POLITECNICO DI TORINO

Laurea magistrale in ingegneria informatica
orientamento Data Analytics

UNSUPERVISED ANOMALY DETECTION SU METRICHE DI PERFORMANCE DI APPLICATION SERVER

Relatore

Prof. DANIELE APILETTI

Ing. ANTONIO GRECO

Ing. ALESSANDRO TEDESCO

Candidato

STEFANO BERGIA

DICEMBRE 2021

Sommario

Con il termine Anomaly detection si intendono tutte quelle tecniche di data analytics volte ad individuare comportamenti o caratteristiche inattese all'interno di un insieme di dati; nel caso specifico di questa tesi i dati analizzati sono metriche di performance di un server weblogic; un application server sviluppato da Oracle che permette di eseguire applicazioni web aziendali.

Le metriche rappresentano il variare di alcuni parametri di sistema nel tempo, quali la percentuale di memoria occupata, il numero di connessioni aperte o il numero di richieste ricevute, questi tipi di dati vengono definiti serie temporali e hanno una serie di proprietà interessanti, ad esempio il trend e la stagionalità, che possono essere sfruttati per fare assunzioni sul comportamento normale della serie stessa.

L'obiettivo della ricerca è quello di sviluppare dei modelli che sfruttando le caratteristiche matematiche delle serie temporali e alcuni modelli di machine learning già consolidati, siano in grado di rilevare la presenza di anomalie all'interno delle metriche di performance.

L'interesse nello sviluppare questi modelli sta nel fatto che riuscire ad identificare le anomalie in maniera automatica e soprattutto non supervisionata (ovvero senza una conoscenza pregressa sul fatto che in un dato intervallo di tempo si sia verificata un'anomalia o meno) sarebbe molto vantaggioso per chi si occupa della manutenzione del server, dato che si potrebbe intervenire in tempi rapidissimi in caso di problemi, prima che diventino bloccanti e causino danni economici.

Ringraziamenti

Ringrazio la mia famiglia, che mi ha sempre sostenuto emotivamente ed economicamente, i miei amici, che mi supportano e soprattutto sopportano da cinque anni, ed infine i colleghi di Mediamente consulting che hanno saputo trasmettermi la loro professionalità e passione per questo lavoro.

Indice

Elenco delle figure

Introduzione

1.1 Spiegazione del caso di studio

Negli ultimi anni l'analisi di grandi quantità di dati attraverso algoritmi di apprendimento automatico sta assumendo un'importanza sempre più rilevante, soprattutto in ambito business.
Le aziende si stanno rendedno conto che l'estrazione di pattern e peculiarità dai loro dati storicizzati ha un'importanza strategica fondamentale e che nel prossimo futuro questo potrà determinare la differenza tra un'azienda in crescita e una destinata a scomparire.

Tra la miriade di modelli e approcci che rientrano nella definizione generale di "machine learning", l'anomaly detection assume una posizione di spicco per quanto riguarda il potenziale miglioramento che può apportare nel modo di lavorare dell'azienda, sia in termini di efficienza che di scalabilità.

Con il termine "anomaly detection" si intende l'insieme di tutti quegli approcci, modelli e tecniche che permettono di rilevare o prevedere la presenza di elementi, all'interno di un insieme di dati, che si discostano dal comportamento che ci si aspetta dato il contesto nel quale i dati stessi sono stati raccolti.

Nel caso di questa tesi, l'obiettivo è stato quello di studiare ed implementare dei modelli non supervisionati che permettano di rilevare possibili anomalie presenti in un dataset di metriche di performance relative ad un application server aziendale.

Lo scopo di quest'attività è fornire all'azienda uno studio di fattibilità per un tool di alerting che in futuro potrà essere in grado di supportare i consulenti che si occupano della manutenzione dei server, rendendo il loro intervento più rapido ed efficace, e di conseguenza minimizzando i tempi di inoperatività della piattaforma, garantendo un notevole risparmio per il cliente.

1.2 Presentazione azienda

L'attività di tesi si è svolta presso l'azienda Mediamente Consulting s.r.l, una società di consulenza informatica specializzata nell'ambito della business analytics avanzata. L'azienda si occupa dell'analisi e gestione del dato in tutti i suoi aspetti:

- progettazione infrastrutturale e sistemistica: consiste nel progettare e mantenere la piattaforma di raccolta dati del cliente.
- data integration: consiste nella pulizia, l'elaborazione e l'integrazione dei dati raccolti.
- business intelligence e data visualization: consiste nell'estrazione di informazioni utili dai dati aziendali per il supporto alle decisioni strategiche del cliente e nella presentazione di tali risultati in maniera efficace e fruibile. Quest'ultima fase si avvale di modelli di data science e machine learning, focus principale dell'attività di tesi.

L'azienda è nata nel 2013 come startup presso l'incubatore di imprese del Politecnico di Torino e conta oggi una cinquantina di dipendenti.

I clienti tipici sono aziende di medie e grandi dimensioni di settori variegati, spaziando dal Retail, E-Commerce, Food and Beverage, Manufacturing, Fashion, Banking Insurrance, Healthcare e Telco.

Attualmente l'azienda è fortemente specializzata nell'installazione e manutenzione di prodotti Oracle, dal diffusissimo Oracle database a prodotti meno conosciuti ma comunque molto utilizzati come l'application server weblogic, le cui metriche sono state oggetto di analisi nel progetto di tesi.

L'attività di tesi è stata svolta principalmente a cavallo tra le business unit di infrastrutture e business intelligence in quanto conoscenze da entrambe sono state necessarie per affrontare il problema.

In generale, nonostante la necessaria divisione burocratica nelle varie business unit, l'azienda supporta e incoraggia la collaborazione e lo scambio di conoscenze ed esperienza tra business unit diverse, fatto che contribuisce, assieme alla professionalità dei dipendenti, a rendere l'esperienza di tesi aziendale estremamente formativa.

1.3 Flusso di lavoro

La realizzazione del progetto di tesi ha richiesto diverse fasi, sia sequenziali che parallele, mostrate nella figura 1.1.

La prima fase è stata caratterizzata da uno studio degli strumenti e delle piattaforme utilizzate nell'azienda, in particolare l'application server weblogic e il database Oracle, che costituiscono la base infrastrutturale da cui verranno rispettivamente prelevati e immagazzinati i dati da analizzare.
Una rapida panormaica sugli aspetti tecnici degli strumenti utilizzati verrà esposta nel capitolo 2.

Parallelamentte a questa fase si è svolta una ricerca riguardante lo stato dell'arte degli algorimi di anomaly detection attualmente esistenti, specie quelli relativi alle serie temporali in modo da poterli utilizzare come base per sviluppare un modello specifico per le metriche del server weblogic. Un'analisi dello stato dell'arte si può trovare nel capitolo 3.

Successivamente si passa alla fase di raccolta e pulizia dei dati, fondamentale per la costruzione di un dataset solido per permettere lo studio e la costruzione dei modelli. Una descrizione del dataset viene trattata nel capitolo 4.

Infine c'è stata la fase di sviluppo vero e proprio di nuovi modelli per risolvere il problema in questione. Quest'ultima fase è stata in realtà "ciclica", ovvero i modelli proposti sono stati parametrizzati e adattati il più possibili al dataset preso in esame e successivamente confrontati tra di loro. Alla fine solo i modelli più solidi sono stati mantenuti.Quest'ultima fase viene trattata nel capitolo 5.

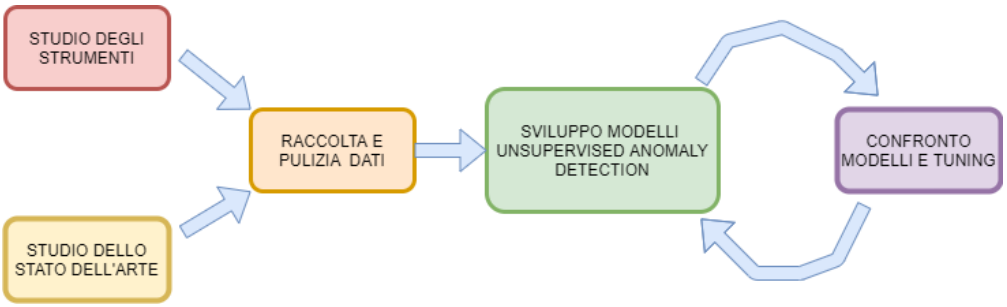

Figura 1.1: Flusso di lavoro generale

Strumenti

In questo capitolo verranno descritti brevemente le piattaforme e gli strumenti utilizzati per la realizzazione del progetto di tesi.

2.1 Oracle Weblogic

Weblogic è una piattaforma software sviluppato da Oracle che permette il deployment su server web di business application basate su java.

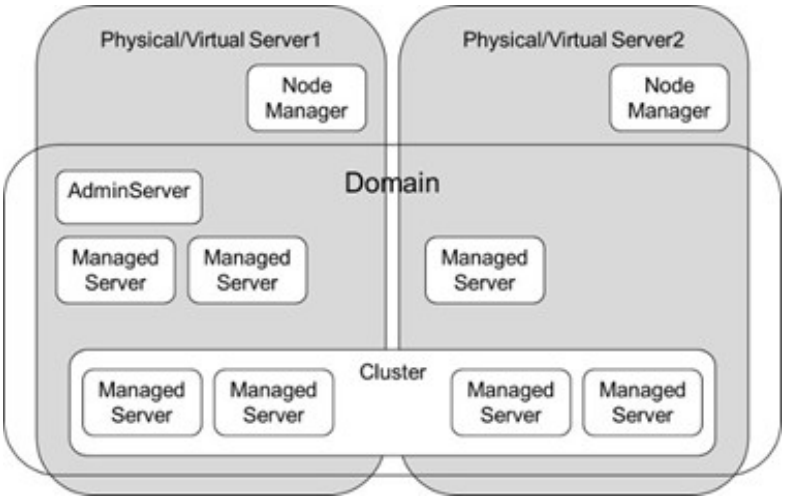

Figura 2.1: Struttura architettura weblogic [1]

Il suo scopo è quello di rendere trasparente e scalabile la comunicazione tra le applicazioni che implementano la logica di business e l'infrastruttura hardware e software necessaria per il loro funzionamento (ad esempio comunicazione con il database, servizi di messaggistica etc. . .).

Quando il client fa una richiesta ad un'applicazione web, tutta l'esecuzione viene gestita dal server, delegando l'elaborazione del servizio richiesto. Ad esempio se il client richiede di leggere dei dati da un database, weblogic renderà questo processo trasparente senza che l'applicazione debba conoscere cosa sia stato eseguito in background.

Il grande vantaggio di weblogic è che questo sistema è facilmente scalabile e quindi è adatto per gli ambienti di produzione, dove sono presenti molti client che effettuano centinaia di richieste al minuto ma vi è la necessità che quest'ultime vengano soddisfatte in tempi molto brevi senza problemi di performance.

Di seguito viene spiegata ad alto livello la struttura generica di una piattaforma weblogic rappresentata nella figura 2.2:

- Physical/Virtual server: weblogic è una piattaforma che può operare in modo distribuito, ovvero un'istanza del software può essere installata su più server ma essere gestita come una piattaforma unica.
- Node Manager: è il processo che si occupa della gestione di ogni istanza di weblogic, in particolare permette di avviare e stoppare un Managed Server remoto, monitorare lo stato di salute dei managed server e stopparli/riavviarli in caso di problemi.
- AdminServer: Processo che fornisce un unico punto centrale per la gestione dei managed server appartenenti ad un Domain.
- Domain: è un'entità logica che racchiude tutti i server contenenti le applicazioni relative ad un'unica piattaforma
- Managed Server: entità su cui vengono installate le applicazioni web e gestiscono altre risorse come le connessioni con il database.
- Cluster: consiste in un insieme di managed server che vengono eseguiti simultaneamente (sullo stesso server fisico o su server diversi) per fornire scalabilità e affidabilità. Un client vede un cluster come una singola istanza.

Il sistema operativo di riferimento per l'installazione di weblogic e degli altri prodotti Oracle è "Oracle linux", una distribuzione basata su red hat, prevalentemente diffuso in ambienti server aziendali. Di conseguenza la gestione della piattaforma si effettua principalmente attraverso la modifica di file di configurazione e l'esecuzione di shell scripts.

Weblogic diagnostic framework

il weblogic diagnostic framework (WLDF) è un insieme di tool messi a disposizione dalla piattaforma per poter raccogliere una serie di metriche di performance utili per effettuare analisi molto approfondite del suo funzionamento e poter risalire alla causa di vari problemi tecnici che possono presentarsi.

Il numero di metriche diverse che si possono raccogliere è molto elevato, ed è variabile a seconda dei moduli che sono attivi su uno speifico server, il numero di managed server e le applicazioni installate su ognuno di essi. Per poterle analizzare è stata per cui fatta una cernita escludendo tutte le metriche non valorizzate (ovvero che assumono sempre un valore costante nell'intervallo di tempo preso in esame) e quelle che non portano beneficio nell'ottica dell'anomaly detection, come ad esempio la metrica "Uptime" che semplicemente tiene traccia del numero di secondi trascorsi dall'avvio del server.

Le metriche nel WLDF sono organizzate in MBean. Un MBean è una classe Java che descrive una particolare categoria di metriche. Ogni MBean può essere istanziato più volte a seconda dei componenti attivi nel domain, ad esempio uno stesso MBean può essere attivato una volta per ogni server.
Inoltre è possibile selezionare una frequenza di campionamento per ogni metrica, quella di default è di un minuto.

Weblogic permette di archiviare queste metriche; sia sotto forma di file di testo, sia sotto forma tabellare in un database. La scelta è ovviamente ricaduta su quest'ultima possibilità in quanto permette un accesso ai dati molto più flessibile ed efficace. Per fare ciò è stato necessario creare un JDBCConnector all'interno del server weblogic per potersi collegare con un db Oracle sul quale storicizzare queste metriche per poterle analizzare.

Ovviamente i dati non saranno direttamente utilizzabili ma sarà necessario elaborarli e riformattarli in modo che per ogni istante di tempo in cui avviene il campionamento, sia associata la lista di tutti i valori. Questa operazione, che ha richiesto l'utilizzo di viste sql e indici verrà spiegata in maniera approfondita nel capitolo 4 relativo alla raccolta e pulizia dati.

2.2 Oracle Database

Il database Oracle è una delle piattaforme RDBMS (relational database management system) più diffuse e consolidate al mondo. La caratteristica principale degli RDBMS è che organizzano i dati in strutture tabellari, ognuna corrispondente ad un'entità logica che si vuole rappresentare, le entità vengono messe in relazione attraverso campi comuni tra tabelle diverse.

Spiegare in modo dettagliato le caratteristiche degli RDBMS e il funzionamento del database Oracle esula dall'obbiettivo di questa tesi, tuttavia nei seguenti paragrafi verranno elencati concetti tecnici principali riguardanti il funzionamento del database Oracle.

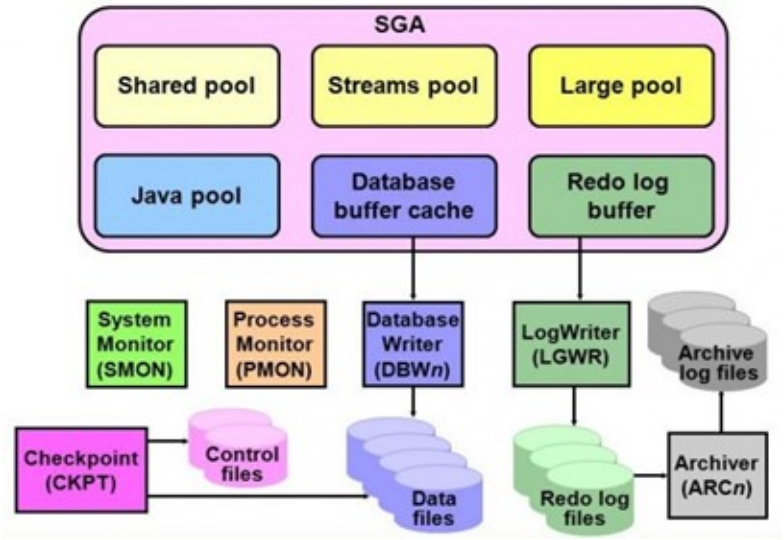

Figura 2.2: Struttura architettura Oracle Database [2]

Un'istanza di Database è un insieme di strutture in memoria che gestiscono file di database. Un database è un insieme di file su disco. Per poter accedere e modificare il file di database l'istanza deve essere associata al database fisico.

Nel momento in cui un'istanza di database viene creata viene istanziata una zona di memoria chiamata System Global Area (SGA) in cui verranno allocate tutte le strutture necessarie per la gestione dei file di database, viene anche creata la PGA che conterrà invece i processi in background che verranno analizzati in seguito. Inoltre è possibile fare in modo che un'unica istanza gestisca file di database diversi.

File di database

Sono i file fisici che costituiscono il database, ne esistono vari tipi:

- Control files: contengono metadati riguardanti la struttura del database, sono fondamentali perché senza di essi i processi non saprebbero come accedere ai dati veri e propri
- Data file: contengono fisicamente i dati delle tabelle del DB
- Online redo log files: permettono di effettuare l'instance recovery del database in caso di crash improvviso dell'istanza (anche chiamato soft reset)
- Archive log files: simili all'online redo log ma per recuperare i dati persi in caso di un crash dell'intero sistema che rischia di corrompere i data files
- Parameter file: definiscono come deve essere configurata l'istanza quando viene avviata
- Trace file: contiene un dump degli errori

Processi

- Database Writer Process (DBWn): scrive il contenuto dei buffer nei data file. Nei buffer sono presenti tutti i record che sono stati modificati ma ancora non scritti in un data file permanentemente (dirty bit=1)
- Log Writer Process (LGWR): si occupa di scrivere il redo log. Il redo log è un buffer circolare, nel momento in cui il processo scrive una entry dal buffer al redo log file, nuove entry possono essere inserite nel buffer. Solitamente questo processo è abbastanza veloce in modo da far si che ci sia sempre sufficiente spazio nel redo log buffer
- Checkpoint Process (CKPT): i check point corrispondo a istanti di tempo in cui si ha la certezza che tutte le transazioni committate fino a quel punto sono scritte in modo permanente sui data files, nel momento in cui si raggiunge un checkpoint, Oracle deve aggiornare gli header di tutti i data files, quest'operazione viene effettuata dal checkpoint process (nota: il processo CKPT non scrive mai direttamente sui data files, quest'operazione viene effettuata solo dal processo DBWn)
- System Monitor Process (SMON): si occupa di effettuare la recover quando necessario (ad esempio durante lo startup) ed è responsabile della pulizia dei segmenti temporanei che non vengono più utilizzati e di unire i segmenti contigui nei tablespaces

- Process Monitor Process (PMON): simile all'SMON ma effettua la recovery nel momento in cui è un singolo processo a crashare ed è responsabile della pulizia della buffer cache e liberare le risorse che il processo stava usando. Ad esempio resetta lo status della "active transaction table", rilascia i lock e rimuove l'id del processo dalla lista dei processi attivi

Table space e data files

Ogni database è suddiviso in strutture logiche chiamate tablespaces, ogni database è logicamente suddiviso in uno o più tablespace, di conseguenza per ogni tablespace esisterà almeno un datafile. Il tablespace è diviso in segmenti.

Gli oggetti che costituiscono un database, come tabelle e indici, vengono memorizzate all'interno di un segmento del tablespace. Ogni segmento contiene uno o più extents che consiste in un blocco dati contiguo (l'unità minima accessibile dal DB) e di conseguenza un extent può esistere in un solo data file.

Quando il database richiede dei data block al sistema operativo, quest'ultimo si occupa del mapping effettivo con il file system e il blocco su disco. La dimensione del data block può essere impostata in fase di creazione del DB ma il valore di default (8KB) è solitamente adatto alla maggior parte dei database. Dimensioni più grandi sono utili nel caso vengano effettuate letture di molti blocchi assieme come nel caso di data wharehouse, mentre nel caso di un database relazionale che effettua molte piccole transazioni è meglio specificare una block size più piccola per non leggere dati "inutili" ad ogni transazione.

2.3 Google Colab

Per la parte di sviluppo dei modelli è stato deciso di utilizzare il linguaggio di programmazione python dato che la maggior parte delle librerie che facilitano il lavoro dell'analisi dati sono disponibili in questo linguaggio. Google Colab è una piattaforma online che si pone lo scopo di fornire un ambiente in grado di eseguire codice python in modo rapido e interattivo senza tutte le complicazioni e problematiche derivanti dal dover configurare un ambiente di sviluppo in locale.

Un altro vantaggio della piattaforma è che permette di eseguire il proprio codice su delle CPU e GPU remote gestite da Google. Questo è l'ideale nel caso si vogliano studiare dei modelli di machine learning creando dei prototipi da testare in poco tempo, in quanto lo sviluppatore non deve preoccuparsi nè di avere a disposizione dell'hardware performante nè di perdere tempo in configurazioni complicate per poter parallelizzare l'esecuzione di questi modelli.

Inoltre su google colab è molto semplice gestire la visualizzazione di grafici e di dati in quanto non è necessario impostare una applicazione che deve funzionare nel sistema operativo (con tutte le dipendenze del caso), ma la stampa viene fatta direttamente nella pagina web. A questa immediatezza si aggiunge la possibilità di eseguire nuovo codice senza dover ricompilare e rieseguire tutto il codice passato. Questo è molto utile nel contesto del machine learning perchè permette di separare la fase di training dei modelli, che solitamente richiede molto tempo, dalla fase di visualizzazione dei dati che invece può richiedere molte variazioni e modifiche (anche banali come un semplice cambio di colore) e che in un ambiente convenzionale richiederebbero di rieseguire l'intera applicazione o necessiterebbero di complessità extra per trovare dei workaround. L'elenco delle librerie utilizzate è il seguente:

- numpy: permette una gestione più efficace di matrici e array ed è di fonadmentale importanza per le altre librerie utilizzate
- pandas: permette la gestione di dataset e fornisce una serie di funzioni molto utili per la data exploration come il calcolo della media, la varianza, valori minimi e massimi etc..
- pandas-profiling: permette di generare automaticamente report di data exploration
- sklearn: implementa al suo interno moltissimi modelli e algortimi legati al machine learning già studiati e consolidati
- matplotlib: libreria che permette di generare grafici e plot

Stato dell'Arte

In questo capitolo verranno esposti i principali concetti teorici dietro le serie temporali e la loro scomposizione, nonchè i principali algoritmi esistenti in grado di operare su questo tipo di dati, molti dei concetti esposti in in questo capitolo sono spiegati nel libro di Rob J Hyndman and George Athanasopoulos "Forecasting: Principles and Practice" [3].

3.1 Serie temporali

Le serie temporali sono un insieme di dati campionati ad intervalli di tempo regolari. Esse vengono utilizzate per rappresentare l'andamento di particolari fenomeni nel tempo (come riferimento si veda [4]).
Nel mondo attuale le serie temporali costituiscono una grossa porzione dei dati disponibili alle aziende, quindi riuscire ad analizzarle e comprenderle è di estrema importanza, sia per studiare la storia passata di un particolare fenomeno, sia per prevedere con una certa accuratezza cosa ci si può aspettare dal futuro.

Nel caso di questa tesi le serie temporali sono il formato dati principale su cui i vari modelli e algoritmi analizzati andranno ad operare, In particolare il dataset utilizzato sarà formato da una serie di metriche di performance generate dalla piattaforma weblogic server, ognuna di esse campionata una volta al minuto.

L'obiettivo è quello di analizzare tali serie temporali e, applicando appositi algoritmi di data analysis e machine learning, riuscire ad individuare eventuali istanti (o intervalli) di tempo in cui le metriche assumono valori anomali, possibile indicatore di problemi nell'infrastruttura, e conseguentemente prevedere il verificarsi di tali anomalie con un certo anticipo.

Nella figura 3.1 si può osservare una serie temporale relativa alla metrica *OpenSocketsCurrentCount* che mostra la variazione del tempo del numero di socket aperti

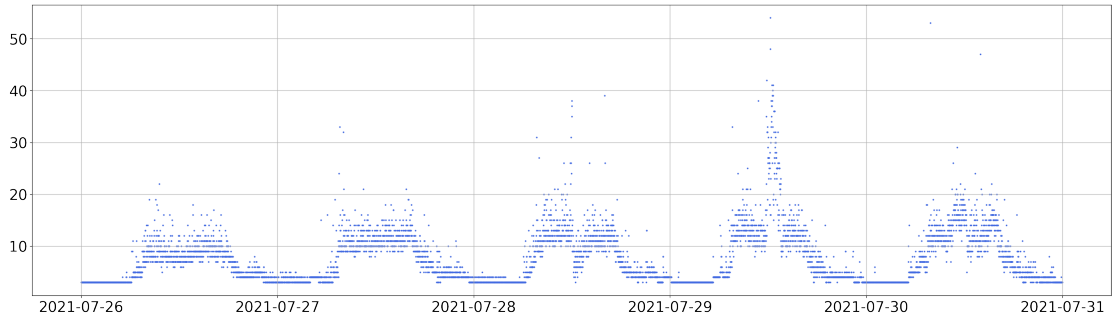

Figura 3.1: Serie temporale relativa alla metrica *OpenSocketsCurrentCount*

sul server weblogic.

Ogni serie temporale è scomponibile in tre elementi:

- Trend: componente del segnale che rappresenta la tendenza dello stesso ad incrementare o decrementare nel tempo, anche in modo non lineare e non periodico, nel caso dell'anomaly detection il trend permette di avere una sorta di baseline rispetto alla quale confrontare ogni punto della serie.
- Stagionalità: componente del segnale dovuta alla ripetizione ad intervalli di tempo costanti di particolari eventi, ad esempio con cadenza mensile o settimanale. La stagionalità va presa in considerazione quando si analizzano le anomalie in quanto i valori del segnale considerati anomali in un particolare periodo di tempo potrebbero non esserlo in un altro. Ad esempio un picco di una particolare metrica potrebbe essere normale nei giorni lavorativi perché dovuto al normale carico del sistema nel suo funzionamento per il business, mentre lo stesso picco potrebbe essere indice di un problema nel weekend quando pochi impiegati e clienti lo stanno utilizzando.
- Residuo o componente randomica: componente della serie che non ricade in nessuno dei due segnali precedenti ed è dovuto ad oscillazioni randomiche del segnale o eventi imprevedibili

Altri due concetti importanti relativi alle serie temporali sono:

- Stazionarietà: una serie temporale si dice stazionaria se le sue proprietà non dipendono dall'intervallo di tempo in cui viene osservata, ovvero sia la media che la varianza del segnale rimangono costanti per qualunque intervallo di tempo osservato. Alcuni modelli di forecast, come ARIMA, richiedono che una serie temporale sia stazionaria per poter funzionare.
- Ciclicità: concetto simile alla stagionalità ma con frequenza non fissa, ovvero consiste un pattern che si ripete ad intervalli non periodici.

3.2 STL Deomposition

l'STL Decomposition (seasonal and trend decomposition using loess) [5] è una tecnica che permette di scomporre la serie temporale nei suoi 3 componenti: il trend, la stagionalità e il residuo.

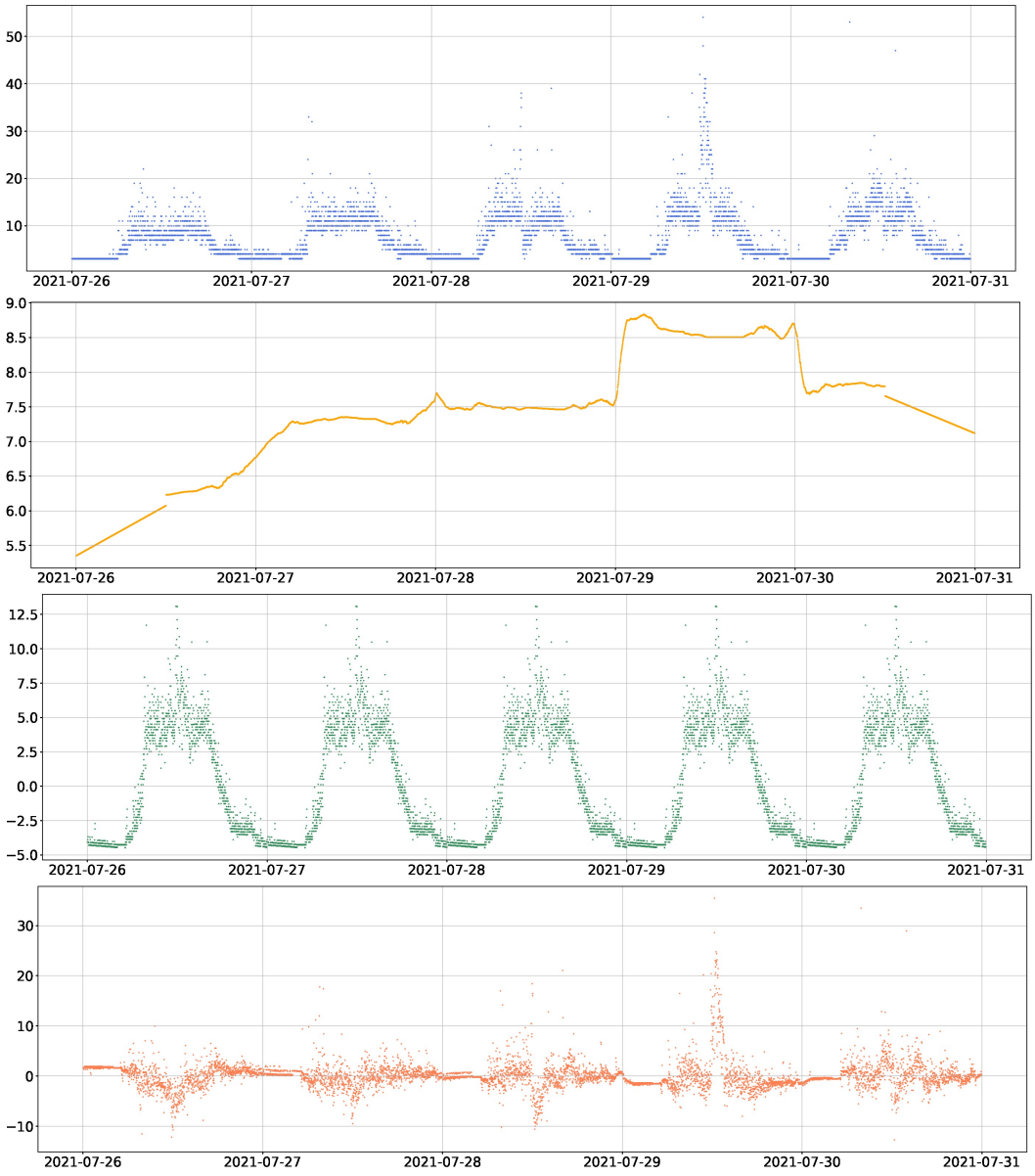

Figura 3.2: STL decomposition , in alto il segnale originale, successivamente in ordine il trend, la stagionalità e il residuo

Dall'immagine 3.2 è possibile osservare un fatto interessante, ovvero che in generale la componente randomica ha media nulla e varianza limitata, per cui rientra nella categoria di segnale stazionario. Questo non è vero per l'intera durata del segnale, infatti possiamo osservare un picco di valori tra il 29-07 e il 30-07. Questo può essere un'indicazione che nell'intervallo di tempo sia presente un'anomalia.

A riprova di ciò si può constatare che le anomalie, se presenti, dovrebbero essere "contenute" nel residuo, in quanto per definizione non possono essere parte né del trend né della stagionalità dato che queste due componenti assieme descrivono il comportamento che ci si aspetta dal segnale.

Il segnale originale può essere scomposto in due modi:

- Additive decomposition: Il segnale (M) è dato dalla somma delle 3 componenti della decomposizione, ovvero il trend (T) la stagionalità (S) e il residuo(R) $M = T + S + R$

- Multiplicative decomposition: Il segnale (M) è dato dal prodotto delle 3 componenti della decomposizione, ovvero il trend (T) la stagionalità (S) e il residuo(R) $M = T * S * R$

In generale l'additive decomposition è la più utilizzata delle due.

STL si basa su sulla Loess (Local regression) smoothing [6]; ovvero una tecnica che permette di trovare l'approssimazione $g(x)$ di una funzione $f(x)$, come una serie temporale, a partire da una serie di campioni della stessa. I passaggi per trovare l'approssimazione sono descritti nell'Algortimo 1.

Algorithm 1 LOESS Smoothing Algorithm: data una funzione ne trova una approssimazione

procedure LOESS(x, q)
x è un vettore contenente tutti i campioni della serie temporale
q è il numero di campioni da utilizzare per la local regression
for x[i] in x **do**
$v[0 : q/2] \leftarrow x[i - q/2, i]$
$v[q/2 : q - 1] \leftarrow x[i + 1, i + q/2]$
▷ Recupera i q punti più vicini a xi
for v[j] in v **do**
$v[j] \leftarrow (1 - |d[j]|^3)^3$
▷ d[j] è la distanza del punto v[j] da x[i]
▷ ogni punto viene pesato in base alla sua distanza da x[i]
end for
$L \leftarrow QuadraticRegressor(v)$
▷ Viene calcolato un regressore quadratico sui punti pesati
$g[i] \leftarrow L[q/2]$
end for
return g
▷ g è la serie temporale approssimata
end procedure

Per individuare le tre componenti del segnale, STL realizza due cicli. Uno esterno, in cui un peso viene assegnato ad ogni punto in base al valore del residui (Questo verrà utilizzato nella successiva loess smoothing regression e permette di ridurre o eliminare gli effetti degli outlier) e uno interno che aggiorna iterativamente la componente di trend e stagionalità per ogni punto.

Questo viene fatto sottraendo la stima corrente del trend dalla serie originale. La serie temporale è successivamente partizionata in "cycle-subseries", una per ogni periodo della stagionalità (ad esempio se la stagionalità è mensile e sono presenti un anno di dati la serie temporale verrà suddivisa in 12 cycle-subseries).

Ad ognuna delle cycle-subseries viene applicata la loess smoothing e successivamente un filtro passa basso per eliminare gli outlier. Infine la componente stagionale viene sottratta dai dati originali. Al risultato viene applicata nuovamente la loess smoothing ottenendo così la componente di trend. Ciò che rimane costituirà il residuo.

L'algoritmo 2 descrive con pseudocodice più dettagliato questa procedura.

Algorithm 2 STL Decomposition Algorithm: data una serie temporale ne individua il trend, la sagionalità e i residui

1.

procedure STL_DECOMPOSITION($Y(x), Np, Ni, No, Nl, Ns, Nt$)
▷ Y(x): serie temporale da scomporre
▷ Np: periodicità della seasonality
▷ Ni: numero di loop interno, deve essere abbastanza grande da convergere
▷ No: numero di loop esterno: più cicli riducono gli effetti degli outlier (di solito 1)
▷ Nl: larghezza della finestra per il filtro passa basso
▷ Ns: parametro di smoothing per il filtro di stagionalità
▷ Nt: parametro di smoothing per il filtro di trending
$T(x) \leftarrow 0$
▷ inizializza Trend
$R(x) \leftarrow 0$
▷ inizializza Residuo
for i=0;i<No;i++ **do**
$p(x) \leftarrow B(|R(x)|/(6 * median|R(x)|))$
▷ $B(u) = (1 - u^2)^2$ per 0<u<1 B(u)=0 per ogni altro valore di u
▷ calcola il peso per ogni valore del residuo, alla prima iterazione p(x)=0
for k=0;k<Ni;k++; **do**
$D(x) \leftarrow Y(x) - T(x)^k$
▷ Detrend della serie temporale
for j=0;j<Np;j++ **do**
$C(x)^j \leftarrow D(x)[j * Np : (j + 1) * Np]$
▷ Estraggo le cycle-subseries
$L(x)^j \leftarrow LowPassFilter(LOESS(C_tmp(x)^j, Ns), Nl)$
▷ Il filtro passa basso è realizzato attraverso 2 Moving avarage con lag=Nl e una successiva riapplicazione di LOESS
$S^j = C^j - L^j$ ▷ Detrend delle cycle subseries
end for
$S(x) \leftarrow S(x)^j\_ricompattati$
$T(x) \leftarrow LOESS(Y(x) - S(X), Nt)$
$R(x) \leftarrow Y(x) - T(x) - S(x)$

end for
end for
return T(x),S(x),R(x)
end procedure

3.3 ARIMA

Con ARIMA[7] si indica una classe di modelli per il forecasting di time series. L'acronimo ARIMA sta per autoregressive integrated moving average ed è l'unione di due modelli: AR (auto regressive) e MA (moving average).

Entrambi i modelli richiedono che la serie temporale sia stazionaria, ovvero che le sue proprietà non dipendano dall'istante di tempo in cui vengono osservate. Per cui le serie con trend o stagionalità (che sono quelle di utilità nel mondo reale) per definizione non sono stazionarie. Occorre quindi un metodo per rendere una serie temporale con componente trend e stagionalità non nulle stazionaria. l'acronimo "I" in "ARIMA" sta per Integrated, che si riferisce prorio alla procedura appena introdotta.

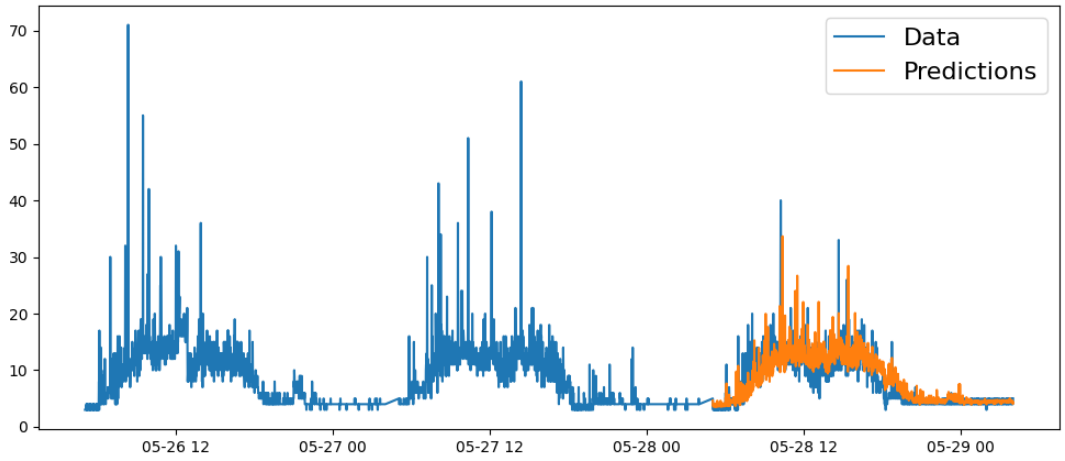

Figura 3.3: Esempio di forecast con arima sulla feature *OpenSocketsCurrentCount* vedi figura 3.1

Per rendere una serie temporale stazionaria viene usata una semplice tecnica chiamata "differencing" ovvero sostituire ogni punto con la differenza tra il punto stesso e quello precedente. Questa operazione ha la conseguenza di rendere costante la media della nuova serie e allo stesso tempo limitata la sua varianza e conseguentemente eliminare trend e stagionalità del segnale originale.

Nel caso venga fatta la differenza con il termine immediatamente precedente nella serie si parla di "first order differencing". Talvolta una differenza del primo

ordine non è sufficiente e occorre differenziare i dati una seconda volta per ottenere una serie stazionaria attraverso una "second order differencing"

Teoricamente si potrebbe avere un differencing di qualunque ordine ma nella pratica è estremamente raro dover ricorrere ad un ordine più grande del secondo. Un altro tipo di differencing è il cosiddetto "seasonal differencing" il quale, come si può intuire dal nome" prende in considerazione serie con una componente stagionale e consiste nell'assegnare ad ogni punto la differenza con il punto corrispondente del periodo precedente.

Per valutare l'accuratezza di un modello di forecasting esistono diversi metodi, uno dei più utilizzabili è il Mean absolute percent error (MAPE) score.
Come suggerito dal nome, Il MAPE score si può ottenere facendo la differenza in valore assoluto punto punto tra la serie temporale originale, e il forecast del modello, rapportandola con il vaolre della serie di partenza. Infine viene fatta la media di tutti i valori puntuali.

$$MAPE = \frac{1}{n} \sum_{i=0}^{n} \frac{|Forecast_i - Original_i|}{Original_i}$$

dove n è il numero totale di campioni nella serie temporale di cui si è fatto il forecast.

Un forecast è tanto più accurato quanto più il MAPE score si avvicina a zero. Nonostante il MAPE score sia molto facilmente interpretabile, presenta criticità nel momento in cui la serie originale assume un valore prossimo allo zero.

In questa situazione si ha che il punteggio tende ad infinito a prescindere dall'accuratezza del forecast per via della divisione per un valore nullo. Per ovviare a questo problema spesso si effettua un'offset della serie temporale (prima del training del modello) in modo che il valore minimo sia maggiore di zero.

3.3.1 modello AR

Come già accennato, ARIMA è dato dall'insieme di un autoregressive model e un moving average model. L'autoregressive model permette di effettuare una previsione dei futuri valori della serie (forecast) utilizzando una combinazione lineare dei valori in input (per questo si chiama auto-regression, ovvero regressione della serie con sé stessa).

Una serie temporale può quindi essere rappresentata come :

$$y = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + +\phi_p y_{t-p} + \epsilon_t$$

In questa formula ϵ_t è un termine di errore dovuto all'approssimazione della regressione, considerato rumore bianco, ovvero una serie stazionaria con media nulla. I termini $y_{t-1}y_{t-2}...y_{t--p}$ corrispondono ai p valori passati della serie, quelli su cui viene calcolata la regressione. in questo caso si parla di modello AR di ordine p. I coefficienti di questi termini vengono calcolati attraverso un algoritmo di regressione lineare.

La regressione lineare è il modello di regressione più semplice, data una serie di punti l'output del modello saranno i coefficienti dell'iperpiano (ovvero i ϕ del modello AR) che meglio approssimano tutti i punti forniti; questo si traduce nel minimizzare una certa funzione di costo che nel caso della regressione lineare è la somma degli errori quadratici medi:

$$MSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\Big(d_i - f_i\Big)^2}$$

in cui n è il numero di punti e $d_i - f_i$ è la differenza tra il punto assegnato dal modello e il punto reale della serie in un certo istante di tempo.

3.3.2 modello MA

Il modello moving average à simile al modello auto regressive, entrambi si basano su una regressione lineare, il modello MA utilizza come input non i valori passati della serie ma gli errori di forecast nei precedenti step. La formulazione è la seguente:

$y = c + \theta_1 y_{t-1} + \theta_2 y_{t-2} + ... + +\theta_p y_{t-p} + \epsilon_t$

È possibile dimostrare che qualunque modello AR(p) è equivalente a un modello MA(∞):

$y = \phi_1 y_{t-1} + \epsilon_t$
$= \phi_1(\theta_1 y_{t-2} + \epsilon_{t-1}) + \epsilon_t$
$= \phi_1^2 y_{t-2} + \phi_1 \epsilon_{t-1} + \epsilon_t$
$= \phi_1^3 y_{t-3} + \phi_1^2 \epsilon_{t-2} + +\phi_1 \epsilon_{t-1} \epsilon_t$
...

Se il valore assoluto di ϕ è minore di 1 ϕ_∞ tenderà a zero e rimarrà soltanto:

$y_t = \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_{t-2}^2 + \phi^3 \epsilon t - 3 + ...$

che è la formulazione di un modello MA(∞)

Inoltre se è possibile dimostrare anche la relazione opposta, ovvero che un modello MA(p) può essere scritto come AR(∞), il modello viene chiamato invertibile; questo introduce una serie di proprietà matematiche rilevanti.

Ad esempio, se si prende in considerazione un modello MA(1): $y = \epsilon_t + \theta_1 \epsilon_{t-1} \infty$ l'ultimo errore effettuato utilizzando la rappresentazione AE(∞))sarà :

$\epsilon_t = \Sigma_{j=0}^{n} \Big(- \theta^j y_{t-j} \Big)$

Quando $|\theta| > 1$ le osservazioni più distanti avranno un'importanza più rilevanti, mentre quando $|\theta| = 1$ i pesi sono costanti. Entrambe queste condizioni sono indesiderabili in quanto le osservazioni più recenti dovrebbero essere le più rilevanti Questo corrisponde alla condizione $|\theta| < 1$, necessaria per determinare se un modello è invertibile.

Nella prossima sezione verrà descritto una metodologia per testare questa condizione.

3.3.3 Unit roots

Per verificare se una serie temporale è stazionaria esistono una serie di test statistici chiamati "unit root test".
Impotizzando di avere un modello AR(1):
$y_t = \varphi y_{t-1} + \epsilon_t$
e ricordando che un modello AR(1) può essere scritto come un modello MA(∞)
$y_t = \varphi_t y_0 + \Sigma_{k=0}^{t-1} \varphi_t \epsilon_{t-k}$

si può calcolare la media del modello come:
$E(y_t) = \varphi E(y_{t-1} = \varphi^2 E(y_{t-2}) = \varphi^3 E(y_{t-3}) = \varphi^t y_0$

e la varianza come:
$Var(y_t) = \sigma^2(\varphi^0 + \varphi^2 + \varphi^4 + ...\varphi^{2(t-1)})$

Si hanno per cui 3 casi distinti nel momento in cui t tende a infinito:

- $|\varphi| < 1, E(t_o) \to 0, Var(t_0) \to \frac{\sigma^2}{(1-\varphi^2)}$ soddisfa le condizioni di stazionarietà
- $|\varphi| > 1, E(t_o) \to \infty$ non soddisfa le condizioni di stazionarietà
- $|\varphi| < 1, E(t_o) \to 0, Var(t_0) \to t\sigma^2$ non soddisfa le condizioni di stazionarietà

In una serie temporale si ha una unit root quando $|\varphi| = 1$.

Per testare la presenza di unit root, e quindi di non stazionarietà, esistono diversi test statistici. Uno dei più utilizzati è il KPSS (Kwiatkowski-Phillips-Schmidt-Shin). Questo test restituisce un p-value calcolato sulla time series, se il p-value è minore di un valore soglia, solitamente 0.05, si può affermare con una certa sicurezza che la serie non è stazionaria.

il p-value è un punteggio di probabilità usato in statistica per determinare la rilevanza di un evento osservato, nel caso del KPSS viene testata la null- hypothesis del problema della stazionarietà (ovvero che non sia presente la stazionarietà)

3.3.4 modello ARIMA

combinando i modelli appena presentati si ottiene una rappresentazione di questo tipo:

$y_t = c +_{t-1} +... + \varphi_p y_{t-p} + \theta\epsilon_{t-1} + ... + \theta_p\epsilon_{t-q} + \epsilon_t$

Indicata anche come ARIMA(p,d,q) dove p è l'ordine dell'autoregressione, d il differencing order e q l'ordine della moving average.

La scelta dei parametri p,d,q è di estrema importanza per ottenere un buon modello di forecasting, sebbene esistano dei metodi per trovarli automaticamente, analizzarli è molto utile per capire il comportamento del modello.

Esiste una relazione tra la costante c presente nel modello e il parametro di differencing d

- se c=0 e d=0, il forecast tende a zero
- se c=0 e d=1 il forecast tende a una costante diversa da 0
- se c=0 e d=2 il forecast tende ad una retta
- se c≠0 e d=0 il forecast converge verso la media dei dati
- se c≠0 e d=1 il forecast tende ad una retta
- se c≠0 e d=2 il forecast sequirà un trend quadratico

Il valore di d influenza anche gli intervalli di confidenza, più alto il valore di d più gli intervalli di confidenza crescono, se d=0 la deviazione standard tenderà ad essere uguale alla deviazione standard della serie originale.

Determinare i valori corretti di q e p non è immediato, per stimarli è molto utile calcolare i grafici di autocorrelation(ACF) e partial autocorrelation (PACF).

Il diagramma di autocorrelazione misura la correlazione di ogni punto della serie temporale con tutti i punti precedenti. Con correlazione si intende l'indice di Pearson:

$\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y}$

L'indice di correlazione di Pearson associa ad ogni variabile (nel caso delle serie temporali i punti a diversi istanti di tempo) un valore compreso tra 0 e 1.

I valori vicini ad uno indicano un'alta correlazione, quelli vicino a zero una correlazione inversa mentre valori di correlazione nell'intorno di 0.5 indicano l'assenza di alcuna correlazione. Con correlazione si intende la correlazione lineare, ovvero se una delle due variabili raddoppia anche l'altra subisce lo stesso effetto.

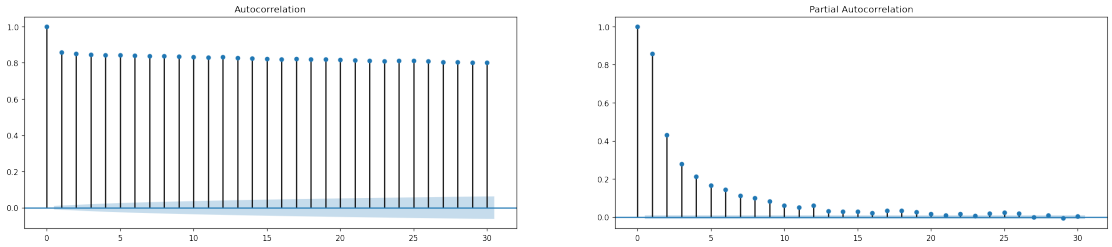

Figura 3.4: Grafici di autocorrelation (sinistra) e partial autocorrelation (destra) per la feature *OpenSocketsCurrentCount* vedi figura 3.1. Nota: Nei grafici la curve blu indicano che tutto ciò che sta al di sotto di tali curve non è statisticamente rilevante.

L'ACF considera le correlazioni in sequenza e non direttamente le correlazioni tra istanti distanti tra di loro. Ad esempio y_t e y_{t_1} sono correlati e y_{t-1} e y_{t-2} allora esisterà una correlazione anche tra y_t e y_{t-2} per "transizione" nonostante y_{t-1} possa non avere nessun tipo di correlazione diretta con y_t.

Per ovviare a questo problema esiste la partial autocorrelation (PACF) che opera in maniera molto simile alla ACF ma considera le correlazioi tra punti distanti senza la sequenza di punti intermedi.
Come già accennato è possibile stimare p e q dai grafici ACF e PACF.

L'ordine di autoregressione può essere impostato a p se valgono contemporanemente:

- L' ACF decade esponenzialmente oppure è sinusoidale
- È possibile osservare un picco significativo al lag p nel PACF, ma nessuno oltre il lag p

L'ordine di moving average può essere impostato a q se valgono contemporanemante:

- il PACF decade esponenzialmente oppure è sinusoidale
- È possibile osservare un picco significativo al lag q nel PACF, ma nessuno oltre il lag q

3.3.5 Auto ARIMA

Quelle appena descritte sono linee guida generali, la scelta dei parametri del modello ARIMA deve sempre essere fatta con cautela perchè ogni serie temporale è un caso a sè. Fortunatamente in python esiste una libreria chiamata "auto.arima" che permette in modo automatico di trovare la migliore configurazione di parametri per un modello ARIMA usando l'algoritmo Hyndman-Khandakar. Questo algoritmo si basa sull'utilizzo di unit root tests e minimizzazione dell'AIC (Akaike information criterion).

L'AIC è un estimatore di quanto il modello si adatti ai dati di training. La formula per calcolarlo è la seguente:

$AIC = sk - 2ln(L)$

dove k è il numero di parametri mentre L è la maximum likelihood estimation(MLE) del modello.

Se il numero di parametri aumenta anche la MLE aumenta ma al costo di overfittare sui dati di training, di conseguenza il valore dell'Aic aumenta a sua volta (il contributo dell'aumento della MLE non sarà rilevante per via del logaritmo). Viceversa nel caso di una riduzione di parametri, se k diventa troppo basso la MLE diventerà molto bassa, il che si si tradurrà in valori molto negativi del logaritmo e quindi nuovamente un valore per l'Aic elevato.

Un modello per cui è tanto migliore quanto l'equilibrio tra queste due componenti è mantenuto, il che si riflette in un valore dell'Aic più vicino allo zero possibile; di conseguenza confrontando modelli diversi, il più adatto è quello che più rispetta questa condizione.

Tornando all' algoritmo Hyndman-Khandakar i passi generali che vengono seguiti sono i seguenti:

1. Viene scelto l'ordine di differencing d tramite KPSS tests ripetuti. In ogni caso 0<=d<=2
2. I valori di p e q sono scelti minimizzando il valore dell'AIC dopo aver differenziato d volte. Non viene considerata ogni possibile combinazione ma vengono generati quattro modelli iniziali:
   - ARIMA(0,d,0)
   - ARIMA(2,d,2)
   - ARIMA(1,d,0)
   - ARIMA(0,d,1)
   - Tutti questi modelli tengono in considerazione una costante, se d=2 viene trainato un ulteriore modello ARIMA(0,d,0) senza costante
3. Tra i modelli scelti nel punto 2 viene scelto quello con l'AIC minore
4. Vengono trainati altri modelli variando p e q (aggiungendo +1/-1) partendo da quello scelto e includendo/escludendo la costante
5. Si ritorna allo step 2 finchè non si trova un modello con l'AIC minore

Auto Arima è una libreria molto utile per trovare un modello adatto ma non esplora tutte le soluzioni possibili e talvolta non sceglie i parametri migliori in assoluto. Per questo motivo è utile capire come selezionare un modello manualmente.

Questo procedimento ovviamente non è del tutto meccanico ma necessita anche dell'esperienza del data scientist. Esistono però delle linee guida da seguire per la selezione.

1. Visualizzare i dati in un grafico per identificare eventuali anomalie
2. Se necessario trasformare i dati per stabilizzare la varianza
3. Se i dati sono non stazionari, applicare il differencing finché non lo diventano
4. Esaminare i plot ACF e PACF e selezionare p e q come descritto in precedenza
5. Partire da questo modello e variare i parametri cercando di minimizzare l'AIC
6. Visualizzare i residui, dovrebbero avere una distribuzione di rumore bianco, se non lo sono occorre trainare un nuovo modello.

3.3.6 SARIMA

Il modello fin'ora descritto non tiene conto in alcun modo della stagionalità. Esiste però un'estensione di arima che può essere applicata nel caso di dati stagionali come quelli analizzati in questa tesi. Questo modello prende il nome di SARIMA , ovvero "Seasonal ARIMA" e prevede 4 parametri aggiuntivi per gestire la stagionalità:

- m: numero di campioni in un singolo periodo
- P: seasonal autoregressive order
- D: seasonal differencing order
- Q:seasonal moving average order

P,D e Q hanno lo stesso significato di p,d,q del modello ARIMA tradizionale ma vengono applicati sui periodi invece di singoli lag.

Come in precedenza, la selezione dei parametri P,D,Q e m può essere effettuata osservando i grafici ACF e PACF. Ad esempio un modello ARIMA(0,0,0)(0,0,1)12 presenterà:

- Un picco a lag 12 nel grafico ACF ma nessun'altro picco significatico
- Decadimento esponenziale nei lag stagionali(ex 12,24,36...) nel grafico PACF

Mentre un modello ARIMA(0,0,0)(1,0,0)12 presenterà:

- Decadimento esponenziale nei lag stagionali(ex 12,24,36...) nel grafico ACF
- Un picco a lag 12 nel grafico PACF ma nessun'altro picco significativo

SARIMA ha purtroppo lo svantaggio che il tempo di esecuzione cresce di molto per via del fatto che tutte le operazioni fatte per un punto vengono ripetute anche per tutti i periodi. Risulta per cui complicato testarlo e applicarlo in situazioni in cui si hanno molti campioni.

3.4 Prophet

Prophet è un modello di time series forecasting sviluppato da due ricercatori di facebook Taylor SJ e Letham B. [8] come progetto open source per rendere più semplice l'analisi e il forecasting di serie temporali, anche ai non esperti di questa disciplina.

Tra i punti di forza di prophet ci sono la robustezza del modello nel caso di dati mancanti, di frequenti cambiamenti di trend e presenza di outlier (questo ovviamente è il caso più interessante nell'ambito di questa tesi).

Il modo in cui Prophet rappresenta la serie temporale è molto simile a quello spiegato nella STL decomposition, in particolare il segnale viene rappresentato come la somma di tre componenti: trend, stagionalità e "holidays":

$y(t) = g(t) + s(t) + h(t) + e(t)$

- g(t): trend, modella i cambiamenti non periodici, può essere logaritmico

- s(t): stagionalità , modella i cambiamenti che si ripetono a intervalli regolari di tempo (mensile, settimanale, annuale), è inoltre possibile avere più di una stagionalità nella stessa serie

- h(t): holidays, modella eventi irregolari che modificano temporaneamente la serie temporale

- e(t): termine di errore, rappresenta i cambiamenti nella serie temporale che non vengono catturati dal modello, e(t) viene considerata come una distribuzione normale.

Essenzialmente prophet risolve il problema del forecasting come un problema di "curve-fitting", a differenza di ARIMA che considera le dipendenze di ogni istante di tempo con i precedenti tramite l'autocorrelation e la partial autocorrelation.

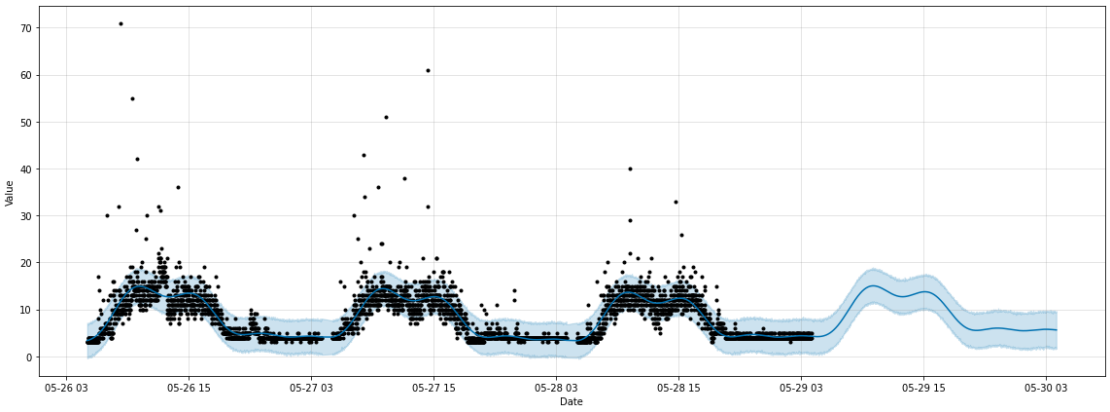

Figura 3.5: Prophet: forecast di un periodo della serie temporale, le bande blu sono le regioni di confidenza, ovvero il range di valori in cui il il modello è sicuro all'80% (valore di default) sia presente il valore predetto

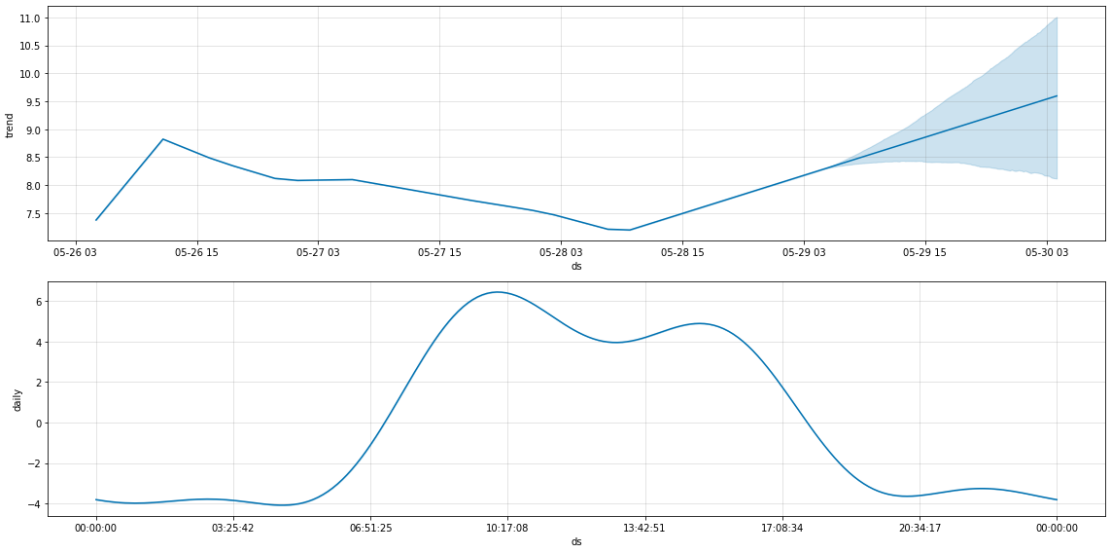

Figura 3.6: Prophet: componenti del modello, in alto il trend, con la regione di confidenza per il forecast; in basso la componente stagionale calcolata con la trasformata di fourier

Trend

Prophet fornisce due modi diversi per modellare il trend, un "saturating growth model" e un "piecewise linear model"

- Saturating growth model: viene utilizzato nel caso in cui il fenomeno che si sta modellando presenti una componente di saturazione, ovvero un valore massimo che il trend può raggiungere.

  Un esempio di un fenomeno di questo tipo può essere il numero di persone con accesso ad internet in una particolare area geografica, valore limitato chiaramente dal numero totale di persone che vivono in quella data area.

  Questo tipo di trend può essere rappresentato con questa formula:

  $g(t) = \frac{C}{1+exp(-k(t-m))}$

  Dove C è la carrying capacity (il limite alla crescita della serie temporale), k è il tasso di crescita e m un parametro di offset.

  C e k possono non essere costanti ma essere a loro volta delle variabili dipendenti dal tempo. Introdurre queste due variabili temporali direttamente nella formula complicherebbe troppo il modello.
  Per questo motivo è importante introdurre il concetto di changepoint, ovvero di un punto nella serie temporale in cui il valore costante k può variare (e di conseguenza k sarà costante tra due changepoint consecutivi).

  Supponendo che ci siano S changepoints con j=1,...,S; Prophet definisce un vettore di "rate adjustments" $\delta \in \mathbb{R}^S$ dove δ_j è la variazione di k che avviene al tempo s_j, Il rate al tempo t è per cui dato dal rate iniziale più la somma degli adjustment fino a quell'istante:

  $k + \sum_{j:t>s_j} \delta_j$

  Si può semplificare questa formulazione introducendo un vettore a(t) tale che:

  $$a_j(t) = \begin{cases} 1 & \text{if } t \geqq s_j, \\ 0 & \text{altrimenti} \end{cases}$$

  il rate può per cui essere rappresentato come: $k + a(t)^T\delta$

Quando il parametro k viene modificato, anche il parametro m va modificato in modo da far connettere gli estremi dei segmenti di k(t).
Si può per cui calcolare l'Adjustment totale come:

$\gamma_j = (s_j - m - \sum_{i<j} \gamma_i)(1 - \frac{k+\sum_{i<j} \delta_i}{k+\sum_{ij} \delta_i})$

La formulazione finale con tutte le correzioni sarà:

$g(t) = \frac{C(t)}{1+exp(-(k+a(t)^T\delta)(t-(m+a(t)^T\gamma)))}$

- Piecewise linear model: Questo modello è molto più semplice e viene utilizzato nei casi in cui si vuole avere un rate di crescita costante con un massimo ben definito.
  La formulazione è la seguente:

  $g(t) = (k + a(t)^T\delta)t + (m + a(t)^T\gamma)$

  I parametri hanno lo stesso significato descritto nel paragrafo precedente, con la differenza che il termine di adjustment non è necessario se non per rendere la funzione continua. il termine di correzione va impostato in questo modo: $-s\delta_j$

Selezione automatica dei changepoint

I changepoint possono essere specificati manualmente se si conosce la loro posizione temporale, ma più comunemente possono anche essere selezionati automaticamente data una serie di punti candidati.

Gli sviluppatori del modello consigliano di selezionare un grande numero di changepoints e impostare il vettore di adjustment come: $\delta_j \sim Laplace(0, \tau)$
Dove τ è un parametro che controlla la flessibilità del modello nel modificare il rate di crescita. La distribuzione di Laplace può essere osservata nel grafico della figura 3.7

NOTA: dato che δ_j non influisce sul parametro k, più τ si avvicina a zero, più il modello si avvicina a una regressione standard senza changepoints.

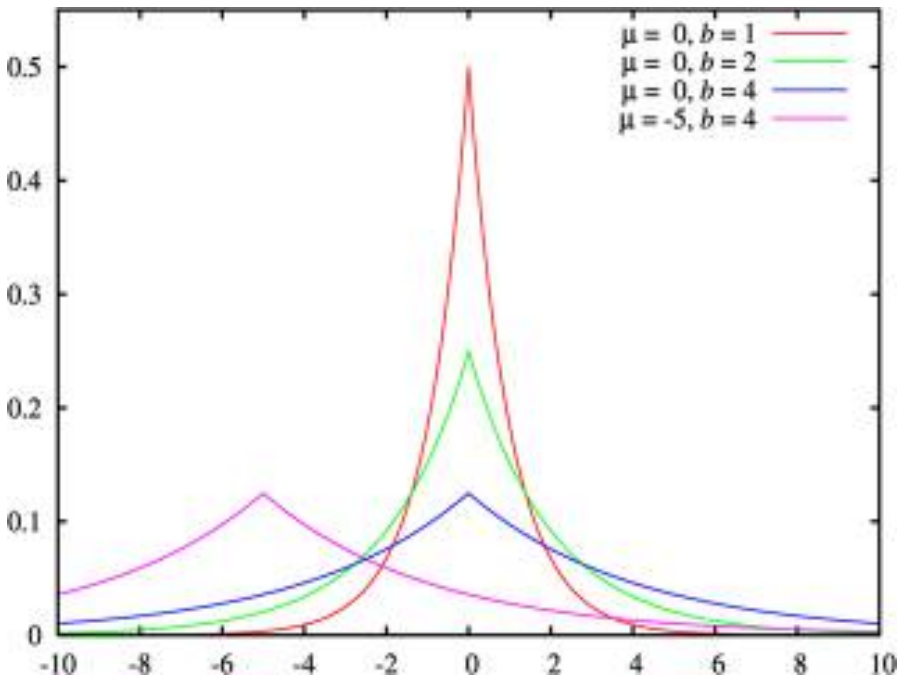

Figura 3.7: Distribuzione di Laplace, la b di questo grafico corrisponde al parametro τ [9]

Trend Forecast

Se si utilizza la funzione g(t) estrapolata dai dati passati per il forecast, tutti i dati generati avranno un tasso di crescita (k) costante, senza changepoints.

Per generare anche i changepoints, mantenedo lo stesso "pattern" del passato, occorre rimpiazzare il parametro τ del modello (usato per la fase learning) con la maximum likelihood estimate del rate change parameter basata sugli adjustment δ del modello:

$\lambda = \frac{1}{S} \sum_{j=1}^{S} |\delta_j|$

I changepoint vengono campionati randomicamente in modo che la frequenza media sia la stessa dei dati di training.

$$\forall j > T = \begin{cases} \delta_j = 0 \text{ w.p. } \frac{T-S}{T} \\ \delta_j \sim Laplace(0,) \text{ w.p. } \frac{S}{T} \end{cases}$$

L'assunzione che i dati futuri mantengano la stessa frequenza di rate change viene usata per calcolare i range di incertezza che vengono visualizzati nella Figura 3.6.

Uno svantaggio di questo approccio è che l'assunzione è molto forte, quindi è possibile che gli intervalli di incertezza non siano completamente accurati, ma questo non è un grande problema nella maggior parte dei casi.

Se il parametro τ cresce il modello è più flessibile nella fase di learning ma i range di incertezza saranno più grandi.

Stagionalità

Prophet prevede la presenza di molteplici componenti stagionali: annuali, settimanali, giornaliere e sub-giornaliere. Per modellare queste stagionalità prophet si basa sul teorema di fourirer che permette di rappresentare un segnale periodico come la somma di segnali periodici.

Il numero di campioni in un periodo viene indicato con P e la componente stagionale può essere rappresentata con:

$s(t) = \sum_{n=1}^{N}(a_n cos(\frac{2\pi nt}{P}) + b_n sin(\frac{2\pi nt}{P}))$

La fase di learning consisterà per cui nello stimare i 2n parametri (an, bn), questo viene fatto costruendo una matrice di vettori di stagionalità per ogni valore di t per i dati passati e futuri., ad esempio se N=10

$X(t) = [cos(\frac{2\pi(1)t}{365.25}), ..., sin(\frac{2\pi(10)t}{365.25})]$

$s(t) = X(t)\beta$ (con β una distribuzione normale (0, σ_2) per imporre uno "smoothing prior" che serve per evitare l'overfitting del modello.

Troncare la serie di fourier a N applica un filtro passa basso alla stagionalità, quindi si può scegliere N per impostare la flessibilità del modello, più N cresce più il modello è flessibile ma con rischio di overfitting.

NOTA: sperimentalmente si è visto che N=10 è un valore buono per la componente annuale e N=3 per la componente settimanale

Holidays

Con holidays si intendono eventi eccezionali nella serie temporale che "rompono" la stagionalità. Questi eventi non possono essere rilevati automaticamente dal modello in quanto per definizione sono "imprevedibili", devono per cui essere definiti manualmente.

Per ogni holiday i, si definisce D_i l'insieme di tutte le date passate e future di quell'evento e si definisce il parametro κ_i che corrisponde al cambiamento di previsione per ogni holiday.

Come per la stagionalità viene realizzato generando una matrice di regressori:

$Z(t) = [1(t \in D_1), ...,1(t \in Dl)$

$(t) = Z(t)\kappa$

NOTA: spesso quando si definiscono le holidays bisogna tenere in considerazione l'effetto di tale evento sui campioni precedenti e successivi della serie temporale, questo si può fare in modo molto semplice considerando anche questi campioni come holidays a loro volta.

3.5 Isolation forest

L'isolation forest [10] è un algortimo che permette di effettuare unsupervised anomaly detection su un insieme di dati. Come suggerito dal nome si tratta di un modello derivato dalle random forest e di conseguenza dai decision tree .

Gli alberi di decisione e le random forest non sono algoritimi di anomaly detection ma di classificazione supervisionata. La peculiarità principale dei decision tree è quella di selezionare un attributo del dataset e una condizione di separazione (solitamente binaria) e dividere il dataset in due sub-dataset in base alla condizione.

Il processo viene ripetuto ricorsivamente sui subset generandone di sempre più piccoli finchè non si raggiunge una condizione di terminazione, tale condizione può variare in base al contesto ma solitamente si smette di far crescere l'albero quando i nodi foglia hanno superato una certa soglia di purezza (ovvero la maggior parte dei campioni nel subset sono classificati nello stesso modo) oppure si è raggiunta la profondità massima dell'albero (per evitare overfittning).

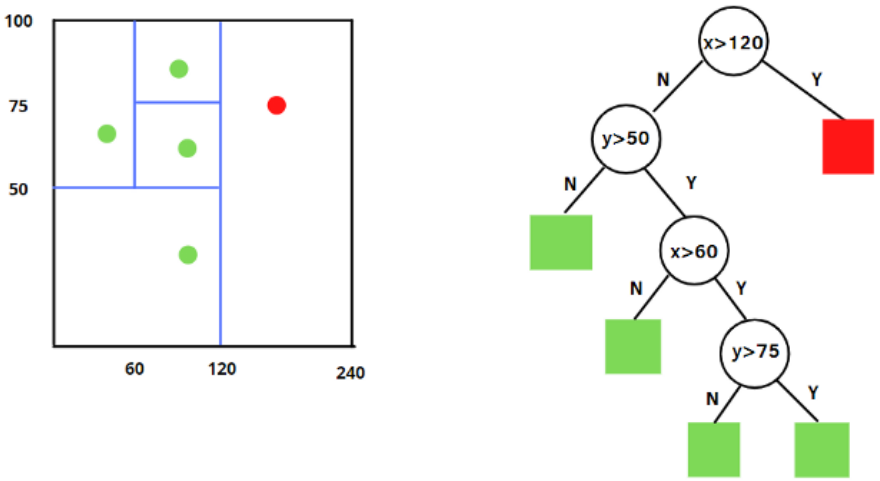

Figura 3.8: Isolation forest: costruzione albero [11]

Le random forest espandono il concetto di decision tree aggiungendo un ulteriore step iniziale. Invece di allenare il modello sull'intero dataset vengono creati un certo numero di subset più piccoli, ognuno generato con random sampling con rimpiazzamento a partire dal dataset iniziale.

Successivamente un decision tree viene trainato per ognuno dei subset selezionando anche un sottoinsieme degli attributi utilizzabili per gli split. La classificazione finale viene effettuata con majority voting di ogni singolo albero.

L'isolation forest unisce le random forest con il concetto di "isolamento" relativo alle anomalie. Con il termine "isolamento" si intende la separazione dei punti anomali dal resto del dataset. In generale si può affermare che i campioni anomali del dataset soddisfano le seguenti condizioni:

- Sono una minoranza rispetto alla totalità dei campioni
- I valori degli attributi dei campioni anomali sono molto diversi rispetto ai campioni non anomali

Si può per cui intuire che i punti anomali saranno probabilmente isolati dal resto dei dati in pochi passaggi del decision tree e quindi si troveranno vicino alla radice di quest'ultimo.

E' per cui possibile assegnare "un'anomaly score" ad ogni punto in base alla distanza di quest'ultimo dalla radice dell'albero, dove per distanza si intende il numero di nodi attraversati dal nodo radice al nodo foglia contenente il punto in questione.

Siccome ogni albero è generato partendo da un subset casuale di campioni e selezionando un subset casuale degli attributi, ogni albero nella foresta sarà diverso e di conseguenza anche la distanza di ogni punto dalla radice sarà diversa, per cui quando si parla di "distanza dalla radice" nel contesto di un insieme di alberi ci si riferisce alla distanza media punto per punto.

Oltre a questa intuizione di base l'isolation forest presenta altre caratteristiche rilevanti:

- Non utilizza misure basate su distanza o densità quindi non soffre della cosiddetta "curse of dimensionality" e può essere utilizzato con dataset aventi un elevato numero di features, incluso se molte di queste feature non sono rilevanti nel contesto dell'anomaly detection
- Molto efficiente sia a livello computazionale (O(n)) che di consumo di memoria
- Siccome la maggior parte dei punti "non anomali" non sono necessari per identificare le anomalie, l'isolation forest lavora bene anche con dataset molto limitati. Un dataset limitato può addirittura essere benefico per questo modello in quanto vengono ridotti gli effetti di "swamping" a "masking".

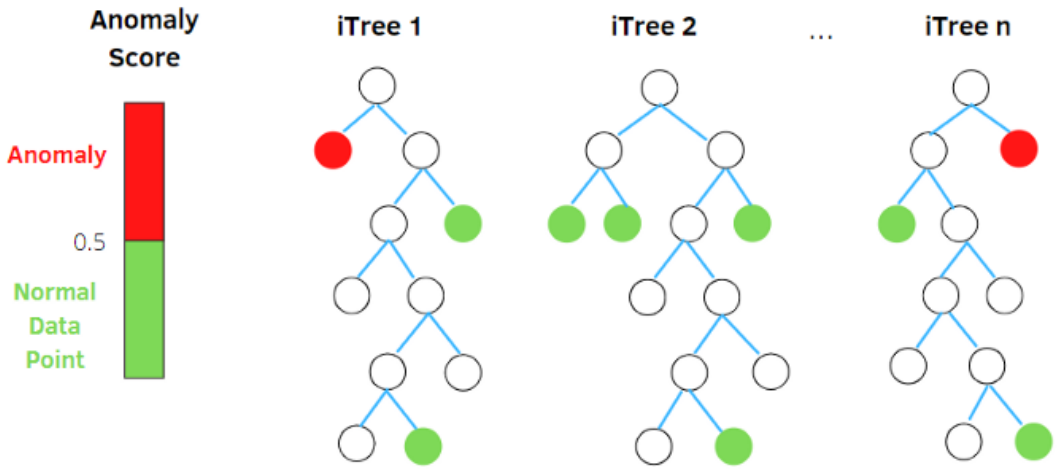

Figura 3.9: Isolation forest: anomaly score [11]

Per swamping si intende la tendenza a classificare punti non anomali come anomali mentre il masking si riferisce alla presenza di troppe anomalie ravvicinate che quindi diventano difficili da rilevare data la premessa dell'isolation forest (cioè che punti anomali appartenenti ad un cluster denso sono difficilmente isolabili in pochi passaggi).

- Per ridurre ulteriormente i problemi di masking e swamping ogni albero della foresta può essere trainato su un dataset contenente un particolare tipo di anomalie

Una volta costruiti gli alberi occorre assegnare un "anomaly score" ad ogni punto che rispecchi il grado di anomalia del punto stesso. Dopo aver assegnato uno score ad ogni punto, si possono classificare le anomalie ordinando i punti in base all'anomaly score e selezionando i k punti con l'anomaly score più alto. Il valore k viene solitamente espresso come percentile di punti con anomaly score più alto viene chiamato "contamination".

L'anomaly score viene calcolato in questo modo: $s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}$

- E(h(x)): numero medio di nodi attraversati per raggiungere il punto partendo dalla radice per ogni albero nella foresta
- $c(n) = 2H(n-1) - (2\frac{2(n-1)}{n})$ distanza media di una ricerca senza successo in un albero con n nodi binario (il raggiungimento del nodo foglia per l'isolamento è a livello di operazioni identico ad una ricerca senza successo in un albero di ricerca binario) H(x) viene definito "armonic number" e può essere approssimato con ln(x)+ 0.5772156649 (costante di Eulero).
  - $E(h(x)) \to 0, s \to 1$; il punto è molto probabilmente un'anomalia
  - $E(h(x)) \to n-1, s \to 0$; il punto è molto probabilmente non è un'anomalia
  - $E(h(x)) \to c(n), s \to 0.5$; zona di incertezza, ma siccome le anomalie sono solitamente poche si può affermare che non sia un'anomalia

Si utilizza questa versione dell'anomaly score piuttosto che semplicemente h(x) per una questione di normalizzazione. Una normalizzazione "tradizionale " (ovvero (x-min)/(max-min)) produrrebbe valori molto distanti tra di loro, rendendo complicato confrontare score diversi tra di loro. Questo è dovuto essenzialmente al fatto che mentre l'altezza assoluta dell'albero cresce nell'ordine di n, l'altezza media di un punto cresce nell'ordine di log(x).

3.6 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)[12] è un algoritmo di clustering non supervisionato basato sulla densità dei punti nel dataset. Per densità si intende il numero di punti in un certo "volume" nello spazio del dataset.

Il clustering è una tecnica che permette il raggruppamento di campioni di un dataset in insiemi tali per cui tutti gli elementi al suo interno siano "simili", ovvero venga minimizzata la distanza tra i punti all'interno del cluster e venga massimizzata tra i punti in cluster diversi.

In questo senso il DBSCAN è molto potente e versatile in quanto non è necessario indicare a priori il numero di cluster ma è sufficiente selezionare i parametri "epsilon" e MinPts. Per capire il significato di questi parametri è necessario spiegare come il DBSCAN classifica i punti del dataset, in particolare ogni punto del dataset può essere:

- Core point: punto interno di un cluster, caratterizzato dal fatto di contenere un numero di punti all'interno di un raggio epsilon dal punto stesso maggiore di MinPts
- Border point: punto che non contiene un numero di punti maggiori di MinPts all'interno del raggio epsilon ma è contenuto dentro al raggio epsilon di un core point
- Outlier: punto al di fuori di ogni cluster

Ovviamente nel caso dell'anomaly detection i punti di interesse saranno proprio gli outlier, ovvero i punti che non possono essere assegnati ad alcun cluster.

L'algoritmo segue i seguenti passi:

1. Si sceglie un punto p del dataset non ancora etichettato e gli si assegna una nuova label
2. Si calcola la distanza di tutti i punti dal punto appena etichettato
3. Si selezionano tutti i punti la cui distanza da p è inferiore a epsilon
4. Se il numero di punti è maggiore uguale a MinPts, a tutti i punti selezionati verrà assegnata la stessa label assegnata a p

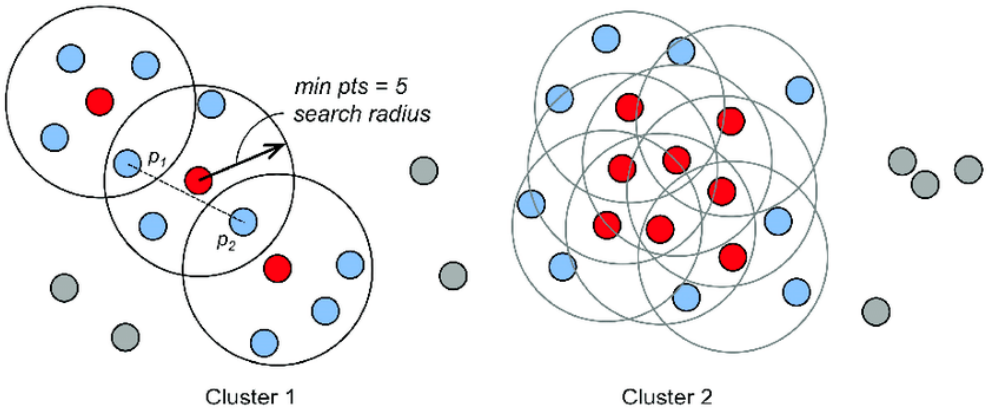

Figura 3.10: DBSCAN: i punti rossi sono i core points, quelli azzurri i border point ed infine gli outlier sono i grigi

5. Si scelgono uno alla volta i punti etichettatati e si riparte dallo step 2
6. Se non ci sono più punti da etichettare con la label corrente si riparte dal punto 1
7. Quando ogni punto ha una label l'algoritmo è concluso, i cluster costituitii da un solo punto saranno gli outlier

Ovviamente la criticità principale di questo modello sta nella selezione dei parametri epsilon e MinPts, scegliendo un epsilon troppo elevato molti outlier verranno probabilmente assegnati ad un cluster, mentre viceversa con un epsilon molto piccolo molti punti non anomaly verranno classificati come outlier.

Di conseguenza il DBSCAN non funzionerà in modo adeguato nel caso di dataset con cluster a densità diversa, per ovviare a questo problema potrebbe essere utile applicare il dbscan più volte con epsilon diversi, eliminando dal dataset di volta in volta i core point trovati.

Un altro problema del DBSCAN, come di tutti i modelli basati sulla distanza (euclidea), è la "curse of dimensionality". Con questo termine ci si riferisce al fenomeno per cui il concetto di distanza perde di significato man mano che il numero di dimensioni in uno spazio di features diventa elevato, inoltre calcolare la distanza tra ogni coppia di punti del cluster può essere dispendioso a livello coputazionale.

Data Preprocessing

I dati necessari allo studio del problema dell'anomaly detection sono stati raccolti da una piattaforma di proprietà di un cliente aziendale che ha gentilmente concesso l'accesso alle metriche di performance del server weblogic.

Come già accennato nel capitolo 2, l'application server weblogic mette a disposizione un framework per la raccolta di metriche di performance (WLDF).
Il numero di features a disposizione è estremamente elevato, e ogni componente aggiuntivo presente sulla piattaforma porta con sé un'insieme di nuove metriche selezionabili.

Per semplificare la raccolta dei dati esistono dei moduli built-in che contengono un insieme delle metriche più importanti e soprattutto più generali, ovvero che sono presenti in ogni installazione di weblogic a prescindere dai componenti aggiuntivi e dalle applicazioni web installate.

Le metriche built-in sono raggruppate nelle seguenti categorie:

- Statistiche della java virtual machine
- Statistiche dei Thread (nel contesto di weblogic i thread sono da intendersi come "richieste" di una particolare risorsa e non come i thread del sistema operativo)
- statistiche del sistema di connessioni alle basi di dati (JDBC)
- Statistiche del JMS (Java messaging system) ovvero lo standard usato da java per il passaggio di informazioni tra oggetti
- Statistiche di logging
- Statistiche JTA (Java transaction API)

Il fatto che i dati siano stati raccolti da un'application server realmente utilizzato porta con se vantaggi e svantaggi.

Il vantaggio principale è ovviamente che le metriche rispecchiano il comportamento reale del server e di conseguenza i modelli sviluppati su questi dati dovrebbero essere scalabili anche in altri contesti simili.

Tuttavia è impossibile, per motivi di business piuttosto che tecnici, avere un controllo dei moduli attivi sul server per decidere accuratamente le metriche selezionabili dato che ciò andrebbe a interferire con il business del cliente su un server di sua proprietà, né fare stress test per osservare il comportamento delle metriche in situazioni critiche, che sono quelle di interesse per l'anomaly detection nella maggior parte dei casi.

Inoltre il caso di studio preso in considerazione è relativo ad un cliente il cui sistema è raramente sotto sforzo, questo è ottimo per il suo business ma è un ostacolo per quanto riguarda il progetto di tesi perchè comporta che molte delle metriche raccolte dal modulo built-in siano in realtà non valorizzate e quindi inutili per i modelli; di conseguenza il pool di metriche che sono state utilizzabili è abbastanza ridotto e questo può incidere sull'efficacia dei modelli.

Nei successivi paragrafi di questo capitolo verranno descritte nel dettaglio tutte le metriche raccolte e le operazioni effettuate per rendere questi dati digeribili dagli algoritmi.

4.1 Metriche selezionate

In questa sezione verranno descritte ed elencate tutte le metriche rimaste dopo il processo di filtraggio di quelle nulle e selzione delle più rilevanti.

Il filtraggio è stato necessario in quanto alcune metriche sono date dalla combinazione di altre più semplici oppure rappresentano lo stesso concetto con un valore diverso, ad esempio percentuale di memoria libera e valore assoluto di memoria libera.

le metriche raccolte sono:

- `Open_Sockets_Current_Count` : rappresenta il numero di socket (ovvero di connessioni tcp) aperte
- `Open_Sessions_Current_Count` : rappresenta il numero di sessioni aperte
- `Execute_Thread_Idle_Count`: rappresenta il numero di Thread (entità che eseguono le richieste) in stato idle, ovvero avviati ma in attesa di una qualche richiesta
- `Pending_User_Request_Count`: rappresenta il numero di richieste utente in attesa di essere gestite
- `Standby_Thread_Count`: Numero di Thread creati ma al momento non utilizzati
- `Throughput`: numero di richieste soddisfatte al minuto
- `Heap_Free_Percent`: percentuale di memoria ram libera (nello specifico la heap della java virtual machine)
- `Heap_Size_Current`: dimensione massima della memoria heap della java virtual machine

I dati sono stati raccolti con una frequenza di un campione al minuto, uguale per ogni metrica, per un periodo di tempo di 5 settimane dal 24 luglio 2021 al 28 agosto 2021.

Questo intervallo di tempo è purtroppo problematico in quanto è presente la settimana di ferragosto che introduce un effetto che rompe la stagionalità, dovuto al basso carico del sistema nelle ferie estive. Questo effetto si può osservare nelle figure 4.1 e 4.2

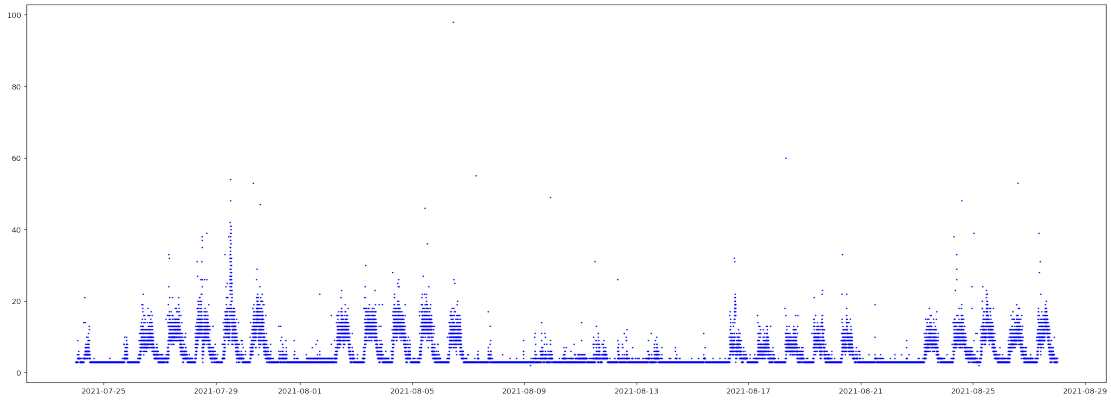

Figura 4.1: Intervallo di tempo relativo alla feature *OpenSocketCurrentCount*

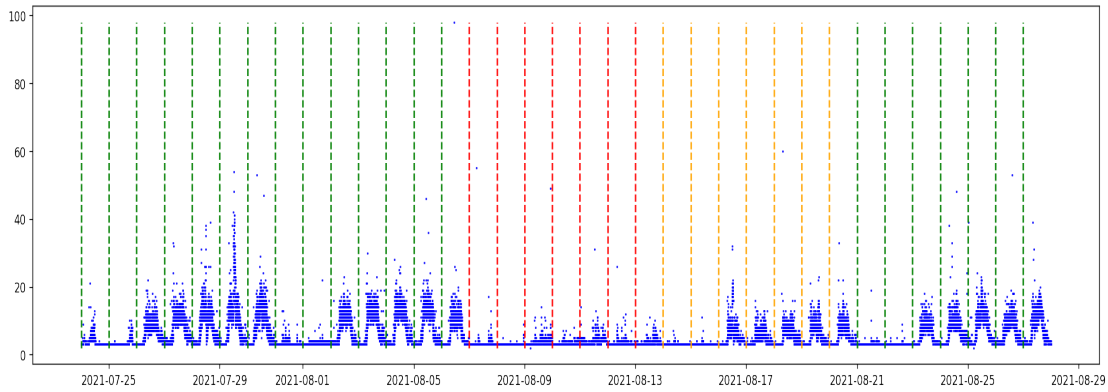

Figura 4.2: Intervallo di tempo relativo alla feature *OpenSocketCurrentCount* con la settimana con basso carico evidenziata in rosso

Come evidenziato dalle immagini appena mostrate, la settimana dal 7 ottobre al 14 ottobre presenta un quasi annullamento della metrica, indice del fatto che pochissimi socket sono stati aperti in questi giorni dato che il sistema è stato sostanzialmente inutilizzato.

Questo fenomeno si ripete per tutte le metriche e rappresenta un problema in quanto la rottura della stagionalità comporta una maggiore complessità dei modelli e rende più difficile la rilevazione di anomalie, soprattutto in luce del fatto che il periodo di tempo preso in esame non è molto esteso e la settimana con poco carico rappresenta un quinto di tutti i campioni.

Per questi motivi nell'ambito del progetto di tesi si è ritenuto più utile ignorare la presenza di questa settimana in quanto la sua gestione avrebbe sviato

l'attenzione dall'obiettivo finale, ovvero lo sviluppo di modelli per l'anomaly detection.

Per fare questo la settimana che nella figura 4.2 è indicata in rosso è stata eliminata e le settimane successive, a partire da quella in giallo, sono state scalate in avanti di sette giorni.

Questa operazione "rompe" il significato intrinseco della metrica in quanto alcuni campioni vengono associati ad un timestamp che non è relativo alla vera misurazione di quel campione, ma è necessaria perchè le time series per essere analizzate devono essere continue, senza dati mancanti, specie se a mancare è un'intera settimana.

Questa operazione nella pratica non è problematica in quanto, come si può osservare dai dati, il comportamento delle ultime due settimane è molto simile a quello che ci si aspetterebbe in un mese normale, senza eventi particolari al suo interno come la settimana di ferragosto.

Chiaramente in uno scenario di produzione sarebbe necessario selezionare il periodo in cui vengono campionati i dati di training con più cognizione di causa, tuttavia nell'ambito della tesi questo è stato l'unico periodo di tempo analizzabile e quindi è stato necessario effettuare questo piccolo taglio.

La metrica risultante si può osservare nella figura 5.38

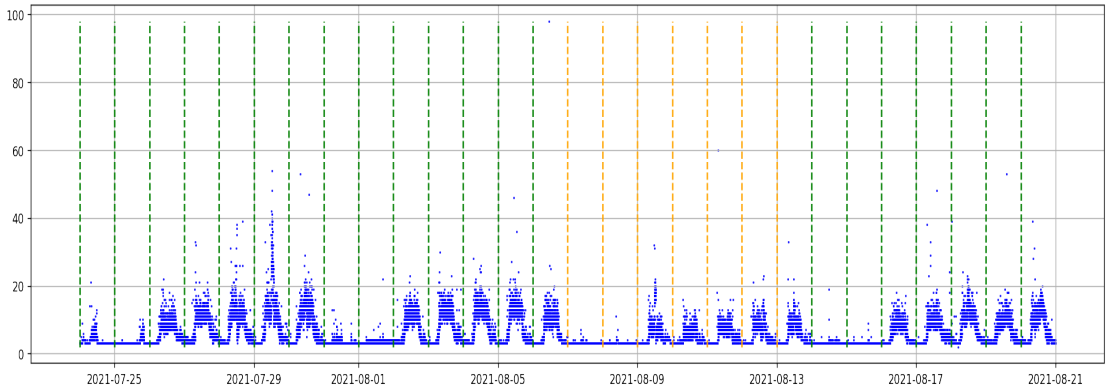

Figura 4.3: Intervallo di tempo relativo alla feature *OpenSocketCurrentCount* dopo il taglio

Nelle segenti pagine verranno mostrati i grafici di tutte le metriche utilizzate.

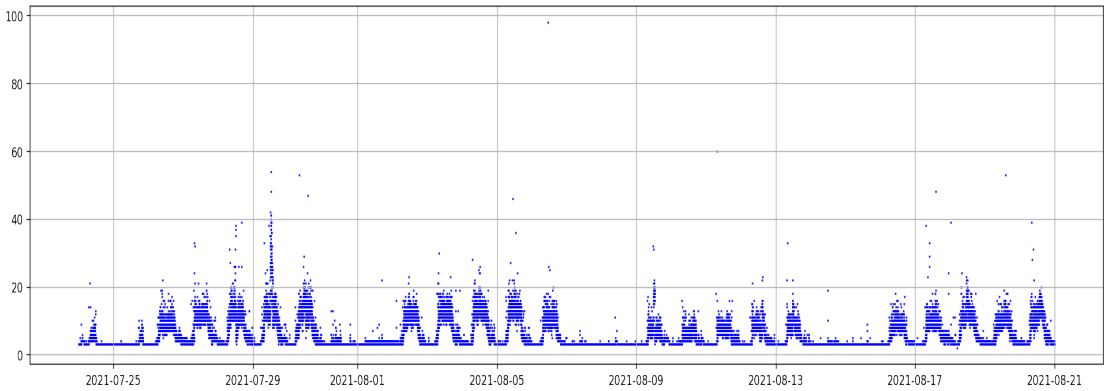

Figura 4.4: Grafico relativo alla metrica *OpenSocketCurrentCount*

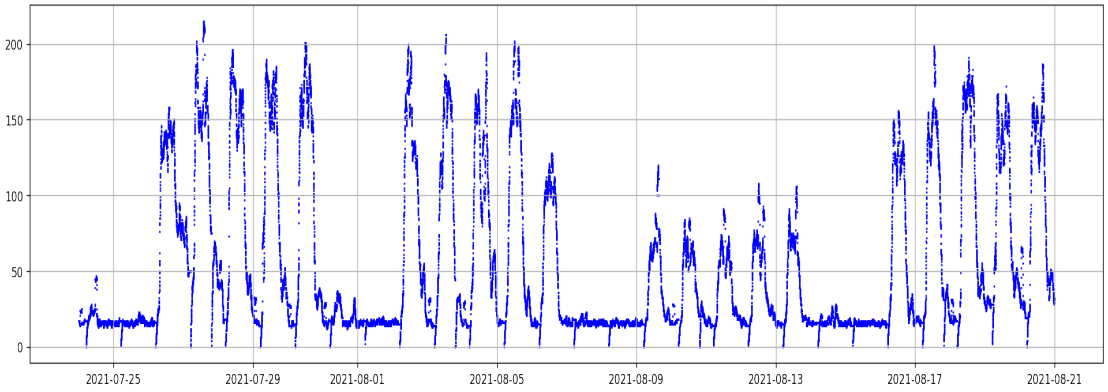

Figura 4.5: Grafico relativo alla metrica *OpenSessionCurrentCount*

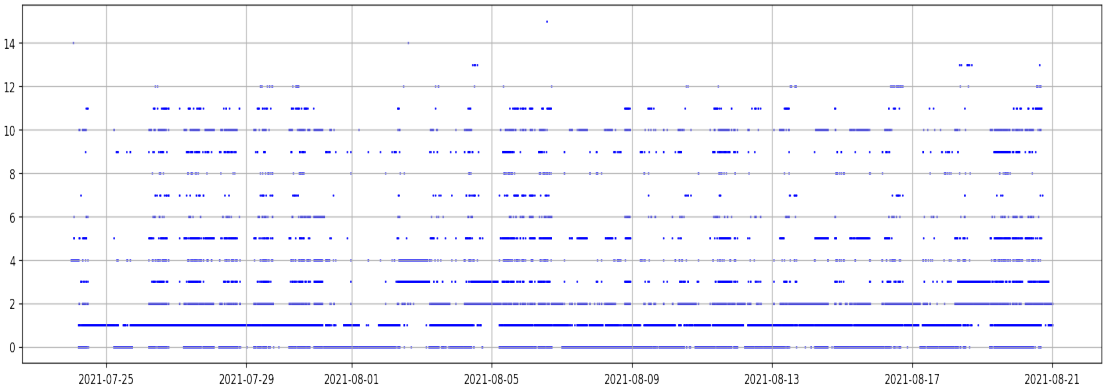

Figura 4.6: Grafico relativo alla metrica *ExitThreadIdleCount*

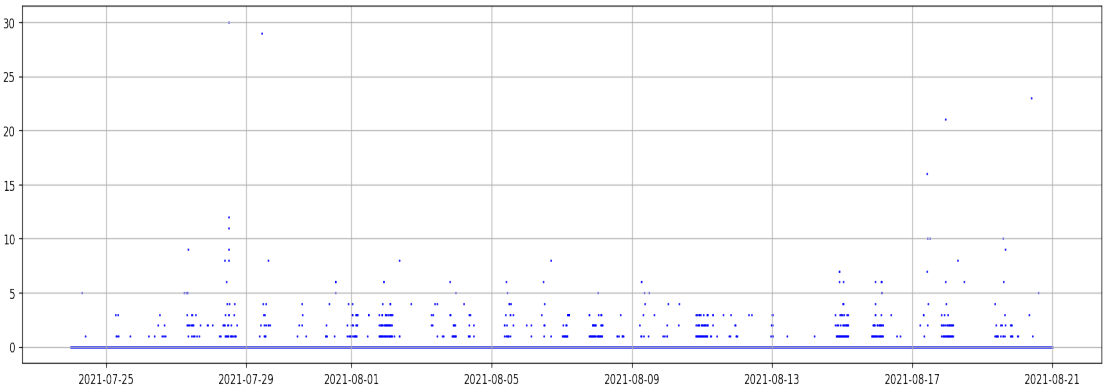

Figura 4.7: Grafico relativo alla metrica *PendingUserCount*

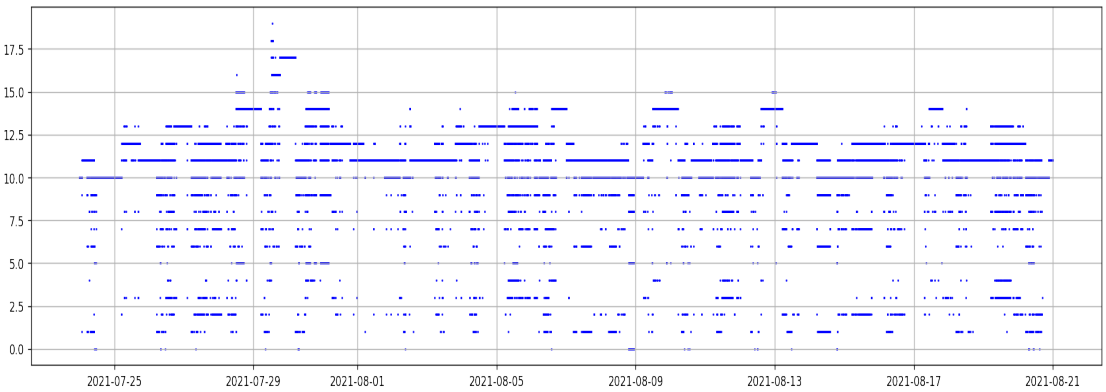

Figura 4.8: Grafico relativo alla metrica *StandByThreadCount*

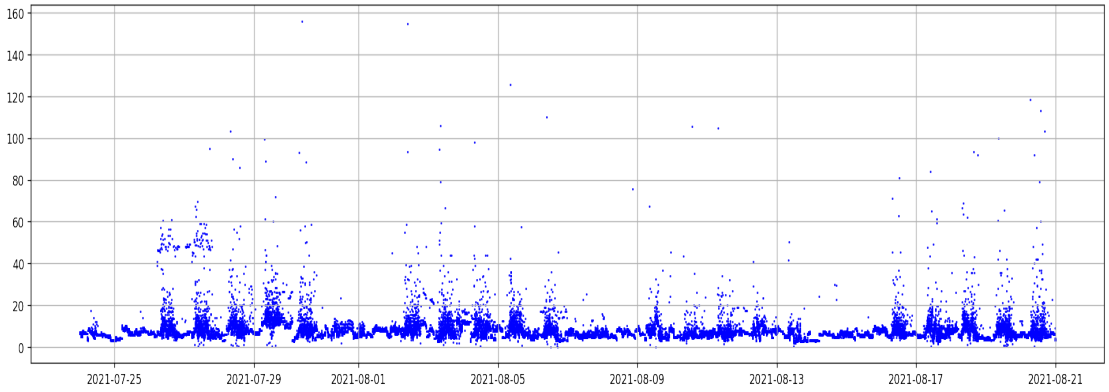

Figura 4.9: Grafico relativo alla metrica *Throughput*

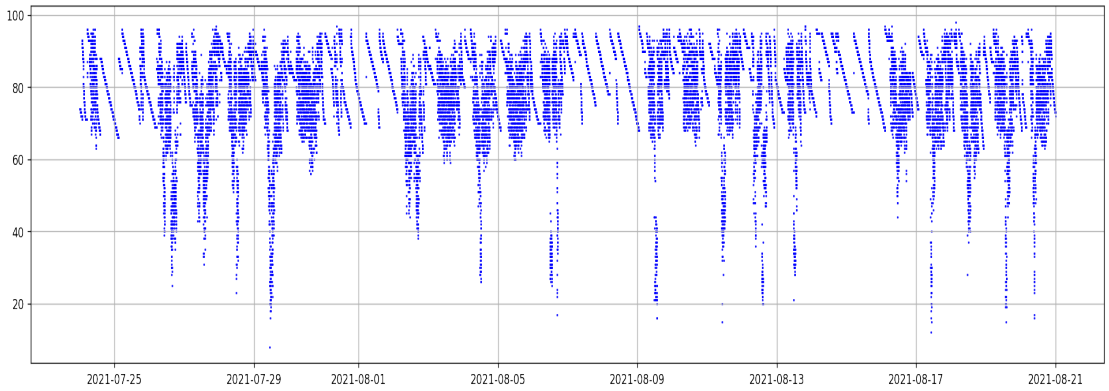

Figura 4.10: Grafico relativo alla metrica *HeapfreePercent*

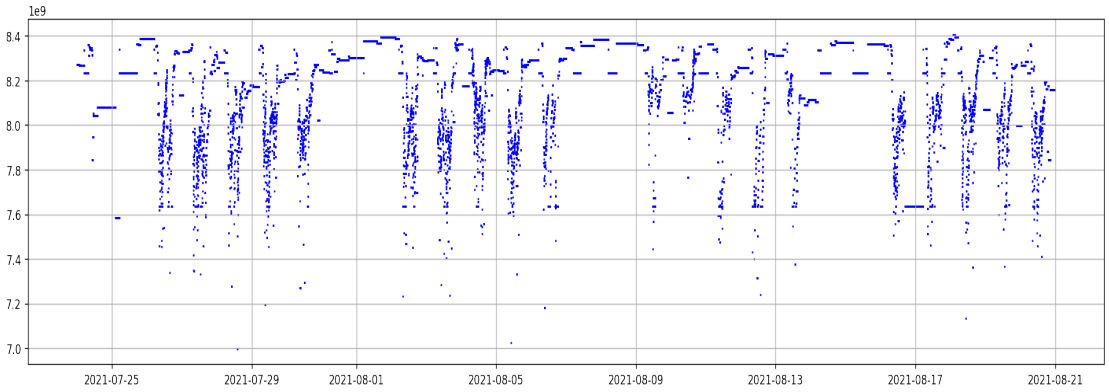

Figura 4.11: Grafico relativo alla metrica *HeapSizeCurrent*

4.2 Formato dei dati

Prima di poter essere rappresentati come appena descritto nel paragrafo precedente a questo, i dati grezzi provenienti dal server weblogic vanno prima elaborati in modo da rimuovere rumore e informazioni addizionali inutili per le analisi successive.

Il connettore JDBC che trasporta i dati li inserisce all'interno di una tabella di un DB chiamata WLS_HVST, la struttura della tabella è la seguente:

```
create table WLS_HVST
(
    RECORDID       NUMBER(20) not null,
    TIMESTAMP      NUMBER(20)     default NULL,
    DOMAIN         VARCHAR2(250)  default NULL,
    SERVER         VARCHAR2(250)  default NULL,
    TYPE           VARCHAR2(250)  default NULL,
    NAME           VARCHAR2(500)  default NULL,
    ATTRNAME       VARCHAR2(250)  default NULL,
    ATTRTYPE       NUMBER(10)     default NULL,
    ATTRVALUE      VARCHAR2(4000) default NULL,
    WLDFMODULE     VARCHAR2(250)  default NULL,
    PARTITION_ID   VARCHAR2(250)  default NULL,
    PARTITION_NAME VARCHAR2(250)  default NULL
)
```

Gli unici attributi necessari per l'analisi della anomalie sono:

- `TIMESTAMP` che indica l'istante di campionamento
- `ATTRVALUE` che contiene il valore della metrica
- `TYPE,NAME,ATTRNAME` che identificano univocamente il nome della metrica: `TYPE` contiene il nome dell'Mbean, `NAME` che contiene il nome dell'istanza (ad esempio il nome del server o il nome della componente di weblogic che ha generato la metrica), `ATTRNAME` contiene il nome della metrica vera e propria (metriche in MBean diversi possono avere lo stesso `ATTRNAME` per questo occorrono tutti e 3 gli attributi.

I dati di questa tabella vanno filtrati per due motivi, il primo è che in weblogic tutte le metriche del modulo built-in (vedere capitolo 3) vengono raccolte, a meno che non vengano scelte manualmente, operazione infattibile dovuto all'enorme mole di metriche e la mancanza di documentazione sulla maggior parte di esse.

Inoltre la tripletta `TYPE,NAME,ATTRNAME` è decisamente di difficile lettura in quanto consiste in una stringa troppo lunga e complicata:

weblogic.management.runtime.WebAppComponentRuntimeMBean,
"com.bea:ApplicationRuntime=W02JDP920,Name=w02wls3_/jde,
ServerRuntime=w02wls3,Type=WebAppComponentRuntime",OpenSessionsCurrentCount

Per ovviare a questi problemi è stata creata una tabella aggiuntiva sul DB per associare un flag ed un ID univoco alla tripletta `TYPE,NAME,ATTRNAME`.

il flag servirà in una vista successiva per filtrare le metriche inutili (quelle con flag impostato manualmente a 0) e l'id per associare un valore numerico semplice alla tripletta e rendere i dati più fruibili.

Il codice SQL della tabella e della vista sono molto semplici:

```
create table FEAT_FLAG
(
    TYPE     VARCHAR2(250),
    NAME     VARCHAR2(500),
    ATTRNAME VARCHAR2(250),
    FLAG     NUMBER,
    ID       NUMBER generated as identity
)

create view WLS_HVST_FILTERED as
SELECT *
FROM WLS_HVST where
                    CONCAT(TYPE,CONCAT(NAME,ATTRNAME))
                        in (SELECT CONCAT(TYPE,CONCAT(NAME,ATTRNAME))
                            from FEAT_FLAG where flag=1)
```

A questo punto i dati sono filtrati ma rimane ancora la semplificazione dell'id da applicare. Inoltre sarebbe utile avere tutti gli `ATTRVALUE` delle varie metriche corrispondenti ad uno stesso `TIMESTAMP`.

Per effettuare l'operazione appena descritta il linguaggio SQL fornisce un comando molto utile chiamato `PIVOT` che sostanzialmente permette di ruotare i valori della tabella facendoli diventare attributi, eseguendo le dovute aggregazioni.

Il codice sql per effettuare il pivot è il seguente:

```
create view PIVOT_WLS_HVST as
select  to_char(to_date('1970-01-01 02','yyyy-mm-dd hh24') +
                ("TIMESTAMP")/1000/60/60/24 , 'YYYY-MM-DD HH24:MI:SS')
                    as TS,
                  "272","430","431","436","480","482","776","777","780
                  ","781","782","783","784","907","908","909","910"
from (
         SELECT TIMESTAMP,ATTRVALUE,ID
         from WLS_HVST_FILTERED,FEAT_FLAG
         WHERE CONCAT(WLS_HVST_FILTERED.TYPE,
                CONCAT(WLS_HVST_FILTERED.NAME,WLS_HVST_FILTERED.ATTRNAME))
               =CONCAT(FEAT_FLAG.TYPE,
                 CONCAT(FEAT_FLAG.NAME,FEAT_FLAG.ATTRNAME)))
pivot (
    min(ATTRVALUE)
    for ID in (272,430,431,436,480,482,776,777,
               780,781,782,783,784,907,908,909,910)
)
order by TS
```

l'output della vista sarà una tabella in questo formato:

TS	**272**	**430**	**776**	**780**	**782**	**784**	**908**	**909**
2021-07-21 16:58:36	10	137	0	0	16	6.5	79	8148484096
2021-07-21 16:59:36	11	135	1	0	15	10.0	74	8148484096
2021-07-21 17:00:36	9	132	1	0	15	7.5	72	8148484096

I nomi delle colonne della tabella sono gli id delle metriche associati alla tripletta `TYPE,NAME,ATTRNAME,`inoltre il timestamp è stato convertito in un formato di data più facilmente comprensibile.

Anomaly detection

Nell'ambito del machine learning si possono distinguere principalmente due tipi di approcci all'apprendimento: supervisionato e non supervisionato.

La principale differenza tra i due consiste nell'utilizzo di dataset etichettati nel caso supervisionato e viceversa di dataset non etichettati nel caso non supervisionato.

Con etichette si intende un insieme di metadati che descrive i singoli campioni del dataset.

Un'esempio di etichette nell'ambito dell'anomaly detection potrebbero essere l'insieme di timestamp in cui una particolare metrica presenta un valore anomalo.

Poter applicare un approccio supervisionato in questo progetto sarebbe estremamente utile in quanto si potrebbe valutare in modo automatico l'accuratezza dei modelli, purtroppo però non esiste al momento un metodo per etichettare automaticamente il dataset e questa operazione andrebbe necessariamente effettuata manualmente da un esperto di settore.

Occorre per cui utilizzare un approccio non supervisionato, non basato sulle label ma basato solamente sui dati grezzi.

Un'altro aspetto molto importante da considerare è il tipo di anomalia. Infatti se ne possono indivituare di due tipi:

- Anomalia puntuale: interessa un singolo o pochi istanti temporali
- Anomalia contestuale: interessa un intervallo di tempo in cui la serie temporale presenta un comportamento anomalo

Il tipo di anomalia da individuare dipende dalla metrica e dal contesto di quest'ultima.

Una particolare metrica potrebbe presentare un'anomalia puntuale, ovvero un punto che si discosta dal resto della serie temporale, ma non avere la necessità di essere segnalata in quanto un singolo valore anomalo potrebbe non costituire un problema infrastrutturale.

Viceversa per una anomalia contestuale i singoli punti non risulterebbero anomali presi singolarmente, in quanto all'interno di una zona in cui la serie temporale ha un comportamento "continuo", ma l'andamento della serie in quel punto potrebbe non essere quello corrispondente ad un funzionamento normale del sistema.

Nel proseguio di questo capitolo verranno analizzati 3 modelli per la point anomaly detection, basati su ARIMA, PROPHET, ISOLATION FOREST e un modello per il rilevamento delle anomalie contestuali basato su ISOLATION FOREST.

Tutti i modelli presenteranno al loro interno uno studio della densità basato su DBSCAN o altri approcci.

5.1 Anomaly detection con ARIMA

Il primo algoritmo per la unsupervised point anomaly detection si basa sul modello ARIMA.

Come spiegato nel capitolo 3, il modello ARIMA è nato come modello di forecast, ovvero il suo scopo è quello di riuscire a prevedere i valori futuri di una serie temporale basandosi sui suoi valori passati.

Questa non è una operazione necessaria visto l'obiettivo dell'algoritmo, che è quello di individuare i punti anomali all'interno delle metriche che già si hanno a disposizione, per cui non serve effettuare un forecast nel futuro.

Il modello ARIMA è stato invece utilizzato per la sua capacità di ottenere un fitting molto accurato della serie temporale, dove per fitting si intende un'espressione matematica tale per cui la distanza dai punti della funzione rispetto a quelli della seire temporale sia minimizzata.

I passaggi principali dell'algoritmo sono i seguenti:

1. Normalizzazione della serie temporale
2. Fitting della serie temporale con il modello ARIMA con relativa banda di confidenza
3. Rimozione dei punti all'interno della regione di confidenza
4. DBSCAN sui punti al di fuori della regione di confidenza per ottenere i cluster di punti vicini tra loro
5. I punti che non appartengono a nessun cluster sono le anomalie

La normalizzazione viene applicata per fare in modo che ogni metrica abbia lo stesso range di valori, per poter confrontare metriche diverse a prescindere dalla loro scala.

In questo caso è stata utilizzata la Normalizzazione MinMax, che scala i campioni in modo da avere un range compreso tra 0 e 1:

$MinMax(x) = \frac{x - min(x)}{max(x) - min(x)}$

In seguito alla normalizzazione viene effettuato il fitting del segnale con ARIMA e viene ricavata la banda di confidenza sui dati di training; la banda indica

la regione del grafico all'interno della quale il modello è sicuro con una certa percentuale (90%) che la predizione sia corretta.

La banda può essere ottenuta chiedendo al modello di fare il forecast per lo stesso periodo di tempo su cui il modello è stato trainato.
In un modello tradizionale (con l'obiettivo di fare forecast accurati su dati mai visti) questa operazione sarebbe chiaramente errata in quanto risulterebbe in overfitting e di conseguenza in un modello poco accurato.

Tuttavia in questo caso l'obbittivo è quello di isolare i punti anomali; l'intuizione di partenza consiste in due affermazioni: la prima è che i punti anomali sono in estrema minoranza rispetto al totale di punti presenti nella serie temporale e per definizione non sono incasellabili all'interno di un pattern; la seconda è che un modello "overfittato" riesce a far rientrare al suo interno la maggior parte dei punti della serie temporale ma non ogni singolo punto.

La seconda affermazione regge in virtù del fatto che il modello ARIMA non è abbastanza complesso (ovvero non ha abbastanza parametri trainabili) per poter overfittare alla perfezione ogni singolo punto di una data serie temporale, specie se il numero di campioni è molto elevato come nel caso in questione.

A riprova di ciò si possono osservare i grafici delle metriche alla fine di questa sezione notando che il numero di punti che coincidono esattamente con la funzioni di fitting sono estremamente ridotti ma allo stesso tempo si può dire che qualitativamente la serie temporale trovata da ARIMA si adatta molto bene a quella originale, infatti la maggior parte dei campioni si trovano all'interno della regione di confidenza del modello.

I campioni al di fuori della regione di confidenza sono per cui candidati ad essere anomalie in quanto nemmeno il modello overfittato è stato in grado di modellarle facendole rientrare nella regione di confidenza.

Ogni campione al di fuori della regione di confidenza viene successivamente pesato per fare in modo che la distanza tra i campioni lontani dalla regione di confidenza (e quindi più probabilmente anomali) e quelli più vicini venga accentuata per rendere più semplice il lavoro del DBSCAN applicato nel passaggio successivo.

il DBSCAN ha lo scopo di clusterizzare i punti candidati ad essere anomali.
I punti vicini tra di loro verranno inseriti all'interno dello stesso cluster mentre quelli più isolati costituiranno dei cluster a sé stanti e verranno per cui etichettati come anomalie puntuali.

Algorithm 3 ARIMA based point anomaly detection

procedure ARIMA_ANOMALY_DETECTION (x, eps, m)
x è un vettore contenente tutti i campioni della serie temporale
eps è il raggio tramite il quale il DBSCAN calcola la densità
m è il numero di campioni in un periodo
$min \leftarrow$min(x)
$max \leftarrow$max(x)
for x[i] in x **do**
$x[i] \leftarrow \frac{x[i]-min}{max-min}$
▷ Normalizzazione
end for
$p, d, q \leftarrow auto\_arima(x, m)$
▷ auto arima per trovare i parametri ottimali
$model \leftarrow ARIMA(x, p, d, q, m)$
▷ Fitting del modello ARIMA
$upper\_band, lower\_band \leftarrow model.predict(x)$
▷ Predict sull'intervallo di training per ottenere la regione di confidenza
$out\_of\_bounds \leftarrow []$
for x[i] in x **do**
if $(x[i] > upper\_band[i])\ or\ (x[i] < loewr\_band[i])$ **then**
$out\_of\_bounds.append(\frac{distance(x[i],band)^2}{2*(upper_band[i]-lowe_band[i])})$
▷ vengono recuperati e pesati i punti al di fuori della regione di confidenza
end for
$clusters \leftarrow DBSCAN(out\_of\_bounds, eps)$
$anomalies \leftarrow []$
for cluters[i] in clusters **do**
if $(len(clusters[i]) == 1)$
$anomalies.append(clusters[i][0]))$
▷ le anomalie sono i punti all'interno dei cluster costituiti da un solo punto
end for
$return\ anomalies$
end procedure

Nelle seguenti pagine verranno riportati i grafici delle metriche con l'output dell'algoritmo appena descritto.

la funzione in blu rappresenta il fitting della funzione fatta da arima, la zona grigia è la banda di incertezza, i punti in giallo sono i punti all'interno della regione di confidenza, quelli rossi sono fuori dalla regione di confidenza, infine quelli viola sono i punti identificati come anomalie.

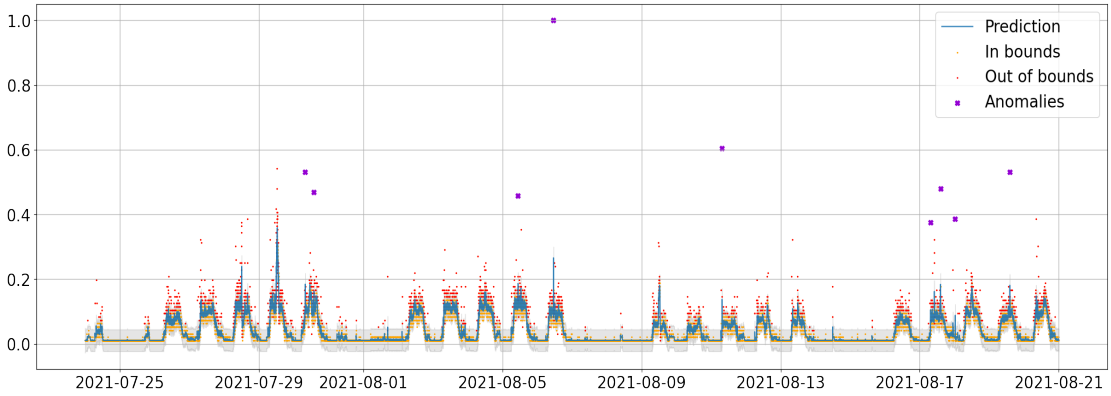

Figura 5.1: ARIMA anomaly detection sulla metrica *OpenSocketCurrentCount*

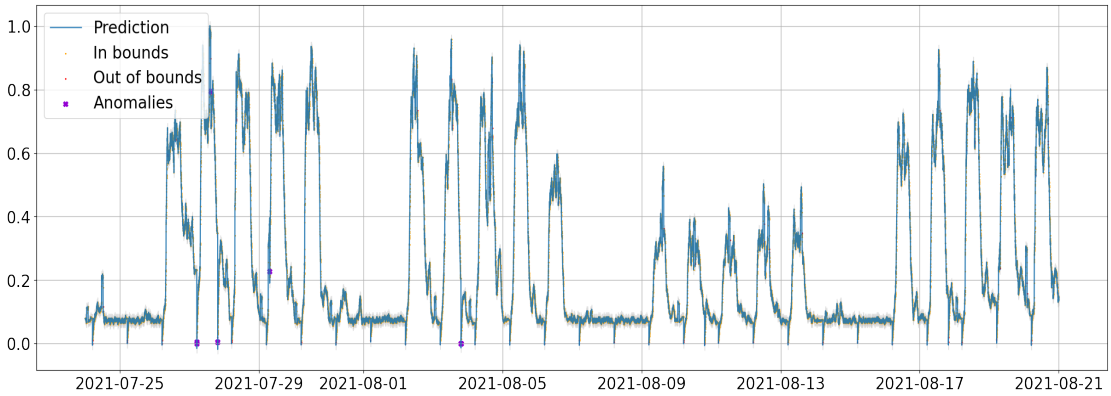

Figura 5.2: ARIMA anomaly detection sulla metrica *OpenSessionCurrentCount*

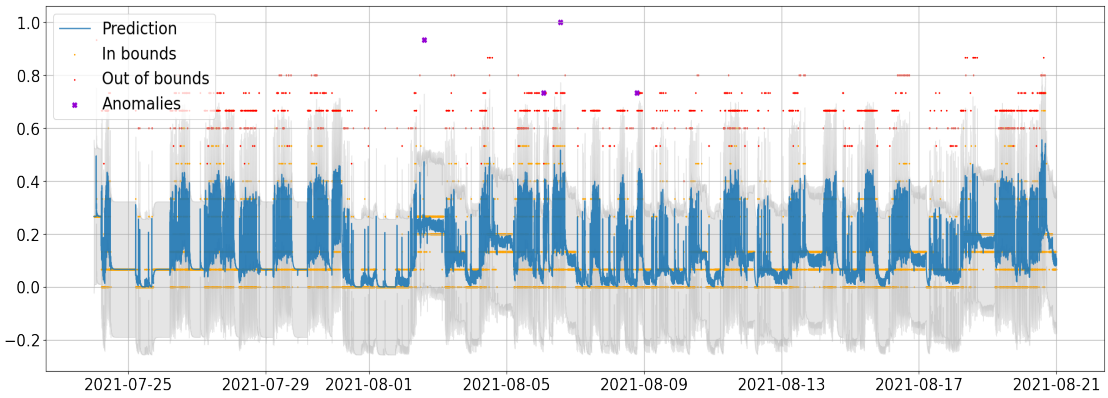

Figura 5.3: ARIMA anomaly detection sulla metricaa *ExitThreadIdleCount*

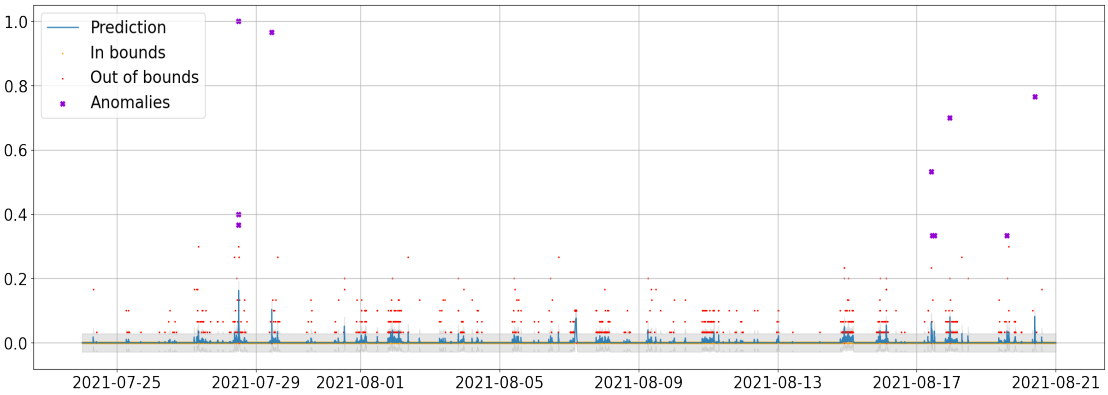

Figura 5.4: ARIMA anomaly detection sulla metrica *PendingUserCount*

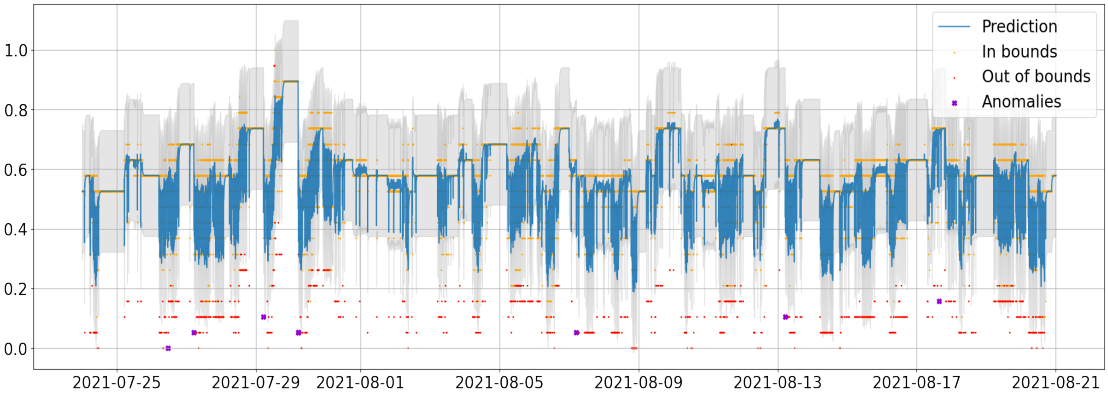

Figura 5.5: ARIMA anomaly detection sulla metrica *StandByThreadCount*

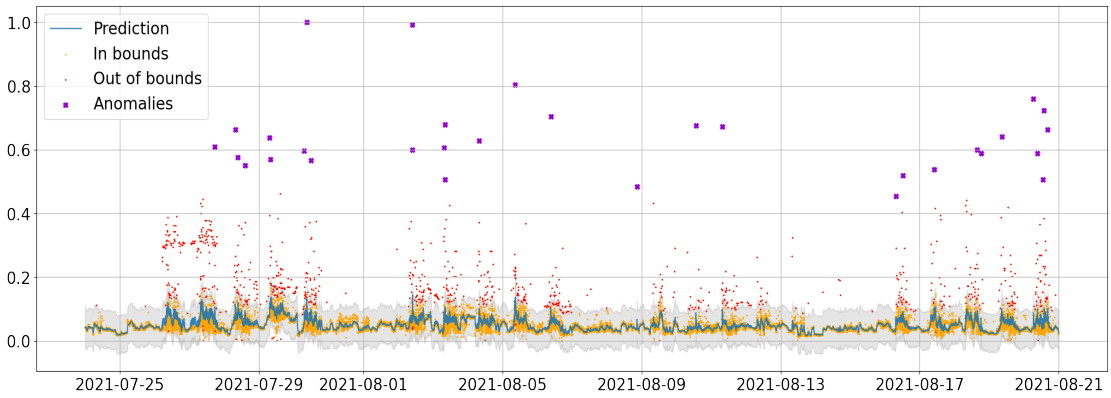

Figura 5.6: ARIMA anomaly detection sulla metrica *Throughput*

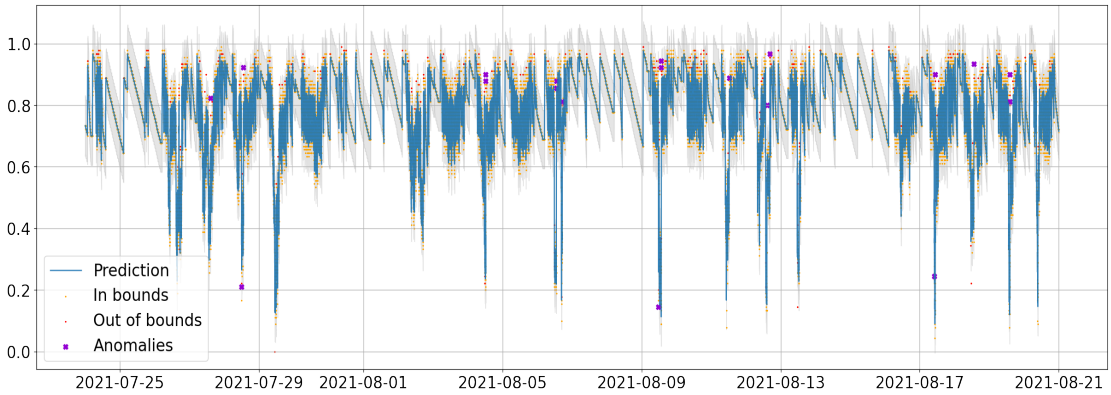

Figura 5.7: ARIMA anomaly detection sulla metrica *HeapfreePercent*

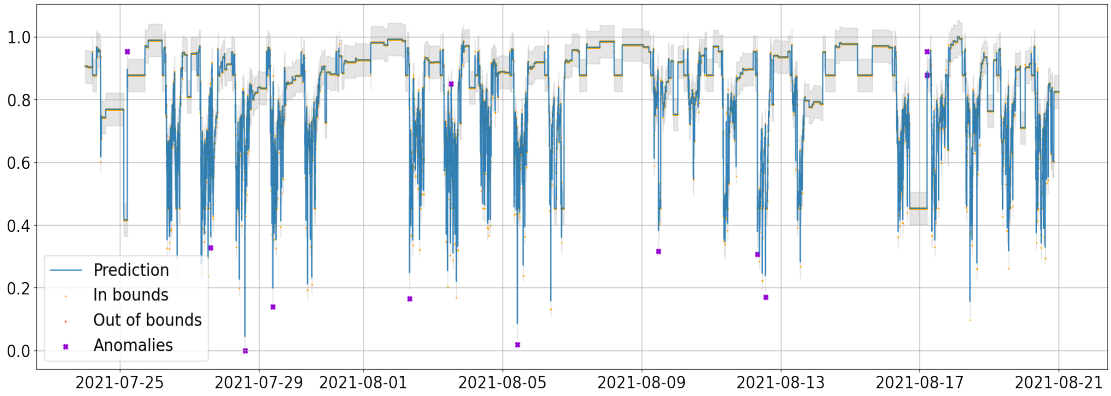

Figura 5.8: ARIMA anomaly detection sulla metrica *HeapSizeCurrent*

5.2 Anomaly detection con PROPHET

Il secondo modello che è stato sviluppato è invece basato su Prophet.

Il principio di funzionamento è lo stesso del modello sviluppato con ARIMA, ovvero:

1. Normalizzazione della serie temporale
2. Fitting della serie temporale con il modello Prophet con relativa banda di confidenza
3. Rimozione dei punti all'interno della regione di confidenza
4. DBSCAN sui campioni al di fuori della regione di confidenza per ottenere i cluster di punti vicini tra loro
5. I punti che non appartengono a nessun cluster sono le anomalie

Prophet concettualmente svolge la stessa funzione di ARIMA, offero il fitting di una funzione sulla serie temporale tale da minimizzare l'errore che si compirebbe per ogni punto se si utilizzasse la funzione di fitting al posto dei campioni reali.

Ovviamente il training del modello Prophet è differente rispetto a quello di ARIMA, di conseguenza anche la funzione di fitting sarà diversa.

Una differenza qualitativa che si può osservare è che Prophet da un peso molto maggiore alla componente stagionale (questo si può osservare negli intervalli di tempo in cui la stagionalità viene interrotta, come i fine settimana) e genera una funzione di fitting quanto più possibile continua, a differenza di ARIMA che si basa sull'autoregressione e la media mobile e di conseguenza genera una funzione non continua, con molti picchi e oscillazioni.

Algorithm 4 PROPHET based point anomaly detection

procedure PROPHET_ANOMALY_DETECTION (x, eps)
x è un vettore contenente tutti i campioni della serie temporale
eps è il raggio tramite il quale il DBSCAN calcola la densità
$min \leftarrow$min(x)
$max \leftarrow$max(x)
for x[i] in x **do**
$x[i] \leftarrow \frac{x[i]-min}{max-min}$
▷ Normalizzazione
end for
$model \leftarrow PROPHET(x, p, d, q, m)$
▷ Fitting del modello PROPHET
$upper\_band, lower\_band \leftarrow model.predict(x)$
▷ Predict sull'intervallo di training per ottenere la regione di confidenza
$out\_of\_bounds \leftarrow []$
for x[i] in x **do**
if $(x[i] > upper\_band[i])\, or\, (x[i] < loewr\_band[i])$ **then**
$out\_of\_bounds.append(\frac{distance(x[i],band)^2}{2*(upper_band[i]-lowe_band[i])})$
▷ vengono recuperati e pesati i punti al di fuori della regione di confidenza
end for
$clusters \leftarrow DBSCAN(out\_of\_bounds, eps)$
$anomalies \leftarrow []$
for cluters[i] in clusters **do**
if $(len(clusters[i]) == 1)$
$anomalies.append(clusters[i][0]))$
▷ le anomalie sono i punti all'interno dei cluster costituiti da un solo punto
end for
$return\, anomalies$
end procedure

Nelle seguenti pagine verranno riportati i grafici delle metriche con l'output dell'algoritmo appena descritto, valgono le stesse considerazioni sui colori fatte nel paragrafo 5.1

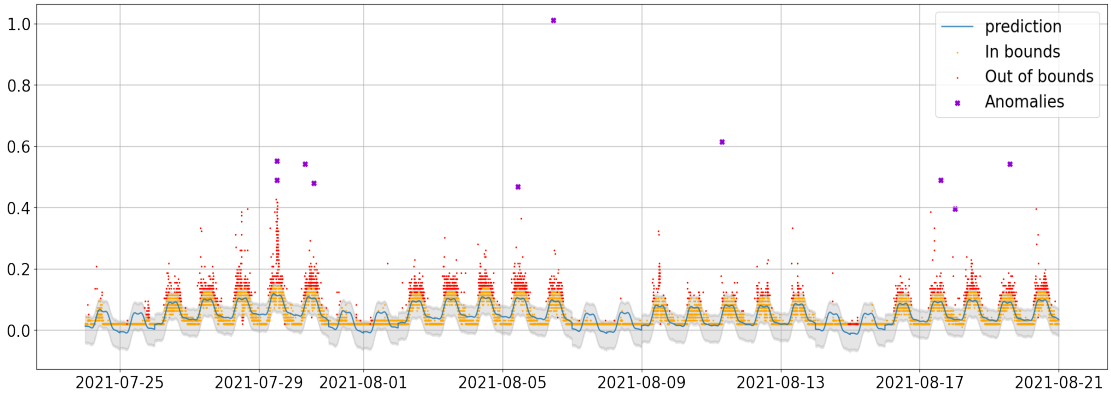

Figura 5.9: PROPHET anomaly detection sulla metrica *OpenSocketCurrentCount*

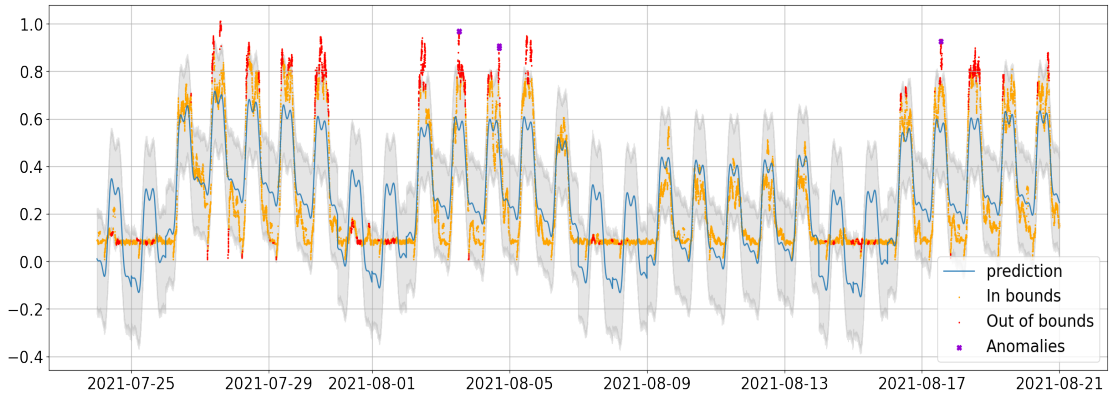

Figura 5.10: PROPHET anomaly detection sulla metrica *OpenSessionCurrentCount*

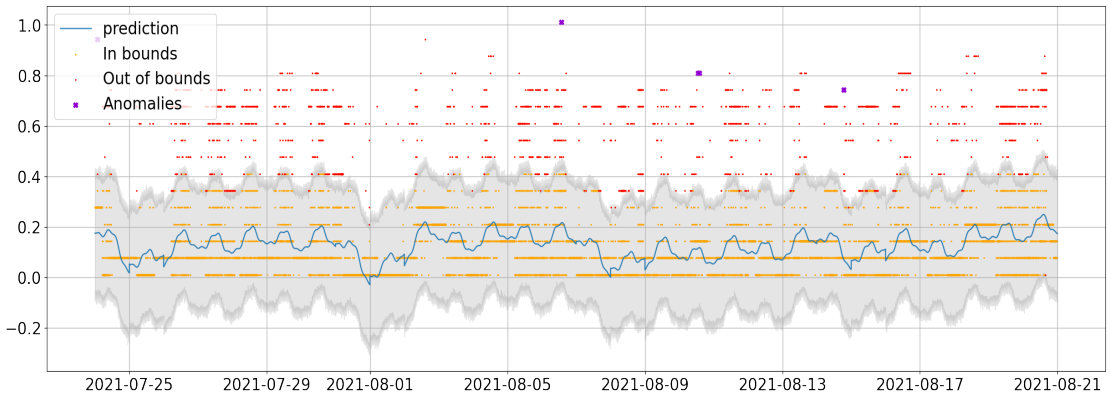

Figura 5.11: PROPHET anomaly detection sulla metrica *ExitThreadIdleCount*

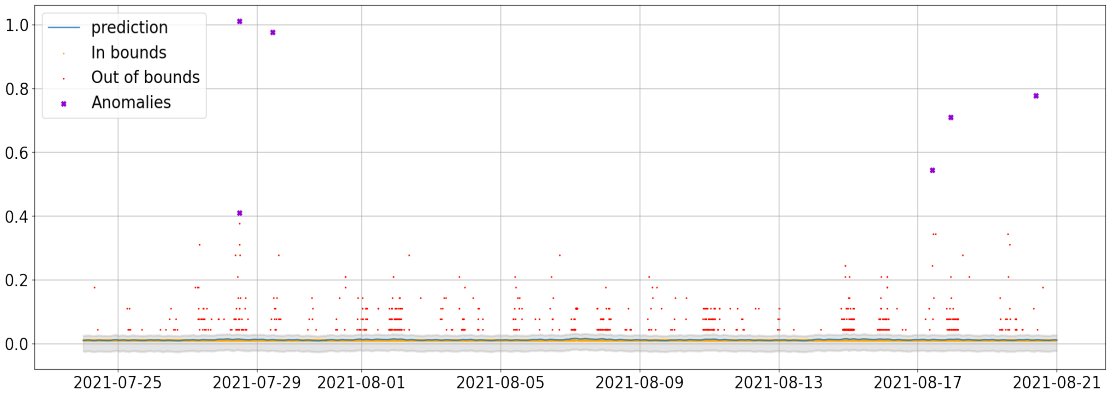

Figura 5.12: PROPHET anomaly detection sulla metrica *PendingUserCount*

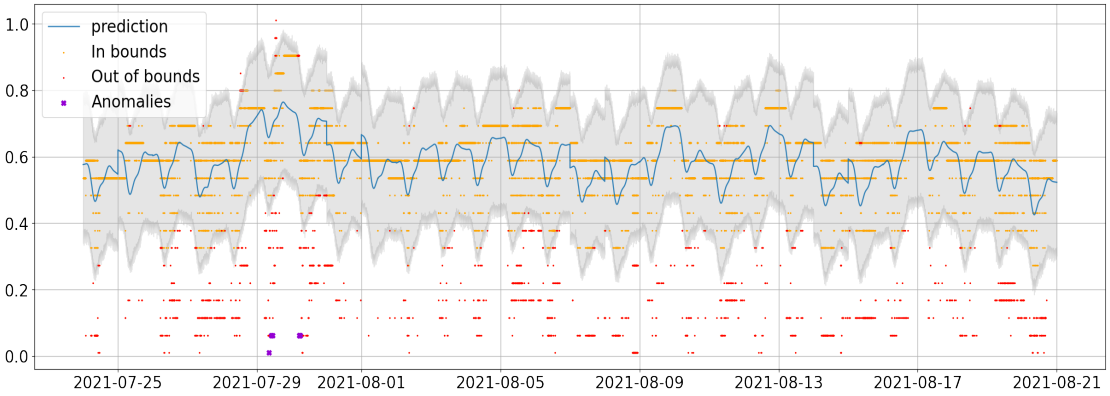

Figura 5.13: PROPHET anomaly detection sulla metrica *StandByThreadCount*

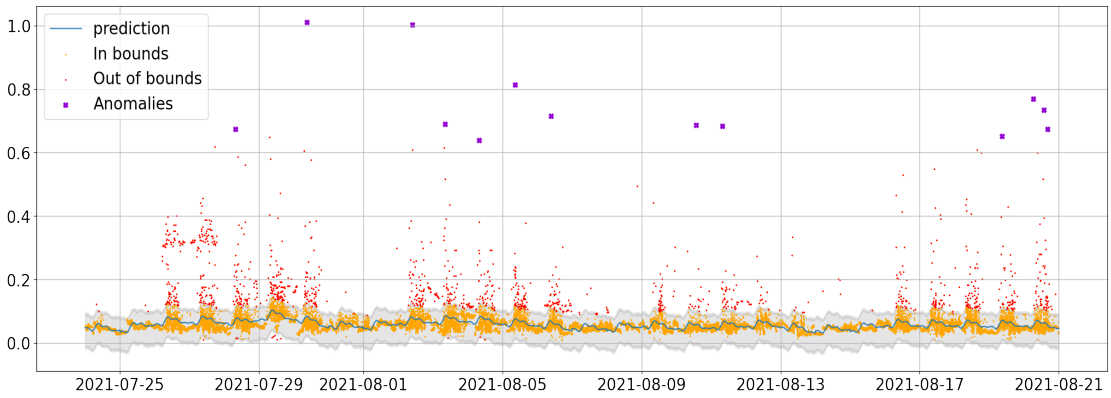

Figura 5.14: PROPHET anomaly detection sulla metrica *Throughput*

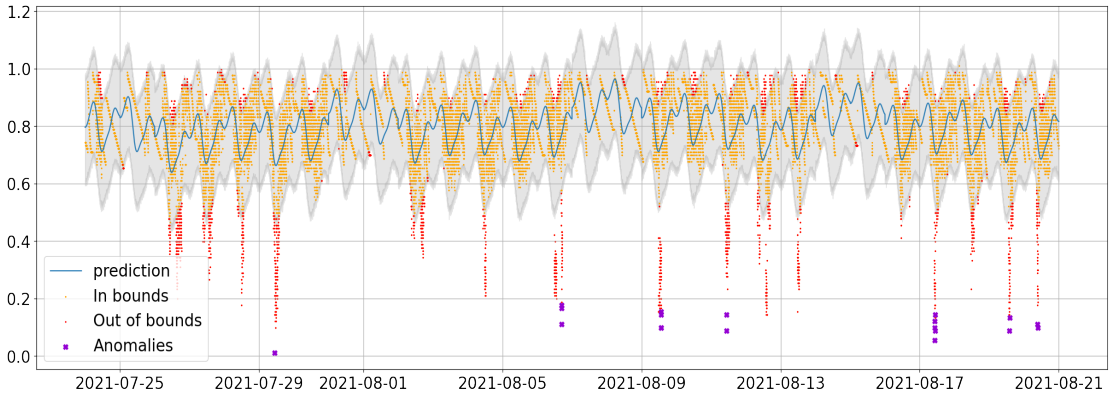

Figura 5.15: PROPHET anomaly detection sulla metrica *HeapfreePercent*

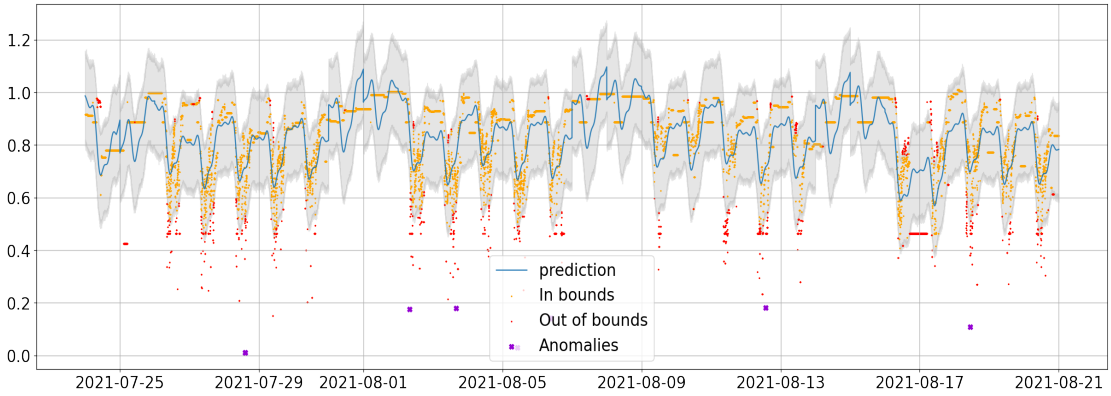

Figura 5.16: PROPHET anomaly detection sulla metrica *HeapSizeCurrent*

5.3 Anomaly detection con ISOLATION FOREST

L'ultimo modello per la point anomaly detection che è stato analizzato nell'ambito di questa tesi è basato sull'isolation forest.

Il procedimento è molto simile a quello adottato nei due modelli precedenti ma con un passaggio aggiuntivo fondamentale:

1. Normalizzazione della serie temporale
2. STL decomposition della serie temporale ricavando il residuo
3. ISOLATION FOREST sul residuo
4. Rimozione dei punti non considerati anomali dall'isolation forest
5. DBSCAN sui punti al di fuori della regione di confidenza per ottenere i cluster di punti vicini tra loro
6. I punti che non appartengono a nessun cluster sono le anomalie

L'isolation forest è un tipo di algortimo completamente diverso da ARIMA e PROPHET, infatti non ha l'obiettivo di trovare una funzione di fitting, ma cerca direttamente i punti anomali in base al parametro contamination (si veda 3.5).

Una nota importante è che l'isolation forest non tiene conto della dimensione temporale, ovvero dell'ordine e delle autocorrelazioni tra i campioni, ma considera solo i valori assunti dalle metriche.

Per questo motivo applicare l'isolation forest direttamente sulla serie temporale non aiuterebbe ad isolare i punti anomali ma isolerebbe semplicemente gli estremi superiori e inferiore della matrica.

Questo accade perchè l'isolation forest identifica le anomalie in base alla semplicità (intesa come numero di condizioni da soddisfare per raggiungere il punto negli alberi di decisione della foresta) di isolare i singoli punti, se viene applicato direttamente sulla metrica i punti più facilmente isolabili sono gli estremi (ovvero i picchi).

Isolare semplicemente gli estremi non risolve il problema dell'anomaly detection in quanto non è detto che un estremo lo sia, ad esempio un picco molto elevato

che però si ripete a intervalli regolari di tempo non deve essere classificato come anomalia dato che per definizione si tratterebbe di un comportamento stagionale e quindi atteso, allo stesso modo se il trend è in crescita si avranno valori sempre più elevati ma non necessariamente anomali.

Viceversa un picco singolo che non si ripete nel tempo dovrebbe essere categorizzato come anomalia, anche se avesse un valore inferiore rispetto al caso periodico precedente.

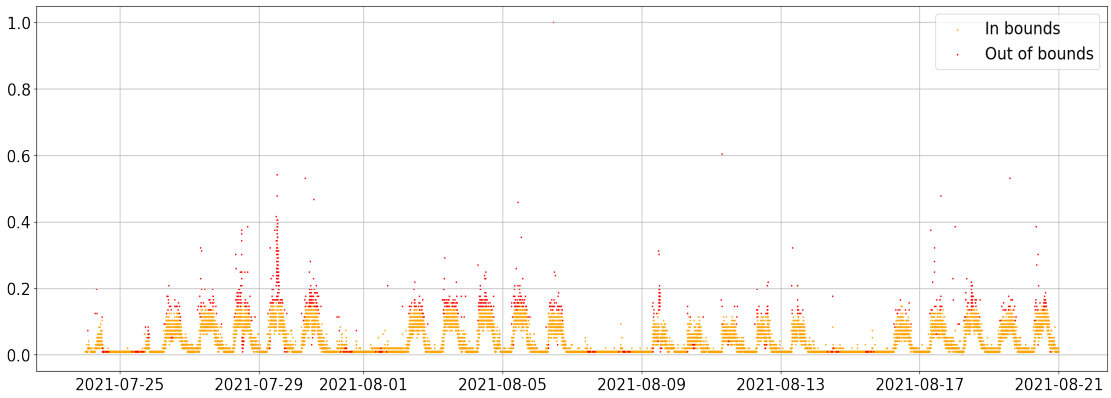

Figura 5.17: Risultato dell'isolation forest sulla metrica *OpenSocketCurrentCount*

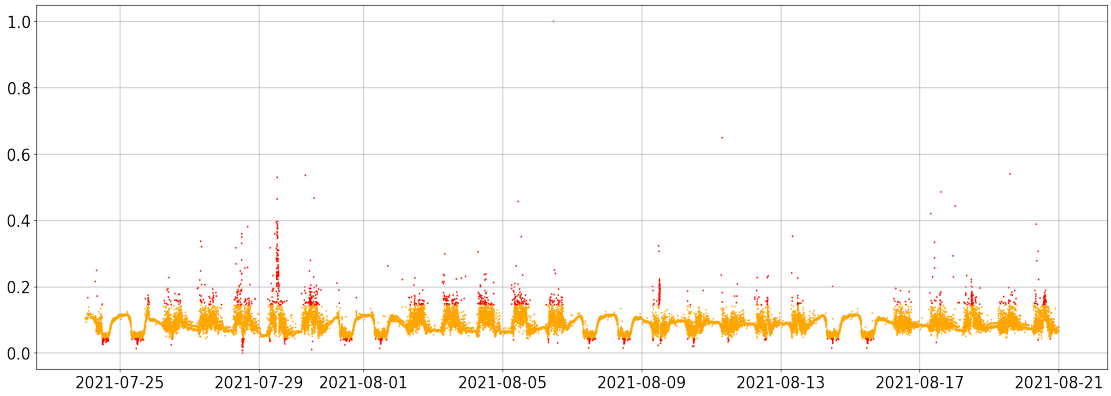

Figura 5.18: Risultato dell'isolation forest sul residuo della metrica *OpenSocketCurrentCount*

Per risolvere questo problema si più applicare l'isolation forest non sulla metrica

originale ma sulla componente di residuo del segnale, ottenuta tramite la scomposizione STL.

Infatti il residuo per definizione non contiene né la componente di trend del segnale, eliminando il problema dei picchi dovuti a trend crescenti (che non sono anomalie), né la componenete stagionale, risolvendo il problema di picchi periodici nel tempo.

Quindi, per esclusione, i punti anomali, come il picco "isolato" dell'esempio, dovrebbero essere contenuti solo nel residuo; Inoltre il residuo, in un segnale privo di anomalie, dovrebbe avere una distribuzione simile al rumore bianco con media nulla e varianza costante.

Questa caratteristica però non vale nel caso delle anomalie, le quali vengono riportate così come sono nei residui; di conseguenza, essendo il rumore bianco a media nulla, tanto più un punto è isolato nel residuo, tanto più è probabile che sia una anomalia.

In conclusione, applicando l'isolation forest sul residuo del segnale, verranno identificati gli estremi dello stesso che, come appena spiegato, corrisponderanno alle zone a media non nulla del residuo e di conseguenza alle zone non modellate da trend e varianza , ovvero le anomalie.

La figura 5.18 rappresenta il residuo della metrica *OpenSocketCurrentCount* rappresentata invece nella figura 5.17.

l'Isolation forest è stato applicato sul residuo e i punti considerati anomali sono stati segnati in rosso su quest'ultimo e poi riportati sul segnale originale.

Da queste figure è importante notare la media costante del residuo per la maggior parte della serie, tuttavia il 29 luglio si può osservare un picco sia in corrispondenza del residuo che della metrica originale.

Tale picco non è periodico e quindi deve essere considerato anomalo, tant'è che viene evidenziato di rosso dall'isolation forest.

I successivi passaggi del modello sono identici a quelli dei modelli precedenti, quindi viene applicato il DBSCAN e i punti anomali che non appartengono a nessun cluster vengono identificati.

Nel caso della point anomaly il picco del 29 luglio verrà poi scartato dalla clusterizzazione visto che presenta un'alta densità di punti ma sarà interessante analizzarlo nell'ambito della context anomaly detection.

Algorithm 5 ISOLATION FOREST based point anomaly detection

procedure ISOLATION_FOREST_ANOMALY_DETECTION (x, c, eps, m)
x è un vettore contenente tutti i campioni della serie temporale
c è il parametro contamination dell'isolation forest
eps è il raggio tramite il quale il DBSCAN calcola la densità
m è il numero campioni in un periodo
    $min \leftarrow$min(x)
    $max \leftarrow$max(x)
    for x[i] in x **do**
        $x[i] \leftarrow \frac{x[i]-min}{max-min}$
        ▷ Normalizzazione
    end for
    $resid \leftarrow STL\_decomposition(x, m)$
    ▷ Scomposizione segnale per ottenere i residui
    $candidates \leftarrow ISOLATION\_FOREST(resid, c)$
    ▷ candidates contiene i punti candidati ad essere anomalie
    $clusters \leftarrow DBSCAN(candidates, eps)$
    $anomalies \leftarrow []$
    for cluters[i] in clusters **do**
        if $(len(clusters[i]) == 1)$
            $anomalies.append(clusters[i][0]))$
        ▷ le anomalie sono i punti all'interno dei cluster costituiti da un solo punto
    end for
    $return\, anomalies$
end procedure

Nelle seguenti pagine verranno riportati i grafici delle metriche con l'output dell'algoritmo appena descritto, valgono le stesse considerazioni sui colori fatte in precedenza.

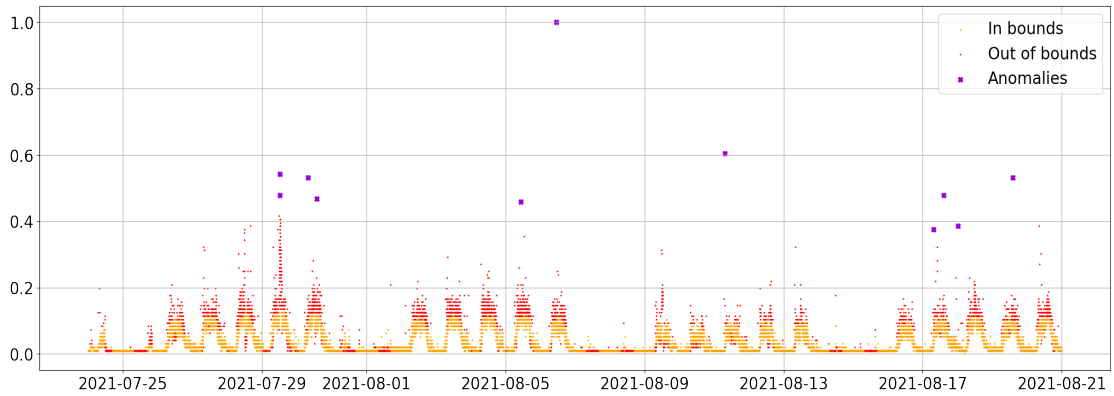

Figura 5.19: ISOLATION FOREST anomaly detection sulla metrica *OpenSocketCurrentCount*

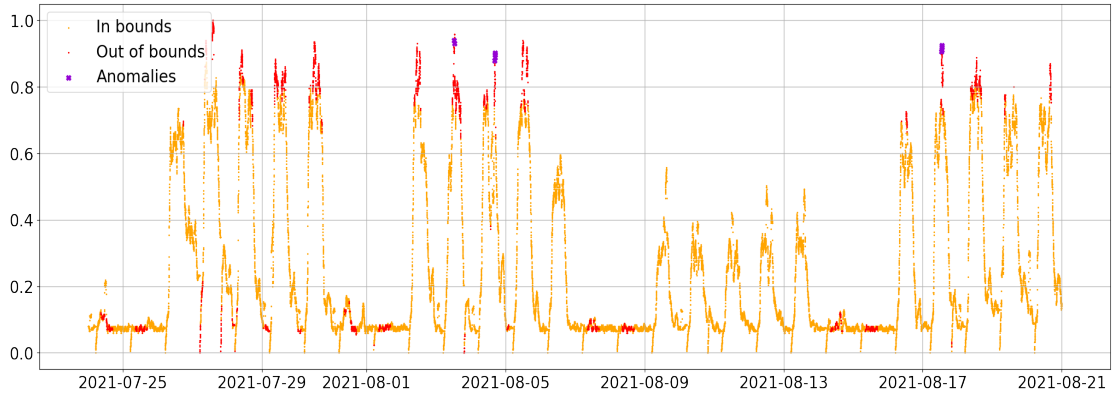

Figura 5.20: ISOLATION FOREST anomaly detection sulla metrica *OpenSessionCurrentCount*

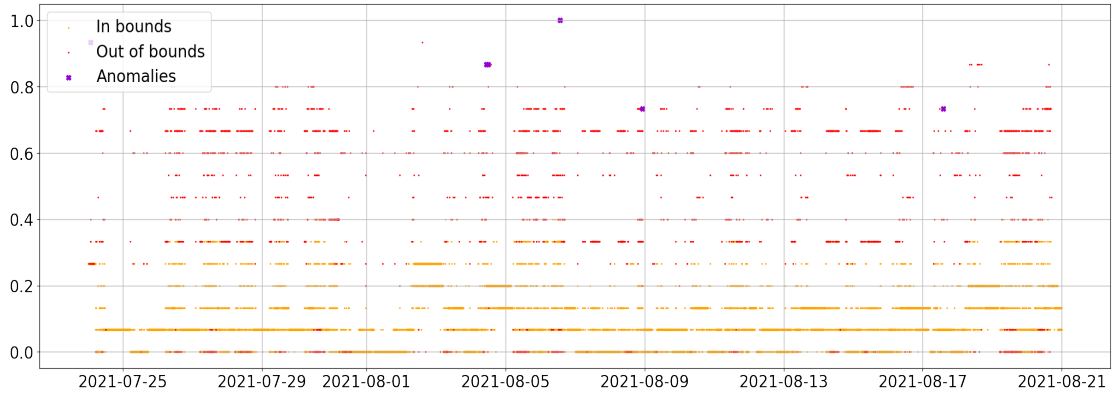

Figura 5.21: ISOLATION FOREST anomaly detection sulla metrica *ExitThreadIdleCount*

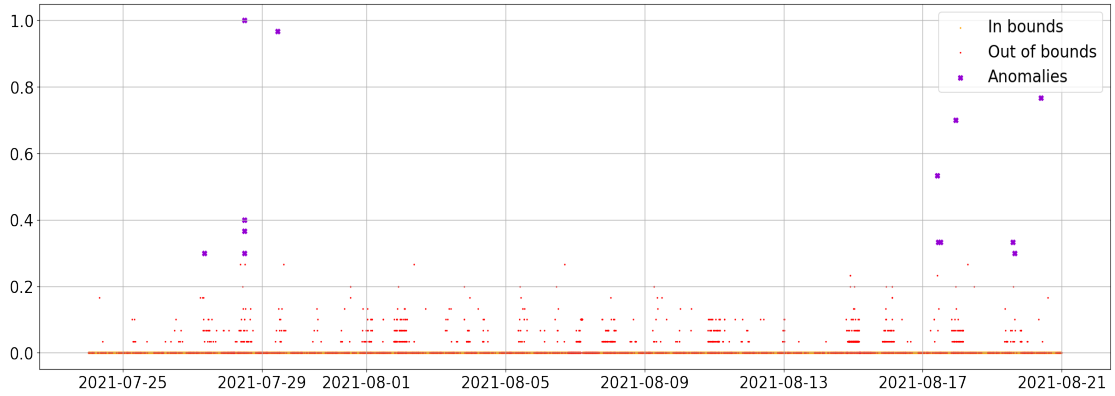

Figura 5.22: ISOLATION FOREST anomaly detection sulla metrica *PendingUserCount*

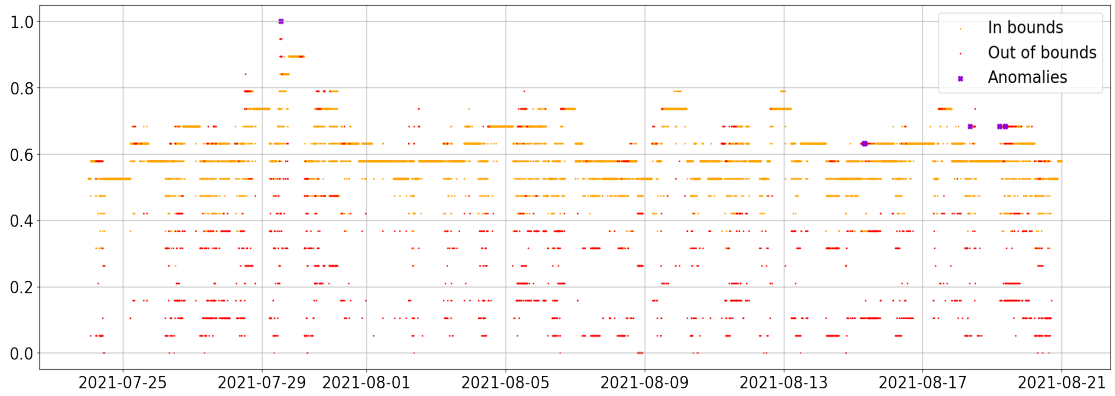

Figura 5.23: ISOLATION FOREST anomaly detection sulla metrica *StandByThreadCount*

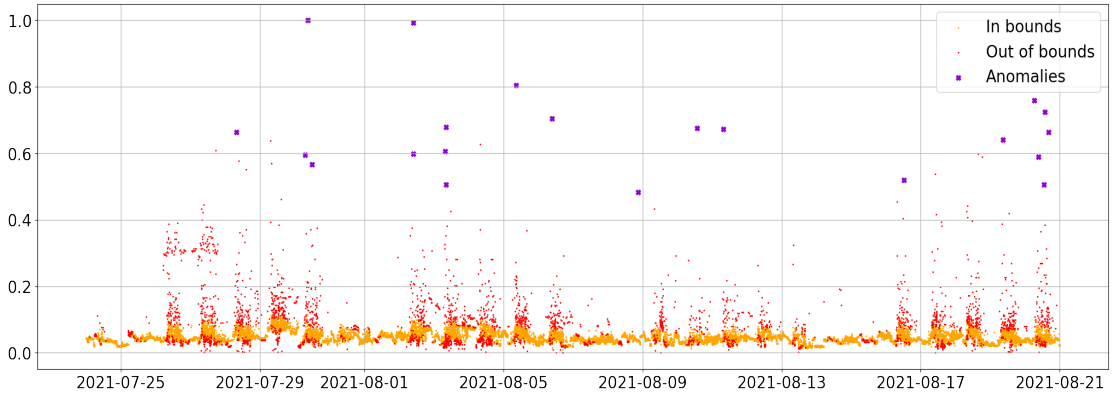

Figura 5.24: ISOLATION FOREST anomaly detection sulla metrica *Throughput*

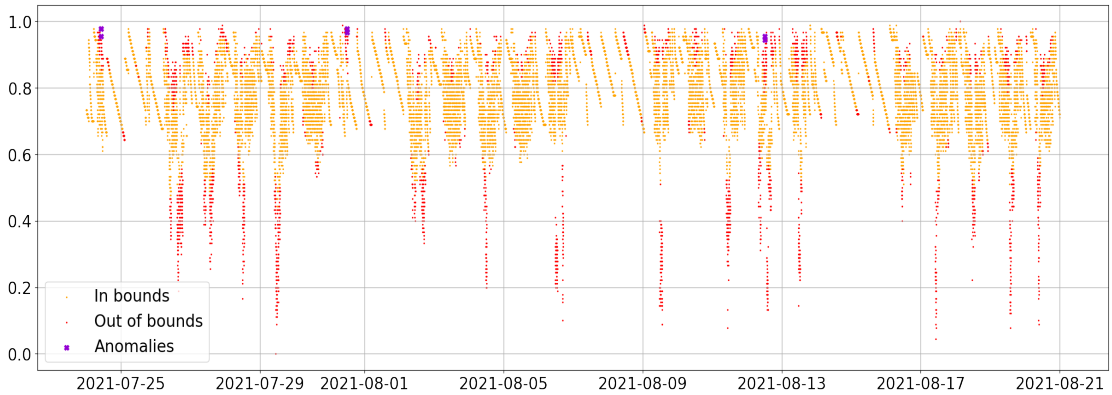

Figura 5.25: ISOLATION FOREST anomaly detection sulla metrica *HeapfreePercent*

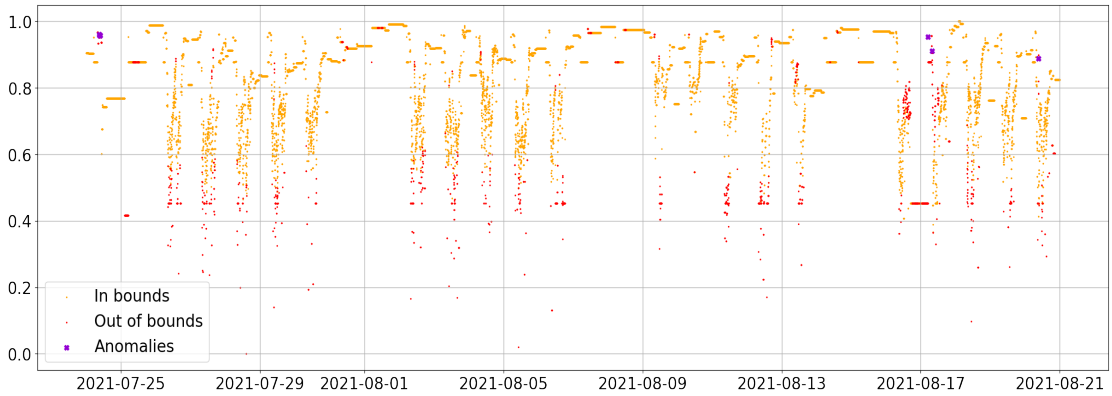

Figura 5.26: ISOLATION FOREST anomaly detection sulla metrica *HeapSizeCurrent*

5.4 Confronto tra i modelli di Point Anomaly Detection

A causa della natura non supervisionata del dataset, non è possibile trovare una metrica oggettiva per valutare l'accuratezza dei modelli.

L'unico modo di valutarne la correttezza è un'attenta valutazione da effettuare con gli esperti di settore, operazione che si traduce nell'efftuare il tuning dei parametri in modo da far combaciare l'output dei modelli con le situazioni che l'esperienza umana sostiene siano sintomo di un problema tecnico da risolvere.

Un'altra criticità fondamentale è la qualità del dataset di training, dove per qualità non si intende solamente il tipo di metriche raccolte, ma soprattutto la presenza di anomalie nel periodo di tempo in cui sono state storicizzate.

Nell'ambito dello sviluppo aziendale questi problemi non sono di facile risoluzione in quanto non si può conoscere a priori la quantità e la variabilità delle anomalie che si verificheranno.

Aumentare il tempo di campionamento ovviamente può mitigare questo problema ma ha come conseguenza un ritardo nello sviluppo della piattaforma e quindi possibili perdite economiche.

Nell'ambito di questa tesi, non avendo a disposizione il tempo necessario per effettuare una validazione reale dei dati e avendo a disposizione campionamenti per un periodo di tempo limitato, il tuning dei parametri è stato effettuato tramite la visualizzazione dei risultati, impostando i parametri dei modelli in modo da segnalare le anomalie che sono individuabili visivamente.

Nelle seguenti tabelle è possibile osservare un confronto tra i 3 modelli basati su ARIMA (A), PROPHET (P) e ISOLATION FOREST (I).

Nella tabelle viene indicato il numero di anomalie individuate giorno per giorno dai 3 modelli per ogni feature, le righe più scure indicano una corrispondenza per tutti i 3 modelli,quelle più chiare una corrispondenza tra due modelli.

Le feature nella tabella sono indicate con il relativo codice in modo da risparmiare spazio (si veda il capitolo 4)) ma l'ordine è il solito che si è mantenuto fin'ora quando sono state elencate le metriche.

	272			430			776			780		
TIMESTAMP	**A**	**P**	**I**	**A**	**P**	**I**	**A**	**P**	**I**	**A**	**P**	**I**
07-24			1					1	1			
07-25												
07-26												
07-27				5								1
07-28										3	2	3
07-29		2	1	1						1	1	1
07-30	2	2	2									
07-31												
08-01												
08-02							1					
08-03				2	4	2						
08-04					2	4			3			
08-05	1	1	1									
08-06	1	1	1				2	1	1			
08-07												
08-08							1		1			
08-09												
08-10								2				
08-11	1	1										
08-12												
08-13												
08-14								1				
08-15												
08-16												
08-17	2	1	1		1	4			1	4	2	3
08-18	1	1										
08-19	1	1	1							1		1
08-20										1	1	1

	782			784			908			909		
TIMESTAMP	**A**	**P**	**I**	**A**	**P**	**I**	**A**	**P**	**I**	**A**	**P**	**I**
07-24									3			13
07-25										2		
07-26	1											
07-27	1			1			1			1		
07-28				3	1	1	2			1	3	
07-29	1	3	1	2				1		1		
07-30	5	4		3	1	3						
07-31									5			
08-01												
08-02				2	1	2				1	1	
08-03				3	1	3				1	1	
08-04				1	1		2					
08-05				1	1	1				1	1	
08-06				1	1	1	3	3			5	
08-07												
08-08				1		1						
08-09							3	4		1		
08-10				1	1	1						
08-11				1	1	1	1	2				
08-12							2			2	1	
08-13	1											
08-14												
08-15			3									
08-16				2		1						
08-17	1			1			2	5		2		2
08-18			1	2			1				1	
08-19			4	1	1	1	2	2				
08-20				5	3	3		2				1

Come mostrato dalle tabelle, esiste una corrispondenza piuttosto forte tra i tre modelli, specie tra quello basato su ARIMA e quello basato su PROPHET.

Questo ha senso anche in virtù del fatto che i due modelli sono molto simili tra di loro, allo stesso tempo il modello basato su ISOLATION FOREST si discosta dai primi due in quanto opera principalmente sui residui del segnale e non sulla serie completa.

Un'altro confronto che vale la pena effettuare è quello sul tempo di esecuzione dei modelli, per ogni modello viene riportato il tempo di esecuzione medio per feature sulle 4 settimane di campioni:

- ARIMA: 33 secondi/feature (nota: non comprende il tempo necessario alla selezione dei parametri con auto arima)
- PROPHET: 123 secondi/feature
- ISOLATION FOREST: 5 secondi/feature

I tempi sopra indicati sono da considerarsi solo per un confronto relativo tra i tre modelli visto che sono stati eseguiti sulla piattaforma Google Colab senza un controllo dell'hardware assegnato.

Nonostante questo è chiaro che il modello più rapido è l'isolation forest; per questo motivo e per il fatto che l'analisi dei residui rimuove le complessità dovute alla presenza di trend e stagionalità, l'isolation forest sarà il modello di partenza per la Context anomaly detection che verrà descritta nella prossima sezione.

5.5 Context anomaly detection

Come già affermato nell'introduzione di questo capitolo, le anomalie di contesto non si riferiscono ad un singolo istante temporale ma interessano intervalli di tempo composti da campioni contigui.

L'individuazione delle anomalie contestuali è molto importante nell'ambito dell'analisi di metriche di performance in quanto esse rappresentano il caso su cui è effettivamente possibile implementare strategie di intervento proattive prima che il problema causi un rallentamento del sistema.

Se ad esempio si analizzasse la metrica relativa all'utilizzo di memoria, difficilemente si verificherà un'anomalia puntuale tale che la percentuale di utilizzo passi dal 50% al 100% in un singolo istante per poi ritornare in un valore normale subito dopo.

Sarebbe molto più probabile invece osservare un aumento di memoria graduale che dura più istanti di tempo successivi, sebbene questo comportamento dipenda anche dalla frequenza di campionamento con cui si storicizza la metrica, che deve essere abbastanza fine da poter osservare questo fenomeno.

L'approccio utilizzato per l'identificazione di questo tipo di anomalie riprende quello descritto nel modello di point anomaly detection basato su ISOLATION FOREST, i passaggi sono i seguenti:

1. Normalizzazione della serie temporale
2. STL decomposition della serie temporale ricavando il residuo
3. ISOLATION FOREST sul residuo
4. Rimozione dei punti non considerati anomali dall'isolation forest
5. Calcolo delle densità locali
6. Calcolo dei pesi locali
7. Calcolo dell' "anomaly score" sull'intervallo di tempo
8. Identificazione valori "anomaly score" sopra la treshhold

I primi quattro punti sono identici al modello appena nominato, dato che valgono le stesse considerazioni fatte sulla componente residuo del segnale, ovvero che quest'ultimo, per definizione, contiene le anomalie non modellate dalla componente

trend e stagionale.

Inoltre le anomalie, nel caso siano contestuali, saranno caratterizzate da picchi nella componente residuo che possono essere individuate applicando l'ISOLATION FOREST con un opportuno parametro di contamination (si faccia riferimento alle figure 5.17 e 5.18).

L'ISOLATION FOREST da solo non è in grado di identificare correttamente le anomalie contestuali, tuttavia è molto utile per restringere il campo e concentrasi sulle regioni più estreme del residuo.

Queste regioni, in generale, saranno molto rumorose, con parecchi punti non anomali vicini al segnale residuo.
Al contempo ci saranno altrettanti punti sparsi e isolati che non constituiscono anomalie contestuali ma sono più probabilmente anomalie puntuali (e quindi possono essere individuate dai modelli spiegati in precedenza)..

Occorre per cui fare delle valutazioni sia sulla distanza dei punti candidati ad essere anomali rispetto a quelli non anomali esclusi dall'isolation forest, sia sulla densità locale di quest'ultimi.

Per effettuare quest'operazione si calcolano le densità locali tramite un'approccio "a griglia", che consiste nel suddividere la serie temporale in rettangoli di dimensioni costanti sia lungo l'asse del tempo sia lungo l'asse del valore della metrica.1

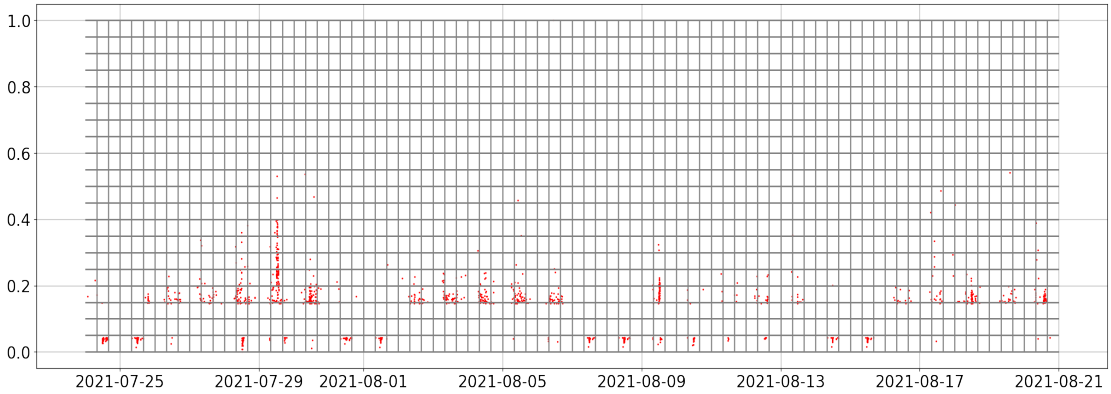

Figura 5.27: Griglia per il calcolo delle densità

Nella figura 5.27 le dimensioni della griglia sono di 8 ore per 0.05 unità, chiaramente questi valori costituiscono dei parametri di questo modello e vanno tunati

in base al contesto di applicazione.

Una volta costruita la griglia il modello conta il numero di campioni all'interno di ogni rettangolo, moltiplicando questo valore per la distanza del rettangolo dalla media del residuo in quell'intervallo.

Questo produce una matrice come quella nell'immagine 5.28 in cui le celle con valore più alto sono più scure.

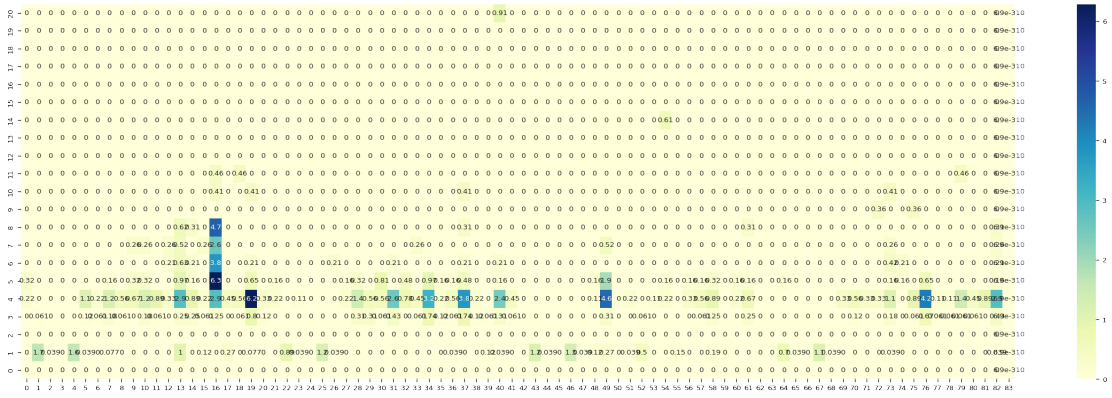

Figura 5.28: Pesi di ogni elemento della griglia

Tanto più grande il valore della cella tanto più alta la probabilità che quell'intervallo corrisponda a un'anomalia contestuale.

Il passo successivo consiste nel sommare i valori appena calcolati lungo l'asse delle y, ottenendo un vettore con "l'anomaly score" corrispondente ad ogni intervallo.

Infine occorre scegliere un soglia per determinare quali intervalli sono anomali in base al valore di "anomaly score".

Per rendere il modello più scalabile questo parametro viene impostato automaticamente al 95 percentile della distrobuzione dei pesi, ma questo valore può essere calcolato in modi diversi in base alla necessità oppure essere impostato manualmente.

Nella figura 5.29 si può osservare questo processo notando che gli intervalli che superano la soglia sono 3, con altri che si avvicinano molto, a riprova del fatto che il tuning dei parametri con affiancamento di esperti è fondamentale per evidenziare le anomalie corrette.

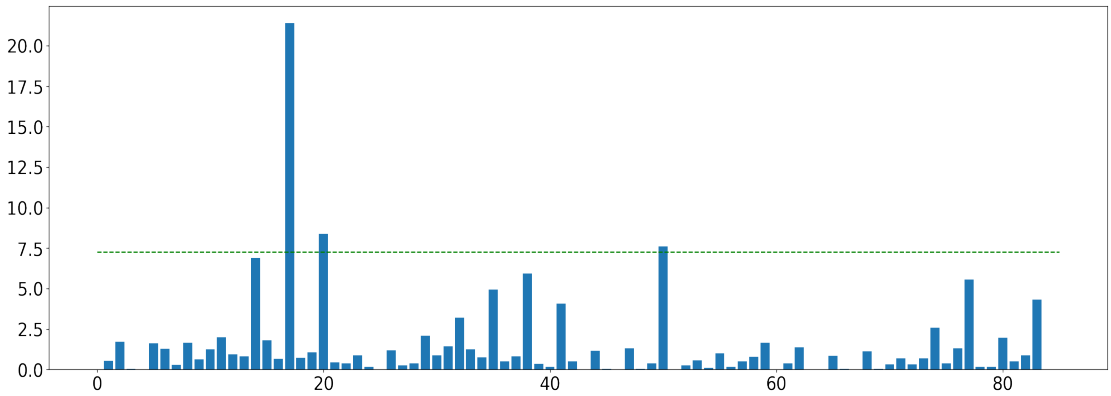

Figura 5.29: Pesi di ogni intervallo con soglia impostata al 95 percentile

Come ultimo passaggio si individuano gli estremi corretti degli intervalli, unendo gli intervalli contigui e selezionando i timestamp del primo e dell'ultimo punti anomali al loro interno.

Il risultato finale può essere osservato nella figura 5.30

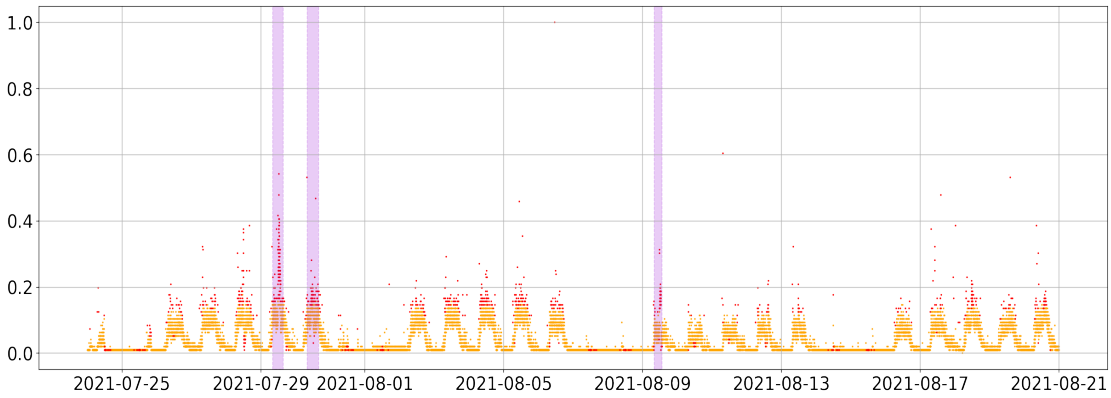

Figura 5.30: Risultato finale del modello di Context anomaly detection sulla metrica *OpenSocketCurrentCount*

Da questa immagine si può constatare qualitativamente l'efficacia del modello, osservando soprattutto i picchi del 07-29 e del 08-08 siano stati evidenziati.

Inoltre è interessante vedere come il giorno 07-28 non sia stato evidenziato ma che il suo anomaly score nella figura 5.29 sia molto vicino alla soglia e che quindi, se si ritenesse necessario etichettarlo come anomalia, basterebbe abbassare la soglia

senza necessariamente effettuare nuovamente il training del modello, rendendolo molto facilmente personalizzabile e scalabile.

Algorithm 6 ISOLATION FOREST based context anomaly detection

procedure CONTEXT_ANOMALY_DETECTION (x, c, s_x, s_y, p, m)
x è un vettore contenente tutti i campioni della serie temporale
c è il parametro contamination dell'isolation forest
s_x, s_y sono le rimensioni delle celle della griglia
m è il numero campioni in un periodo
$min \leftarrow$min(x)
$max \leftarrow$max(x)
for x[i] in x **do**
$x[i] \leftarrow \frac{x[i]-min}{max-min}$
▷ Normalizzazione
end for
$resid \leftarrow STL_decomposition(x, m)$
▷ Scomposizione segnale per ottenere i residui
$candidates \leftarrow ISOLATION\_FOREST(resid, c)$
▷ candidates contiene i punti candidati ad essere anomalie
$grid \leftarrow [\frac{len(x)}{size_x}, \frac{1}{size_y}]$
▷ Inizializza la griglia dei pesi con le dimensioni corrette
for i in range(0,len(grid[0]) **do**
for j in range(0,len(grid[1]) **do**
$gird[i, j] \leftarrow count\_elements(i, j) * |\frac{j}{100} - avg\_val(resid, j)|$
▷ count_elements(i,j) conta il numero di elementi nella cella i,j
▷ j/100 è il valore dell'estremo inferiore della cella i,j
▷ avg_val(resid,j) è il valore medio del residuo nell'intervallo corrispondente alla cella i,j
end for
end for
$anomaly\_scores \leftarrow [len(grid[0])]$
for i in range(0,len(grid[0]) **do**
$anomaly\_scores \leftarrow 0$
for j in range(0,len(grid[1]) **do**
$anomaly\_scores[i] \leftarrow anomaly\_scores[i] + grid[i, j]$
end for
end for

$intervals \leftarrow []$
$threshold \leftarrow perrcentile(anomaly_scores, p)$
▷ Calcolo del percentile sugli anomaly scores
for i in range(0,len(grid[0]) **do**
    if $anomaly\_scores[i] \geq threshold$ **then**
    $intervals.append(get\_interval(i))$
    ▷ get_interval(i) restituisce i timestamp corrispondenti agli estremi dell'intervallo con indice i
end for
$l \leftarrow len(grid[0]) - 1$
for i in range(0,l) **do**
    if $intervals[i].upperintervals[i+1].lower$ **then**
    $intervals[i].upper = intervals[i+1].upper$
    $intervals.remove(i+1)$
    $i \leftarrow i - 1$
    ▷ Unione degli intervalli contigui
end for
$anomalies \leftarrow []$
for i in range(0,len(intervals) **do**
    $lower = min(candidates[intervals[i].lower, intervals[i].upper].timestamps)$
    $upper = min(candidates[intervals[i].lower, intervals[i].upper].timestamps)$
    $anomalies.append((lower, upper))$
    ▷ recupera gli estremi degli intervalli dall'output dell'isolation forest
end for
return anomalies
EndProcedure =0

Nelle seguenti pagine verranno riportati i grafici delle metriche con l'output dell'algoritmo di Context Anomaly detection, in rosso i punti candidati ad essere anomalie, in giallo quelli esclusi dall'isolation forest e le bande viola evidenziano gli intervalli anomali trovati.

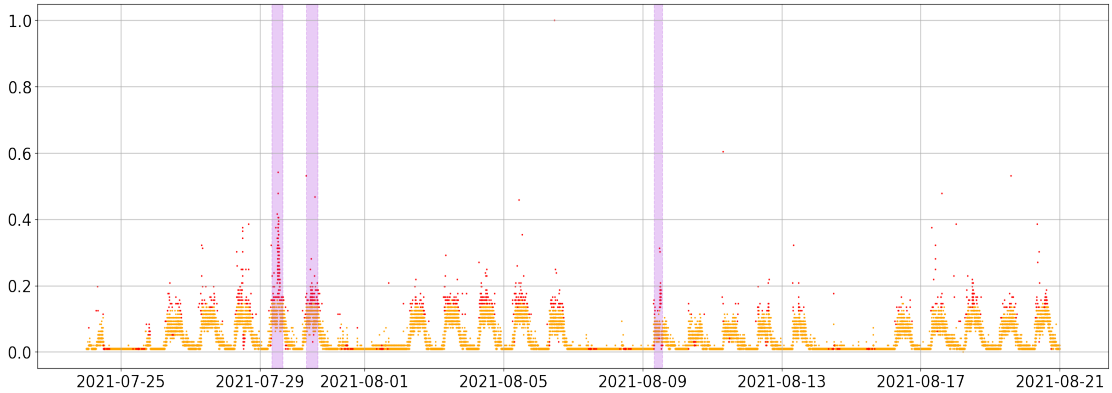

Figura 5.31: Context anomaly detection sulla metrica *OpenSocketCurrentCount*

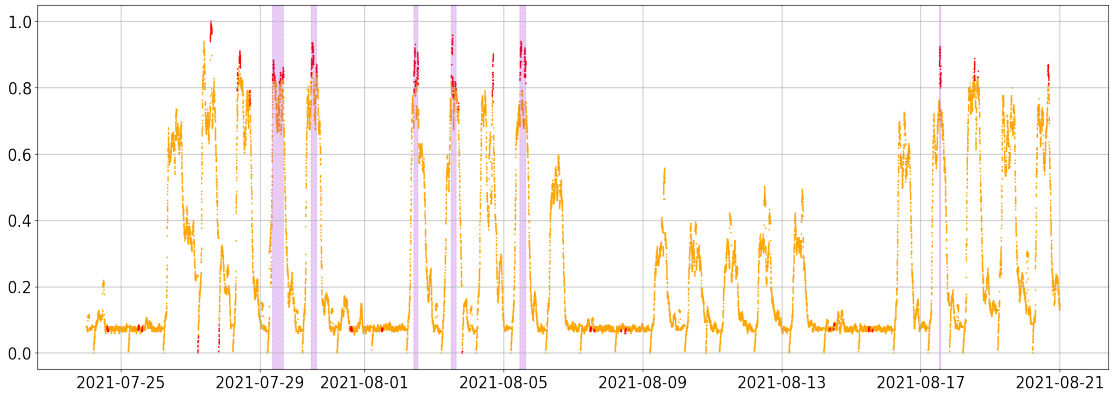

Figura 5.32: Context anomaly detection sulla metrica *OpenSessionCurrentCount*

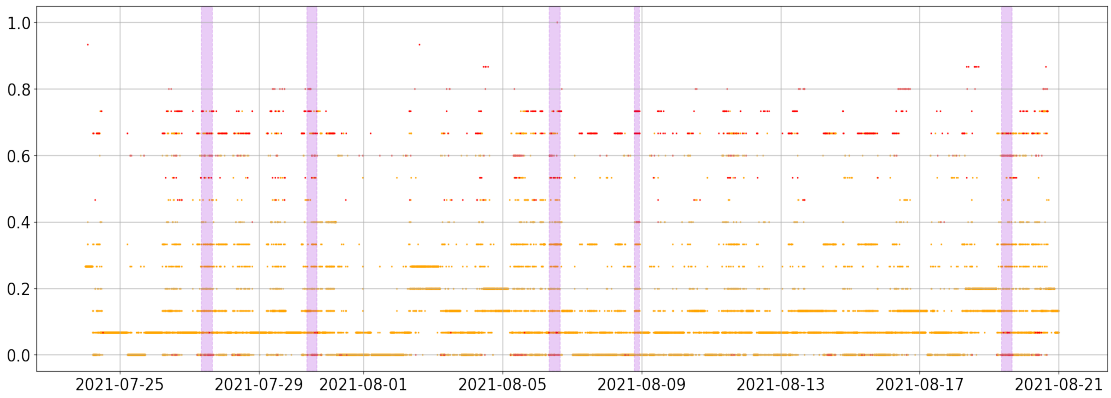

Figura 5.33: Context anomaly detection sulla metrica *ExitThreadIdleCount*

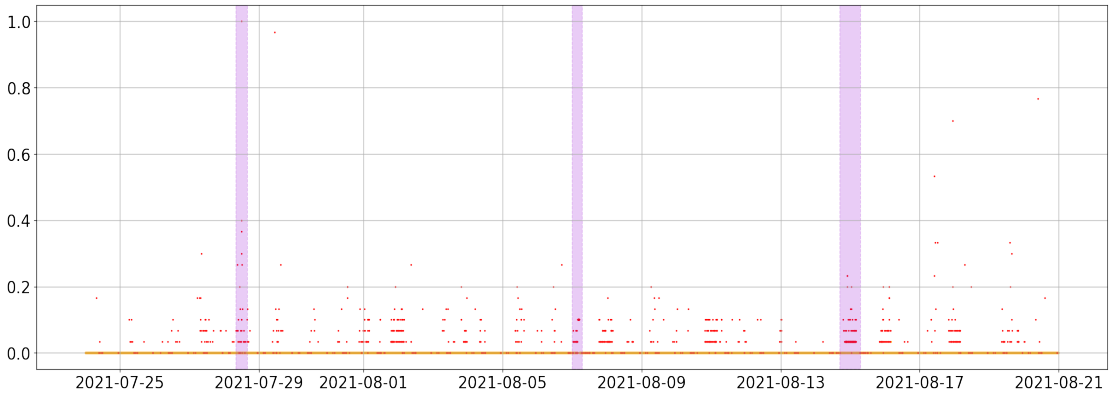

Figura 5.34: Context anomaly detection sulla metrica *PendingUserCount*

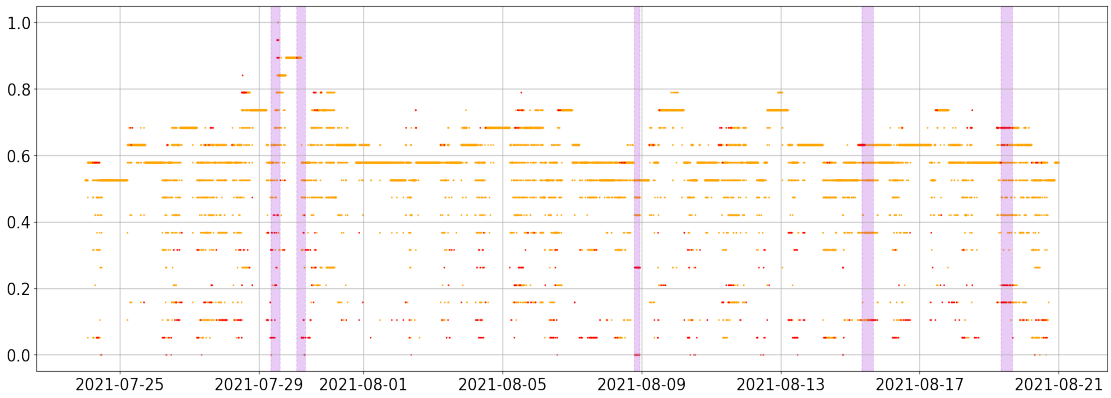

Figura 5.35: Context anomaly detection sulla metrica *StandByThreadCount*

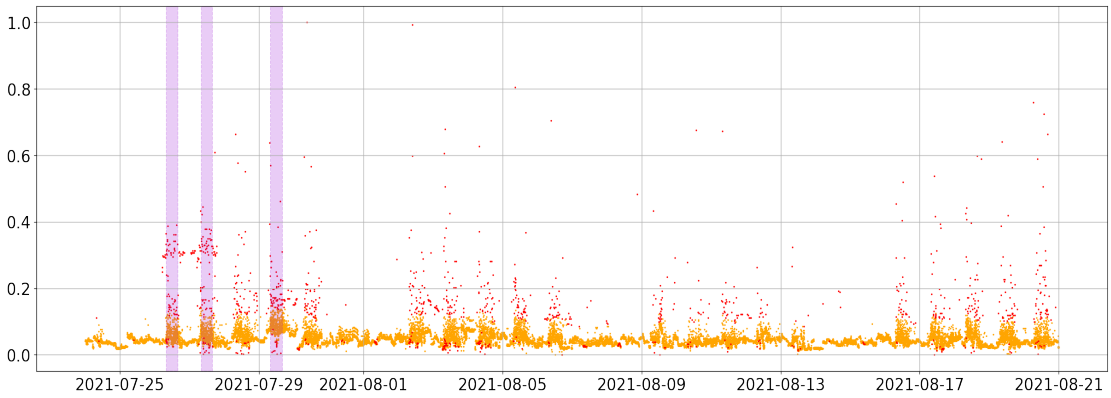

Figura 5.36: Context anomaly detection sulla metrica *Throughput*

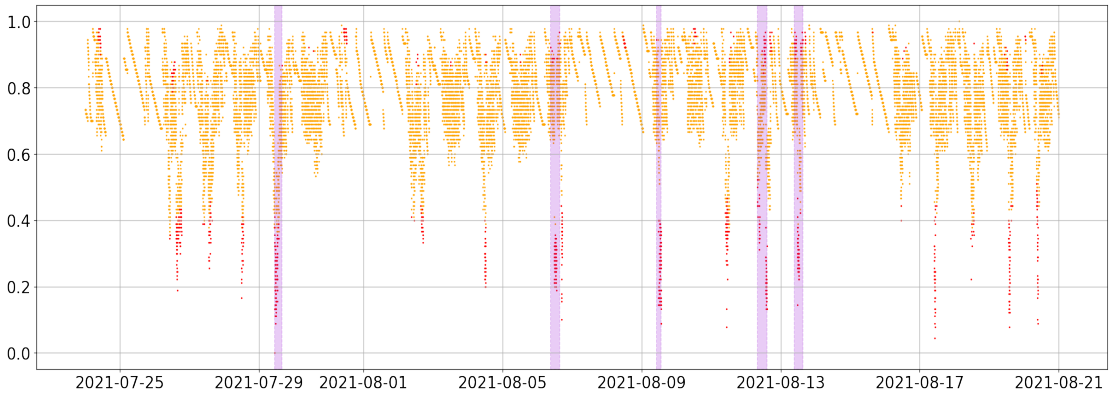

Figura 5.37: Context anomaly detection sulla metrica *HeapfreePercent*

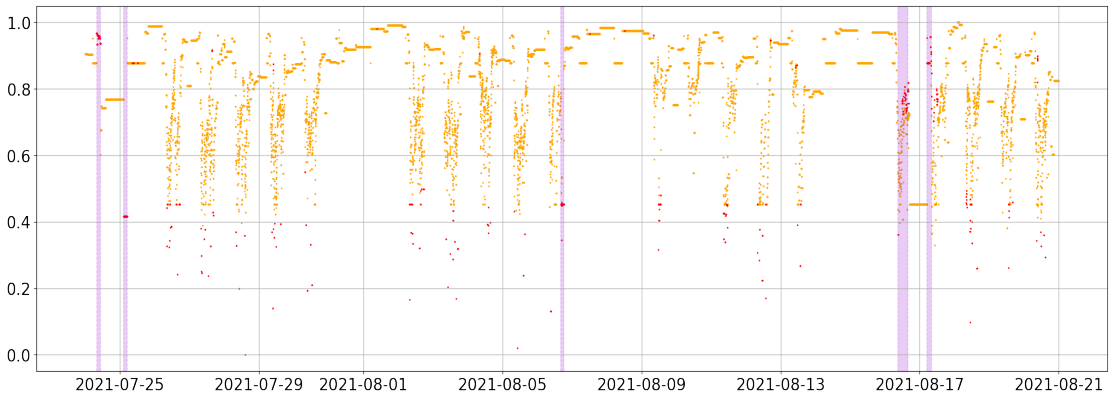

Figura 5.38: Context anomaly detection sulla metrica *HeapSizeCurrent*

Conclusioni

6.1 Sviluppi futuri

Lo studio e lo sviluppo dei modelli descritti all'interno di questa tesi non sono che una base di partenza per la risoluzione del problema dell'anomaly detection.

Per poter utilizzare i modelli in maniera efficace occorre prima di tutto creare un'infrastruttura in grado di prelevare in modo automatico i dati relativi alle metriche ed effettuare tutto il preprocessing necessario; definendo quali metriche selezionare, il loro periodo di campionamento, la dimensione del training set e la sua frequanza di "svecchiamento" con conseguente retraining dei modelli.

Quest'operazione non è banale nè dal punto di vista tecnico nè dal punto di vista di interfacciamento con i clienti vista la complessità nello spiegare perchè l'impiego dei modelli di machine learning porterebbe portare un grande vantaggio rispetto ai metodi tradizionali di manutenzione.

Un altro aspetto critico da affrontare nell'utilizzo aziendale di questi algoritmi è la scalabilità; per essere efficace un modello implementato in un processo aziendale deve essere in grado di lavorare con metriche potenzialmente molto diverse tra di loro con intervento minimo in termini di tuning dei parametri.

Inoltre sarebbbe necessario sviluppare un tool, da integrare all'interno degli strumenti già utilizzati a livello aziendale, che sia in grado di notificare chi ha il compito di effettuare manutenzione dei server in anticipo, senza che sia il cliente a dover notificare un malfunzionamento, migliorando di conseguenza la qualità del servizio e semplificando le azioni da intraprendere in caso di problemi.

Un possibile sviluppo futuro potrebbe quindi essere quello di studiare tecniche che possano automatizzare o quantomeno semplificare la selezione dei parametri del

modello.

Un'ulteriore ambito di approfondimento, degno senza dubbio di una tesi a sè stante, è quello della feature selection, ovvero dell'applicare tecniche per ridurre la dimensionalità del dataset.

Nell'ambito di questa tesi non vi è stata la necessità di effettuare una selezione delle feature raffinata visto lo scarso numero di feature disponibili.

Se però fosse stato possibile avere a disposizione più metriche selezionabili sarebbe stato fondamentale effettuare un'analisi ulteriore, rimuovendo metriche ridondati, raggruppando le metriche per categoria, per similarità di andamento, per correlazione etc...

Anche a livello di modelli moltissime strade sono ancora esplorabili, ad esempio visto i pochi dati disponibili non è stato possibili applicare in modo efficace tecniche più avanzate basate sul deep learning.

Tuttavia in quest'ambito i modelli che possono gestire bene le serie temporali sono svariati, ad esempio i modelli basati su LSMT (Long short term memory cell) che sono stati studiati specificatamente per gestire dati temporali, oppure gli autoencoder che possono estrarre le informazioni fondamentali dai dati in input e quindi potrebbero in linea teorica essere utilizzati per identificare le anomalie all'interno di un segnale visto che per definizione i comportamenti anomali non dovrebbero far parte della rappresentazione fondamentale della serie temporale.

Un'ulteriore approfondimento aggiuntivo che potrebbe avere ricadute positive potrebbe essere quello di passare da un'analisi monovariata a una multivariata, considerando più metriche contemporaneamente e le relazioni che intercorrono tra di esse, anche se questo aggiungerebbe ulteriori complessità dovute all'interpretabilità dei risultati.

6.2 Considerazioni finali

In conclusione, l'attività di tesi aziendale è stata estremamente formativa e interessante, soprattutto vista la volontà dell'azienda di innovare e di inserire quanto ricercato all'interno dei suoi processi di produzione.

L'anomaly detection, e più in generale tutte quelle tecnologie che ricadono sotto il nome di "machine learning", stanno assumendo sempre più importanza nella nostra società ed economia e non possono essere ignorate.

In questa tesi si è esplorata solo una piccola parte di quanto si potrebbe studiare e applicare, tuttavia è certamente stato un ottimo punto di partenza per effettuare il passaggio da mondo universitario a mondo del lavoro, unendo alle necessità "pragmatiche" dell'ultimo l'innovazione e la voglia di scoprire del primo, sintetizzando il meglio dei due mondi.

Bibliografia

[1] *An Oracle DBA's Guide to WebLogic Server*. https://oracle-base.com/articles/misc/an-oracle-dbas-guide-to-weblogic-server (cit. a p. 4).

[2] *Oracle 10g Database Structure*. https://dbainseconds.wordpress.com/2012/02/08/oracle-10g-database-structure/ (cit. a p. 7).

[3] Hyndman R.J.; Athanasopoulos G. *Forecasting: Principles and Practice*. https://otexts.com/fpp2/. 2018 (cit. a p. 11).

[4] Dario Radečić. *Time Series From Scratch — Decomposing Time Series Data*. https://towardsdatascience.com/time-series-from-scratch-decomposing-time-series-data-7b7ad0c30fe7. 2021 (cit. a p. 11).

[5] Robert B Cleveland; William S. Cleveland; jean E. McRae; Irma Terpenning. *STL : A Seasonal-Trend Decomposition Procedure Based on Loess*. https://www.scb.se/contentassets/ca21efb41fee47d293bbee5bf7be7fb3/stl-a-seasonal-trend-decomposition-procedure-based-on-loess.pdf. 1990 (cit. a p. 13).

[6] William S. Cleveland. *Robust Locally Weighted Regression and Smoothing Scatterplots*. :http://www.jstor.org/stable/2286407. 1979 (cit. a p. 14).

[7] Selva Prabhakaran. *ARIMA Model – Complete Guide to Time Series Forecasting in Python*. https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/. 2021 (cit. a p. 17).

[8] Letham B. Taylor SJ. *Forecasting at scale*. https://doi.org/10.7287/peerj.preprints.3190v2. 2017 (cit. a p. 27).

[9] *Distribuzione di Laplace*. https://amp.it.what-this.com/6351830/1/distribuzione-di-laplace.html (cit. a p. 31).

[10] Kai; Ming Ting Fei Tony; Liu. *Isolation Forest*. https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/icdm08b.pdf?q=isolation-forest. 2008 (cit. a p. 34).

[11] Eugenia Anello. *Anomaly Detection With Isolation Forest*. `https://betterprogramming.pub/anomaly-detection-with-isolation-forest-e41f1f55cc6`. 2020 (cit. alle pp. 34, 36).

[12] Anant Ram; Sunita Jalal; Anand S. Jalal; Manoj Kumar. *A Density based Algorithm for Discovering Density Varied Clusters in Large Spatial Databases*. `https://www.researchgate.net/publication/44250717_A_Density_Based_Algorithm_for_Discovering_Density_Varied_Clusters_in_Large_Spatial_Databases`. 2010 (cit. a p. 38).