UNIVERSITÀ DI PISA

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

Rilevazione di Correlazioni tra Serie Temporali

Relatore:
Luca Deri

Candidato:
Nicola Coltelli

ANNO ACCADEMICO 2020/2021

Sommario

In questa tesi viene proposto un algoritmo per l'individuazione di correlazioni tra serie temporali distinte, utilizzando come ausilio l'individuazione di anomalie. L'algoritmo fa uso, come base di partenza, di metodologie e di parti di altri algoritmi analizzati, ma modificati e rielaborati in modo da realizzare uno strumento efficiente, in particolare su un alto numero di serie temporali parallele, e che non presenti prerequisiti sulle serie temporali da analizzare.

Indice

1 Introduzione

La rilevazione di correlazioni è uno degli aspetti più importanti nell'analisi delle serie temporali. In uno scenario reale complesso, dove molteplici serie temporali sono monitorate, l'osservazione manuale da parte di un operatore umano dovrebbe essere evitata in quanto prona ad errori o addirittura impossibile a causa del vasto quantitativo di dati; l'intervento umano dovrebbe iniziare con l'analisi di dati già elaborati, i quali danno un feedback immediato su possibili problemi all'interno del sistema. Se, ad esempio, un'anomalia viene rilevata all'interno di un oggetto monitorato, semplicemente segnalarla all'operatore non è sufficiente, in quanto esso dovrà verificare l'entità del danno che l'anomalia potrebbe aver causato sull'intero sistema. Attualmente ci sono molti strumenti e algoritmi che permettono di individuare queste correlazioni, ma nessuno di essi è capace né di sfruttare la rilevazione di comportamenti anomali, né di analizzare i dati in tempo reale.

1.1 Obiettivo

L'obiettivo di questa tesi è definire un algoritmo che permetta di analizzare in modo parallelo molteplici serie temporali, individuando correlazioni tra di esse ed eventuali anomalie al loro interno. L'algoritmo deve permettere di effettuare analisi in tempo reale, per cui deve rispettare requisiti di efficienza tali da permettere l'elaborazione di serie temporali parallele. L'algoritmo non vuole stabilire un rapporto di causa effetto nel caso che una correlazione sia individuata, ma segnalare in modo efficace quali oggetti monitorati risultano simili tra loro, e hanno subito conseguenze a causa di uno stesso evento.

2 Stato dell'Arte

2.1 Luminol

Luminol[1] è una libreria scritta in Python per l'analisi di serie temporali utilizzata da LinkedIn. La libreria non necessita di nessuna soglia predefinita riguardo i valori della serie temporale, ma l'algoritmo proposto prevede l'assegnazione di uno score a ciascun punto della serie[2]. L'assegnazione è effettuata mediante il metodo *bitmap detector*[3]. Inizialmente viene generato un *Symbolic Aggregate approXimation*[4] (SAX) della serie temporale, che viene poi scorso tramite due finestre temporali e utilizzato per costruire un dizionario di *chunks*, dei quali viene conteggiata la frequenza.

La generazione del SAX avviene prendendo il valore minimo e massimo della serie temporale, stabilendo così il range dei valori. A seconda della precisione desiderata, il range viene diviso in parti uguali: maggiore è la precisione richiesta, più sezioni sono create. Di seguito viene assegnata una lettera per ogni sezione, generalmente partendo con "A" per la sezione con valori minori. Le lettere sono quindi assegnate a gruppi di valori adiacenti nella serie temporale, basandosi sulla

loro media, oppure ai singoli valori, come avviene in Luminol.

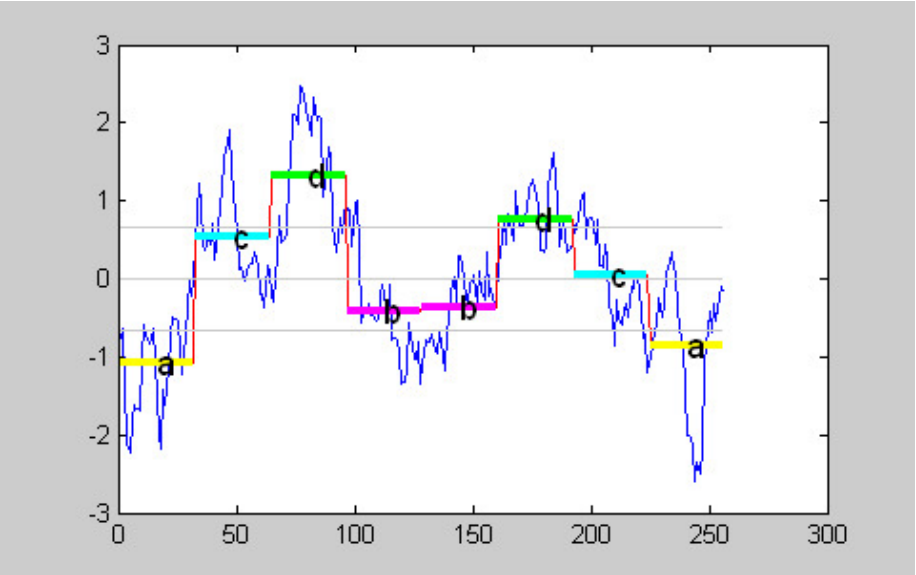


Figura 1: Approssimazione di una serie temporale tramite SAX.

Il dizionario di chunks viene costruito utilizzando due finestre temporali adiacenti: la *lagging window* e la *future window*. Queste finestre scorrono la serie temporale, e per ogni scorrimento viene creato un indice nei rispettivi dizionari, il quale memorizza il numero di istanze di ogni chunk.

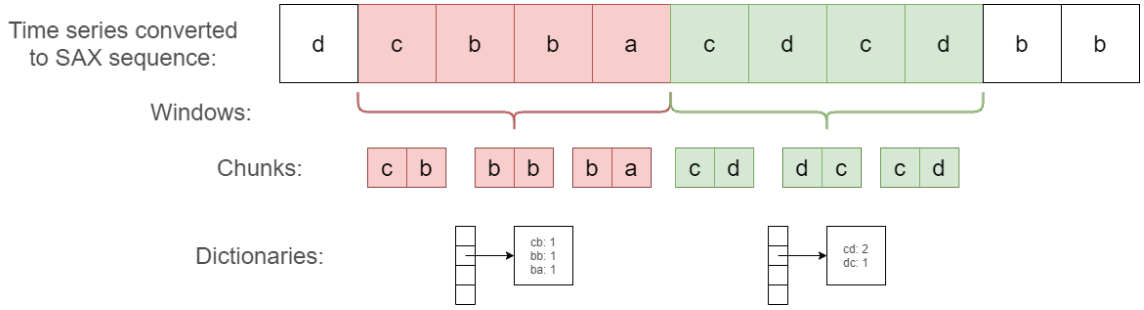


Figura 2: Creazione di un indice nei due dizionari.

A questo punto viene calcolato lo score di ogni punto computando la distanza tra le frequenze dei chunks delle due finestre posizionate al momento del timestamp del punto, e se lo score è maggiore di una soglia, il punto (o l'insieme di punti) è classificato come anomalia. La distanza, per ogni chunk del punto, viene calcolata effettuando la differenza tra la frequenza del chunk nella lagging window e la frequenza del chunk nella future window elevata alla seconda; lo score del punto sarà dato dalla somma di tutte le distanze dei chunk presenti nella due finestre di quel punto. Il procedimento è ripetuto per ogni punto della serie temporale.

Una volta classificate le anomalie, si può procedere ad effettuare la correlazione con un'altra serie temporale tramite l'algoritmo *Cross Correlation*[5]. La correlazione avviene prendendo due serie temporali, o parti di esse, e facendo scorrere temporalmente una delle due. Per ogni delay temporale, viene calcolata la cross correlation r tramite la seguente formula:

$$\frac{\sum_i[(x(i) - mx) * (y(i - d) - my)]}{\sqrt{\sum_i(x(i) - mx)^2}\sqrt{\sum_i(y(i - d) - my)^2}}$$

dove x(i) è un punto della prima serie, y(i-d) un punto della seconda

serie che differisce temporalmente da x(i) per un delay d, mx è la media dei valori della prima serie temporale, e my della seconda.

Il problema di questa metodologia è che prevede di avere tutta la serie a disposizione e non può, quindi, funzionare in "modalità online". Tuttavia, l'algoritmo potrebbe essere modificato per ovviare a questo problema, ma soffrirebbe comunque di un consistente lag tra produzione dei dati e individuazione dell'anomalia. A seconda del tipo dei dati e analizzati, l'algoritmo potrebbe inoltre risultare impreciso in quanto le finestre utilizzate per individuare le anomalie devono essere molto grandi per garantire il buon funzionamento dell'algoritmo.

Dal punto di vista della complessità temporale, l'algoritmo, per ogni serie temporale, richiede un primo passaggio per la creazione del dizionario, ed un secondo passaggio per il calcolo dello score. La complessità temporale della generazione del dizionario dipende dall'implementazione, ma è comunque (prendendo m come dimensione delle finestre temporali, $O(1)$ come complessità al caso medio per l'inserimento in un dizionario e $O(m)$ come complessità al caso pessimo per l'inserimento in un dizionario) compresa tra $O(m)$ e $O(m^2)$ per ciascun punto della serie temporale, e quindi compresa tra $O(n \cdot m)$ e $O(n \cdot m^2)$ per tutti i punti della serie temporale. La complessità temporale del

calcolo dello score comprende lo scorrimento di due indici dei dizionari, che sono a loro volta altri dizionari aventi come chiavi i chunks e come valori la rispettiva frequenza; per quanto anche queste operazioni dipendano dall'implementazione, lo scorrimento di un intero dizionario ha complessità $O(m)$, e per ogni chiave deve essere effettuata una ricerca nell'altro dizionario che ha complessità compresa tra $O(1)$ e $O(m)$, rendendo la complessità totale di questa operazione compresa tra $O(2m)$ e $O(2m^2)$, quindi compresa tra $O(m)$ e $O(m^2)$. Per quanto al caso medio queste operazioni abbiano complessità asintotica inferiore a quadratica, l'accesso in scrittura o in lettura di un dizionario non ha necessariamente un costo di tempo trascurabile e potrebbe generare problemi di performance in caso di molteplici serie temporali, in particolar modo se vengono usate dimensioni di finestre temporali (m) alte.

Dal punto di vista della complessità dello spazio, la generazione del SAX per ogni punto richiede di raddoppiare lo spazio occupato dalle serie temporali. La creazione dei dizionari richiede, al caso ottimo (tutti i chunks sono uguali) $O(n)$, dove n è la dimensione di una serie temporale, in quanto ogni indice del dizionario conterrà a sua volta un dizionario avente una sola chiave (il chunk ripetuto all'interno di tutta

la serie temporale), mentre al caso pessimo (tutti i chunks sono diversi) $O(n \cdot m)$, in quanto ogni dizionario contenuto come indice nel dizionario della serie temporale conterrà m chiavi diverse, aventi frequenza 1, con una complessità nello spazio al caso pessimo di $O(n \cdot m)$.

2.2 Minimum Bounding Rectangles

Il metodo proposto[6] prevede la generazione di un modello tramite l'utilizzo del metodo dei MBR, applicando uno dei tre algoritmi proposti a delle serie temporali di cui si assume un comportamento "normale". Una volta generato il modello, esso viene comparato ad una serie temporale della quale si vuole verificare la presenza di anomalie; in caso in cui ne vengano trovate alcune, viene inoltre assegnato un punteggio basato sulla distanza dal modello precedentemente generato (più è alto lo score, più è grave l'anomalia).

Il modello si propone di definire il comportamento normale, essendo una generalizzazione delle serie temporali di training. La tecnica con cui viene generato, ossia MBR, prevede di unire punti adiacenti in modo iterativo, racchiudendoli in *scatole* (*box*). L'algoritmo itera fintanto che non si raggiunge un numero k di box. Per ragioni di ef-

ficienza l'algoritmo utilizzato è *Greedy*; una versione ottimale avrebbe complessità maggiore. La complessità dell'algoritmo, se applicato utilizzando *code di priorità* come strutture dati, è di $O(n \cdot log(n))$, dove n è la lunghezza della serie temporale.

Il modello può essere generato tramite 3 algoritmi: *order - dependent Greedy - Split algorithm*, *order - independent Greedy - Split algorithm*, *order - independent Greedy-Split algorithm without all - series constraint*, dove l'all - series constraint indica il vincolo, in particolare *generalization bias*, che ogni box debba contenere almeno un punto proveniente da ogni serie temporale di training. Si assume che le serie temporali di training abbiano un comportamento normale.

order - dependent Greedy - Split algorithm prevede di applicare l'algoritmo Greedy - Split inizialmente ad una singola serie temporale di training. Questo avviene approssimando la sequenza di n punti della serie temporale con una sequenza di *n-1* box, ognuna racchiudente una coppia di punti adiacenti. In seguito, l'algoritmo unisce iterativamente le due box adiacenti che minimizzano l'incremento di volume fintanto che non si hanno k box. Effettuato il procedimento in una delle s serie temporali, l'algoritmo continua espandendo le box generate al fine di includere anche le altre *s-1* serie temporali di training, processando

una di esse alla volta. Ciò è eseguito in due passaggi per ogni serie temporale. Nel primo passaggio viene assegnata ad ogni punto della serie la box più vicina, mentre nel secondo passaggio le box vengono espanse al fine di includere i punti a cui sono state assegnate. Fare due passaggi distinti è necessario in quanto i punti delle serie temporali tendono ad essere vicini tra loro, e fare un solo passaggio potrebbe risultare in un modello patologico in cui una singola box cresce in piccoli step fino ad inglobare tutti i punti. Questo algoritmo ha complessità di spazio O(k), dove k è il numero di box, in quanto viene elaborata una serie temporale alla volta; questo però comporta una complessità nel tempo di $O(s \cdot n)$, da aggiungere alla complessità di Greedy - Split applicato inizialmente di $O(n \cdot log(n))$.

In figura 3 è visibile un esempio con una serie temporale composta da 4 punti, e approssimata con 3 box. Se $k < 3$, l'algoritmo approssimerà ulteriormente la serie temporale con 2 box, scegliendo la combinazione di box che minimizza l'incremento di volume.

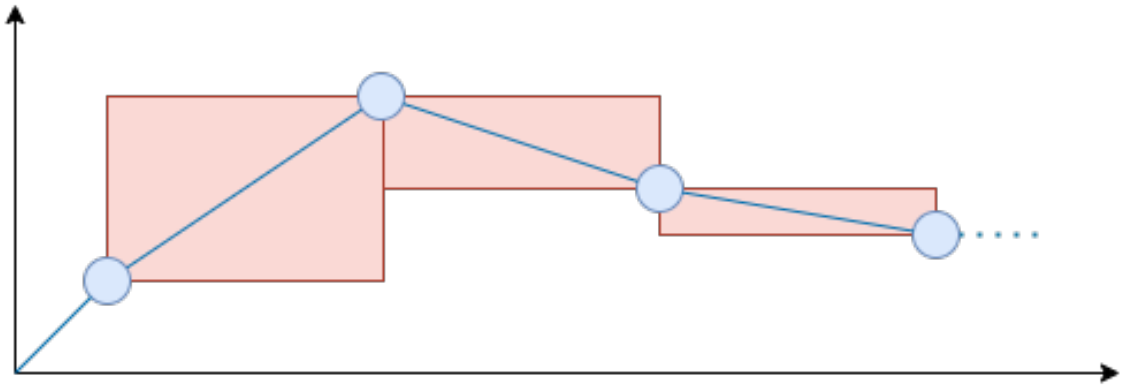


Figura 3: Serie temporale composta da 4 punti, approssimata con 3 box.

Le due combinazioni possibili sono visibili in nelle figure 4 e 5. L'algoritmo sceglierà l'approssimazione in figura 4, unendo le prime due box, in quanto minimizza l'incremento di volume complessivo.

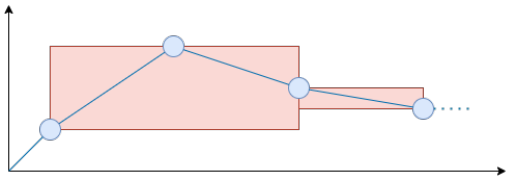


Figura 4: Combinazione ottenuta unendo le prime due box.

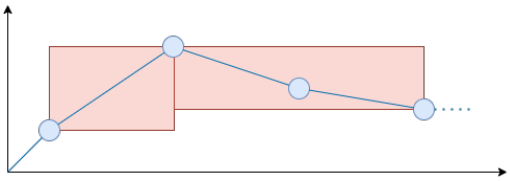


Figura 5: Combinazione ottenuta unendo le ultime due box.

order - independent Greedy - Split algorithm è utilizzato nel caso in cui non sia desiderabile che l'ordine delle serie temporali utilizzate per il training influenzi il modello. L'idea è di approssimare ogni serie temporale utilizzando le box (non più una singola serie temporale), e

poi unire box provenienti da altre serie temporali. Dopo aver generato *k1* box per ogni serie, per ogni box in ognuna delle s serie viene individuata la box più vicina nelle altre *s-1* serie; la distanza può anche essere negativa nel caso in cui una delle box inglobi l'altra. In seguito, l'algoritmo unisce le box abbinate nel passo precedente. Infine, fintanto che ci sono più di k box, l'algoritmo unisce iterativamente le due box più vicine. La complessità temporale dell'algoritmo è $O(k1^3)$.

order - independent Greedy - Split algorithm without the all - series constraint è utilizzato nel caso in cui non sia desiderabile che l'ordine delle serie temporali utilizzate per il training influenzi il modello, ma non è necessario imporre che ogni box del modello copra almeno un punto di ogni serie temporale di training. Per ottenere questo risultato è sufficiente eseguire il primo e l'ultimo passo dell'algoritmo order - independent Greedy - Split algorithm, ossia l'approssimazione di ogni serie temporale di training con box seguita dall'unione iterativa delle due box più vicine fintanto che ci sono più di k box. La complessità temporale di questo algoritmo è dominata da quella di Greedy - Split, ed è pertanto $O(n \cdot log(n))$.

Durante la fase di testing, ad ogni punto della serie temporale viene assegnato un *anomaly score* in modo online. Se le box del modello

hanno un ordinamento, può essere usato sia *stateful testing* che *stateless testing*, altrimenti può essere usato solamente *stateless testing*. L'algoritmo order - dependent restituisce un ordinamento, mentre gli altri due algoritmi potrebbero non restituirlo. In caso di stateful testing viene tenuta traccia della box in cui eravamo al passo precedente; se il punto attuale si trova all'interno di questa box o della successiva, il suo score è 0, altrimenti ne viene computata la distanza dalla box più vicina tra corrente o successiva. In caso di stateless testing, se il punto si trova all'intero di una box il suo score è 0, altrimenti viene individuata la box più vicina e ne viene computata la distanza.

Questa metodologia può essere applicata per risolvere il problema proposto in questa tesi utilizzando la serie temporale in cui è stata individuata un'anomalia come serie temporale di training. In seguito le altre serie temporali sono comparate al modello generato, e se vi è una correlazione tra modello e serie testata si può assumere che l'anomalia si sia propagata anche nella serie temporale testata.

Una delle problematiche riscontrabili in questa tecnica è il fatto che nel caso d'uso del problema proposto in questa tesi verrebbe utilizzata una sola serie temporale di training, e questo potrebbe non essere adeguato a generare un modello sufficientemente accurato. Inoltre, il

requisito che le serie temporali di training debbano avere un comportamento normale e il fatto che queste serie siano finite per loro stessa natura, suggerisce che questa tecnica è utilizzabile solo quando si hanno a disposizione serie temporali il cui comportamento è ripetuto nel tempo, ossia contiene una stagionalità, assunzione che non si vuole mantenere nel tentativo di risolvere il problema di questa tesi. Infine, anche se l'algoritmo di training venisse modificato per funzionare in modalità online, l'elevato numero di calcoli renderebbe difficile scalare l'algoritmo su un alto numero di serie temporali.

2.3 Correlation & Coherence

La ricerca[7] presenta tecniche per l'investigazione di coppie di serie temporali per ricercare caratteristiche anomale simili e simultanee. Per quanto la ricerca sia pesantemente orientata alle conseguenze che un terremoto ha su due serie temporali distinte rappresentanti i livelli di radon, l'autore precisa che "ciò non implica che le tecniche siano ristrette solamente alle serie temporali di radon".

Il metodo proposto fa utilizzo di *correlation* e *coherence*. Tramite la correlazione viene effettuata un'analisi della forma della serie tempo-

rale in relazione al tempo, mentre la "coherence" analizza il contenuto delle serie temporali in relazione alla frequenza. Viene citato l'esempio secondo il quale prendendo due sinoidi, di cui uno fuori fase per un quarto di ciclo, la loro correlazione sarà uguale a 0, in quanto in ogni quarto i due sinoidi andranno per metà nella stessa direzione, e per metà in direzioni opposte. Per quanto la correlazione suggerisca che tra le due serie temporali ci sia nessuna o bassa correlazione, questo potrebbe non essere vero e per tanto viene introdotta la coherence.

Nel testo la correlazione viene definita come *covarianza normalizzata*, e quindi indipendente dalla scala di valori utilizzati, e la sua definizione è molto simile a quella proposta dall'algoritmo Luminol. Formalmente, la covarianza con lag, $\sigma_{xy}(k)$, e cross - correlation con lag, $R_{xy}(k)$, sono definite nel seguente modo:

$$\sigma_{xy}(k) = \sum_{n=1}^{N}(x_n - \mu_x)(y_{n-k} - \mu_y), \ for \ k \leq N$$

$$R_{xy}(k) = \frac{\sigma_{xy}(k)}{\sigma_x \sigma_y}$$

dove:

x_i, y_i sono gli i-esimi membri delle serie temporali $\{x_i\}, \{y_i\}$

μ_x, μ_y sono i valori medi delle serie temporali $\{x_i\}, \{y_i\}$
σ_x, σ_y sono le deviazioni standard delle serie temporali $\{x_i\}, \{y_i\}$.

Analogamente sono definibili autocovarianza con lag, $\sigma_{xx}(k)$, e autocorrelazione con lag, $R_{xx}(k)$:

$$\sigma_{xx}(k) = \sum_{n=1}^{N}(x_n - \mu_x)(x_{n-k} - \mu_x),$$

$$R_{xx}(k) = \frac{\sigma_{xx}(k)}{\sigma_{xx}(0)}$$

L'autore effettua un'analisi dei limiti della correlazione, affermando che la *cross - correlation con delay* è utile per verificare se è presente un lag tra le serie temporali, mentre la *cross - correlation senza delay* è utile per fornire una misura di similarità complessiva, ma entrambe le tecniche non rivelano nessuna dipendenza dal tempo riguardo la loro similarità, ossia quali sono le sezioni in cui le serie temporali sono più o meno correlate.

Per definire la coherence[8], è necessario prima introdurre la *Power Spectral Density*, ottenuta tramite *Discrete Fourier Transform* della

autocovarianza. La Power Spectral Density rappresenta la proporzione di una certa frequenza sulla serie temporale. Viene definita nel seguente modo:

$$G_{xx}(k) = \sum_{n=0}^{N-1} \sigma_{xx}(n) e^{-i\frac{2\pi}{N}nk}$$

Inoltre, similmente è definita la *Power Cross - Spectral Density*, ottenuta dalla Discrete Fourier Transform della cross - covarianza:

$$G_{xy}(k) = \sum_{n=0}^{N-1} \sigma_{xy}(n) e^{-i\frac{2\pi}{N}nk}$$

A questo punto è possibile definire la coherence, e quindi il *coherence - coefficient*, definito come *Cross - Spectral Density* normalizzata dal prodotto delle *Spectral Densities* individuali, ossia:

$$C_{xy}(k) = \frac{|G_{xy}(k)|^2}{G_{xx}(k)G_{yy}(k)}$$

La coherence è reale ed il suo valore varia tra 0, indicante nessun componente comune nelle serie temporali alla frequenza k, ed 1, che

indica uguale proporzione tra le due serie temporali alla frequenza k.

L'utilizzo della Cross - Correlation in questa metodologia e all'interno di Luminol è sicuramente un'indicazione che la tecnica è efficace nella correlazione di serie temporali distinte. Inoltre, una critica sollevata dall'autore di questa ricerca afferma che la Cross - Correlation non rivela nessuna dipendenza dal tempo riguardo la similarità tra le serie temporali, ossia quali sono le sezioni in cui le serie temporali sono più o meno correlate; questa limitazione può essere, però. facilmente risolta partendo inizialmente dall'individuazione di un'anomalia in una serie temporale, per poi cercare in un intorno, centrato nel timestamp dell'anomalia individuata, di un'altra serie temporale l'eventuale correlazione tramite l'uso di Cross - Correlation. L'algoritmo Cross - Correlation ha complessità temporale $O(n)$, dove n è il numero di punti su cui applicare l'algoritmo.

2.4 Local Indicators of Spatial Association

Questo metodo[9] si bassa su *Local Indicator for Spatial Association* (LISA)[10], assegnando un LISA score ad ogni punto di ogni serie temporale indicante quanto è distante dagli altri punti contemporanei.

Inoltre, introduce il concetto di *Dynamic Time Warping*[11], ossia un algoritmo di preprocessing delle serie temporali avente fine di trovare il miglior allineamento tra due serie temporali comprimendole ed espandendole nel tempo.

Al fine di trovare il miglior mappaggio tra due serie temporali $X = [x_1, x_2, ..., x_n]$ e $Y = [y_1, y_2, ..., y_m]$ di lunghezze diverse n e m, viene creata una matrice distanza $D \in \mathbb{R}^{n \times m}$. Il percorso ottimale in questa matrice ha la distanza complessiva minore, e definisce che i propri valori $(i, j) \in D$ rappresentano un allineamento tra coppie di punti (x_i, y_i). Il *Warping Path* è una sequenza $W = (w_1, w_2, ..., w_L)$ con $w_l = (i_l, j_l) \in [1 : n] \times [1 : m]$ e $l \in [1 : L]$.

Warping path validi devono soffisfare le seguenti condizioni:

(i) **Boundary condition**: $w_1 = (1, 1)$ e $w_L = (m, n)$.

(ii) **Monotonicity condition**: $i_1 \leq i_2 \leq ... \leq i_L$ e $j_1 \leq j_2 \leq ... \leq j_L$.

(iii) **Step size condition**: $w_l - w_{l+1} \in \{(1, 1), (1, 0), (0, 1)\}$ con $l \in [1 : L - 1]$.

Si nota che la condizione *ii* non è necessaria in quanto implicata dalla condizione *iii*.

La *Warping Path Distance* (*DTW distance*) dalla prima cella $(1, 1)$ all'ultima cella (n, m) è calcolata utilizzando la seguente equazione:

$$d(W) = \sum_{i=1}^{L} d(w_i)$$

$$d(w_l) = d(x_{il}, y_{jl})$$

La DTW distance è il percorso ottimale tra X e Y, avente la distanza totale minima tra tutti i possibili percorsi, come mostrato dalla seguente equazione:

$$DTW(X, Y) = \min_{\forall W}\{d(W)\}$$

In quanto calcolare ogni percorso possibile ha complessità esponenziale al crescere di n, viene introdotto il concetto di *Accumulated Distance Matrix*, computabile con complessità $O(nm)$. Ogni elemento nella matrice contiene il rispettivo valore sommato alla minore distanza precedentemente accumulata, con l'ordinamento della precedenza definito come da *Step size condition*. Il valore nella cella (n, m) sarà

quindi la DTW Distance tra X e Y.

La tecnica statistica LISA permette di misurare il grado di correlazione spaziale in locazioni specifiche. Può essere applicata direttamente senza bisogno di effettuare training. LISA soddisfa due requisiti:

- Il LISA score assegnato ad ogni osservazione indica il grado di clustering spaziale di valori simili intorno all'osservazione.
- La somma dei LISA score di tutte le osservazioni è proporzionale all'indicatore globale di associazione spaziale.

Considerando un insieme di serie temporali $X = \{X_1, X_2, ..., X_m\}$ dove la p-esima serie temporale è un insieme di n valori v_{pi} ordinati rispetto ai loro timestamps t_{pi} e definita come $X_p = \{(t_{p1}, v_{p1}), (t_{p2}, v_{p2}), ..., (t_{pn}, v_{pn})\}$, e assumendo $z_{pi} = \frac{v_{pi} - \overline{v}_i}{\sigma_i}$, il LISA score di un valore v_{pi} è definito come:

$$L(v_{pi}) = z_{pi} \cdot \sum_{q=1, q \neq p}^{k \leq m} w_{pq_i} \cdot z_{qi}$$

dove k è il numero di tutti i valori v_{*i} in posizione i, $\overline{v}_i$ è la loro media e σ_i la loro deviazione standard. Il *Similarity Weight* tra due serie

temporali X_p e X_q ad indice i è espresso come w_{pq_i}. Gli score LISA sono fortemente impattati dal peso w_{pq_i} tra serie temporali, per cui la scelta dei pesi è estremamente importante.

Algorithm 1: LISA

Input: $\{X_1, ..., X_m\}$ insieme di m serie temporali dove $X_p = [v_{p1}, ..., v_{pn}]$ è una serie di valori ordinati temporalmente v_{pi};
$p \in \{1, ..., m\}$: l'indice della serie temporale su cui effettuare l'individuazione delle anomalie;
δ: soglia che definisce lo score limite oltre il quale viene considerato un'anomalia.
Output: $L = [l_1, l_2, ..., l_n]$: serie di etichette predittive $l_i \in [0, 1]$ *per* X_p.

```
for i = 1 to n do
    if L(v_pi) < δ then
        l_i := 1;
    else
        l_i := 0;
    end
    L := l_i;
end
return L;
```

I pesi w_{pq_i} possono essere derivati anche con tecniche ben definite, come la correlazione effettuata su una finestra temporale di dimensione w. Vengono, quindi, presi in considerazione solo il valore ad indice i e i precedenti $w - 1$ valori, per cui non potrà essere associato un LISA score ai primi $w - 1$ valori della serie temporale.

Algorithm 2: Pearson-based LISA

Input: $\{X_1, ..., X_m\}$ insieme di m serie temporali dove $X_p = [v_{p1}, ..., v_{pn}]$ è una serie di valori ordinati temporalmente v_{pi};
W: la lunghezza della finestra temporale;
$p \in \{1, ..., m\}$: l'indice della serie temporale su cui effettuare l'individuazione delle anomalie;
δ: soglia che definisce lo score limite oltre il quale viene considerato un'anomalia.
Output: $L = [l_1, l_2, ..., l_n]$: serie di etichette predittive $l_i \in [0, 1]$ *per* X_p.

```
for i ∈ {w, w+1, ..., n} do
    sum := 0;
    z_pi := (v_pi − v̄_i) / σ_i;
    X'_p := [v_p(i−w+1), ..., v_pi];
    for q = 1 to m ∧ q ≠ p do
        X'_q := [v_p(i−w+1), ..., v_pi];
        w_pq_i := |corr((X'_p, (X'_q)|;
        z_qi := (v_qi − v̄_i) / σ_i;
        sum := sum + w_pq_i · z_qi;
    end
    L(v_pi) := z_pi · sum;
    if L(v_pi) < δ then
        l_i := 1;
    else
        l_i := 0;
    end
    L := l_i;
end
return L;
```

Come precedentemente introdotto, in quanto le serie temporali possono essere spostate nel tempo tra di loro, la tecnica del Dynamic Time Warping può comportare un ulteriore miglioramento. L'algoritmo *Pearson-based LISA* è quindi modificato nel seguente modo.

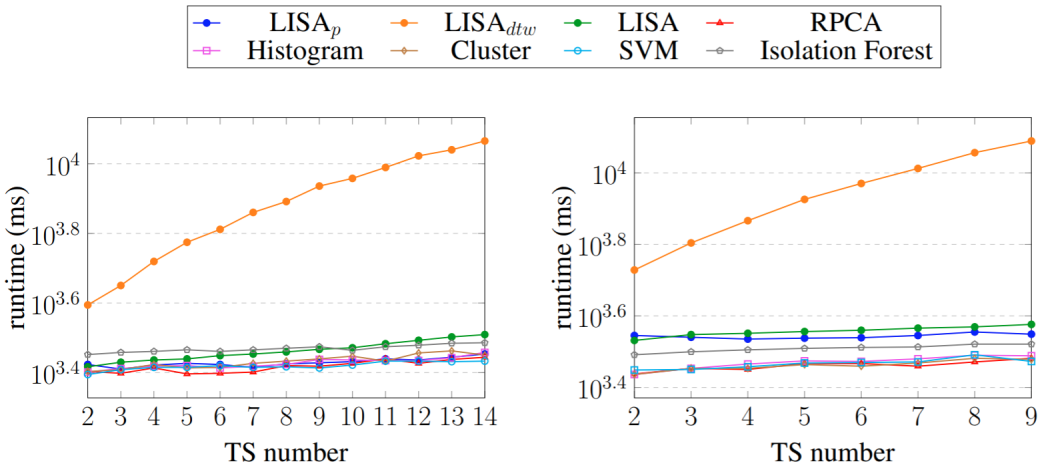
Algorithm 3: DTW-based LISA

Input: $\{X_1, ..., X_m\}$ insieme di m serie temporali dove $X_p = [v_{p1}, ..., v_{pn}]$ è una serie di valori ordinati temporalmente v_{pi};
W: la lunghezza della finestra temporale;
$p \in \{1, ..., m\}$: l'indice della serie temporale su cui effettuare l'individuazione delle anomalie;
δ: soglia che definisce lo score limite oltre il quale viene considerato un'anomalia.
Output: $L = [l_1, l_2, ..., l_n]$: serie di etichette predittive $l_i \in [0, 1]$ *per* X_p.

```
for i ∈ {w, w+1, ..., n} do
    sum := 0;
    z_pi := (v_pi − v̄_i) / σ_i;
    X'_p := [v_p(i−w+1), ..., v_pi];
    for q = 1 to m ∧ q ≠ p do
        X'_q := [v_p(i−w+1), ..., v_pi];
        W_{X'_p X'_q} := DTW(X'_p X'_q);
        for w = (j, k) ∈ {W_{X'_p X'_q}} do
            X''_p := v_p(i−w+k);
            X''_q := v_q(i−w+j);
        end
        w_pq_i := |corr((X'_p, (X'_q)|;
        z_qi := (v_qi − v̄_i) / σ_i;
        sum := sum + w_pq_i · z_qi;
    end
    L(v_pi) := z_pi · sum;
    if L(v_pi) < δ then
        l_i := 1;
    else
        l_i := 0;
    end
    L := l_i;
end
return L;
```

Questi metodi presuppongono una correlazione tra le serie temporali analizzate, e partendo da questa correlazione cercano di individuare anomalie quando le serie temporali si discostano l'una dall'altra. Per quanto il metodo contenga spunti interessanti come il Dynamic Time Warping, di fatto non risolve il problema proposto in questa tesi, in

quanto non si vuole dare come precondizione l'esistenza di una correlazione tra le serie temporali da analizzare. Inoltre, in termini di complessità si nota dalla figura sottostante che DTW-based LISA è l'algoritmo avente il runtime maggiore, con una complessità temporale più che lineare.



(a) Temp1 dataset ($n = 1000$). (b) Hum1 dataset ($n = 1500$).

Figura 6: Efficienza all'incremento del numero di serie temporali.

3 Algoritmo

3.1 Caso d'Uso

L'algoritmo è applicabile su serie temporali multiple. Le serie temporali non devono necessariamente presentare una correlazione logica, ma possono rappresentare oggetti monitorati completamente indipendenti tra loro. Non è presente nessun requisito riguardo l'uniformità degli intervalli temporali dei punti tra serie temporali distinte. L'obiettivo della soluzione è l'individuazione di comportamenti simili tra serie temporali: l'algoritmo fornisce informazioni riguardo la somiglianza tra coppie di serie temporali, e informazioni riguardo comportamenti classificati come anomali che una serie temporale può presentare. Nel caso che una serie temporale manifesti anomalie, l'algoritmo verifica la presenza di somiglianze tra la porzione anomala della serie temporale con intorni temporali, centrati nel timestamp dell'anomalia individuata, appartenenti alle altre serie temporali. L'algoritmo è stato poi esteso ed adattato per integrarsi con serie temporali rappresentanti host. Questa versione sfrutta il fatto di conoscere la tipologia di serie temporali su cui applicare la soluzione per migliorare l'efficienza dell'algoritmo, effettuando confronti solamente tra serie temporali

omogenee. L'obiettivo di questa versione è individuare il grado di correlazione tra host monitorati e generare una visualizzazione in cui gruppi di host aventi comportamento simile sono situati vicini tra loro, ossia clusterizzati.

3.2 Architettura

Dall'analisi dello stato dell'arte è risultato che l'utilizzo della Cross Covariance è un metodo efficace per stabilire la somiglianza tra due serie temporali. Altre metodologie proposte sono, invece, risultati prettamente teorici e difficili da applicare in caso di serie temporali multiple, in quanto richiedenti un numero elevato di computazioni e, comunque, richiedenti di avere a disposizione l'intera serie temporale. L'algoritmo descritto in questa tesi, invece, è stato pensato per poter funzionare contestualmente alla lettura di nuovi dati, e quindi senza necessariamente avere a disposizione le serie temporali complete. Inoltre, dallo stato dell'arte è stato preso come ispirazione il Dynamic Time Warping, ma, al fine di ridurre la complessità, è stato semplificato con un sistema di allineamento delle serie temporali più semplice, descritto in seguito.

L'elaborato di questa tesi è costituito da due algoritmi. Il primo algoritmo non ha precondizioni sulle serie temporali da analizzare, e pertanto non fa assunzioni riguardo quali serie temporali correlare, mentre il secondo algoritmo può effettuare correlazioni in modo più efficiente in quanto opera su serie temporali rappresentanti host, e quindi aventi una struttura ben nota. Il secondo algoritmo, inoltre, elabora ulteriormente i risultati ottenuti tramite le correlazioni per visualizzare quali gruppi di host presentano somiglianze reciproche. Per questa ragione i due algoritmi presentano un'architettura di base molto simile, ma anche differenze che devono essere discusse separatamente.

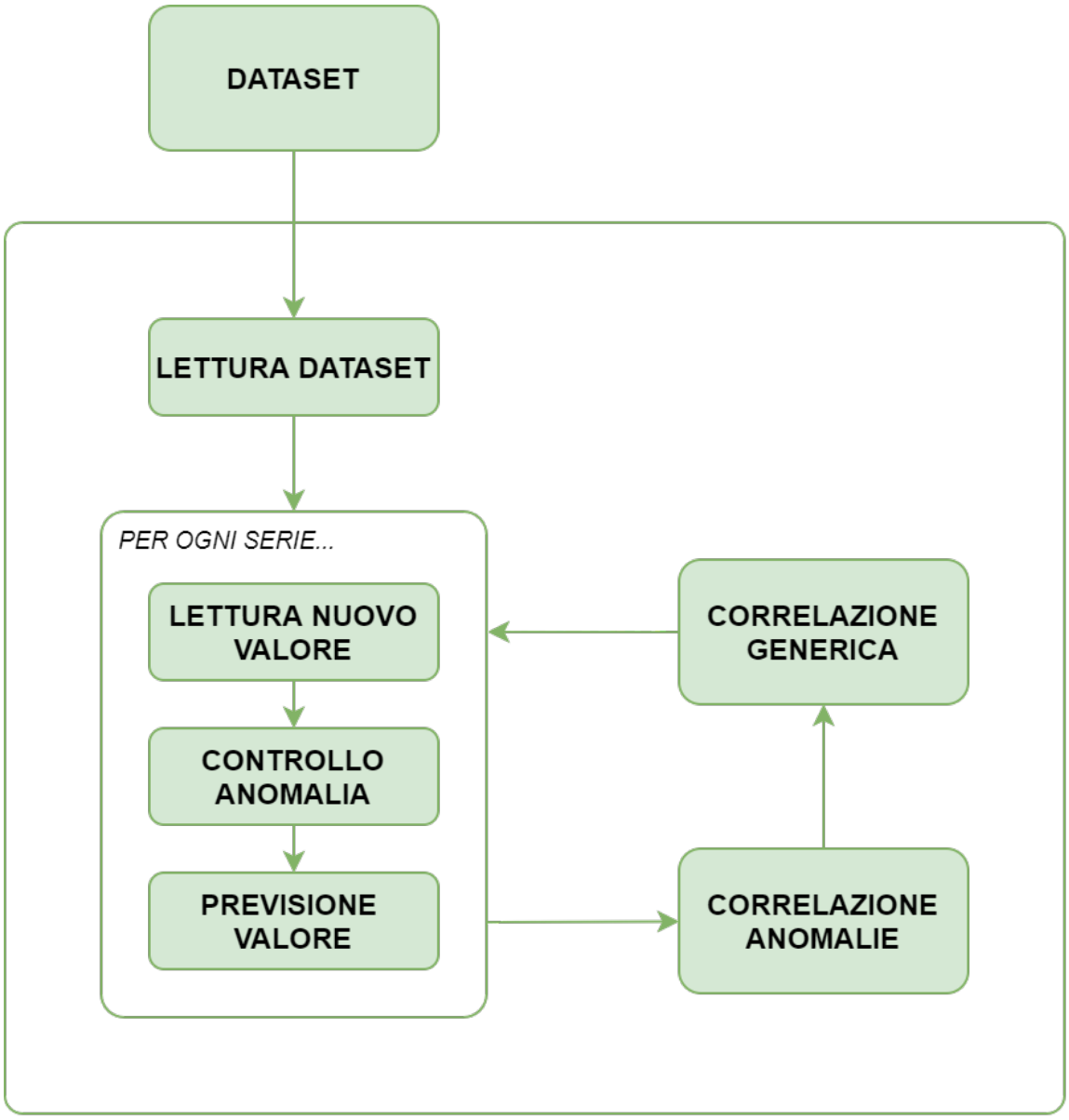


Figura 7: L'architettura base degli algoritmi.

3.2.1 Ricerca di Correlazioni tra Serie Temporali Generiche

L'algoritmo prende in input serie temporali in modalità *streaming*, ossia leggendo i valori uno alla volta e rispettando l'ordine dettato dal tempo in cui sono stati prodotti, memorizzato sotto forma di timestamp. Funzionare in modalità streaming significa operare sui dati non appena questi vengono prodotti, invece che aspettare di avere a disposizione l'intera serie temporale. Il primo passo che viene eseguito per ogni nuovo valore letto è verificare se questo si tratta di un'anomalia[12]. Per verificarlo confrontiamo il nuovo valore con un valore di predizione ottenuto tramite *Double Exponential Smoothing*, calcolandone la differenza, ossia l'errore della predizione.

Double Exponential Smoothing[13] è un metodo di predizione di serie temporali univariate che si basa sui valori precedentemente letti, dando progressivamente meno importanza ai valori più vecchi; in particolare, la versione *Double* tiene conto dell'eventuale trend che i dati possono avere, a differenza della versione *Single*.

A questo punto l'algoritmo effettua un confronto tra la deviazione standard di tutti gli errori di predizione precedenti e il nuovo errore di predizione e classifica il punto come anomalia in base a quanto i due

valori si discostano l'uno dall'altro. Dopo aver verificato se il punto appena letto costituisce un'anomalia, l'algoritmo procede ad utilizzarlo per calcolare una nuova predizione con Double Exponential Smoothing.

In seguito ad aver letto tutti i nuovi valori al tempo t, l'algoritmo procede a verificare la presenza di correlazioni sfruttando le anomalie appena individuate, se presenti; in particolare, per ogni anomalia ancora da correlare, l'algoritmo verifica se sono presenti anomalie in altre serie temporali situate in un intorno temporale vicino, e nel caso procede a correlarle.

L'algoritmo di correlazione viene applicato, oltre ai punti classificati come anomalie (rappresentati in rosso nelle figure), anche a 5 punti che precedono e 5 punti che seguono l'anomalia (rappresentati in blu nelle figure). In figura 8 è visibile un esempio di due serie temporali in cui è stata individuata un'anomalia ciascuna.

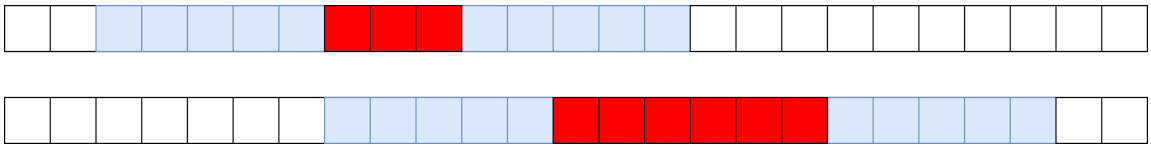


Figura 8: Porzioni di serie temporali da correlare.

La scelta di aggiungere 5 punti è stata effettuata mediante test: con un valore minore non verrebbe lasciato all'algoritmo un intorno

sufficiente per individuare correlazioni manifestate in ritardo in alcune serie temporali, mentre un valore maggiore di 5 costituirebbe un rallentamento della performance senza nessun particolare miglioramento dell'efficacia dell'algoritmo.

La correlazione avviene calcolando il coefficiente di correlazione, che è definito dalla seguente formula:

$$CC = \frac{\sum_{n=1}^{N}(x_n - \mu_x)(y_n - \mu_y)}{\sigma_x \sigma_y}$$

dove:

x_i, y_i sono gli i-esimi membri delle serie temporali $\{x\}, \{y\}$

μ_x, μ_y sono i valori medi delle serie temporali $\{x\}, \{y\}$

σ_x, σ_y sono le deviazioni standard delle serie temporali $\{x\}, \{y\}$.

Il calcolo del coefficiente di correlazione richiede porzioni di serie temporali di uguale dimensione. Nel caso in cui i punti utilizzati per il confronto non siano in egual numero nelle due serie temporali, la correlazione viene calcolata con delay, ossia la serie temporale più lunga viene tagliata in porzioni aventi la stessa lunghezza della serie temporale più corta, e per ogni combinazione viene effettuato il confronto

e calcolato il relativo coefficiente di correlazione. Successivamente, verrà considerato solamente il coefficiente di valore massimo. Le figure seguenti esemplificano questo concetto.

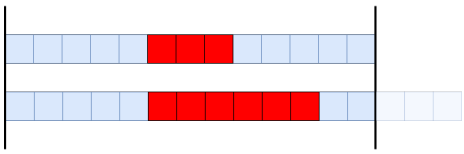


Figura 9: Delay = 0.

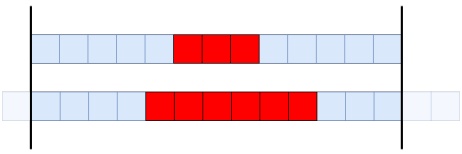


Figura 10: Delay = 1.

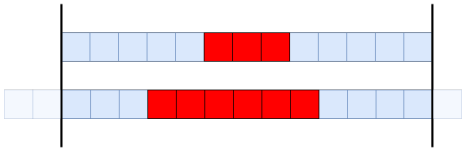


Figura 11: Delay = 2.

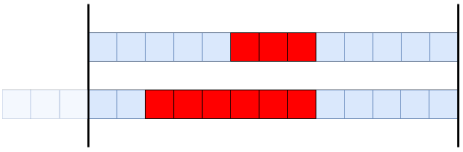


Figura 12: Delay = 3.

Se, invece, in un'altra serie temporale non sono presenti anomalie nell'intorno temporale, l'algoritmo procede a cercare una correlazione tra i punti della serie temporale con l'anomalia, e i punti situati nello stesso tempo della serie temporale senza anomalia, ma aggiungendo ulteriori 5 punti prima e dopo. La figura 13 mostra un esempio in cui sopra si ha una serie temporale in cui è stata individuata un'anomalia e i 5 punti aggiunti prima e dopo, e sotto una serie temporale in cui non è stata individuata un'anomalia: infatti, vengono presi i punti temporalmente equivalenti a quelli nella serie temporale con anomalia,

evidenziati in arancione in figura, e vengono aggiunti altri 5 punti prima e dopo.

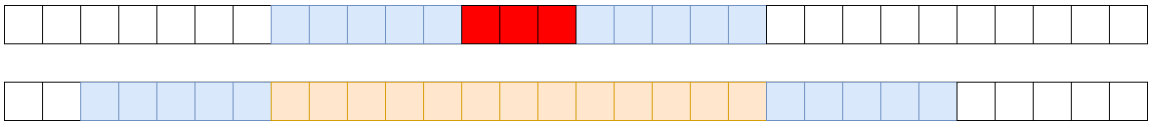


Figura 13: Correlazione tra serie temporale con anomalia e serie temporale senza anomalia.

Il motivo per cui durante la correlazione tra una serie temporale con anomalia e una serie temporale senza anomalia vengono aggiunti altri 5 punti prima e dopo la serie temporale senza anomalia, è per lasciare un margine di ricerca maggiore all'algoritmo. Questo perché, nel caso in cui due serie temporali manifestino entrambe un'anomalia, si ha già un'indicazione molto forte riguardo a dove effettuare la ricerca (ossia in un intorno centrato nell'anomalia individuata), mentre nel caso in cui non sia individuata alcuna anomalia, l'intorno utilizzato è esteso.

Oltre alla ricerca di una correlazione strettamente legata all'individuazione di un'anomalia, l'algoritmo ricerca correlazioni anche tra serie temporali in modo generico. Prima di esporre questa parte della soluzione è necessario introdurre il parametro *store*, che detta in che modo l'algoritmo tiene in memoria i punti delle serie temporali.

Settare il parametro ad n, significa che l'algoritmo tiene in memoria un massimo di $n \cdot 2 - 1$ valori, e al raggiungimento di un $n \cdot 2 - esimo$ valore viene calcolata la media degli n valori aventi timestamp minore e memorizzata al posto di questi. Questa procedura viene ripetuta per due livelli. Al raggiungimento di $n \cdot 2$ medie viene calcolata un'ulteriore media delle n medie aventi timestamp minore. Ad esempio, se store è impostato a 10 e vengono letti 76 valori, solo gli ultimi 16 verranno mantenuti dall'algoritmo; questo perché, al raggiungimento del 20-esimo valore, i primi 10 valori sono rimossi e ne è mantenuta solo la media, (in questo esempio il valore in questione è il 6 all'interno del Lv.1 nella figura 14). Al raggiungimento del 30-esimo valore, invece, vengono rimossi i 10 valori successivi ai 10 già rimossi in precedenza, e ne viene calcolata la media (questa volta rappresentata dal valore 2 all'interno del Lv.1, sempre in figura 14). Al momento della lettura del 76-esimo valore, la memorizzazione dei valori sarà nella situazione rappresentata dalla figura 14 (i numeri sono casuali e servono solamente a titolo di esempio).

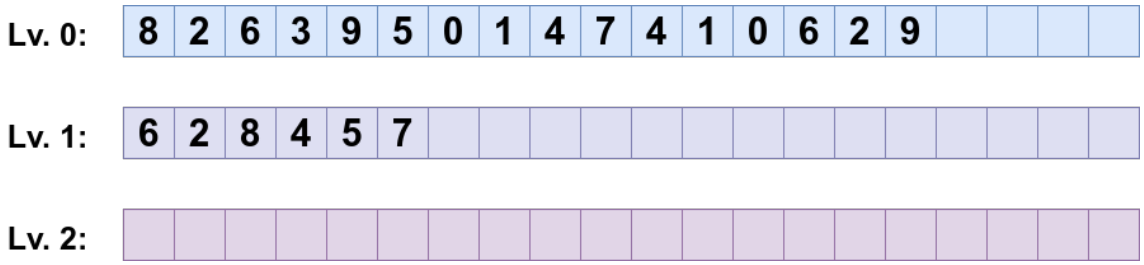


Figura 14: Un esempio con store = 10 dopo la lettura di 76 valori.

Continuando l'esempio precedente, se vengono letti altri 4 valori, per un totale di 80, il Lv.0 sarà riempito e pertanto l'algoritmo procederà a rimuovere i 10 valori più vecchi, calcolandone la media e inserendola come nuovo valore nel Lv.1, come suggerito dalla figura 15

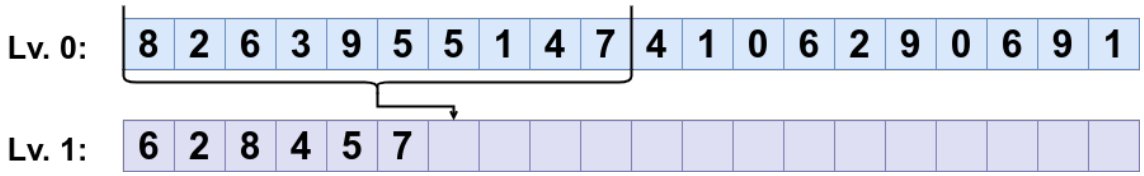


Figura 15: Il Lv.0 è pieno, e viene quindi svuotato dei 10 valori più vecchi.

Il risultato dell'esempio è visibile in figura 16, con i valori più recenti del Lv.0 spostati all'inizio del vettore e un nuovo valore, rappresentante la media dei 10 valori del Lv.0, è aggiunto al Lv.1.

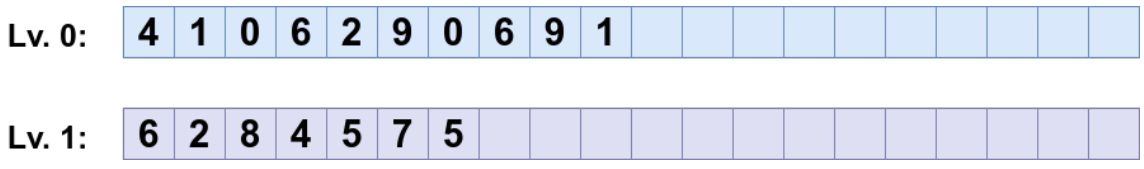


Figura 16: Un nuovo valore è stato aggiunto al Lv.1.

A questo punto è possibile descrivere come l'algoritmo effettua la correlazione generica. La correlazione generica è calcolata in seguito alla lettura da parte dell'algoritmo di *store* nuovi valori. Inizialmente, al fine di evitare confronti inutili tra serie temporali notevolmente dissimili tra loro, viene effettuato un filtraggio in base ai valori di media e deviazione standard delle due serie temporali. Per prima cosa l'algoritmo scala la deviazione standard in base alle medie delle due serie temporali, in modo da consentire il confronto tra valori situati su ordini di grandezza diversi; dopo aver portato le deviazioni standard sullo stesso ordine di grandezza, viene effettuato il confronto tra esse: in base alla loro distanza, viene stabilito se l'algoritmo può procedere a calcolare la correlazione tra le due serie temporali, o semplicemente ignorare questo confronto.

La complessità asintotica di questo passaggio rimane, quindi, $O(n^2)$, ma la riduzione del numero di confronti comporta un miglioramento della performance dell'algoritmo. La figura 17 è un esempio che permette di visualizzare come il numero di confronti effettivamente calcolati (in verde), sia in realtà ridotto rispetto al totale delle possibili combinazioni; in rosso sono rappresentati i confronti esclusi tramite l'ottimizzazione descritta.

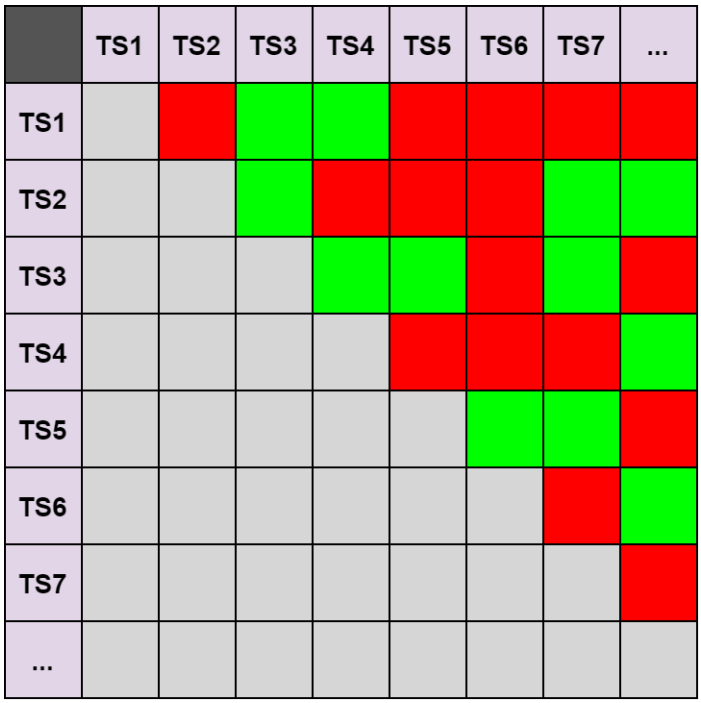


Figura 17: Le correlazioni generiche calcolate.

Nel caso in cui si proceda al confronto, i 3 livelli di memorizzazione delle serie temporali precedentemente descritti vengono appesi l'uno di seguito all'altro, nello stesso ordine con cui i dati sono stati prodotti. A questo punto viene calcolato il coefficiente di correlazione tra le due serie temporali usando i valori appena descritti. In questo modo il coefficiente di correlazione tiene conto sia degli ultimi *store* punti letti, sia dei punti precedenti, che avranno, però, progressivamente sempre meno influenza sul coefficiente di correlazione alla lettura di nuovi punti.

3.2.2 Ricerca di Correlazioni tra Serie Temporali Rappresentanti Host

Il secondo algoritmo che costituisce l'elaborato di questa tesi permette l'analisi di serie temporali multivariate rappresentanti host. L'obiettivo dell'algoritmo è ricercare una correlazione tra serie temporali omogenee appartenenti ad host diversi, generare un singolo valore per ogni coppia di host rappresentante il grado di correlazione tra essi, e verificare la presenza di gruppi di host simili tra loro, visualizzandoli sotto forma di clusters. Per fare questo l'algoritmo sfrutta molte della funzioni dell'algoritmo precedente, ma con opportune modifiche al fine di evitare confronti superflui e migliorare, quindi, la performance.

La fase iniziale è costituita dalla lettura delle serie temporali. Successivamente vengono inizializzate strutture dati necessarie alla memorizzazione specifica degli host e della loro correlazione. A questo punto, analogamente all'algoritmo precedente, vengono letti i valori delle serie temporali uno alla volta e rispettando l'ordine dei relativi timestamps, e per ogni nuovo valore viene verificato se questo costituisce un'anomalia.

Dopo aver letto tutti i valori aventi timestamp t, l'algoritmo verifica

la presenza di correlazioni tra anomalie provenienti da serie temporali distinte, ma omogenee, ossia aventi l'intervallo tra un punto e il successivo uguale tra loro. Nel caso in cui una correlazione tra anomalie sia individuata, l'algoritmo assegna un punteggio rappresentante quanto la correlazione è forte. Il punteggio è, quindi, sommato al grado di correlazione tra gli host a cui le due serie temporali appartengono.

Dopo aver letto *store* nuovi valori, come nell'algoritmo precedente viene calcolata la correlazione generica, sempre rispettando il requisito di omogeneità tra le serie temporali. Anche in questo caso viene assegnato un punteggio rappresentante quanto la correlazione generica tra le due serie temporali è forte, che è sommato al grado di correlazione tra gli host.

Si noti un'importante differenza tra i due algoritmi nella correlazione tra anomalie: mentre nell'algoritmo di correlazione tra serie temporali generiche la correlazione avveniva tra due anomalie (nel caso fossero presenti in entrambe le serie temporali) oppure tra un'anomalia e un insieme di valori situato nello stesso tempo in un'altra serie temporale (come mostrato in figura 13, pagina 37), nell'algoritmo di correlazione tra serie temporali rappresentanti host la correlazione avviene solamente tra due anomalie. Questo perché correlare un'ano-

malia con un insieme di valori in un'altra serie temporale restituirebbe risultati generalmente simili alla correlazione generica, ed in particolare, nel caso in cui una correlazione sia presente, il punteggio ad essa assegnato sarebbe aggiunto due volte al grado di correlazione tra gli host, dando un'indesideratamente eccessiva importanza alla presenza di un'anomalia. Nell'algoritmo di correlazione tra serie temporali generiche questa ridondanza è mantenuta in quanto la correlazione è eseguita a solo scopo informativo e non partecipa alla generazione del grado di correlazione tra gli host.

Un'altra differenza con l'algoritmo di correlazione tra serie temporali generiche è costituita dal fatto che l'algoritmo di correlazione fra serie temporali rappresentanti host non considera correlate serie temporali che hanno la stessa forma, ma hanno valori su ordini di grandezza diversi. Infatti, non sarebbe corretto considerare come simili un host che trasmette, in media, 100Mbit di dati ogni secondo e un altro che ne trasmette 10Kbit ogni secondo. Per questa ragione è stato inserito un filtro che permette di escludere serie temporali rappresentanti metriche di host la cui media si discosta eccessivamente l'una dall'altra.

Dopo aver terminato l'elaborazione di tutte le serie temporali, l'algoritmo procede a creare una visualizzazione rappresentante gli host

come punti in un piano. La posizione degli host sul piano rispecchia il loro grado di correlazione, in quanto maggiore è il grado di correlazione tra due host più essi saranno visualizzati vicini tra loro. Come conseguenza di questa regola di posizionamento, nella visualizzazione possono crearsi dei cluster di host, da interpretare come gruppi di host che tendono ad avere un comportamento simile nel tempo.

4 Implementazione

La soluzione è stata implementata in linguaggio Python ed è composta da due programmi principali: *correlation.py* per la ricerca di correlazioni tra serie temporali generiche e *host_correlation.py* per la ricerca di correlazioni tra host. Le differenze implementative tra i due programmi rispecchiano quanto descritto nella sezione riguardante l'architettura. I programmi fanno uso di tre parametri: *input*, *known*, e *store*. Di seguito sono spiegati in dettaglio.

- --input: Parametro obbligatorio, indica il path in cui applicare il programma. Vengono presi in input tutti i file contenuti nel path e nelle sue sottocartelle in modo ricorsivo.

- --known: Parametro flag che indica al programma di correlare solamente serie temporali aventi lo stesso nome. Non è presente in host_correlation.py in quanto questa funzione è effettuata di default.

- --store: Parametro che indica al programma quanti valori tenere in memoria per calcolare la correlazione generica. Ad esempio, se store è impostato a 60 verranno mantenuti in memoria gli

ultimi $60 \cdot 2 - 1$ punti. Al raggiungimento di $60 \cdot 2$ punti, verrà mantenuta solamente la media dei primi 60. Il valore di default è 60. Si ricorda che se il numero di punti disponibili in una serie temporale è minore di questo valore, non potrà essere calcolata la correlazione generica con questa serie temporale.

Ad esempio, se vogliamo far analizzare al programma un dataset di cui non conosciamo la struttura logica, ossia non abbiamo alcuna conoscenza riguardo quali serie temporali possono essere correlate tra loro, contenuto all'interno della cartella "./my_data/", useremo il comando:

```
python3 correlation.py --input ./my_data/
```

Se, invece, vogliamo analizzare serie temporali rappresentanti host potremmo usare il comando:

```
python3 host_correlation.py --input ./host_data/ --store 10
```

Si noti che nel secondo comando è stato specificato il parametro *store* con un valore inferiore a quello di default. Questo perché le serie temporali relative ad host potrebbero, ad esempio, avere un intervallo temporale tra un punto e l'altro di 300 secondi (5 minuti), e pertanto il

valori di default del parametro store (60) farebbe si che il programma ricerchi la correlazione generica tra host una volta ogni $60 \cdot 300 = 18000$ secondi, ossia 5 ore. Specificando il parametro *store* con un valore di 10, la correlazione generica verrebbe calcolata una volta ogni 50 minuti.

4.1 Classi

La classe principalmente utilizzata nella soluzione è *TimeSeries*, che contiene informazioni riguardanti la serie temporale, le statistiche della serie temporale (in particolare numero di punti letti, media e somma dei quadrati delle differenze dalla media attuale) e lo stato del Double Exponential Smoothing della serie temporale. La classe HostTimeSeries è essenzialmente uguale alla classe TimeSeries, ma è alleggerita di alcuni controlli e contiene referenze al relativo host. Per queste ragioni, *TimeSeries.py* è utilizzato da *correlation.py* e *HostTimeSeries.py* è utilizzato da *host_correlation.py*.

Un'altra classe utilizzata da *host_correlation.py* è *Host*, contenente informazioni riguardanti host e le relative serie temporali.

4.2 Calcolo del Coefficiente di Correlazione

Il calcolo del coefficiente di correlazione tra due serie temporali è stato implementato basandosi sulla seguente formula matematica, descritta nella sezione sull'architettura a pagina 35:

$$CC = \frac{\sum_{n=1}^{N}(x_n - \mu_x)(y_n - \mu_y)}{\sigma_x \sigma_y}$$

Lo snippet di codice che esegue questi calcoli è il seguente.

```
def CC_Calculator(a0, a1, n):

    mean_a0 = mean(a0)
    mean_a1 = mean(a1)
    stddev_a0 = pstdev(a0)
    stddev_a1 = pstdev(a1)

    if (stddev_a0 == 0 or stddev_a1 == 0):
        return 0

    covariance = 0
    for i in range(0, n):
        covariance += ((a0[i] - mean_a0) * (a1[i] - mean_a1))
    covariance /= n

    return covariance / (stddev_a0 * stddev_a1)
```

Figura 18: I parametri rappresentano rispettivamente i due vettori su cui applicare l'algoritmo, e la lunghezza di essi.

Il coefficiente di correlazione tra due porzioni temporali viene calcolato un numero di volte che varia a seconda della differenza di lunghezza delle due porzioni di serie temporali; l'algoritmo si basa, infatti,

sulla *Cross Covarianza normalizzata*, che richiede due vettori aventi la stessa lunghezza. La porzione di serie temporale più lunga è quindi tagliata in parti aventi la stessa lunghezza della porzione di serie temporale più corta, come mostrato nelle figure 9, 10, 11 e 12, pagina 36. Questa elaborazione è eseguita dal seguente snippet di codice.

```
def CrossCovariance(a0, a1):

    if (len(a0) > len(a1)):
        swap = a0
        a0 = a1
        a1 = swap

    delays = len(a1) - len(a0) + 1
    cc_array = []

    a0_len = len(a0)
    for t in range(0, delays):
        cc = CC_Calculator(a0, a1[t:t+a0_len], a0_len)
        cc_array.append(cc)

    return cc_array
```

Figura 19: La variabile delays rappresenta quante volte verrà effettivamente computato il coefficiente di correlazione. Infatti, se le due serie temporali hanno la stessa lunghezza, delays sarà uguale a 1.

Nel caso in cui le due serie temporali non abbiano la stessa lunghezza e venga, quindi, restituito un array contenente più di un coefficiente di correlazione, viene preso il valore assoluto massimo. Questo perché un coefficiente di correlazione è compreso tra $[-1, +1]$, con $+1$ rappresentante perfetta correlazione, e -1 rappresentante perfetta anticorrelazione. Di seguito è mostrata la chiamata a CrossCovariance e

la scelta del miglior coefficiente secondo il criterio appena descritto.

```
cc_array = CrossCovariance(a0_ts[a0_start:a0_end], a1_ts[a1_start:a1_end])
abs_cc_array = [abs(elem) for elem in cc_array]
cc = max(abs_cc_array)
```

Figura 20: La chiamata a CrossCovariance e scelta del miglior coefficiente.

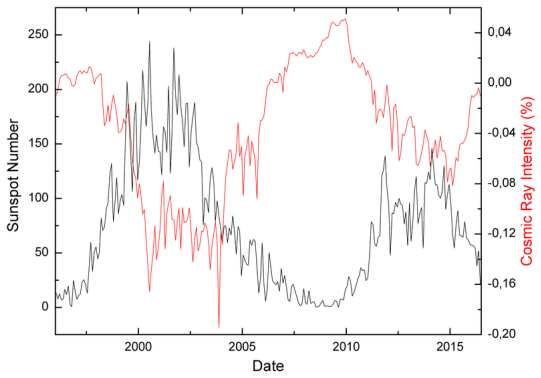


Figura 21: Un esempio di anti correlazione[14].

4.3 Calcolo delle Statistiche delle Serie Temporali.

Alcune serie di valori, come gli errori di predizione del Double Exponential Smoothing e i valori letti nelle serie temporali, necessitano di mantenere statistiche come media e deviazione standard. In quanto il programma funziona in modalità streaming e ogni punto viene letto singolarmente, non è possibile computare queste statistiche in modo classico senza rallentare eccessivamente il programma. Per questo mo-

tivo è stato impiegato un algoritmo di calcolo di media e statistica funzionante, anch'esso, in modalità streaming. L'algoritmo è chiamato *Algoritmo online di Welford*[15], ed è composto da una funzione *update* e una funzione *finalize*. La funzione update prende come parametro lo stato, che viene anche restituito al termine della funzione, e il nuovo valore, e computa le statistiche necessarie per restituire media e deviazione standard. La funzione finalize prende come parametro lo stato ed è utilizzata per ottenere le statistiche tramite i dati precedentemente elaborati, in modo efficiente. In figura 22 ne è fornita un'implementazione.

```
def update(existingAggregate, newValue):
    (count, mean, M2) = existingAggregate
    count += 1
    delta = newValue - mean
    mean += delta / count
    delta2 = newValue - mean
    M2 += delta * delta2
    return (count, mean, M2)


def finalize(existingAggregate):
    (count, mean, M2) = existingAggregate
    if count < 2:
        return float("nan")
    else:
        (mean, variance, sampleVariance) = (mean, M2 / count, M2 / (count - 1))
        return (mean, variance, sampleVariance)
```

Figura 22: L'algoritmo online di Welford.

4.4 Visualizzazione di Host Simili

La memorizzazione degli host, della loro correlazione, e la visualizzazione dei risultati del programma *host_correlation.py* è effettuata mediante l'ausilio della libreria NetworkX[16]. Gli host sono infatti rappresentati come nodi di un grafo, e il loro grado di correlazione è rappresentato tramite gli archi colleganti i rispettivi nodi. Al momento della lettura iniziale degli host, vengono creati i nodi e gli archi, generando il grafo. Ogni volta che viene individuata una correlazione tra due serie temporali di host distinti, l'arco del grafo che collega questi host è aggiornato incrementando il grado di correlazione in base alla correlazione individuata.

Una volta terminata l'elaborazione delle serie temporali, il grafo subisce un'ulteriore elaborazione al fine di essere visualizzato rispecchiando gli obiettivi di questa parte della tesi, ossia evidenziare quali host sono somiglianti tra loro disegnandoli vicini sul piano. NetworkX permette di dare un'indicazione riguardo alla lunghezza fisica degli archi nel disegno, e cerca di rispettarla al momento della visualizzazione. Sfruttando questa funzionalità della libreria, è possibile ottenere la visualizzazione nel modo desiderato.

In particolare, per fare ciò è necessario rendere il grado di correlazione degli host inversamente proporzionale alla lunghezza degli archi, in quanto, maggiore è il grado di correlazione, più gli host devono essere vicini tra loro, e quindi la loro distanza deve essere minore; inoltre, deve essere stabilito un grado di correlazione oltre il quale due host vengono considerati completamente correlati, e le distanze devono essere manipolate in modo da rendere chiaro all'occhio umano quali host sono effettivamente correlati. Al fine di rispecchiare quanto appena descritto, è utilizzata la seguente funzione all'interno del programma, avente il compito di trasformare un grado di correlazione dato in input in una distanza da utilizzare nel disegno del grafo.

```
def ScaledSigmoid(value, scale):
    center = scale/2
    sig_exponent = (-value + center)/(center/6)
    sig_denom = 1 + math.e**sig_exponent
    sig_result = 1/sig_denom
    scaled_result = RoundHalfUp(sig_result * 100, 0)
    return 100 - scaled_result
```

Figura 23: La funzione logistica[17] modificata per lo scopo del programma.

Questa funzione rappresenta la seguente formula matematica:

$$f(x) = L - \frac{L}{1 + e^{k \cdot (x_0 - x)}}$$

dove:

L è il valore massimo della curva. Nel programma è fissato a 100, ma non fa differenza in quanto NetworkX disegna le distanze in modo proporzionale tra loro, indipendentemente dalla scala di valori.

k è la velocità di incremento della curva. Stabilità in modo da risembrare la funzione sigmoide[18], nel programma equivale a $center/6$.

x_0 è il valore di x nel punto medio della funzione. È determinato dividendo *scale* per 2.

x è il valore del grafo da scalare secondo la funzione sigmoide.

Ad esempio, impostando il valore di input *scale* a 200 otteniamo la seguente funzione:

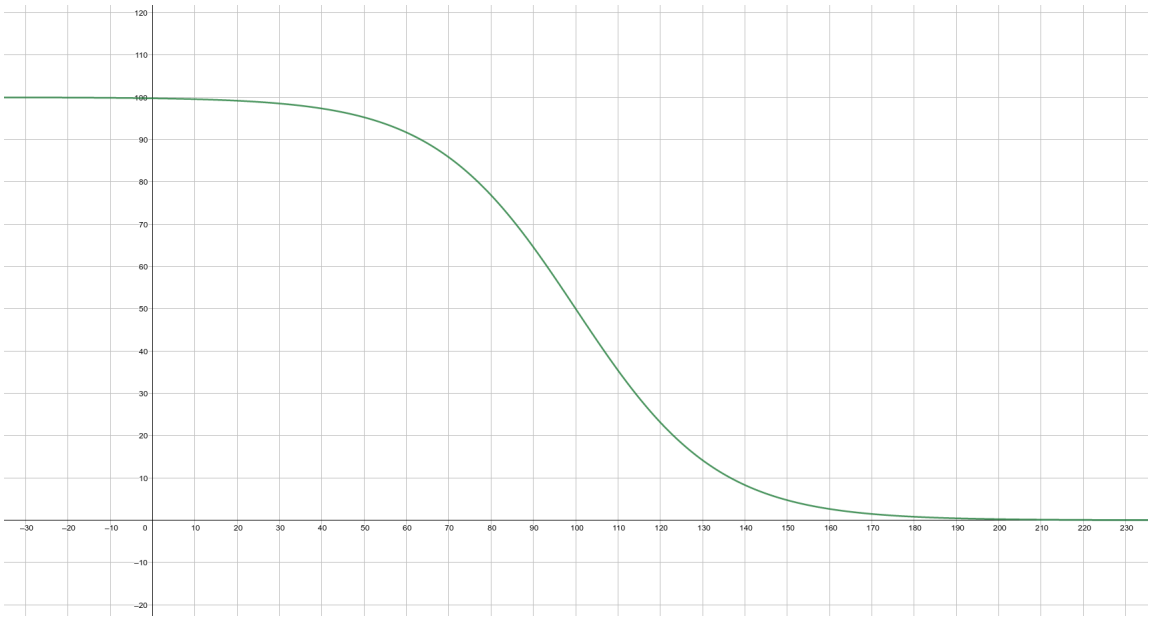


Figura 24: La funzione logistica visualizzata impostando *scale* a 200. Si può notare che la funzione è ribaltata verticalmente rispetto al suo punto medio in modo da renderla inversamente proporzionale al grado di correlazione.

Il motivo per cui viene utilizzata una funzione di questo tipo è per rendere più chiara la visualizzazione. I valori alti tenderanno più velocemente al massimo, e valori bassi tenderanno più velocemente a 0. Ad esempio, in questo modo un grado di correlazione equivalente a $\frac{3}{4}$ del massimo, che rappresenta una correlazione significativa, verrà disegnato in modo pressoché equivalente ad un grado di correlazione massimo, in quanto all'atto pratico queste piccole differenze nel grado di correlazione non sono importanti al fine di stabilire se due host manifestano comportamenti simili o meno. Analogamente, gradi di correlazione

molto bassi, seppur diversi da 0, verranno considerati insignificativi e disegnati in modo simile alla totale assenza di correlazione.

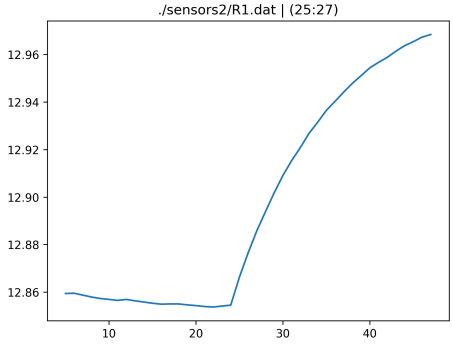
5 Validazione

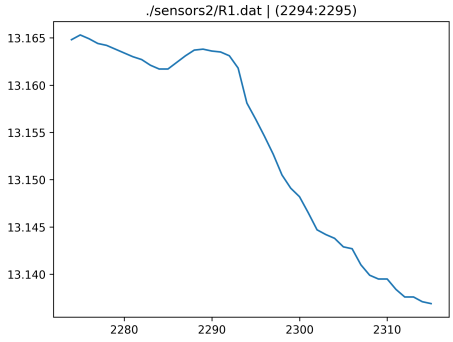
La validazione della ricerca di correlazioni tra serie temporali generiche è stata effettuata utilizzando un dataset contenente registrazioni di un array di 8 sensori del gas, 1 di temperatura e 1 di umidità[19, 20]. I sensori sono stati sottoposti a stimoli esterni come ad esempio del vino, e ne è stata registrata la risposta. Questo dataset è stato scelto in quanto costituito da sensori simili e monitoranti lo stesso ambiente; idealmente le serie temporali generate dovrebbero essere identiche, ma in pratica non lo sono, per quanto siano visivamente molto simili. Per queste ragioni il dataset costituisce un ottimo strumento di validazione, ed è ragionevole aspettarsi che l'algoritmo individui un numero consistente di correlazioni. L'esperimento è stato ripetuto più volte, ma allo scopo di validazione dell'algoritmo è stato utilizzato solamente il primo esperimento, ossia quello avente id 0. Inoltre, al fine di mantenere un numero di risultati ridotto e controllabile manualmente, sono state utilizzate solamente 4 serie temporali rispetto alle 10 disponibili: R1, R2, R3, R4.

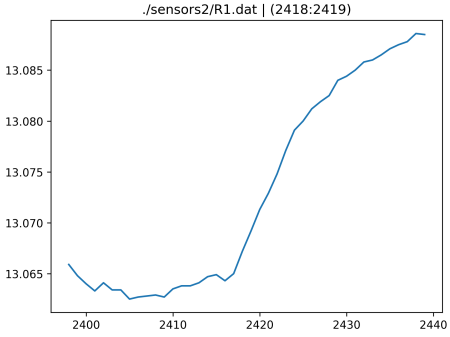
5.1 Individuazione di Anomalie

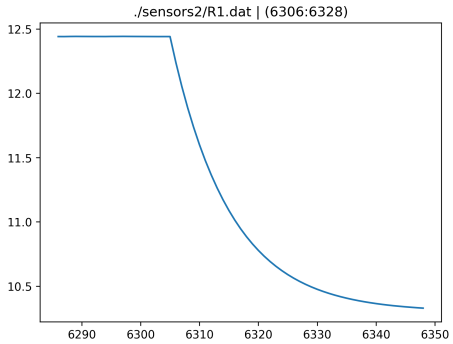
Le anomalie vengono individuate in modo corretto, ad eccezione di alcune situate tra i primi punti della serie, dovute al fatto che l'algoritmo Double Exponential Smoothing è ancora troppo sensibile a piccole oscillazione nei valori delle serie temporali e pertanto segnala una o due anomalie in eccesso. Una soluzione per risolvere questo problema potrebbe essere alzare il numero di punti minimo da attendere prima di iniziare con la ricerca delle anomalie, ma questo comporterebbe un rischio nel caso in cui il programma si trovi ad analizzare serie temporali aventi un numero di punti molto basso. In quanto l'algoritmo funziona in modalità streaming, e non è quindi a conoscenza del numero di punti che andranno a comporre la serie temporale, non è possibile impostare un valore in modo dinamico, ed è quindi necessario lasciare questo valore minimo molto basso per consentire all'algoritmo di poter operare su serie temporali di varia lunghezza.

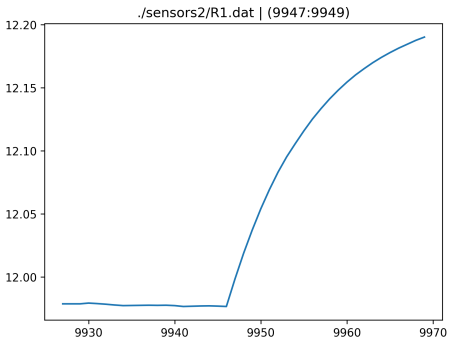
Di seguito sono mostrati alcuni esempi di anomalie individuate. Sopra il grafico è indicata la serie temporale di provenienza e in quali punti l'anomalia inizia e finisce.







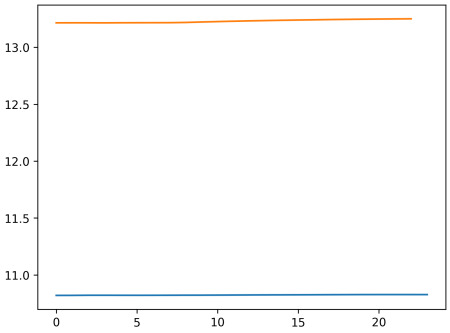


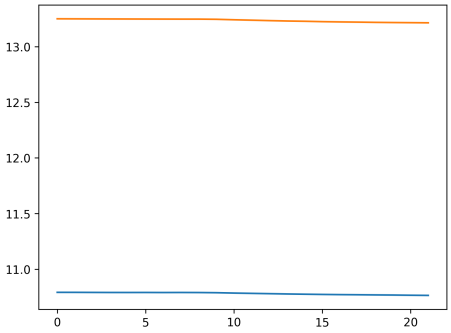


5.2 Correlazione a Seguito di Anomalie

Tra le correlazioni analizzate, alcuni grafici sono stati scartati in quanto non era presente né una chiara correlazione, né una chiara assenza

di correlazione. Per questo motivo non figurano in nessuno tra veri positivi, falsi positivi, veri negativi e falsi negativi. Ad esempio, sono state considerate insignificative le seguenti correlazioni.





Nella matrice di confusione sottostante sono visualizzati i risultati della validazione delle correlazioni effettuate a seguito di anomalie.

Predetti / Reali	Correlate	Non correlate	Somma
Correlate	246	1	247
Non correlate	38	7	45
Somma	284	8	292

Il grado di accuratezza è: $ACC = \frac{246+7}{292} = 0.87 = 87\%$

La probabilità di falso allarme è: $P_{FA} = \frac{38}{38+246} = 0.13 = 13\%$

La probabilità di mancato allarme è: $P_{MA} = \frac{1}{1+7} = 0.13 = 13\%$

Di seguito è mostrato un esempio per ogni combinazione presente nella matrice di confusione di cui sopra.

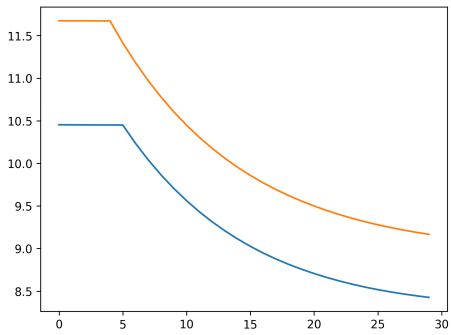


Figura 25: Un vero positivo.

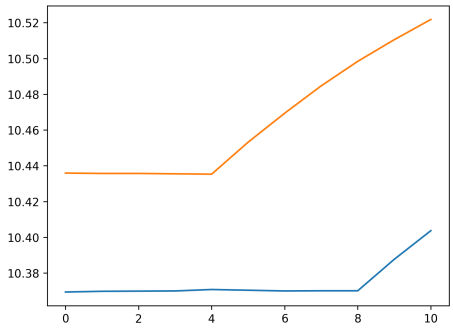


Figura 26: Un falso negativo.

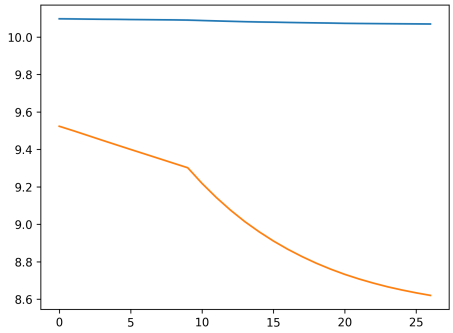


Figura 27: Un falso positivo.

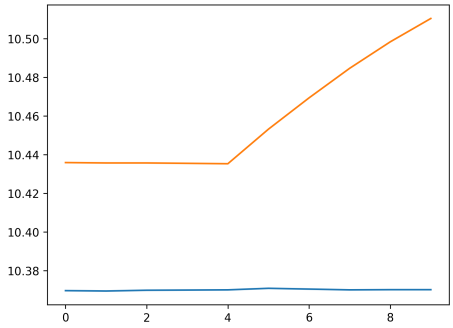


Figura 28: Un vero negativo.

In figura 25 è visibile una correlazione correttamente individuata, dove le due serie temporali hanno un comportamento quasi identico, seppur siano traslate verticalmente. In figura 26 l'algoritmo non ha individuato la correlazione, che invece era presente, anche se il comportamento delle serie temporali non è allineato nel tempo. In figura

27 sono visibili due serie temporali erroneamente segnalate come correlate, per quanto entrambe manifestino una discesa dei valori che però non è della stessa entità nelle due serie temporali, e quindi tale da giustificare una correlazione. In figura 28 sono visibili due serie temporali correttamente non segnalate come correlate dall'algoritmo.

5.3 Correlazione Generica

Considerata la struttura del dataset, le serie temporali risultano, di base, simili tra loro. Per questa ragione nella valutazione sono state considerate correlate solamente le coppie di serie temporali simili per la loro intera lunghezza, e sono state considerate non correlate le coppie di serie temporali che presentano differenze anche solamente in una porzione dei propri valori. Inoltre, per le stesse ragioni descritte nella sezione soprastante, alcune correlazioni sono state considerate insignificative e pertanto non considerate nella validazione.

Nella matrice di confusione sottostante sono visualizzati i risultati della validazione delle correlazioni generiche.

Reali \ Predetti	Correlate	Non correlate	Somma
Correlate	582	25	607
Non correlate	13	69	82
Somma	595	94	689

Il grado di accuratezza è: $ACC = \frac{583+69}{689} = 0.95 = 95\%$

La probabilità di falso allarme è: $P_{FA} = \frac{13}{13+582} = 0.02 = 2\%$

La probabilità di mancato allarme è: $P_{MA} = \frac{25}{25+69} = 0.27 = 27\%$

Ancora una volta, di seguito è mostrato un esempio per ogni combinazione presente nella matrice di confusione di cui sopra.

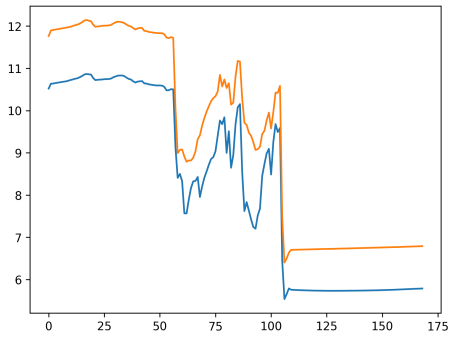


Figura 29: Un vero positivo.

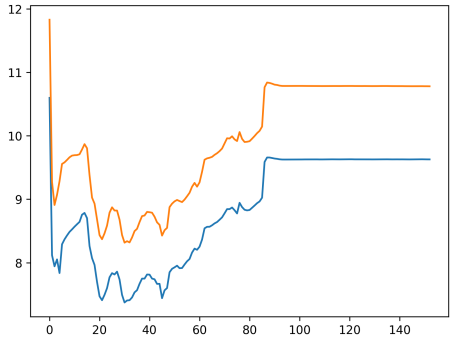


Figura 30: Un falso negativo.

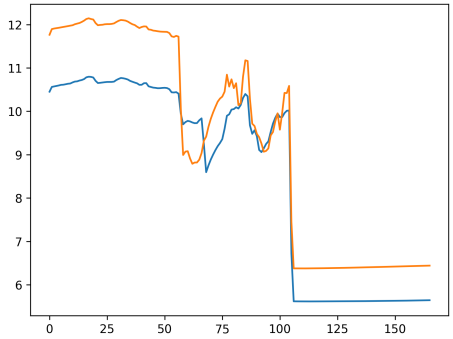


Figura 31: Un falso positivo.

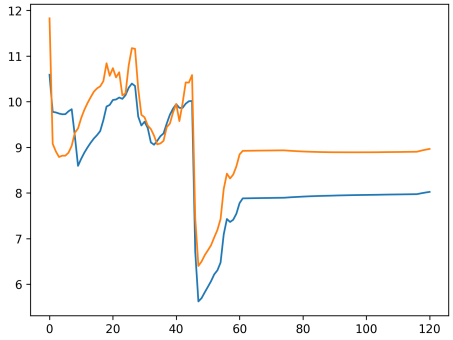


Figura 32: Un vero negativo.

In figura 29 si ha una correlazione correttamente individuata, in cui le due serie temporali hanno un comportamento molto simile, anche se sono traslate verticalmente. In figura 30 si ha una situazione simile, ma l'algoritmo non è riuscito ad individuare la correlazione. In figura 31 sono visibili due serie temporali erroneamente segnalate come correlate, in quanto, seppur simili per gran parte della loro durata, differiscono per circa 10 punti. Analogamente, in figura 32 sono visibili due serie temporali non simili per la loro intera lunghezza, ed infatti correttamente non segnalate come correlate dal programma.

5.4 Confronto con Luminol

Le anomalie individuate da Luminol sono composte da un numero molto superiore di punti, generalmente compreso tra 150 e 200, a differenza di quelle individuate dall'algoritmo che si aggirano intorno a 5 punti. Questo perché Luminol tende ad includere nell'anomalia anche i punti che la precedono, ed in particolare i punti che la seguono; si rileva, quindi, una lentezza da parte di Luminol nell'adattarsi ai cambiamenti all'interno delle serie temporali. Inoltre, Luminol tende ad agglomerare in un'unica anomalia più anomalie distinte, che vengono, invece, correttamente suddivise dall'algoritmo. Utilizzando la serie temporale R1 del dataset sopra descritto, l'algoritmo rileva 39 anomalie, di cui nessuna è un falso positivo, mentre Luminol individua 11 anomalie di cui 2 sono falsi positivi. Tutte le anomalie individuate da Luminol, ad eccezione dei falsi positivi, sono state individuate anche dall'algoritmo; al contrario, Luminol non ha individuato 19 tra le anomalie individuate dall'algoritmo.

I risultati riguardanti la correlazione restano molto simili, perché l'algoritmo di base utilizzato, ossia la Cross Covariance, è lo stesso.

Per quanto riguarda la performance, anche se non è possibile fare

un confronto preciso considerata la struttura ad API di Luminol e la sua difficoltà a scalare in modo automatizzato, dai test effettuati risulta che Luminol impiega intorno al 30% in più di tempo sugli stessi dati. Il confronto è stato effettuato rimuovendo le stampe di output da entrambi i programmi, al fine di evitare che questo interferisca con la performance dei programmi.

5.5 Individuazione di Host Simili

La validazione della individuazione di host simili è stata effettuata utilizzando un dataset prodotto dal software ntopng[21] contenente serie temporali che rappresentano host di un service provider in formato rrd[22]. Il dataset contiene serie temporali rappresentanti 2352 host, dei quali sono state usate 4 serie temporali ciascuno. Il tempo di esecuzione del programma sul dataset è stato di 20 minuti.

Fornire una valutazione dei risultati esatta su un dataset così grande non è possibile, per cui la valutazione è stata effettuata a campione. Sono state considerate come correlate le coppie di host presenti nel grafo di visualizzazione. In base a questo, otteniamo un totale di 135 coppie di host correlati, e 2 764 776 di host non correlati, su un totale

di 2 764 776 coppie di host.

Nelle figure 33 e 34 è possibile vedere un esempio di visualizzazione dei risultati, rispettivamente senza mostrare gli archi che collegano i nodi, e mostrando invece archi con relativo punteggio per ogni metrica analizzata. Per ragioni di privacy gli indirizzi IP degli host sono stati omessi, ma sono altrimenti visibili.

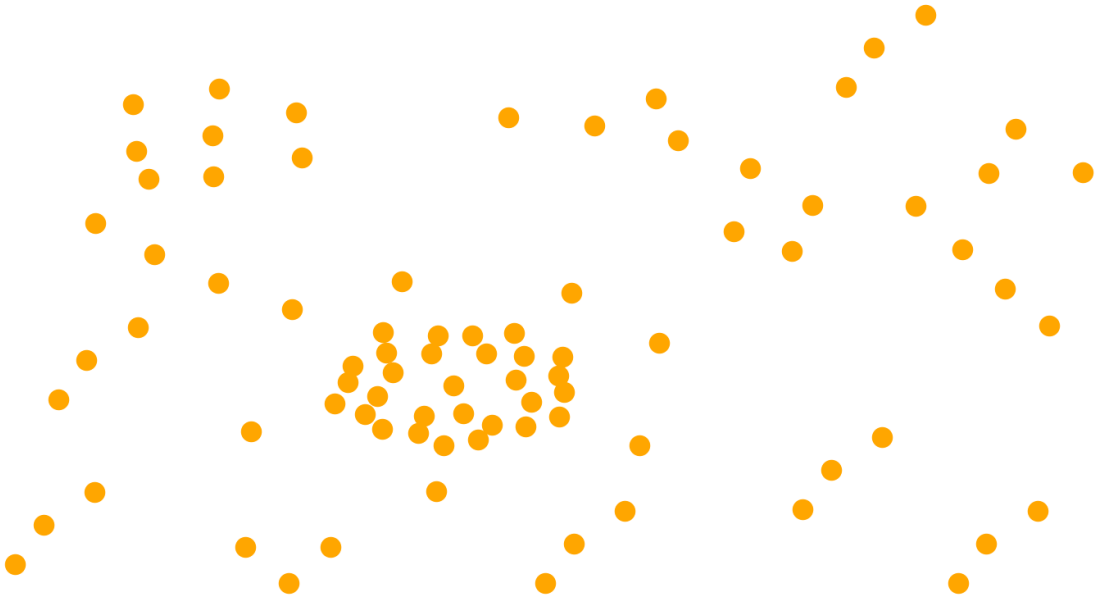


Figura 33: Visualizzazione di host simili senza archi

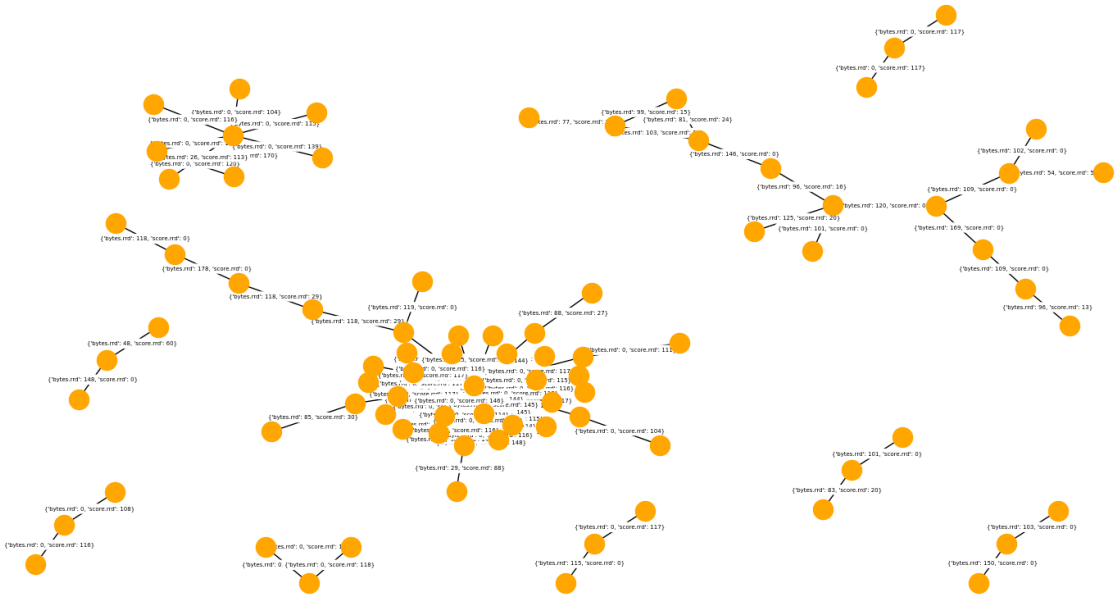


Figura 34: Visualizzazione di host simili con archi e punteggio.

In figura 35 è visibile un esempio di cluster di host formato dall'algoritmo, con i relativi punteggi per ogni metrica. Ancora una volta, gli indirizzi IP degli host sono stati omessi.

Figura 35: Un esempio di cluster con i punteggi sugli archi.

Prendendo una delle coppie di host individuata come correlata dal programma, è possibile vedere un esempio di funzionamento dell'algoritmo. In particolare, nella metrica rappresentante lo score assegnato da ntopng, visibile in figura 36, è possibile rilevare una correlazione partendo dall'individuazione delle anomalie.

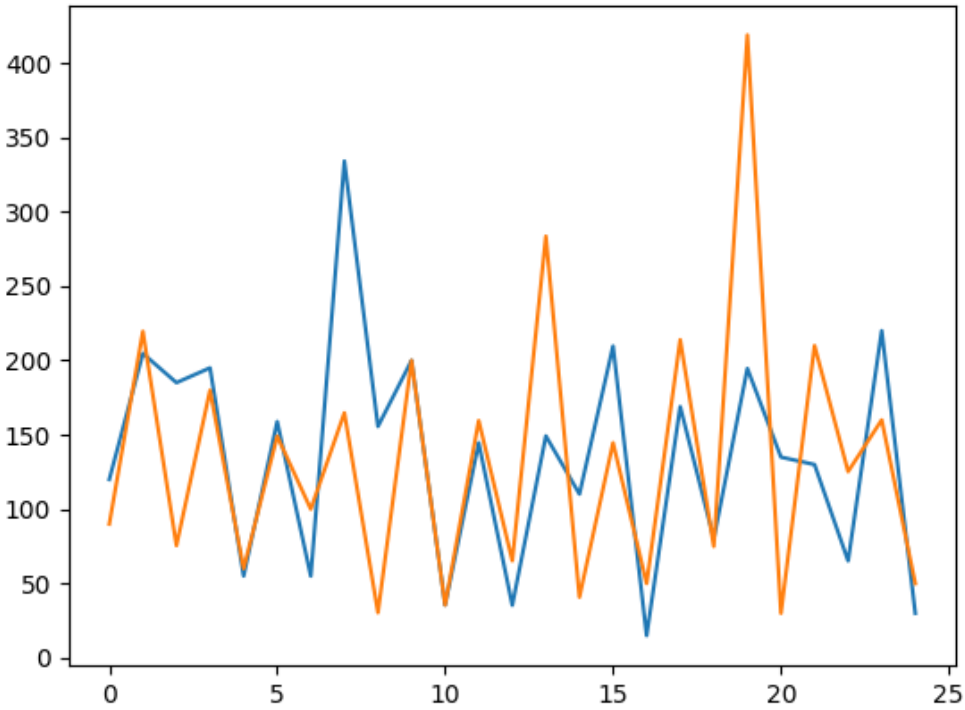


Figura 36: Serie temporali rappresentanti gli score ntopng dei due host.

Le anomalie individuate dall'algoritmo sono visibili in figura 37. Le anomalie sono evidenziate con dei punti rossi, e punti adiacenti sono da intendersi come appartenenti alla stessa anomalia; infatti, seppur i punti anomali siano 4, le anomalie individuate sono due, di cui la prima inizia nel punto 13 e termina nel punto 14, e la seconda inizia nel punto 19 e termina nel punto 20. Si ricorda che l'algoritmo non ricerca anomalie nei primi 10 punti in quanto deve adattarsi alla serie

temporale.

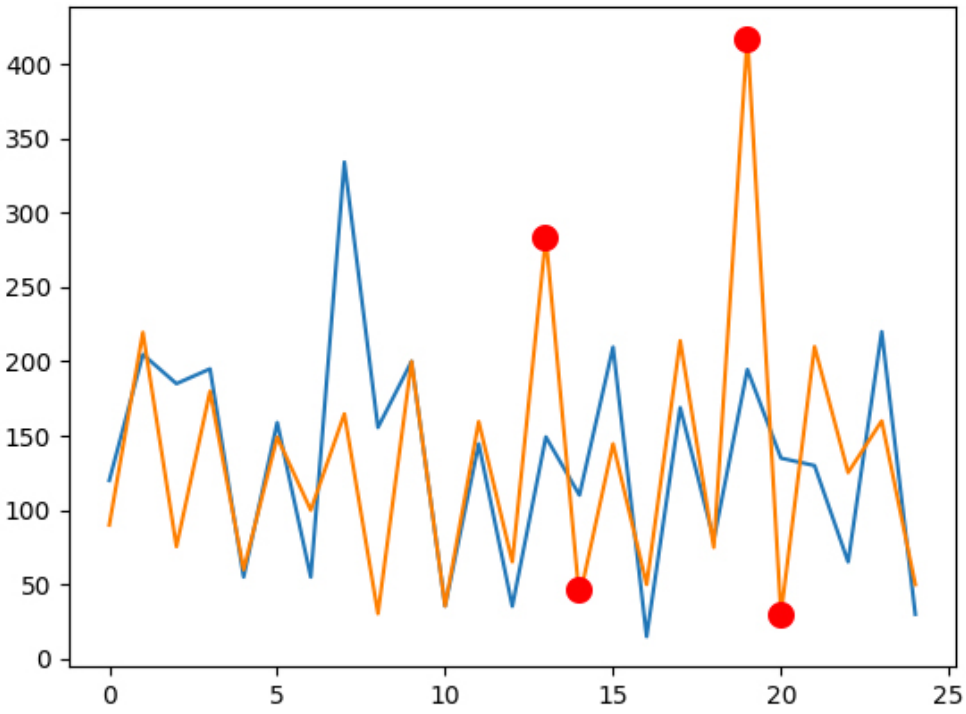


Figura 37: Le anomalie individuate dall'algoritmo.

Delle due anomalie individuate, la seconda non ha prodotto nessuna correlazione con l'altra serie temporale, mentre la prima, una volta allineata dall'algoritmo, presenta una correlazione visibile in figura 38. La correlazione è stata individuata con una confidenza dell'89%.

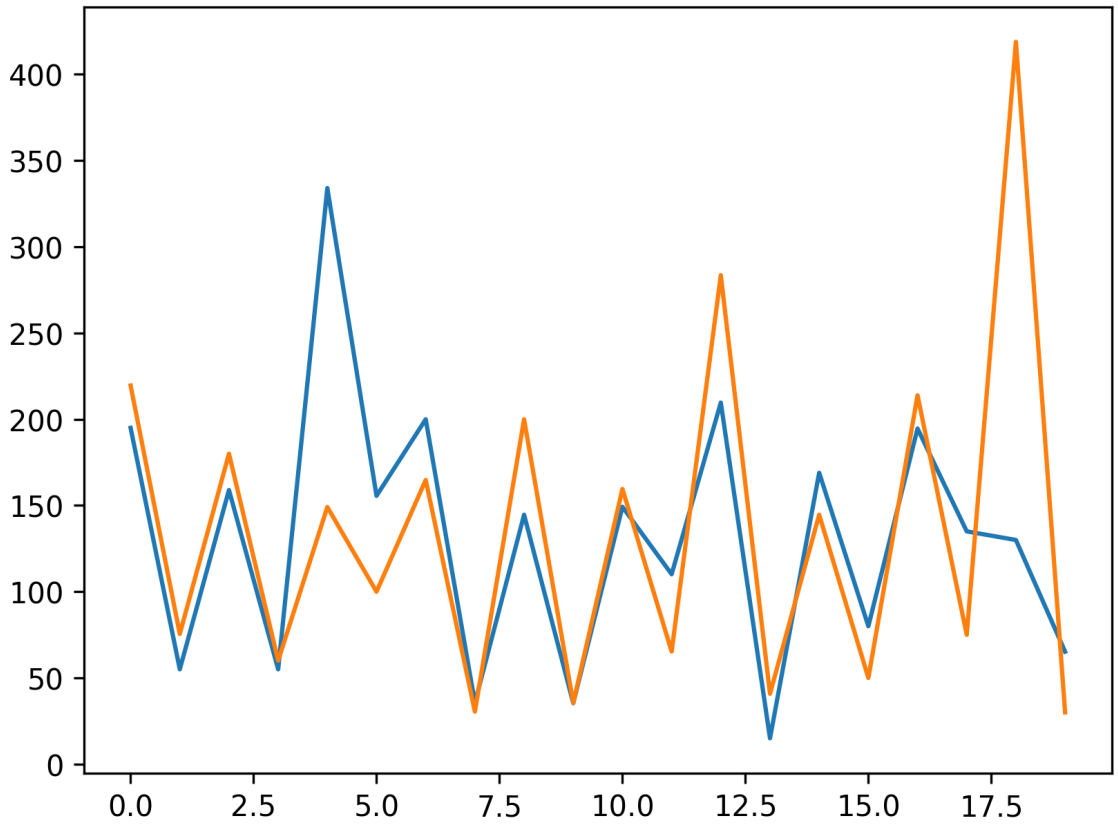


Figura 38: Correlazione tra le due serie temporali.

Prendendo un'altra coppia di host, è possibile vedere la correlazione generica, ossia non dovuta alla rilevazione di anomalie, nelle rispettive serie temporali rappresentanti i bytes, visibili in figura 39.

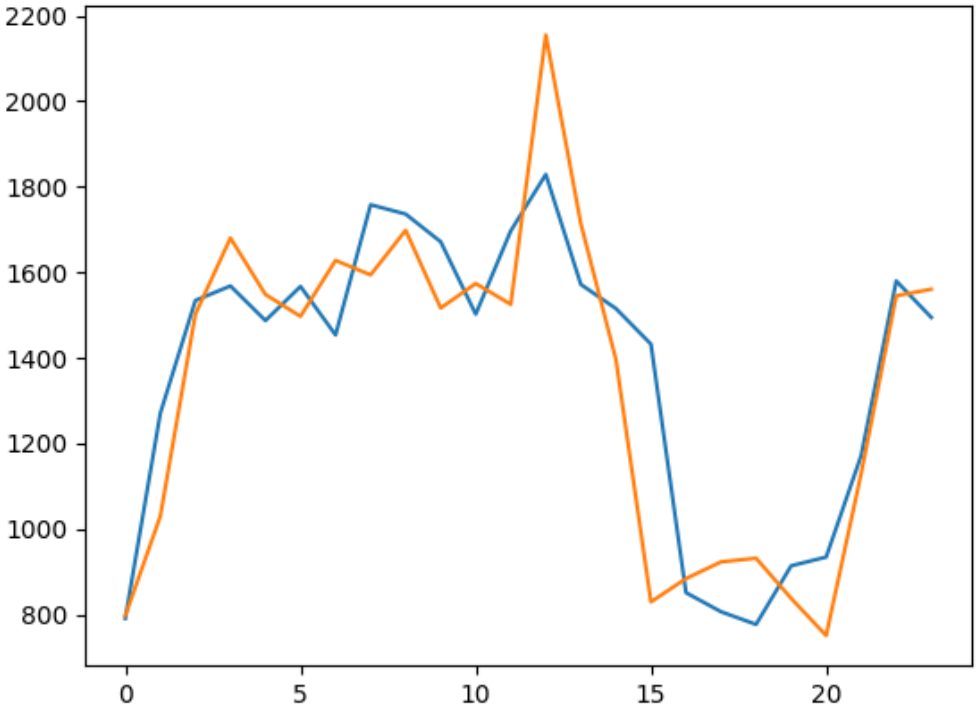


Figura 39: Serie temporali rappresentanti i bytes dei due host.

In particolare, è stata individuata una correlazione generica sia tra i primi 10 punti delle serie temporali, con confidenza del 90% e visibile in figura 40, che tra i 10 punti successivi delle serie temporali, con confidenza del 84% e visibile in figura 41.

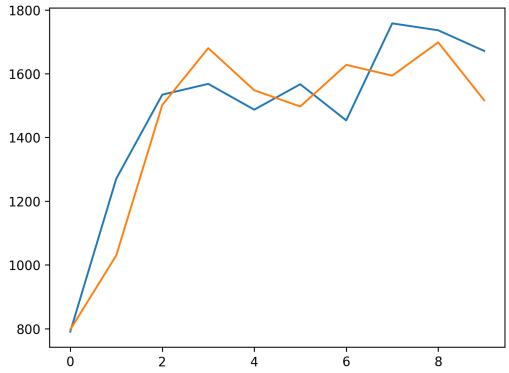


Figura 40: Correlazione generica tra i punti 0-9 delle due serie temporali.

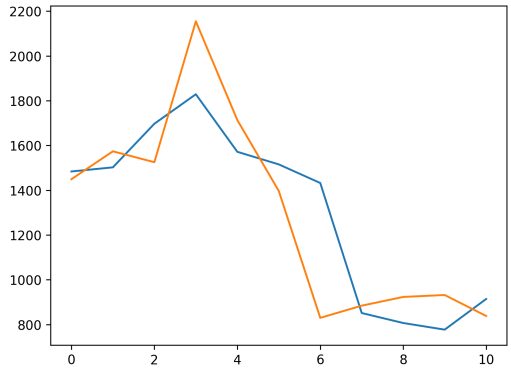


Figura 41: Correlazione generica tra i punti 10-19 delle due serie temporali.

Nella matrice di confusione sottostante sono visualizzati i risultati della validazione dell'individuazione di host simili. Il test è stato effettuato a campione scegliendo casualmente 20 coppie di host che l'algoritmo ha segnalato come simili, e 20 non simili.

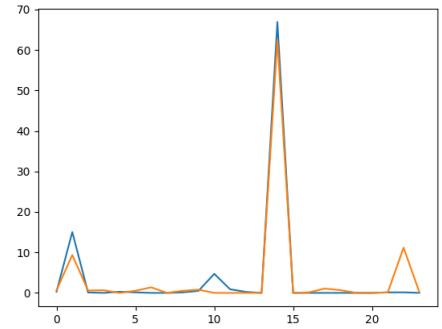
Predetti / Reali	Simili	Non simili	Somma
Simili	20	1	21
Non Simili	0	19	19
Somma	20	20	40

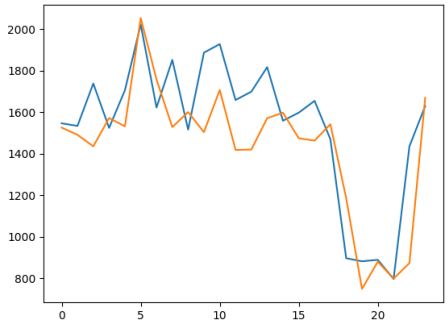
Il grado di accuratezza è: $ACC = \frac{20+19}{40} = 0.975 = 98\%$

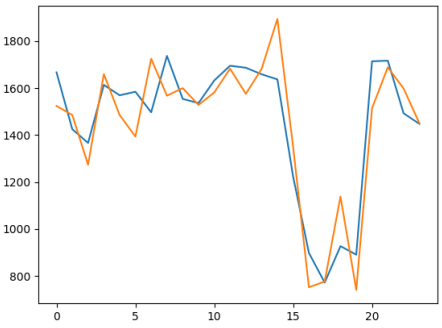
La probabilità di falso allarme è: $P_{FA} = \frac{0}{0+20} = 0 = 0\%$

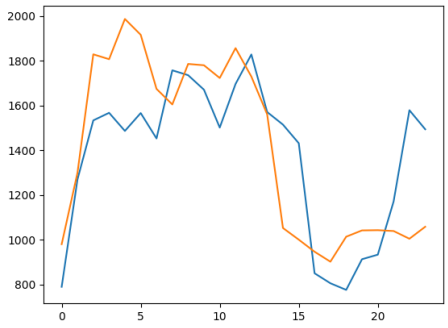
La probabilità di mancato allarme è: $P_{MA} = \frac{1}{1+19} = 0.05 = 5\%$

Di seguito sono mostrati alcuni esempi di correlazioni correttamente individuate.









6 Conclusioni

L'alto numero di oggetti monitorati in un sistema rende impossibile il controllo manuale da parte di un operatore in caso di malfunzionamenti o comportamenti anomali in tempo reale. Anche un'analisi successiva può richiedere tempi molto lunghi, restando comunque soggetta all'errore umano. È quindi necessario l'ausilio della tecnologia, in modo che questi processi siano velocizzati ad automatizzati.

L'algoritmo descritto in questa tesi affronta questo problema, analizzando molteplici serie temporali in modo parallelo e in tempo reale, e fornendo informazioni riguardanti comportamenti ritenuti anomali, correlazioni reciproche tra questi comportamenti, e correlazioni generiche delle serie temporali.

L'algoritmo si inserisce in un contesto povero di risultati pratici, e dimostra efficienza ed efficacia maggiore rispetto all'unico programma che offre funzionalità confrontabili. In particolare, l'algoritmo:

- Individua con precisione comportamenti anomali, sfruttandoli per individuare correlazioni.
- Produce risultati coerenti con le aspettative quando utilizzato su

dataset piccoli e annotati.

- Anche applicato a dataset grandi e di cui non si conosce l'eventuale correlazione degli elementi, produce risultati verificati a campione corretti.

- La performance e l'efficacia dell'algoritmo risulta migliore di quanto attualmente disponibile nello stato dell'arte. Rispetto ad esso, l'algoritmo sviluppato in questa tesi introduce numerose ottimizzazioni al fine di evitare confronti inutili tra serie temporali le cui statistiche sono dissimili tra loro, ed utilizza metodi computazionalmente semplici per ottenere risultati corretti.

L'algoritmo riesce, quindi, a monitorare indistintamente e con successo serie temporali rappresentanti qualsiasi tipo di oggetto, e questo lo rende facilmente applicabile anche nel mondo reale, come nell'ambito del controllo di dispositivi appartenenti all'Internet of Things.

6.1 Lavori Futuri

In futuro potrebbe essere riscritta l'implementazione dell'algoritmo in modo da migliorarne l'efficienza. In particolare, l'algoritmo potrebbe

sfruttare un'implementazione multithread per l'elaborazione delle serie temporali e dei relativi confronti in parallelo, e utilizzare un linguaggio compilato e a più basso livello rispetto all'attuale implementazione in Python.

Inoltre, al fine di migliorare l'utilizzo della memoria e, probabilmente, anche la performance, potrebbe essere riscritta la parte di memorizzazione dei valori, sfruttando un database invece che utilizzare la memoria principale.

Infine, potrebbe essere migliorata la metodologia con cui sono individuate eventuali anomalie, parametrizzando il metodo Double Exponential Smoothing dinamicamente a seconda delle caratteristiche delle serie temporali sottoposte.

A Codice Sorgente

Il codice sorgente che implementa l'algoritmo, insieme ad un file readme che ne illustra l'uso, è di pubblico accesso e può essere trovato al seguente link:

`https://github.com/nicolacoltelli/tesi-unipi-informatica`

Riferimenti bibliografici

[1] Luminol. Linkedin. https://github.com/linkedin/luminol

[2] Luminol README. Linkedin. https://github.com/linkedin/luminol/blob/master/README.md

[3] Bhatkar, Viprali, et al. "Time-series-bitmap based approach to analyze human postural control and movement detection strategies during small anterior perturbations." Proceedings of the ASEE St. Lawrence Section conference, Cornell University, Ithaca, NY, USA. 2006. https://www.researchgate.net/publication/249889424_Time-series-bitmap_Based_Approach_to_Analyze_Human_Postural_Control_and_Movement_Detection_Strategies_during_Small_Anterior_Perturbations

[4] Lin, Jessica, et al. "A symbolic representation of time series, with implications for streaming algorithms." Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery. 2003. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.77.6554&rep=rep1&type=pdf

[5] Bourke, P. (1996). Cross correlation. Cross Correlation", Auto Correlation—2D Pattern Identifica-

tion. https://www.researchgate.net/profile/Paul-Bourke-2/publication/245817141_Cross_correlation_autocorrelation_2d_pattern_identification/links/0046352c36f114ecbf000000/Cross-correlation-autocorrelation-2d-pattern-identification.pdf/

[6] Chan, P. K., & Mahoney, M. V. (2005, November). Modeling multiple time series for anomaly detection. In Fifth IEEE International Conference on Data Mining (ICDM'05) (pp. 8-pp). IEEE. https://cs.fit.edu/~mmahoney/icdm05.pdf

[7] Crockett, R. (2012) Identification of simultaneous similar anomalies in paired time-series. In: D'Amico, S. (ed.) Earthquake Research and Analysis - Statistical Studies, Observations and Planning. Rijeka, Croatia: InTech. pp. 125-142. https://core.ac.uk/reader/1573253

[8] Gardner, W. A. (1992). A unifying view of coherence in signal processing. Signal processing, 29(2), 113-140. https://www.sciencedirect.com/science/article/abs/pii/0165168492900150)

[9] Adrian Hanni. Correlation-based Anomaly Detection in Time Se-

ries. Master Thesis Universiy of Bern. https://exascale.info/assets/pdf/students/2020_MSc_AdrianHaenni.pdf

[10] L. Anselin, "Local indicators of spatial association—lisa," Geographical analysis, vol. 27, no. 2, pp. 93–115, 1995 https://onlinelibrary.wiley.com/doi/epdf/10.1111/j.1538-4632.1995.tb00338.x

[11] Berndt, D. J., & Clifford, J. (1994, July). Using dynamic time warping to find patterns in time series. In KDD workshop (Vol. 10, No. 16, pp. 359-370). https://www.aaai.org/Papers/Workshops/1994/WS-94-03/WS94-03-031.pdf

[12] Cheng, Haibin, et al. "Detection and characterization of anomalies in multivariate time series." Proceedings of the 2009 SIAM international conference on data mining. Society for Industrial and Applied Mathematics, 2009. https://epubs.siam.org/doi/pdf/10.1137/1.9781611972795.36

[13] LaViola, J. J. (2003, May). Double exponential smoothing: an alternative to Kalman filter-based predictive tracking. In Proceedings of the workshop on Virtual environments 2003

(pp. 199-206). http://128.148.32.110/people/jlaviola/pubs/kfvsexp_final_laviola.pdf

[14] Lingri, D., et al. "Forbush decreases during the DeepMin and MiniMax of solar cycle 24." arXiv preprint arXiv:1612.08900 (2016). https://arxiv.org/pdf/1612.08900.pdf

[15] Welford, B. P. (1962). Note on a method for calculating corrected sums of squares and products. Technometrics, 4(3), 419-420. https://arxiv.org/pdf/1510.04923.pdf

[16] Hagberg, A., Swart, P., & S Chult, D. (2008). Exploring network structure, dynamics, and function using NetworkX (No. LA-UR-08-05495; LA-UR-08-5495). Los Alamos National Lab.(LANL), Los Alamos, NM (United States). https://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-08-05495 and https://networkx.org

[17] Reed, L. J., & Berkson, J. (2002). The application of the logistic function to experimental data. The Journal of Physical Chemistry, 33(5), 760-779. https://pubs.acs.org/doi/pdf/10.1021/j150299a014

[18] Han, J., & Moraga, C. (1995, June). The influence of the sigmoid function parameters on the speed of backpropagation learning. In International workshop on artificial neural networks (pp. 195-201). Springer, Berlin, Heidelberg. https://link.springer.com/chapter/10.1007/3-540-59497-3_175

[19] Huerta, Ramon, et al. "Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring." Chemometrics and Intelligent Laboratory Systems 157 (2016): 169-176. https://archive.ics.uci.edu/ml/datasets/Gas+sensors+for+home+activity+monitoring and https://arxiv.org/pdf/1608.01719.pdf

[20] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. http://archive.ics.uci.edu/ml

[21] L. Deri, M. Martinelli, A. Cardigliano. Realtime High-Speed Network Traffic Monitoring Using ntopng Proceedings of LISA 2014 workshop, November 2014. luca.ntop.org/Lisa2014.pdf

[22] Oetiker, T. (2005). RRDtool. https://oss.oetiker.ch/rrdtool/