per ottenere un buon punteggio. La formula dell'F1-Score è:

$$F1\text{-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

In contesti dove l'Accuracy non è indicativa delle prestazioni del modello come in questo caso, l'F1-Score è quindi una metrica particolarmente rilevante. Un F1-Score elevato indica che il modello riesce a bilanciare correttamente Precision e Recall, risultando efficace sia nel rilevare le anomalie sia nel minimizzare i falsi positivi. Nei paragrafi successivi verranno mostrati i risultati di ciascun modello utilizzando le metriche appena descritte.

4.2 Risultati Single-Layer Perceptron

In questo paragrafo verranno mostrati i risultati dell'architettura Single-Layer Perceptron. Essa è stata inserita poiché, nonostante la sua semplicità, fornisce una buona base di confronto per valutare l'efficacia di modelli più complessi. Grazie alla sua struttura lineare e alla ridotta complessità computazionale, il Single-Layer Perceptron permette di ottenere risultati iniziali che sono utili per comprendere meglio le caratteristiche del problema e per stabilire un benchmark di riferimento.

Accuracy	**Precision**	**Recall**	**F1-Score**
0.8088	0.3888	0.7777	0.5185

Tabella 4.1: Risultati Validation-Set Single-Layer Perceptron

Accuracy	**Precision**	**Recall**	**F1-Score**
0.7681	0.3636	0.8	0.5

Tabella 4.2: Risultati Test-Set Single-Layer Perceptron

I risultati sul Validation-Set sono mostrati nella Tabella 4.1, mentre i risultati sul Test-Set sono mostrati nella Tabella 4.2. Analizzando le tabelle si può notare che l'architettura del Single-Layer Perceptron ha raggiunto una **Accuracy** di 0.8088 sul Validation-Set e di 0.7681 sul Test-Set. Questi valori indicano una performance di base relativamente stabile, ma suggeriscono una certa difficoltà nella generalizzazione dei dati di test. Anche osservando le altre metriche emerge un comportamento instabile: la **Precision** sul Validation-Set è di 0.3888, mentre nel Test-Set scende leggermente a 0.3636, suggerendo che il modello ha difficoltà a distinguere accuratamente le classi positive. Invece il valore di **Recall** rimane relativamente alto (0.7777 nel Validation-Set e 0.8 nel Test-Set), indicando che il modello è in grado di individuare una buona parte degli esempi positivi, sebbene lo faccia a discapito della precisione. Il valore dell'**F1-Score**, che rappresenta una media armonica tra Precision e Recall, raggiunge valori modesti (0.5185 sul Validation-Set e 0.5 sul Test-Set), confermando che, sebbene il modello sia abbastanza capace nel trovare esempi positivi, non è altrettanto efficace nel minimizzare i falsi positivi. Questi risultati riflettono i limiti dell'architettura del Single-Layer

Perceptron, che, data la sua struttura lineare, non è in grado di catturare le relazioni più complesse tra le variabili. Ciò è evidente dall'incapacità del modello di raggiungere valori elevati sia di Precision che di Recall contemporaneamente, il che indica un limite nella capacità di discriminare efficacemente le classi in situazioni più complesse. In definitiva, il Single-Layer Perceptron è utile come base per confrontare i modelli più sofisticati, ma i suoi risultati rivelano come la mancanza di capacità di apprendimento non lineare sia un fattore limitante. L'utilizzo di architetture più complesse, in grado di modellare relazioni non lineari, migliorerà significativamente la precisione e la capacità di generalizzazione.

4.3 Risultati Shallow Network

Si mostrano adesso i risultati della Shallow Network, un'architettura leggermente più complessa rispetto al Single-Layer Perceptron.Grazie alla presenza di un layer nascosto aggiuntivo, ci si aspetta che la Shallow Network sia in grado di catturare relazioni non lineari e pattern più complessi nei dati, migliorando le performance complessive rispetto al modello precedente. La presenza di un layer nascosto infatti fornisce al modello una capacità di apprendimento superiore rispetto al Single-Layer Perceptron, consentendo una maggiore capacità di generalizzazione e una riduzione del numero di falsi positivi e falsi negativi.

Accuracy	**Precision**	**Recall**	**F1-Score**
0.9264	0.7	0.7777	0.7368

Tabella 4.3: Risultati Validation-Set Shallow Network

Accuracy	**Precision**	**Recall**	**F1-Score**
0.9347	0.7619	0.8	0.7804

Tabella 4.4: Risultati Test-Set Shallow Network

I risultati sul Validation-Set sono mostrati nella Tabella 4.3, mentre i risultati sul Test-Set sono mostrati nella Tabella 4.4. Analizzando le tabelle si può notare un miglioramento significativo rispetto all'architettura precedente. L'**Accuracy** della Shallow Network raggiunge un valore di 0.9264

sul Validation-Set e 0.9347 sul Test-Set, indicando una migliore capacità del modello di generalizzare sui dati non visti. Anche la **Precision** migliora notevolmente, passando a 0.7 sul Validation-Set e a 0.7619 sul Test-Set. Questo risultato dimostra che il modello è più preciso nel distinguere correttamente le classi positive rispetto al Single-Layer Perceptron. Il **Recall** rimane elevato, con un valore di 0.7777 sul Validation-Set e 0.8 sul Test-Set, indicando che il modello mantiene una buona capacità di identificare la maggior parte degli esempi positivi. Il miglioramento complessivo è confermato dall'aumento dell'**F1-Score**, che raggiunge 0.7368 sul Validation-Set e 0.7804 sul Test-Set. Questo incremento indica una migliore armonizzazione tra Precision e Recall, evidenziando che la Shallow Network è più efficace nel bilanciare la corretta identificazione delle classi positive con la riduzione dei falsi positivi. Questi risultati mostrano che la Shallow Network migliora le prestazioni rispetto al Single-Layer Perceptron, grazie alla sua capacità di catturare pattern non lineari. Ciò evidenzia l'importanza di utilizzare modelli capaci di cogliere relazioni non lineari per affrontare il problema e garantire una maggiore capacità di generalizzazione.

4.4 Risultati Random Forest

Si mostrano adesso i risultati della Random Forest, un modello che, rispetto alle architetture precedenti come il Single-Layer Perceptron e la Shallow Network, offre un approccio alternativo per catturare relazioni non lineari e complesse nei dati. A differenza delle reti neurali, la Random Forest non si basa su una rappresentazione continua dei pesi, ma utilizza un insieme di alberi decisionali, permettendo una gestione più efficace della variabilità nei dati e riducendo il rischio di overfitting. Rispetto al Single-Layer Perceptron, la Random Forest risulta molto più efficace nel modellare relazioni intricate, mentre, rispetto alla Shallow Network, si distingue per la sua robustezza e la capacità di adattarsi a una gamma più ampia di scenari, anche con quantità limitate di dati per l'addestramento.

Accuracy	**Precision**	**Recall**	**F1-Score**
0.9558	0.9285	0.7222	0.8125

Tabella 4.5: Risultati Validation-Set Random Forest

Accuracy	Precision	Recall	F1-Score
0.9420	0.875	0.7	0.7777

Tabella 4.6: Risultati Test-Set Random Forest

I risultati sul Validation-Set sono mostrati nella Tabella 4.5, mentre i risultati sul Test-Set sono mostrati nella Tabella 4.6. Analizzando le tabelle si nota che la Random Forest raggiunge una **Accuracy** di 0.9558 sul Validation-Set e 0.9420 sul Test-Set. Questi valori indicano che il modello ha una buona capacità di generalizzare anche sui dati non visti, garantendo prestazioni stabili su entrambi i set. La **Precision** sul Validation-Set è pari a 0.9285, mentre sul Test-Set è di 0.875, indicando che il modello è molto efficace nel ridurre i falsi positivi, riuscendo così a distinguere accuratamente gli esempi positivi. La **Recall** invece raggiunge il valore di 0.7222 sul Validation-Set e di 0.7 sul Test-Set, mostrando una buona capacità di identificare correttamente gli esempi positivi, seppur mostrando un leggero calo rispetto alla Precision. Infine, l'**F1-Score**, che bilancia Precision e Recall, raggiunge il valore di 0.8125 sul Validation-Set e 0.7777 sul Test-Set. Questo conferma la buona capacità del modello di mantenere un equilibrio tra la corretta classificazione degli esempi positivi e la riduzione dei falsi positivi, risultando più efficace rispetto al Single-Layer Perceptron e alla Shallow Network. In generale, la Random Forest dimostra quindi di essere un modello robusto e in grado di catturare le relazioni complesse nei dati, mostrando performance superiori rispetto agli approcci precedenti, soprattutto nella capacità di generalizzare su dati non visti e nella gestione della variabilità presente nei dati. Questo è evidente dall'elevata Accuracy e dalla buona armonizzazione tra Precision e Recall, come evidenziato dai valori dell'F1-Score.

4.5 Risultati LSTM con Classificatore finale

Di seguito vengono presentati i risultati ottenuti utilizzando la rete Long Short-Term Memory con classificatore finale. Essa rappresenta un approccio diverso rispetto ai modelli precedentemente esaminati, in quanto è progettato specificamente per gestire dati sequenziali e catturare dipendenze temporali. Questo rende l'architettura LSTM particolarmente adatta per problemi in cui le relazioni tra i dati evolvono nel tempo, aspetto che non viene catturato efficacemente dalle architetture precedenti. Rispetto al Single-Layer Perceptron e alla Shallow Network, la LSTM presenta una capacità significa-

tivamente maggiore di modellare relazioni temporali e dinamiche complesse. A differenza della Random Forest, che è costituita da alberi decisionali indipendenti, la LSTM è in grado di memorizzare informazioni a lungo termine e di sfruttare queste informazioni per prendere decisioni più accurate, specialmente in presenza di pattern temporali. Questo rende questa architettura la più adatta per questo task, poiché riesce a sfruttare le dipendenze temporali in modo efficace, migliorando la capacità di previsione e la generalizzazione rispetto agli altri modelli considerati.

Accuracy	**Precision**	**Recall**	**F1-Score**
0.9632	0.8823	0.8333	0.8571

Tabella 4.7: Risultati Validation-Set LSTM con calssificatore finale

Accuracy	**Precision**	**Recall**	**F1-Score**
0.9492	0.8421	0.8	0.8205

Tabella 4.8: Risultati Test-Set LSTM con calssificatore finale

I risultati sul Validation-Set sono mostrati nella Tabella 4.7, mentre i risultati sul Test-Set sono mostrati nella Tabella 4.8. Analizzando le tabelle si nota che l'architettura LSTM raggiunge una **Accuracy** di 0.9632 sul Validation-Set e 0.9492 sul Test-Set. Questi valori indicano una capacità di generalizzazione molto buona su dati non visti. La **Precision** sul Validation-Set è pari a 0.8823, mentre sul Test-Set è di 0.8421. Ciò dimostra che l'architettura è efficace nel ridurre i falsi positivi e distinguere accuratamente gli esempi positivi. La **Recall**, invece, raggiunge il valore di 0.8333 sul Validation-Set e 0.8 sul Test-Set, indicando un'ottima capacità di identificare correttamente la maggior parte degli esempi positivi e dimostrandosi superiore rispetto ai modelli analizzati in precedenza. Infine, l'**F1-Score**, che bilancia Precision e Recall, raggiunge il valore di 0.8571 sul Validation-Set e 0.8205 sul Test-Set, confermando la capacità del modello di mantenere un equilibrio tra la corretta classificazione degli esempi positivi e la riduzione dei falsi positivi. I risultati ottenuti dimostrano la capacità delle LSTM nel gestire dati sequenziali e catturare dipendenze temporali. Questo rende questa architettura particolarmente adatta per problemi in cui le relazioni tra i dati si sviluppano nel tempo, aspetto che non viene catturato efficacemente

dalle architetture precedenti. L'uso dell'architettura LSTM si rivela quindi strategico per l'analisi di serie temporali complesse come in questo caso, permettendo di identificare in modo più accurato schemi e anomalie difficili da rilevare con altri approcci.

4.6 Risultati a confronto

Di seguito vengono mostrate le prestazioni dei vari modelli a confronto. Questo confronto è utile per comprendere i punti di forza e di debolezza di ciascuna architettura, evidenziando quali modelli sono più adatti per questo specifico task. L'analisi comparativa permette quindi di individuare il modello con la migliore capacità di generalizzazione, adattabilità e rilevazione delle anomalie.

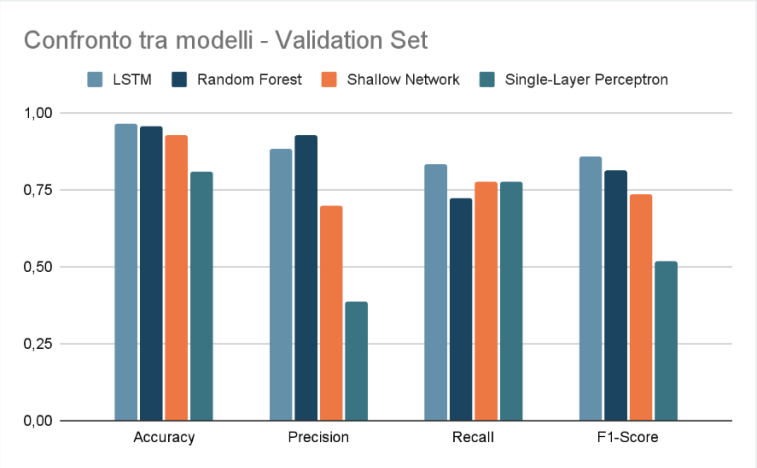

Figura 4.1: Confronto tra modelli - Validation Set

Nella Figura 4.1 è mostrato il confronto tra le prestazioni dei modelli sul Validation Set. Dai risultati emerge chiaramente che i modelli LSTM e Random Forest ottengono le migliori prestazioni complessive, con valori di Accuracy, Precision, Recall e F1-Score generalmente superiori rispetto agli altri modelli. In particolare, l'Accuracy dell'architettura LSTM e della Random Forest sono molto simili e superiori rispetto a quella degli altri modelli, indicando una buona capacità di classificare correttamente i campioni. La Shallow Network ha una Accuracy inferiore, mentre il Single-Layer Perceptron presenta le prestazioni peggiori in questa metrica, suggerendo che la sua semplicità non consente di catturare adeguatamente la complessità dei dati.

Per quanto riguarda la Precision, la Random Forest si distingue nettamente, ottenendo un valore superiore rispetto agli altri modelli. Questo indica che la Random Forest ha una maggiore capacità di ridurre i falsi positivi. La Shallow Network e il Single-Layer Perceptron hanno valori di Precision significativamente inferiori, mentre la LSTM si colloca in una posizione intermedia. Per quanto riguarda invece la Recall, la LSTM e la Shallow Network ottengono i valori migliori, evidenziando una maggiore capacità di individuare correttamente le anomalie presenti nei dati. La Random Forest segue a breve distanza, mentre il Single-Layer Perceptron risulta meno efficace anche in questa metrica, mostrando una capacità limitata di identificare tutte le anomalie. Riguardo l'F1-Score, che bilancia Precision e Recall, la LSTM e la Random Forest ottengono i valori più alti, indicando un buon equilibrio tra la capacità di ridurre i falsi positivi e quella di individuare correttamente le anomalie. In particolare, la LSTM è quella che ottiene i valori migliori. La Shallow Network mostra un valore più basso, e il Single-Layer Perceptron si conferma il modello con le prestazioni peggiori. I risultati indicano quindi che la LSTM è il modello più adatto per il task affrontato, riuscendo a bilanciare Precision e Recall nella maniera migliore possibile. La Random Forest è il secondo modello migliore, eccellendo nella Precision ma mostrando della carenza nella Recall. La Shallow Network mostra prestazioni moderate, mentre il Single-Layer Perceptron non riesce a competere con le altre architetture a causa della sua semplicità e della conseguente limitata capacità di modellare la complessità dei dati.

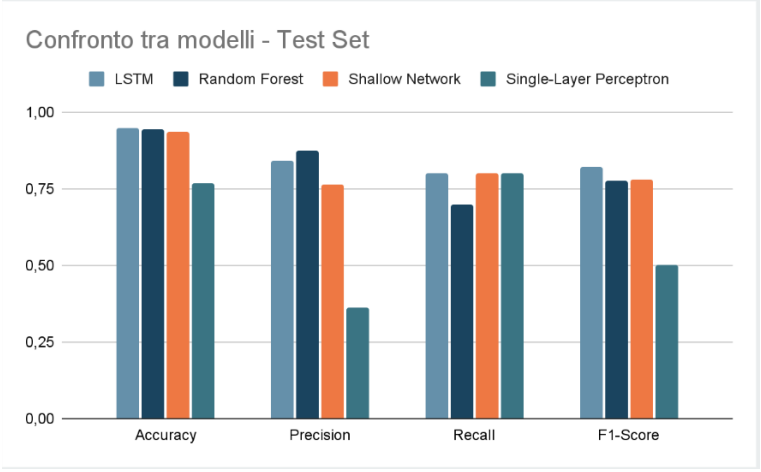

Figura 4.2: Confronto tra modelli - Test Set

Per quanto riguarda il confronto tra le prestazioni dei modelli sul Test Set,

esso è mostrato nella Figura 4.2. Dal grafico si nota che i modelli LSTM, Random Forest e Shallow Network ottengono le prestazioni complessive migliori, con valori di Accuracy, Precision, Recall e F1-Score generalmente superiori rispetto al Single-Layer Perceptron. In particolare, l'Accuracy della LSTM, della Random Forest e della Shallow Network sono molto simili, indicando una buona capacità di classificare correttamente i campioni. Il Single-Layer Perceptron invece mostra un'accuracy inferiore, suggerendo che la sua semplicità limita la capacità di catturare la complessità dei dati. Per quanto riguarda la Precision, la Random Forest si distingue ancora una volta, ottenendo il valore più alto tra i modelli. Ciò indica che la Random Forest ha una capacità maggiore di ridurre i falsi positivi, risultando sotto questo punto di vista la più precisa nella classificazione. La LSTM presenta valori di precision leggermente inferiori, ma comunque migliori rispetto alla Shallow Network, mentre il Single-Layer Perceptron risulta nettamente meno preciso. In termini di Recall, la LSTM e la Shallow Network mostrano i valori più elevati, evidenziando una buona capacità di individuare correttamente le anomalie. La Random Forest invece presenta delle carenze in questa metrica, suggerendo una capacità limitata di identificare tutte le anomalie. Il Single-Layer Perceptron risulta avere buone prestazioni sotto questo punto di vista, a discapito però della Precision come mostrato precedentemente. Per l'F1-Score invece, che rappresenta un equilibrio tra Precision e Recall, la LSTM ottiene i risultati migliori. La Random Forest e la Shallow Network ottengono risultati molto simili, indicando un buon compromesso tra la riduzione dei falsi positivi e l'individuazione corretta delle anomalie. Il Single-Layer Perceptron, invece, mostra un F1-Score inferiore, confermando le sue limitate capacità rispetto agli altri modelli. In conclusione, i risultati del Test Set confermano che la LSTM è il modello più adatto per il compito affrontato, riuscendo a bilanciare Precision e Recall in modo efficace. La Random Forest si posiziona come il secondo miglior modello, eccellendo nella Precision ma mostrando una leggera carenza nel Recall. La Shallow Network mostra prestazioni competitive, mentre il Single-Layer Perceptron non riesce a competere con le altre architetture a causa della sua semplicità e della conseguente limitata capacità di modellare la complessità dei dati.

4.7 Saliency Map per l'identificazione delle Feature Chiave

In questa sezione verranno utilizzate le Saliency Map per identificare le feature chiave che contribuiscono alla classificazione del modello. Questo ti-

po di analisi risulta essere molto utile per fornire delle indicazioni riguardo la natura dell'anomalia all'interno del sistema e comprendere la causa del problema che sta generando l'anomalia. Le Saliency Map sono uno strumento fondamentale per migliorare l'interpretabilità dei modelli di machine learning, in particolare delle reti neurali. Le reti neurali infatti sono spesso considerate delle "black box" a causa della loro complessità e del gran numero di parametri coinvolti, e questo rende difficile capire quali fattori influenzano maggiormente le decisioni del modello. Le Saliency Map aiutano a superare questo ostacolo fornendo una rappresentazione dell'importanza delle diverse feature nella previsione di un determinato output. Il loro concetto di base si fonda sull'analisi dei gradienti, che consente di evidenziare il contributo delle feature rispetto all'output del modello. In pratica, la generazione di saliency map avviene attraverso il calcolo del gradiente dell'output rispetto all'input, mostrando come piccoli cambiamenti nelle feature d'ingresso influenzano l'output del modello. Nel caso di una rete neurale, si calcola la derivata parziale dell'output rispetto a ciascuna feature dell'input. In questo modo, si ottiene una misura che rappresenta quanto una piccola variazione di quella specifica feature influisca sull'output. Maggiore è il valore assoluto del gradiente, più rilevante sarà l'importanza della feature in questione. In altre parole, se una piccola variazione di una determinata feature causa un grande cambiamento nell'output del modello, ciò suggerisce che quella feature ha un ruolo significativo nella previsione. Le saliency map evidenziano quindi quali parti dell'input hanno contribuito maggiormente alla previsione del modello, fornendo un'indicazione chiara delle feature su cui il modello si sta concentrando per prendere una determinata decisione. In un contesto di Anomaly Detection per un sistema informatico, un'analisi del genere può risultare cruciale: essere avvisati in tempo dell'insorgere di un'anomalia e avere delle indicazioni riguardo cosa la sta causando è fondamentale per poter intervenire prontamente e prevenire eventuali danni o interruzioni del servizio. In particolare, un esperto del dominio potrebbe trarre enormi benefici dal ricevere suggerimenti riguardo la natura del problema, poiché sarebbe in grado di interpretare in maniera più accurata e rapida i segnali provenienti dal sistema e identificare le cause del comportamento anomalo. Di seguito verranno mostrati tre casi d'esempio in cui è stata utilizzata la Saliency Map per analizzare la natura dell'anomalia, evidenziando i benefici derivanti dall'applicazione di questa tecnica. In particolare, verrà mostrato come la Saliency Map consenta di identificare le aree critiche del sistema, fornendo chiare informazioni riguardo la causa dell'anomalia.

4.7.1 Saliency Map - Caso d'esempio 1

Nel primo caso d'esempio verrà analizzata l'anomalia mostrata nella Figura 4.3. Per questa finestra temporale anomala si hanno a disposizione dei dati ogni 3 ore, ovvero con Retention Policy *all_data*.

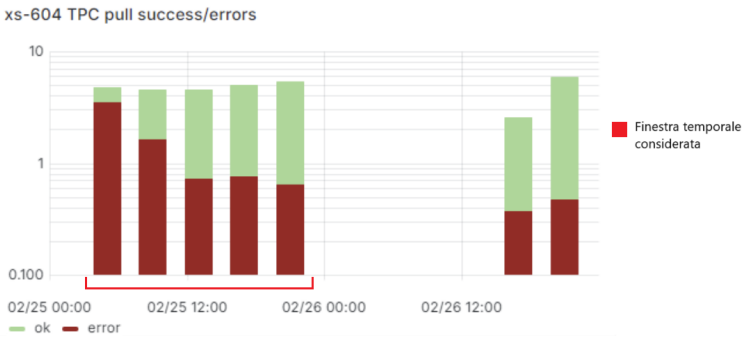

Figura 4.3: Saliency Map - Caso d'esempio 1 - TPC pull success/error

L'anomalia in questione è stata esaminata analizzando le metriche mostrate nella dashboard dell'INFN-CNAF, al fine di comprenderne le cause e identificare il motivo del comportamento anomalo rilevato. L'anomalia ha origine a causa di un aumento dei dispatch attivi da gestire, come mostrato in Figura 4.4. Un aumento dei dispatch attivi indica un incremento nel numero di richieste che il sistema deve assegnare ai vari processi o nodi di calcolo per l'elaborazione. Questo significa che la quantità di lavoro in attesa di essere processata o distribuita tra le risorse disponibili è cresciuta significativamente. L'aumento improvviso e consistente di queste richieste ha progressivamente saturato le risorse del sistema, impedendogli di gestire efficacemente il carico di lavoro. Di conseguenza, si è verificato un sovraccarico critico che ha causato il suo arresto temporaneo, con una conseguente interruzione dei servizi e impossibilità di rispondere correttamente alle richieste future. Nella Figura 4.4 è evidenziato con un cerchio rosso l'istante temporale in cui il sistema si è saturato, causando un crash o un arresto temporaneo. Esso coincide con l'ultimo istante temporale in cui riceviamo dei dati dalla metrica che misura le richieste di TPC pull success/error, mostrata nella Figura 4.3. È inoltre evidenziato con un cerchio blu il punto in cui il sistema è stato riavviato ed è tornato a funzionare.

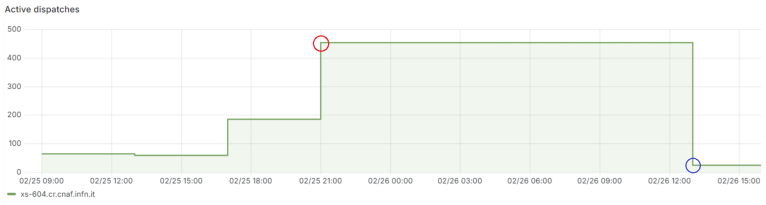

Figura 4.4: Saliency Map - Caso d'esempio 1 - Dispatch Attivi

Il sovraccarico è confermato dall'osservazione di un'altra metrica, ovvero l'utilizzo della pool di thread, come mostrato in Figura 4.5. Infatti, il grafico mostra chiaramente un incremento graduale e consistente dell'utilizzo della pool di thread, partendo da circa il 20% fino a raggiungere il 100%, ovvero la piena saturazione delle risorse di calcolo disponibili. Il costante aumento e la successiva saturazione dell'utilizzo della pool di thread coincide con il picco di dispatch attivi osservato in precedenza, confermando che l'incremento delle richieste ha portato a un sovraccarico delle risorse disponibili, con un conseguente crash del sistema.

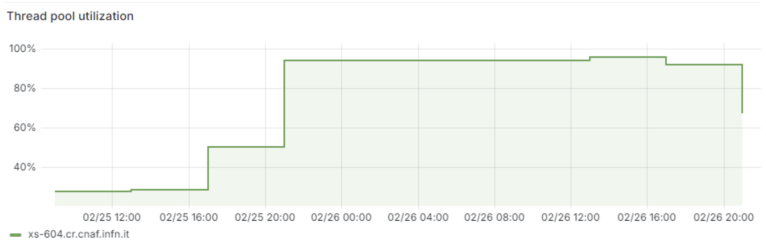

Figura 4.5: Saliency Map - Caso d'esempio 1 - Thread Pool Utilization

Una volta identificata la causa dell'anomalia, l'osservazione anomala viene fornita in input ai modelli precedentemente addestrati. La previsione risultante classifica correttamente l'osservazione come anomala. Grazie alla presenza di un unico neurone nell'output layer e dell'uso della funzione di attivazione sigmoide, è possibile non solo classificare correttamente l'osservazione come anomala, ma anche ottenere la probabilità associata a tale previsione. In questo modo si ottiene un'informazione molto importante: quanto più il valore della previsione si avvicina a 1, tanto maggiore è la confidenza del modello nel classificare l'osservazione come anomala. Di conseguenza, le informazioni estratte dalla Saliency Map saranno più attendibili, in quanto il modello avrà una maggiore sicurezza nella propria predizione. Per il caso

d'esempio analizzato, il modello classifica l'osservazione come anomala con una probabilità dello *0.99975*. Il modello è quindi molto sicuro della sua previsione, e le informazioni che ne derivano di conseguenza possono essere considerate altamente affidabili. Utilizzando la tecnica della Saliency Map, ovvero calcolando il gradiente rispetto alle feature di input che compongono l'osservazione, aggregando i gradienti per metrica e ordinando le metriche per importanza dal valore più alto a quello più basso, è possibile ottenere una lista delle feature più importanti che la rete ha utilizzato per individuare l'anomalia. Per quanto riguarda questo caso d'esempio, sono mostrate le cinque metriche più importanti per l'individuazione dell'anomalia, con la relativa importanza rappresentata dal valore dei gradienti aggregati, nella Tabella 4.9. Le metriche mostrate nella tabella sono ordinate in maniera decrescente, ovvero dalla metrica più importante a quella meno importante.

Metrica	**Gradienti aggregati**
storm.http.thread-pool.utilization.value	0.000650
storm.http.handler.dispatches.m1_rate	0.000631
storm.http.handler.5xx-responses.m1_rate	0.000548
storm.http.handler.4xx-responses.m1_rate	0.000506
cpu.total.nice	0.000470

Tabella 4.9: Caso d'esempio 1 - Le 5 metriche più importanti per l'individuazione dell'anomalia

Confrontando la tabella con l'analisi delle metriche presenti nella Dashboard dell'INFN-CNAF svolta in precedenza, si nota subito che la Saliency Map è riuscita a individuare correttamente le cause dell'anomalia. Essa infatti indica che la metrica più importante risulta essere *storm.http.thread-pool.utilization.value*, ovvero la metrica che misura il tasso di utilizzo della pool di thread, che come mostrato precedentemente è aumentato in maniera costante fino a raggiungere il 100%, saturando completamente le risorse disponibili. La metrica individuata come seconda metrica più importante è *storm.http.handler.dispatches.m1_rate*, ovvero la metrica che misura il tasso di dispatch correttamente gestiti rispetto alla totalità dei dispatch in arrivo. Anche questa metrica, come evidenziato nell'analisi precedente, ha mostrato un aumento progressivo fino al punto di saturazione, risultando una causa diretta del sovraccarico complessivo del sistema e del conseguente crash. La metrica *storm.http.handler.5xx-responses.m1_rate*, ovvero la metrica che mi-

sura il tasso di risposte 5xx (errori del server), restituita dalla Saliency Map come la terza metrica più importante, rappresenta un incremento di errori lato server a causa dell'incapacità di gestire le richieste in arrivo. Questo è probabilmente collegato alla saturazione della pool di thread e all'incremento dei dispatch non gestiti correttamente, poiché un sistema sovraccarico non è in grado di completare correttamente le richieste, restituendo di conseguenza errori di tipo 5xx. In modo simile alla metrica precedente, la metrica *storm.http.handler.4xx-responses.m1_rate*, ovvero la metrica che misura il tasso di risposte 4xx (errori lato client), indicata come la quarta metrica più importante, segnala che le richieste dei client non sono state processate correttamente. Questo avviene probabilmente a causa della saturazione del sistema o di errori dovuti al sovraccarico delle risorse. Infine la metrica *cpu.total.nice*, ovvero la metrica che misura la quantità di tempo che la CPU ha dedicato a eseguire processi a priorità ridotta, indicata come la quinta più importante, potrebbe segnalare che alcuni processi a bassa priorità hanno consumato risorse di CPU, sottraendole ai processi critici. Questo avrebbe ulteriormente contribuito al sovraccarico e al malfunzionamento del sistema, poiché le risorse già limitate sono state utilizzate inefficientemente.

L'utilizzo della Saliency Map risulta quindi essere estremamente importante in un contesto del genere, in quanto permette di identificare le metriche più rilevanti che hanno contribuito all'anomalia. Ciò permette di fornire importanti suggerimenti a un esperto del dominio riguardo a cosa stia andando storto all'interno del sistema, permettendo un intervento tempestivo e mirato per risolvere il problema.

4.7.2 Saliency Map - Caso d'esempio 2

Nel secondo caso d'esempio verrà analizzata l'anomalia mostrata nella Figura 4.6. Per questa finestra temporale anomala si hanno a disposizione dei dati ogni 15 minuti, ovvero con Retention Policy *one_month*. Come mostrato, l'anomalia considerata è quella delle ore 07:00, e la finestra temporale considerata è quella che comprende i cinque istanti temporali precedenti a quello anomalo. L'anomalia è stata esaminata analizzando le metriche mostrate nella Dashboard dell'INFN-CNAF, al fine di comprenderne le cause e identificare il motivo del comportamento anomalo. Anche in questo caso, in maniera simile al caso precedente, l'anomalia ha origine a causa di un aumento dei dispatch attivi da gestire, come mostrato in Figura 4.7. Si osserva quindi un incremento nel numero di richieste che il sistema deve assegnare ai vari processi per l'elaborazione. L'aumento consistente di queste richieste ha progressivamente saturato le risorse del sistema, impedendo al sistema di

gestire efficacemente il carico di lavoro, portandolo a un crash. Come nel caso precedente, l'istante in cui il sistema si è saturato è evidenziato con un cerchio rosso, mentre il momento in cui è stato riavviato e ha ripreso a funzionare è indicato con un cerchio blu.

Figura 4.6: Saliency Map - Caso d'esempio 2 - TPC pull success/error

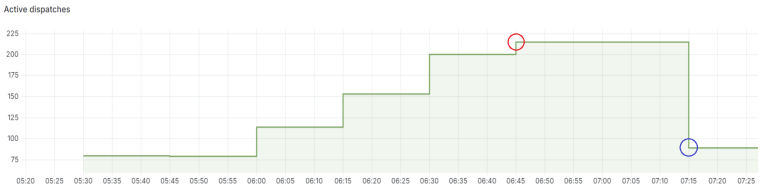

Figura 4.7: Saliency Map - Caso d'esempio 2 - Dispatch Attivi

Anche in questo caso, la metrica direttamente correlata con il sovraccarico è l'utilizzo della pool di thread, come mostrato in Figura 4.8. È mostrato l'incremento consistente della percentuale di utilizzo, fino a raggiungere il 100%, ovvero la piena saturazione delle risorse disponibili. L'anomalia rilevata è quindi molto simile a quella descritta nel caso precedente, con la differenza che questa si sviluppa in un tempo significativamente più breve. Infatti, confrontando i grafici, è possibile notare che questa anomalia si sviluppa esattamente nei cinque istanti temporali precedenti a quello anomalo, mentre nel caso precedente il processo di sviluppo richiedeva diverse ore. Ciò suggerisce che, in questo caso, il sovraccarico del sistema sia stato più repentino e improvviso, probabilmente causato da un aumento più brusco delle

richieste. Di conseguenza, il modello potrebbe incontrare maggiori difficoltà nel rilevare questa anomalia rispetto alla precedente, poiché dispone di una finestra temporale più breve in cui la situazione anomala si evolve rapidamente, riducendo la quantità di informazioni utili per individuare i segnali di sovraccarico.

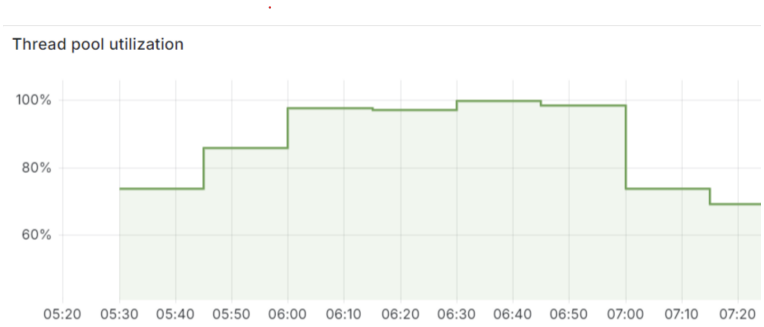

Figura 4.8: Saliency Map - Caso d'esempio 2 - Thread Pool Utilization

Per questo caso d'esempio, il modello classifica l'osservazione come anomala con una probabilità dello *0.75599*. Il modello è quindi meno sicuro della sua previsione rispetto al caso precedente, come prevedibile data la natura più rapida e meno evidente dello sviluppo dell'anomalia. Anche in questo caso però, utilizzando la tecnica della Saliency Map, aggregando i gradienti per metrica e ordinando le metriche per importanza, è possibile notare che il modello ha ben compreso la natura e le cause dell'anomalia, come mostrato nella Tabella 4.10.

Metrica	**Gradienti aggregati**
storm.http.thread-pool.utilization.value	0.677137
storm.http.handler.dispatches.m1_rate	0.676444
memory.used	0.555614
memory.cached	0.553678
iostat.avg-cpu.pct_system	0.514735

Tabella 4.10: Caso d'esempio 2 - Le 5 metriche più importanti per l'individuazione dell'anomalia

Osservando la tabella si può notare che la metrica individuata come più importante risulta essere la metrica *storm.http.thread-pool.utilization.value*, ovvero la metrica che misura il tasso di utilizzo della pool di thread che, come mostrato precedentemente, è aumentato fino a raggiungere il 100%, saturando le risorse disponibili. La seconda metrica più importante risulta essere *storm.http.handler.dispatches.m1_rate*, ovvero la metrica che misura il tasso di dispatch gestiti rispetto a quelli totali. Come evidenziato precedentemente, questo valore è aumentato in maniera progressiva fino a saturare il sistema. La terza metrica più importante è *memory.used*, ovvero la metrica che misura la quantità di memoria attualmente utilizzata dal sistema. Un incremento significativo della memoria utilizzata può indicare che il sistema sta consumando una quantità eccessiva di risorse, probabilmente a causa dell'aumento di richieste in entrata o dei processi in esecuzione, portando il sistema incontro a possibili rallentamenti e difficoltà nella gestione del carico di lavoro. La quarta metrica più importante rilevata è *memory.cached*, ovvero la metrica che rappresenta la quantità di memoria cache utilizzata dal sistema per mantenere dati temporanei in modo da accelerare l'accesso alle risorse. Un uso eccessivo della cache può però sottrarre risorse preziose alla memoria libera disponibile, portando il sistema a una condizione di scarsità di memoria. In un contesto di sovraccarico come quello descritto, l'aumento della memoria cache potrebbe aver peggiorato la situazione, riducendo ulteriormente le risorse disponibili per le operazioni critiche e contribuendo al crash. Infine, la quinta metrica più importante risulta essere *iostat.avg-cpu.pct_system*, ovvero la metrica che misura la percentuale di tempo che la CPU dedica alle operazioni di sistema. Un aumento significativo di questa metrica indica che una porzione rilevante della potenza di calcolo veniva impiegata per gestire le operazioni di sistema, probabilmente a causa del sovraccarico di richieste. In particolare, quando la CPU è impegnata a gestire operazioni di sistema, rimangono meno risorse per elaborare le richieste degli utenti, il che può rallentare le prestazioni e, in casi estremi, portare a un collasso del sistema.

Anche in questo caso quindi, nonostante l'anomalia si sia sviluppata in un periodo più breve e sia di conseguenza più difficile da individuare e comprendere, la Saliency Map si è dimostrata estremamente utile, permettendo di identificare le metriche principali che hanno contribuito all'anomalia. Infatti, nonostante la probabilità con cui il modello ha classificato l'osservazione temporale come anomala sia decisamente inferiore rispetto al caso precedente, esso è stato in grado di comprendere a fondo la natura del problema, fornendo importanti indicazioni riguardo le cause che hanno portato all'anomalia. Fornire queste informazioni a un esperto del dominio permetterebbe un intervento tempestivo e mirato che eviterebbe l'anomalia e salvaguarderebbe la

stabilità e l'efficienza del sistema, prevenendo potenziali malfunzionamenti.

4.7.3 Saliency Map - Caso d'esempio 3

Nel terzo e ultimo caso d'esempio verrà analizzata l'anomalia mostrata nella Figura 4.9. Anche per questa finestra temporale si hanno a disposizione dei dati ogni 15 minuti, ovvero con Retention Policy *one_month*. L'anomalia considerata si verifica dalle ore 10:00 alle ore 10:30, e la finestra temporale analizzata comprende i cinque istanti temporali precedenti a quelli anomali.

Figura 4.9: Saliency Map - Caso d'esempio 3 - TPC pull success/error

Osservando le metriche del sistema, appare evidente che le risorse sono sottoposte a un carico costantemente elevato, con un numero significativo di processi e operazioni in esecuzione. Questo indica una gestione di dati e attività particolarmente intensa. Come mostrato infatti nella Figura 4.9, che indica l'andamento della metrica che rappresenta l'utilizzo della pool di thread, il sistema ha mantenuto un utilizzo quasi costante al 100% fino alle 09:45, ovvero l'istante temporale precedente all'anomalia. Questo suggerisce che, fino a quel momento, la capacità di calcolo fosse prossima al limite, indicando un carico molto elevato sui thread disponibili. Si sospetta quindi che l'anomalia rilevata possa essere una conseguenza diretta di questo carico prolungato, che potrebbe aver sovraccaricato le risorse di calcolo o portato a uno stato di saturazione dei thread. Questo caso d'esempio è differente rispetto ai casi precedenti, in cui l'anomalia era causata da un picco crescente di dispatch che portava alla saturazione del sistema. In questo caso, infatti, il carico rimane elevato e stabile per un lungo periodo di tempo, come mostrato nella Figura 4.9. Ciò indica che il sistema è costantemente sotto sforzo e non

riesce a ridurre il numero di processi attivi, evidenziando una situazione di stress prolungato delle risorse.

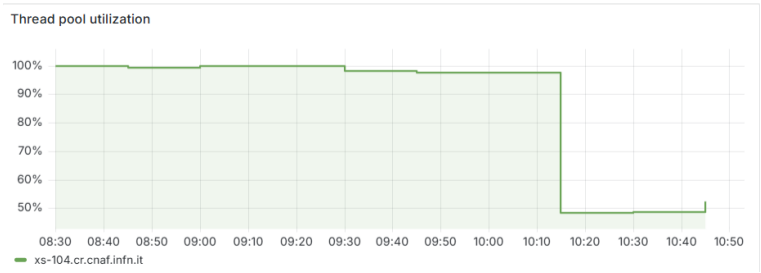

Figura 4.10: Saliency Map - Caso d'esempio 3 - Thread pool utilization

Per questo caso d'esempio, il modello classifica l'osservazione come anomala con una probabilità dello *0.98737*. Il modello è quindi molto sicuro della previsione, e per comprendere meglio le cause dell'anomalia è possibile analizzare le metriche più rilevanti associate a essa, identificate dalla Saliency Map e mostrate nella Tabella 4.11.

Metrica	**Gradienti aggregati**
cpu.ctxt	0.060165
storm.WebDAV.TPC.pull.ok-count.count	0.049626
storm.http.handler.4xx-responses.m1_rate	0.039669
storm.http.handler.dispatches.m1_rate	0.035235
iostat.avg-cpu.pct_iowait	0.034120

Tabella 4.11: Caso d'esempio 3 - Le 5 metriche più importanti per l'individuazione dell'anomalia

Osservando la tabella si nota che la metrica individuata come più importante è *cpu.ctxt*, ovvero la metrica che misura il numero di context switch della CPU. Questa metrica è stata individuata come più importante probabilmente perché un aumento nei context switches indica che la CPU è sottoposta a un alto carico di lavoro ed è costretta a passare rapidamente tra processi per gestire un numero elevato di richieste simultanee. La seconda metrica più importante è *storm.WebDAV.TPC.pull.ok-count.count*, ovvero la metrica che conta il numero di operazioni TPC di tipo pull che

sono state completate con successo. Come mostrato nella Figura 4.9, il numero di richieste TPC pull andate a buon fine è aumentato nei due istanti temporali precedenti all'anomalia, con un trend crescente. Questo indica un uso intensivo del servizio e impone una pressione aggiuntiva sulle risorse, che può causare situazioni anomale quando la capacità di elaborazione non riesce a tenere soddisfare la domanda crescente. La terza metrica più importante è *storm.http.handler.4xx-responses.m1_rate*, che rappresenta il tasso di risposte di tipo 4xx, ovvero errori dovuti a richieste non valide inviate dai client. Questa metrica è mostrata nella Figura 4.11. Come si può notare, negli istanti precedenti l'anomalia, il tasso di errori di tipo 4xx è aumentato, indicando che molte richieste non vengono soddisfatte con successo. Questo può verificarsi quando la capacità del sistema è già stata saturata, rendendo difficile per i client accedere correttamente ai dati, riflettendo una limitata disponibilità di risorse.

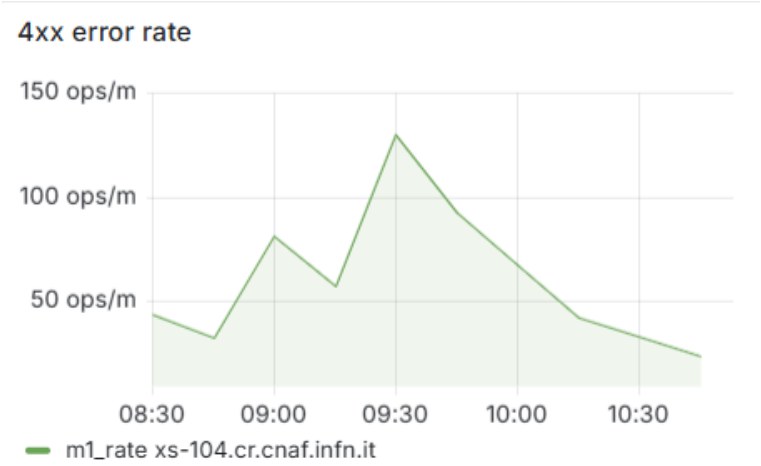

Figura 4.11: Saliency Map - Caso d'esempio 3 - 4xx Error Rate

La quarta metrica più importante è *storm.http.handler.dispatches.m1_rate*, ovvero la metrica che misura il tasso di dispatch gestiti dal sistema. Un incremento di questa metrica riflette l'elevato numero di richieste WebDAV da elaborare. Se il sistema gestisce un volume eccessivo di richieste, la CPU potrebbe faticare a mantenere la rapidità di elaborazione neccessaria, causando rallentamenti e possibili malfunzionamenti. Infine, la quinta metrica più importante è *iostat.avg-cpu.pct_iowait*, ovvero la metrica che misura la percentuale del tempo totale in cui la CPU è in stato di attesa I/O. Un incremento di questa metrica indica che la CPU sta trascorrendo una parte significativa del tempo in attesa di operazioni di input/output, creando

potenziali colli di bottiglia nelle prestazioni complessive del sistema. Queste metriche, nel loro insieme, dipingono il quadro di un sistema WebDAV in condizioni di stress estremo, in cui l'elevata richiesta di risorse CPU e I/O ha causato un malfunzionamento culminato nel crash del sistema. Infatti, l'aumento degli errori di tipo 4xx e il tempo di attesa elevato per le operazioni di I/O suggeriscono che il sistema è stato sovraccaricato al punto da non riuscire più a rispondere alle richieste. La saturazione delle risorse, combinata con un elevato tasso di richieste WebDAV, ha superato le capacità del sistema, portandolo a un collasso completo.

L'integrazione di sistemi di allerta preventiva, basati sui modelli di Machine Learning discussi in precedenza, insieme a un'analisi tempestiva della Saliency Map, permetterebbe quindi di identificare rapidamente situazioni in cui l'aumento continuo delle richieste porta alla saturazione delle risorse, il sistema si trova in condizioni di stress estremo, o si manifestano altri scenari anomali che potrebbero causare un'anomalia. Questo permetterebbe di intervenire tempestivamente, evitando che il sistema raggiunga uno stato critico e prevenendo così malfunzionamenti o interruzioni del servizio.

Capitolo 5

Conclusioni e Lavori futuri

In questo studio è stato affrontato il problema dell'Anomaly Detection in sistemi informatici, in particolare in un servizio WebDAV, utilizzando un'ampia gamma di modelli di Machine Learning, dalle architetture più semplici come il Single-Layer Perceptron, a modelli più avanzati come le Long Short-Term Memory (LSTM). L'obiettivo è stato quello di identificare anomalie nei dati temporali raccolti dai sensori del servizio, affrontando sfide come la natura sbilanciata dei dataset e la necessità di catturare dipendenze temporali e spaziali tra le variabili analizzate. Dalle analisi svolte è emerso chiaramente che, sebbene i modelli più semplici come il Single-Layer Perceptron offrono una base utile per un'esplorazione iniziale dei dati, essi non sono sufficientemente potenti per cogliere le complessità presenti in dataset con caratteristiche che richiedono una modellazione accurata delle dipendenze temporali. I modelli più avanzati, come le LSTM, si sono dimostrati molto più efficaci, grazie alla loro capacità di gestire le relazioni temporali nei dati sequenziali, aspetto cruciale nel contesto di questo studio. Infatti, poché i dati analizzati sono di natura temporale, si è rivelato essenziale catturare le dipendenze tra istanti temporali successivi per poter identificare correttamente diversi tipi di anomalie. L'uso delle LSTM ha permesso di tenere in conto delle relazioni a lungo e breve termine tra le osservazioni, migliorando significativamente la capacità del modello di individuare anomalie complesse e emergenti. Uno degli aspetti cruciali che ha influenzato il successo dei modelli è stato il tuning degli iperparametri. Attraverso un processo di random search e grid search, sono state ottimizzate configurazioni come la funzione di Loss, l'ottimizzatore, il learning rate e, nel caso delle LSTM, il numero di layer e il numero di unità LSTM per layer. Questo ha permesso di migliorare significativamente le performance dei modelli, riducendo l'overfitting e massimizzando la capacità di generalizzazione. In particolare, l'utilizzo di

una Funzione di Loss composta, basata su Binary Cross-Entropy e F1-Score, si è rivelata particolarmente utile nel contesto dell'Anomaly Detection, dove l'obiettivo principale è l'identificazione accurata delle anomalie. Questa funzione ha permesso di bilanciare la minimizzazione degli errori con la selezione dell'epoca ottimale per rilevare le anomalie, migliorando sensibilmente i risultati. I risultati ottenuti, soprattutto da modelli più complessi come le Long Short-Term Memory, sono complessivamente positivi, con valori elevati di Precision e Recall, dimostrando una buona capacità dei modelli di rilevare correttamente le anomalie e ridurre il numero di falsi positivi. Tuttavia, i risultati ottenuti non sono perfetti. Infatti, nonostante i risultati promettenti, ci sono ancora diversi aspetti che possono essere migliorati o ampliati in lavori futuri. Ci sono due limiti principali all'approccio applicato: la scarsa quantità di dati anomali e la bassa granularità dei dati disponibili. Il dataset costruito risulta infatti estremamente sbilanciato e, nonostante siano state esplorate tecniche di bilanciamento come la replicazione dei campioni o l'uso di funzioni di loss che ponderano gli errori in base alla rarità della classe, questi metodi non sono stati sufficienti a garantire una piena generalizzazione del modello. Un aumento dei dati anomali disponibili permetterebbe migliorerebbe significativamente la capacità del modello di apprendere le caratteristiche distintive delle anomalie, migliorando sia la Precision che la Recall. Un ulteriore limite, forse il più rilevante, riguarda la bassa granularità dei dati a disposizione. I dati sui quali sono addestrati i modelli sono infatti campionati ogni 15 minuti e sono ottenuti come media dei dati raccolti ogni 5 minuti. Questa riduzione della granularità potrebbe comportare la perdita di dettagli cruciali e nascondere pattern più sottili e variazioni temporali significative, che potrebbero essere utili per identificare in modo più preciso le osservazioni anomale. Infatti, le anomalie che potrebbero verificarsi su intervalli di tempo più brevi o in picchi improvvisi, rischiano di essere appiattite o mascherate nel calcolo della media, impedendo ai modelli di rilevarle in maniera accurata. Questo compromette la capacità del modello di cogliere pattern temporali più sottili e di rispondere adeguatamente a cambiamenti repentini nel sistema. Una risoluzione temporale più elevata permetterebbe di fornire una rappresentazione più fedele del comportamento del sistema, migliorando significativamente l'efficacia del modello nel rilevamento delle anomalie. In conclusione, il lavoro svolto ha dimostrato come l'utilizzo di tecniche di Machine Learning possa rappresentare un metodo efficace e automatizzato per la rilevazione delle anomalie in sistemi informatici complessi, consentendo un monitoraggio continuo e una risposta tempestiva ai cambiamenti inattesi nelle metriche di sistema. L'approccio adottato si è rivelato utile nell'individuare pattern anomali che, altrimenti, sarebbero difficili da rilevare con metodi tradizionali di analisi manuale, i quali richiederebbero

un monitoraggio costante da parte di esperti del dominio. Questo lavoro fornisce quindi una solida base per futuri sviluppi, che potranno beneficiare di dati più completi e dettagliati, consentendo di migliorare ulteriormente le prestazioni dei modelli e rafforzare la capacità di rilevazione delle anomalie in modo più preciso e affidabile.

Bibliografia

[1] Laura Viola, Elisabetta Ronchieri, and Claudia Cavallaro. Combining log files and monitoring data to detect anomaly patterns in a data center. *Comput.*, 11(8):117, 2022.

[2] Claudia Cavallaro and Elisabetta Ronchieri. Identifying anomaly detection patterns from log files: A dynamic approach. In Osvaldo Gervasi, Beniamino Murgante, Sanjay Misra, Chiara Garau, Ivan Blecic, David Taniar, Bernady O. Apduhan, Ana Maria A. C. Rocha, Eufemia Tarantino, and Carmelo Maria Torre, editors, *Computational Science and Its Applications - ICCSA 2021 - 21st International Conference, Cagliari, Italy, September 13-16, 2021, Proceedings, Part II*, volume 12950 of *Lecture Notes in Computer Science*, pages 517–532. Springer, 2021.

[3] Zezhou Li, Jing Zhang, Xianbo Zhang, Feng Lin, Chao Wang, and Xingye Cai. Natural language processing-based model for log anomaly detection. In *2022 IEEE 2nd International Conference on Software Engineering and Artificial Intelligence (SEAI)*, pages 129–134, 2022.

[4] Dongjiang Li, Jing Zhang, Xianbo Zhang, Feng Lin, Chao Wang, and Liang Chang. Logps: A robust log sequential anomaly detection approach based on natural language processing. In *22nd IEEE International Conference on Communication Technology, ICCT 2022, Nanjing, China, November 11-14, 2022*, pages 1400–1405. IEEE, 2022.

[5] Christophe Bertero, Matthieu Roy, Carla Sauvanaud, and Gilles Trédan. Experience report: Log mining using natural language processing and application to anomaly detection. In *28th IEEE International Symposium on Software Reliability Engineering, ISSRE 2017, Toulouse, France, October 23-26, 2017*, pages 351–360. IEEE Computer Society, 2017.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[7] Haixuan Guo, Shuhan Yuan, and Xintao Wu. Logbert: Log anomaly detection via BERT. In *International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021*, pages 1–8. IEEE, 2021.

[8] Xiao Han, Shuhan Yuan, and Mohamed Trabelsi. Loggpt: Log anomaly detection via GPT. In Jingrui He, Themis Palpanas, Xiaohua Hu, Alfredo Cuzzocrea, Dejing Dou, Dominik Slezak, Wei Wang, Aleksandra Gruca, Jerry Chun-Wei Lin, and Rakesh Agrawal, editors, *IEEE International Conference on Big Data, BigData 2023, Sorrento, Italy, December 15-18, 2023*, pages 1117–1122. IEEE, 2023.

[9] Senming Yan, Lei Shi, Jing Ren, Wei Wang, Yaxin Liu, Limin Sun, Xiong Wang, and Wei Zhang. Log-based anomaly detection with transformers pre-trained on large-scale unlabeled data. In *IEEE International Conference on Communications, ICC 2024, Denver, CO, USA, June 9-13, 2024*, pages 1–6. IEEE, 2024.

[10] Hongcheng Guo, Xingyu Lin, Jian Yang, Yi Zhuang, Jiaqi Bai, Tieqiao Zheng, Bo Zhang, and Zhoujun Li. Translog: A unified transformer-based framework for log anomaly detection. *CoRR*, abs/2201.00016, 2022.

[11] Jin Wang, Yangning Tang, Shiming He, Changqing Zhao, Pradip Kumar Sharma, Osama Alfarraj, and Amr Tolba. Logevent2vec: Logevent-to-vector based anomaly detection for large-scale logs in internet of things. *Sensors*, 20(9):2451, 2020.

[12] Yuanyuan Wei, Julian Jang-Jaccard, Wen Xu, Fariza Sabrina, Seyit Camtepe, and Mikael Boulic. Lstm-autoencoder based anomaly detection for indoor air quality time series data. *CoRR*, abs/2204.06701, 2022.

[13] Stephen Githinji and Ciira Wa Maina. Anomaly detection on time series sensor data using deep lstm-autoencoder. In *IEEE AFRICON 2023, Nairobi, Kenya, September 20-22, 2023*, pages 1–6. IEEE, 2023.

[14] Minh-Hao Ho, Nhu-Y. Tran-Van, and Kim-Hung Le. A multi-input bi-lstm autoencoder model with wavelet transform for air quality pre-

diction. In *International Conference on Multimedia Analysis and Pattern Recognition, MAPR 2024, Da Nang, Vietnam, August 15-16, 2024*, pages 1–6. IEEE, 2024.

[15] Robin Frehner, Kesheng Wu, Alexander Sim, Jinoh Kim, and Kurt Stockinger. Detecting anomalies in time series using kernel density approaches. *IEEE Access*, 12:33420–33439, 2024.

[16] Michael R. Lindstrom, Hyuntae Jung, and Denis Larocque. Functional kernel density estimation: Point and fourier approaches to time series anomaly detection. *Entropy*, 22(12):1363, 2020.

[17] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 413–422. IEEE Computer Society, 2008.

[18] Aleksandar Petkovski and Visar Shehu. Anomaly detection on univariate sensing time series data for smart aquaculture using k-means, isolation forest, and local outlier factor. In *12th Mediterranean Conference on Embedded Computing, MECO 2023, Budva, Montenegro, June 6-10, 2023*, pages 1–5. IEEE, 2023.

[19] Hongzuo Xu, Guansong Pang, Yijie Wang, and Yongjun Wang. Deep isolation forest for anomaly detection. *IEEE Trans. Knowl. Data Eng.*, 35(12):12591–12604, 2023.

[20] Orieb AbuAlghanam, Hadeel Alazzam, Esraa Alhenawi, Mohammad Qatawneh, and Omar Adwan. Fusion-based anomaly detection system using modified isolation forest for internet of things. *J. Ambient Intell. Humaniz. Comput.*, 14(1):131–145, 2023.

[21] Elham Enayati, Reza Mortazavi, Abdolali Basiri, Javad Ghasemian, and Mahmoud Moallem. Time series anomaly detection via clustering-based representation. *Evol. Syst.*, 15(4):1115–1136, 2024.

[22] Chang Dong, Jianfeng Tao, Qun Chao, Honggan Yu, and Chengliang Liu. Subsequence time series clustering-based unsupervised approach for anomaly detection of axial piston pumps. *IEEE Trans. Instrum. Meas.*, 72:1–12, 2023.

[23] Bowen Chen. DGNN: dynamic graph neural networks for anomaly detection in multivariate time series. In Shi-Kuo Chang, editor, *The 35th*

International Conference on Software Engineering and Knowledge Engineering, SEKE 2023, KSIR Virtual Conference Center, USA, July 1-10, 2023, pages 415–420. KSI Research Inc., 2023.

[24] Hongtai Guo, Zhangbing Zhou, Deng Zhao, and Walid Gaaloul. EGNN: energy-efficient anomaly detection for *IoT* multivariate time series data using graph neural network. *Future Gener. Comput. Syst.*, 151:45–56, 2024.

[25] Ziyu Guo, Weiyang Kong, and Yubao Liu. Grouped graph neural networks for anomaly detection in time series. In *International Joint Conference on Neural Networks, IJCNN 2024, Yokohama, Japan, June 30 - July 5, 2024*, pages 1–8. IEEE, 2024.

[26] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. Anomaly detection in time series: A comprehensive evaluation. *Proc. VLDB Endow.*, 15(9):1779–1797, 2022.

[27] Aoqian Zhang, Shuqing Deng, Dongping Cui, Ye Yuan, and Guoren Wang. An experimental evaluation of anomaly detection in time series. *Proc. VLDB Endow.*, 17(3):483–496, 2023.

[28] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386—-408, 1958.

[29] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001.

[30] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[31] Joan Pastor-Pellicer, Francisco Zamora-Martínez, Salvador España Boquera, and María José Castro Bleda. F-measure as the error function to train neural networks. In Ignacio Rojas, Gonzalo Joya Caparrós, and Joan Cabestany, editors, *Advances in Computational Intelligence - 12th International Work-Conference on Artificial Neural Networks, IWANN 2013, Puerto de la Cruz, Tenerife, Spain, June 12-14, 2013, Proceedings, Part I*, volume 7902 of *Lecture Notes in Computer Science*, pages 376–384. Springer, 2013.