Scuola di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Tesi di Laurea

ESPLORAZIONE DI METODI TIME SERIES PER ANOMALY DETECTION: VALUTAZIONE E ANALISI

EXPLORING TIME SERIES METHODS FOR ANOMALY DETECTION: EVALUATION AND ANALYSIS

ALESSANDRO PISCOPO

Relatore: *Tommaso Zoppi*

Anno Accademico 2022-2023

INDICE

1 Introduzione 5
2 Stato dell'Arte 7
  2.1 Machine Learning 7
    2.1.1 Algoritmi utilizzati 8
    2.1.2 Metriche di valutazione 13
  2.2 Anomaly Detection 15
  2.3 Time Series 16
3 Metodologie 19
  3.1 Descrizione Dataset 19
    3.1.1 Dataset unifi-filtered 20
    3.1.2 Dataset all3 21
  3.2 Preprocessing 22
  3.3 Strategie 23
    3.3.1 Approccio Classico 23
    3.3.2 Approccio Time Series con Media Mobile 24
    3.3.3 Approccio Time Series con Differenze 25
  3.4 Window e Shuffle 25
    3.4.1 Window 25
    3.4.2 Shuffle 26
4 Analisi Risultati 27
  4.1 Progressione MCC, Error Rate e Speed Score 27
  4.2 MCC dataset All3 27
  4.3 Progressione MCC al variare della finestra temporale 27
  4.4 Confronto shuffle True/False 27
5 Conclusioni 29

ELENCO DELLE FIGURE

Figura 1 Logistic Regression 9
Figura 2 Linear Discriminant Analysis 9
Figura 3 Schema Albero Decisionale 10
Figura 4 Esempio Albero Decisionale 11
Figura 5 Schema Random Forest 12
Figura 6 Confusion Matrix 13
Figura 7 Balanced e Unbalanced Datasets 15
Figura 8 Esempio Anomalia 16
Figura 9 Composizione Dataset unifi-filtered 21
Figura 10 Composizione Dataset my-all3 22
Figura 11 Funzione per preprocessing datasets 23
Figura 12 Funzione per trasformazione dataset con features media mobile 24
Figura 13 Funzione per trasformazione dataset con features differenze 25

1

INTRODUZIONE

Nell'era in cui viviamo l'analisi dei dati riveste un ruolo cruciale in molti settori della nostra società. In questo contesto il Machine Learning si è dimostrato un potente strumento per estrarre informazioni utili e conoscenza da dati complessi e voluminosi. Il *Machine Learning (ML)* è un sottoinsieme dell'intelligenza artificiale (AI) che si occupa di creare sistemi che apprendono e migliorano le proprie performance in base ai dati che utilizzano.

In particolare lo scopo del lavoro di Tesi è stato analizzare due approcci diversi all'apprendimento automatico: un approccio classico e un approccio time series. Una *time series* [11] può essere definita come un insieme di osservazioni ordinate rispetto al tempo. La differenza sostanziale tra i due approcci è che con un approccio time series si hanno informazioni non solo sull'istante di tempo corrente, ma anche su un numero di istanti di tempo precedenti scelto arbitrariamente. Il fine ultimo del presente lavoro di Tesi è stato quello di comparare le performance di 4 modelli di machine learning in condizioni diverse: Logistic Regression, Linear Discriminant Analysis, Random Forest, XGBoost. Le condizioni diverse sopracitate sono state date dall'addestramento dei modelli su set di dati differenti in base all'approccio utilizzato, classico o time series. L'ipotesi che abbiamo voluto dimostrare è che un approccio time series migliori le performance dei modelli rispetto ad un approccio classico, avendo a dispozione informazioni anche su una finestra di istanti di tempo precedenti e non solo sull'istante di tempo corrente, quindi più dati disponibili durante l'apprendimento.

Il lavoro è organizzato nel seguente modo:

- Capitolo 2: fornisce una panoramica su Machine Learning, Anomaly Detection e Time Series

- Capitolo 3: descrive le metodologie e le strategie utilizzate nel lavoro di Tesi
- Capitolo 4: analisi dei risultati ottenuti
- Capitolo 5: conclusioni della Tesi

2

STATO DELL'ARTE

2.1 MACHINE LEARNING

Il Machine Learning (ML) è una sottodisciplina dell'intelligenza artificiale che è nata nel corso degli ultimi decenni del XX secolo. Nel campo dell'informatica, l'apprendimento automatico è un'alternativa alla programmazione tradizionale, in cui a una macchina viene data l'abilità di imparare autonomamente dai dati, senza richiedere istruzioni esplicite [15]. I metodi principali per l'apprendimento automatico sono due [6]:

- **Apprendimento Supervisionato**: vengono utilizzati dati etichettati per effettuare il training del modello di ML. Nel set di dati è quindi associata un'etichetta ad ogni osservazione, ovvero ad un insieme di features, e il compito del modello sarà associare l'etichetta corretta all'insieme di features in arrivo.
- **Apprendimento Non Supervisionato**: vengono utilizzati dati non etichettati durante l'addestramento. Non viene resa esplicita quindi nessuna relazione tra i dati, sarà l'algoritmo ad estrarre le informazioni necessarie a classificare o predire i risultati attesi tramite tecniche di clustering.

Nel nostro caso, abbiamo utilizzato l'apprendimento supervisionato avendo a disposizione dei set di dati etichettati. Esistono principalmente due compiti dell'apprendimento automatico supervisionato[1]:

- **Classificazione**: un algoritmo (classificatore) è addestrato a classificare i dati di input su variabili discrete. Durante il processo di training questi algoritmi vengono esposti a dati di input, a ognuno dei quali è associata un'etichetta di classe e dovranno essere in grado, una volta addestrati, di restituire la classe di appartenenza di nuovi input forniti al modello.

- **Regressione**: un algoritmo (regressore) ha come scopo quello di individuare una relazione funzionale tra i dati di input e l'output. Il valore di output non è discreto come nella classificazione, ma è continuo.

2.1.1 *Algoritmi utilizzati*

Gli algortitmi utilizzati nel presente lavoro di tesi sono tutti classificatori, avendo preso come caso di studio un problema di classificazione, in particolare un problema di classificazione binaria. Sono stati utilizzati in totale 4 algoritmi di classificazione, 2 per ciascuna classe di modelli:

- **Modelli Statistici**: Logistic Regression, Linear Discriminant Analysis
- **Modelli basati su Alberi Decisionali**: Random Forest, XGBoost

Modelli Statistici

La **Logistic Regression (regressione logistica)** è uno dei modelli statistici più utilizzati nell'ambito del ML. Per regressione logistica si intende l'analisi di regressione che si conduce quando la variabile dipendente è binaria, ad esempio anomalia rilevata o non rilevata [14]. Il modello di regressione logistica può essere utilizzato per trovare la relazione tra una variabile binaria dipendente Y e una o più variabili indipendenti X_i. La Logistic Regression si compone delle seguenti variabili:

- **Y, variabile dipendente binaria**: assume valore 0 quando l'evento non si verifica (anomalia non rilevata) e valore 1 quando l'evento si verifica (anomalia rilevata).
- X_i**, variabili indipendenti o regressori**: queste possono essere di qualsiasi natura, qualitative o quantitative e influenzano la variabile risposta Y.

Il modello da stimare è dato dall'espressione[17]:

$$P(Y = 1|X_1, ...X_n) = \frac{1}{1 + e^{-(\beta_0+\beta_1X_1+\beta_2X_2+...+\beta_nX_n)}} \tag{2.1}$$

Dove i coefficienti $\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ sono i coefficienti di regressione

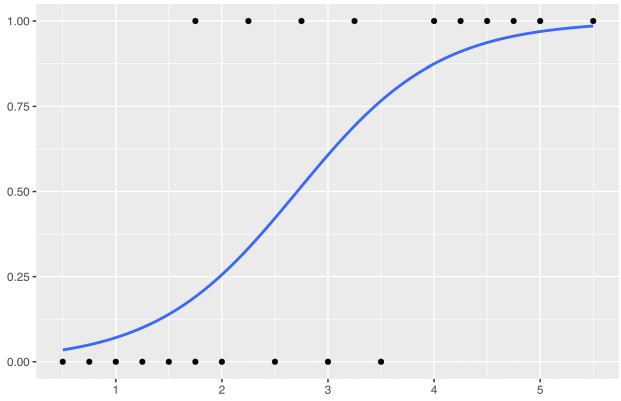

Figura 1: Logistic Regression

L'altro modello statistico utilizzato è stato la **Linear Discriminant Analysis (LDA)**. Questo modello, cerca di trovare una combinazione lineare di features che separino al meglio le classi nel dataset. La LDA funziona proiettando i dati su uno spazio a minore dimensionalità che massimizza la separazione tra le classi. Ciò avviene trovando un insieme di discriminanti lineari che massimizzano il rapporto tra la varianza inter-classe e la varianza intra-classe [18]. In altre parole, trova le direzioni nello spazio delle features che meglio separano le diverse classi di dati, nel nostro caso anomalia rilveata o non rilevata.

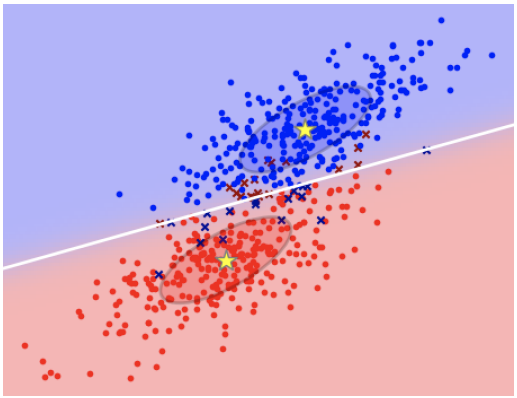

Figura 2: Linear Discriminant Analysis

Modelli basati su Alberi Decisionali

I modelli non statistici utilizzati sono entrambi basati sugli alberi decisionali, quindi prima di definire i singoli modelli, procederemo con un'introduzione su cosa sono gli alberi decisionali. Un **Albero Decisionale (DT)** è un algoritmo ampiamente utilizzato nell'apprendimento supervisionato, adatto sia per attività di classificazione che per problemi di regressione. Questo modello adotta una struttura ad albero gerarchica, caratterizzata da un nodo radice, rami, nodi interni e nodi foglia[13].

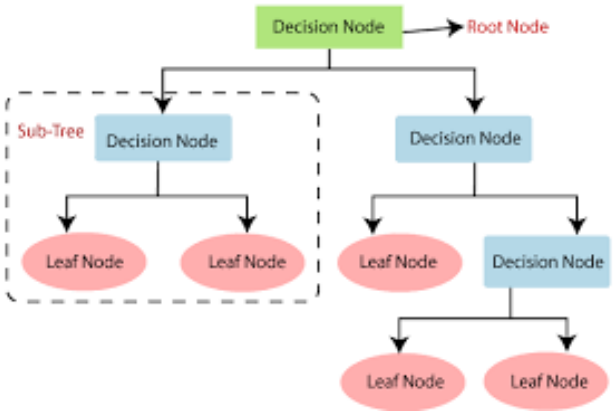

Figura 3: Schema Albero Decisionale

Come rappresentato nel diagramma soprastante, un Albero Decisionale inizia con un nodo radice, che è contraddistinto dalla mancanza di rami in ingresso. I rami che si dipartono dal nodo radice alimentano i nodi interni, noti anche come nodi decisionali. Sia il nodo radice che i nodi decisionali contribuiscono nella creazione di sottoinsiemi omogenei all'interno del dataset, in particolare, i nodi foglia rappresentano tutte le possibili previsioni o risultati nel set di dati. Il training dell'albero decisionale utilizza una strategia "dividi et impera" per cercare i punti di suddivisione ottimali all'interno di un albero. Vediamo nella figura sottostante un diagramma, molto semplice e a titolo di esempio, di un albero decisionale per determinare la presenza o meno di un'anomalia all'interno di un calcolatore.

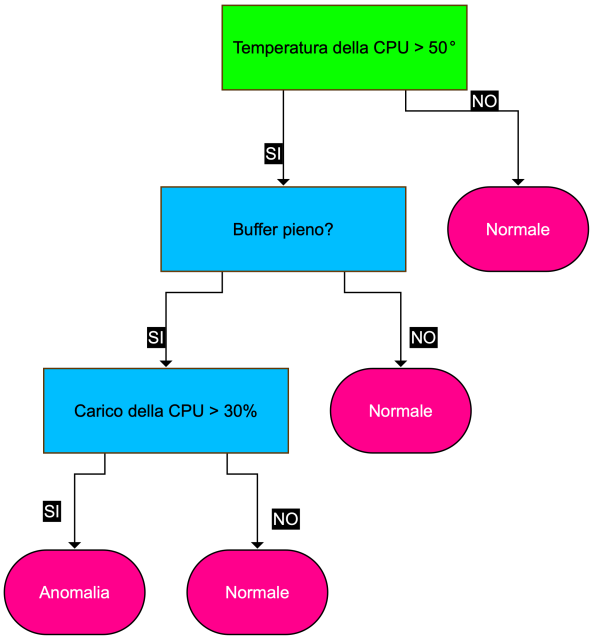

Figura 4: Esempio Albero Decisionale

Nel nostro caso non abbiamo usato direttamente degli alberi decisionali, ma dei modelli basati su di essi. L'algoritmo **Random Forest (RF)** è un classificatore d'insieme basato su alberi decisionali, ovvero è costituito da un insieme di classificatori, in questo caso DT, e le loro previsioni vengono aggregate per identificare il risultato più diffuso [7]. In particolare, l'algoritmo RF utilizza una tecnica dell'apprendimento d'insieme, chiamata bagging, in cui più DT vengono addestrati su insiemi di dati diversi, ciascuno ottenuto dal dataset di addestramento iniziale. La casualità delle caratteristiche su cui vengono addestrati i singoli alberi della foresta garantisce una bassa correlazione tra le singole strutture ad albero, riducendo così uno dei maggiori problemi degli alberi decisionali, l'overfitting. L'overfitting è un problema che si verifica quando un modello si adatta esattamente ai suoi dati di addestramento. Quando questo accade, l'algoritmo non funziona correttamente in presenza di dati non osservati in precedenza e non è quindi in grado di generalizzare, risultando inutile. Possiamo schematizzare il comportamento dell'algoritmo Random Forest nella figura seguente.

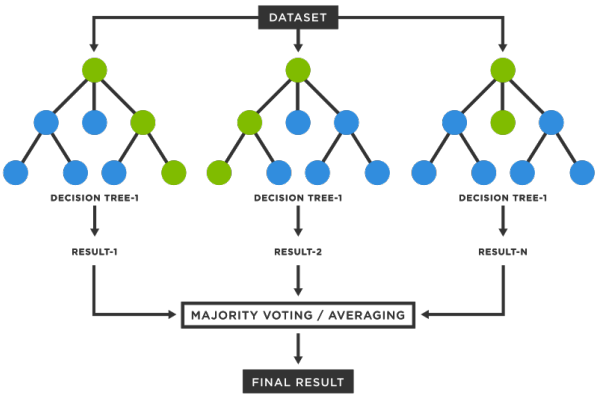

Figura 5: Schema Random Forest

Nel nostro caso, ovvero nel caso della classificazione, il risultato del modello corrisponderà al voto di maggioranza dei singoli alberi sulla classe prevista.

L'altro algoritmo basato su alberi decisionali utlizzato nel presente lavoro di Tesi è **XGBoost**, il cui nome completo è Extreme Gradient Boosting. Solitamente l'approccio più frequente quando si costruiscono dei modelli predittivi è addestrare un singolo modello forte. Un approccio differente potrebbe essere quello di costruire un insieme di modelli deboli che combinati costruiscano una previsione d'insieme forte. Quest'ultima idea è alla base di tutti gli algoritmi di ensemble learning, quindi anche di XGBoost e Random Forest [16]. Come si deduce dal nome dell'algoritmo, XGBoost è basato sul boosting, ovvero un metodo di apprendimento d'insieme, proprio come lo è il bagging per Random Forest. Il boosting combina una serie di modelli deboli che vanno a formare un modello forte per ridurre al minimo gli errori di addestramento [9]. Nel boosting, viene effettuata una selezione casuale dal set di dati e il modello viene addestrato in modo sequenziale, ovvero ogni modello successivo nella sequenza cerca di migliorare rispetto al precedente. I modelli deboli, che non sono nient'altro che alberi decisionali, vengono combinati per ottenere una previsione d'insieme forte. La differenza principale con la tecnica di bagging utilizzata da Random Forest è che nel bagging i modelli deboli sono addestrati in parallelo, mentre nel boosting apprendono in sequenza. Il miglioramento dei modelli durante l'apprendimento

sequenziale avviene grazie all'algoritmo di discesa del gradiente (notare il termine *Gradient* in *Extreme Gradient Boosting*), ovvero un algoritmo di ottimizzazione molto utilizzato nel ML. XGBoost (Extreme Gradient Boosting) è un'implementazione di Gradient Boosting, nome utilizzato poiché combina l'algoritmo di discesa del gradiente e il metodo di boosting, progettata per velocità e scalabilità.

2.1.2 *Metriche di valutazione*

In questa sezione illustriamo le metriche di valutazione utilizzate per valutare le performance dei modelli. Prima di menzionare le singole metriche, è necessario introdurre il concetto di *confusion matrix (matrice di confusione)*. La matrice di confusione è una matrice utilizzata per valutare le prestazioni di un algoritmo di ML, che può essere così schematizzata [10]:

		Predicted	
		Negative **(N)** -	Positive **(P)** +
Actual	Negative -	True Negative **(TN)**	**False Positive (FP) Type I Error**
	Positive +	**False Negative (FN) Type II Error**	True Positive **(TP)**

Figura 6: Confusion Matrix

Ora che abbiamo introdotto la confusion matrix possiamo meglio definire le metriche di valutazione. In totale sono state utilizzate 4 metriche di valutazione:

- **Accuracy (ACC)**: il valore è dato dal numero delle classificazioni corrette diviso il numero totale di classificazioni. Formalmente, guardando alla confusion matrix, il valore è dato da:

$$ACC = \frac{TP + TN}{P + N} \tag{2.2}$$

- **Error Rate (1-ACC)**: il valore è dato dalla differenza tra 1 e l'Accuracy, ovvero $1 - ACC$

- **Matthews correlation coefficient (MCC)**: questa metrica di valutazione, rispetto alle precedenti, è sicuramente la più robusta per valutare le performance dei modelli. Questo perché, a differenza dell'accuracy, questa metrica non è affetta dal problema degli *unbalanced datasets (set di dati sbilanciati)*, ma torneremo su questo problema tra poco. Guardando alla matrice di confusione il valore dell'MCC è dato da:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.3}$$

- **Speed Score (SS)**: questa metrica è stata creata ed utilizzata appositamente per analizzare le performance dei nostri modelli sui dataset usati nel lavoro di Tesi. Lo scopo di questa metrica è quello di misurare la velocità con cui il modello rileva l'anomalia. Ovviamente nell'Anomaly Detection è molto importante che le anomalie vengano rilevate nel minor tempo possibile. Per comprendere la formula del SS basta sapere che le anomalie sono sempre presenti per 5 istanti di tempo (5 secondi), ovvero 5 righe del dataset (la descrizione del dataset nel dettaglio è presente nel Capitolo 3). La formula per lo speed score è una somma pesata delle frequenze di rilevamento con decrescita quadratica e non lineare, per dare maggiore importanza alle rilevazioni nei primi istanti di tempo, diviso il totale delle anomalie rilevate.

$$SS = \frac{x_0 \cdot 1 + x_1 \cdot 0.8^2 + x_2 \cdot 0.6^2 + x_3 \cdot 0.4^2 + x_4 \cdot 0.2^2}{TotaleAnomalie} \tag{2.4}$$

  Con x_0, x_1, x_2, x_3, x_4 che indicano rispettivamente il numero di anomalie rilevata al primo, secondo, terzo, quarto e quinto istante di tempo.

Riprendiamo per un momento il problema degli unbalanced datasets. Questo problema avviene quando, in una classificazione binaria ad esempio,la frequenza di una delle due etichette risulta sbilanciata rispetto all'altra, ovvero una delle due etichette risulta molto più frequente nel

dataset rispetto all'altra. Per comprendere al meglio il problema facciamo un esempio che risalti gli aspetti critici degli unbalanced datasets. Supponiamo di avere un dataset dove il 95% dei dati è etichettato come 'normale', mentre il restante 5% è etichettato come 'anomalia'. Se avessimo un modello chiamato *dumb model* che classifica come 'normale' tutti i dati in input, avrebbe un'ACC del 95% sul nostro dataset. Se prendessimo l'ACC come metrica di riferimento, il nostro dumb model sembrerebbe un ottimo classificatore di anomalie, quando a stento potremmo definirlo classificatore. Abbiamo preso quindi l'MCC come metrica di riferimento perché molto più robusta, non risentendo degli effetti dei set di dati sbilanciati.

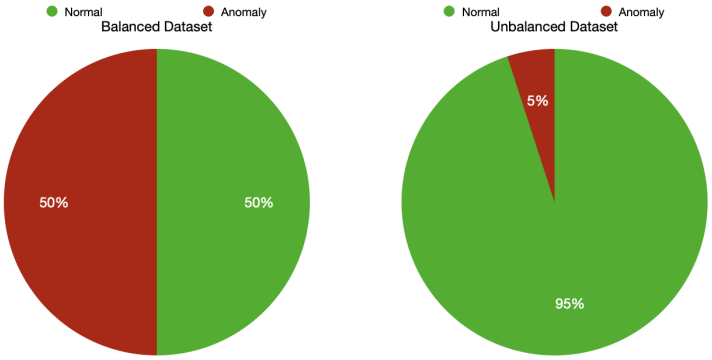

Figura 7: Balanced e Unbalanced Datasets

2.2 ANOMALY DETECTION

L'*Anomaly Detection* (o *rilevamento di anomalie* in italiano) consiste nella rilevazione di eventi rari che non rientrano nella definizione di comportamento normale dei dati. Il rilevamento di anomalie è particolarmente importante nel settore della sicurezza informatica, ma anche nei settori quali finanza, medicina e molti altri ancora.

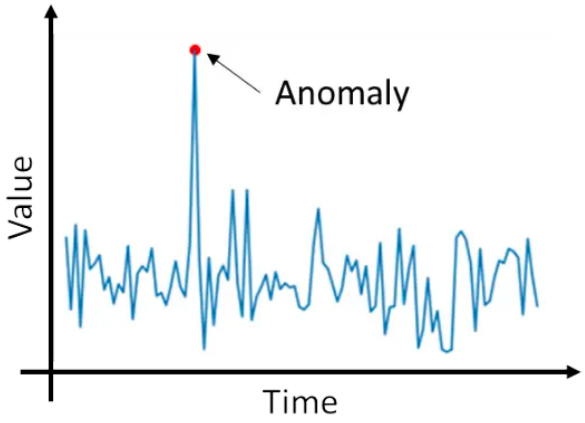

Figura 8: Esempio Anomalia

Un tempo chi si occupava di anomaly detection era solito eseaminare manualmente i dati, alla ricerca di comportamenti fuori dal normale, spesso non trovando le cause principali delle anomalie. Ad oggi il rilevamento di anomalie si basa quasi totalmente sul machine learning. Per definizione le anomalie sono eventi rari e quindi avremo a che fare spesso con dataset sbilanciati, con maggior presenza di dati etichettati come 'normali' rispetto a quelli etichettati come 'anomalie'. Le 3 categorie principali di tecniche di anomaly detection sono [8] :

- **Anomaly Detection Supervisionata**: richiedono un set di dati etichettato in due classi, 'normale' e 'anomalia', e implicano l'addestramento di un classificatore. È il caso preso in analisi nel presente lavoro di Tesi.
- **Anomaly Detection Semi-Supervisionata**: richiedono che una porzione dei dati sia etichettata.
- **Anomaly Detection Non Supervisionata**: i dati non sono etichettati e sono sicuramente le tecniche più diffuse oggigiorno.

2.3 TIME SERIES

Il tempo è una variabile fondamentale per fare delle previsioni sul futuro. Per introdurre questo fattore nei nostri modelli ricorriamo alle *Time Series*(o *serie storiche* in italiano), che definiamo come un insieme di osservazioni ordinate rispetto al tempo [11]. L'analisi delle serie storiche non coincide meramente con l'atto di raccogliere e analizzare dati nel tempo.

Ciò che contraddistingue una serie strorica da altri tipi di dati è che in una serie storica è possibile vedere come le variabili o features cambino nel tempo. Il tempo è una variabile cruciale e fornisce delle informazioni aggiuntive per i modelli. Le serie storiche vengono solitamente analizzate con modelli specifici per *time series forecasting*, che consiste nell'utilizzare un modello per predire valori futuri basandosi sui valori precedentemente osservati , come ad esempio il modello ARIMA. In letteratura, difficilmente vengono utilizzati algoritmi di ML come Random Forest o XGBoost per l'analisi di serie storiche [2]. Quello che ci siamo proposti in questo lavoro di Tesi è stato unire un problema di classificazione binaria (Anomaly Detection Supervisionata) con un approccio time series per dare maggiori informazioni ai modelli in fase di training, aspettandoci un miglioramento nelle performance dei modelli stessi.

3

METODOLOGIE

3.1 DESCRIZIONE DATASET

I dataset utilizzati nel presente lavoro di Tesi sono il risultato della ricerca dell'articolo (*commento per il Professore: non credo che il paper sia già stato pubblicato, posso citarlo?*) qui citato [19]. Le features contenute nel dataset sono state ottenute monitorando degli indicatori di performance di un dispositivo chiamato ARANCINO [12], che è il nome commerciale per una famiglia di schede IoT e embedded che risiedono sull'omonima architettura. In istanti di tempo casuali, il dispositivo è stato sottoposto a delle *error injections* della durata costante di 5 secondi. Gli errori con cui è stato attaccato il dispositivo sono di 8 tipi:

- CPU usage
- RAM usage
- Deadlock
- Redis read
- Redis write
- Stuck arancino
- Stuck node-red

Gli indicatori di performance monitorati sono stati presi a livello hardware, a livello di sistema, dai sensori, dall'ambiente e a livello di applicazione. In particolare le feature presenti nel dataset possono essere suddivise nelle seguenti 7 categorie:

- Network: 32 features

- Chip Temperature: 1 feature
- Virtual Memory: 116 features
- Memory Info: 38 features
- IO Stats: 6 features
- Python Indicators: 55 features
- Redis DB: 25 features

Ogni osservazione è dotata di un'etichetta che fornisce l'informazione sullo stato del dispositivo: normale o anomalo (durante le *error injections*). Se lo stato è normale l'etichetta corrisponderà a 'normal' altrimenti sarà una stringa che descrive l'*error injection* perpetrata in quel momento nel sistema. I dataset a disposizione sono stati:

- *unifi-filtered*: 69000 osservazioni. Il dispositivo è connesso al WiFi dell'Università degli Studi di Firenze.
- *homenet-filtered*: 72000 osservazioni. Il dispositivo è connesso al WiFi di un appartamento residenziale.
- *mobile-filtered*: 13000 osservazioni. Il dispositivo è connesso a una WiFi privata (ad esempio usando il tethering da una connessione mobile).
- *all3*: 154000 osservazioni, è l'unione dei primi 3 dataset

3.1.1 *Dataset unifi-filtered*

Il dataset su cui sono stati addestrati i modelli è *unifi-filtered*. Sono presenti 69000 osservazioni totali, di cui 11989 osservazioni etichettate come anomalie.

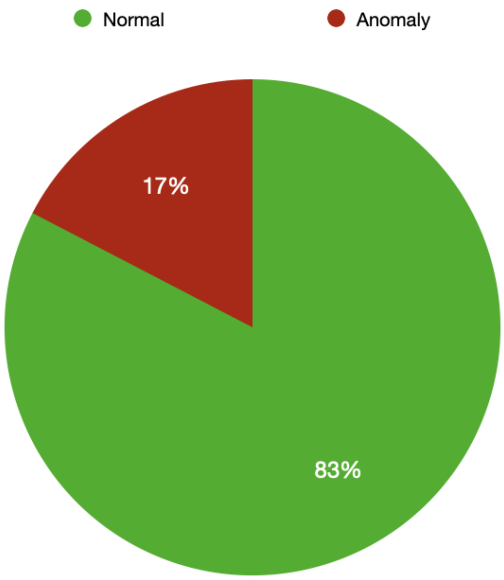

Figura 9: Composizione Dataset unifi-filtered

3.1.2 *Dataset all3*

Il dataset *all3* è stato utilizzato come test set per valutare le performance dei modelli su istanze diverse del problema. In realtÃ , non è stato utilizzato il dataset *all3*, ma una sua versione modificata. Avendo *all3* tutte le osservazioni al suo interno, comprese quelle su cui sono stati addestrati i modelli, se fosse stato utilizzato come test set avremmo testato i modelli anche su input su cui erano già stati addestrati, ottenendo come risultato delle metriche inaccurate. Per risolvere questo problema abbiamo creato una versione custom di *all3* chiamata *my-all3*, unendo il dataset *homenet-filtered* e *mobile-filtered* con solo il set di dati di test di *unifi-filtered*, senza i dati di addestramento. In *my-all3* sono presenti 105700 osservazioni totali, di cui 18245 etichettate come anomalie.

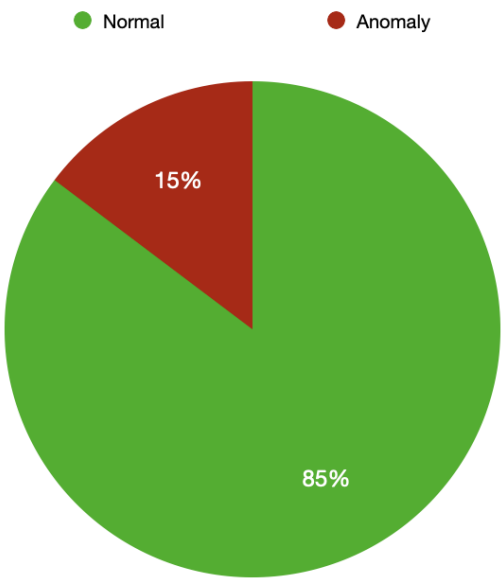

Figura 10: Composizione Dataset my-all3

3.2 PREPROCESSING

I dataset utilizzati erano già stati in parte preprocessati, ad esempio non erano presenti righe con valori NaN e i dati categorici erano già stati convertiti in dati numerici. Le operazioni di preprocessing aggiuntive sono state:

- Eliminazione delle feature con valori costanti. Ad esempio erano presenti features in cui il valore non variava mai e rimaneva sempre uguale a 1, quindi non fornivano nessuna informazione utile ai modelli.
- Trasformazione delle etichette con la descrizione delle diverse anomalie in *'anomaly'*. In questo modo è stato trasformato un problema di classificazione multiclasse in un problema di classificazione binaria.
- Mapping delle etichette da *'normal'* e *'anomaly'* a 0 e 1, rispettivamente. In questo modo anche l'etichetta è diventata un valore numerico e abbiamo soltanto dati numerici nel dataset.

```python
# Inizializzazione e Preprocessing
# Preprocessing: eliminate feature con valori costanti (1)
# Preprocessing: cambiate le varie anomalie con nomi diversi in 'anomaly' (2)
# Preprocessing: mapping 0: normal 1:anomaly (3)
def preprocessing_base(csv_path):
    # Inizializzazione
    df = pd.read_csv(csv_path)

    # Preprocessing 1
    df=df.loc[:, (df != df.iloc[0]).any()]

    # Preprocessing 2
    df.loc[df['label'] != 'normal', 'label'] = 'anomaly'

    # Preprocessing 3
    df['label'] = df['label'].map({'normal': 0, 'anomaly': 1})

    y = df['label']
    X = df.drop(columns='label')
    return X,y,df
```

Figura 11: Funzione per preprocessing datasets

L'ultima operazione di preprocessing è stata standardizzare i dataset attraverso l'utilizzo della classe *StandardScaler()* presente nel modulo *scikit-learn*. Questa operazione di preprocessing permette di elaborare la riduzione in scala dei dati, trasformando le serie di valori in una distribuzione normale standard con media uguale a 0 e deviazione standard uguale a 1 [4]. Standardizzare i dati è un'operazione di preprocessing con la quale solitamente si ottengono delle performance migliori nel processo di apprendimento.

3.3 STRATEGIE

Le strategie adottate per l'addestramento dei modelli sono state 3 in totale. Un approccio classico, senza alcuna modifica al dataset *unifi-filtered*, e 2 approcci time series, intervenendo sulla struttura del dataset. In generale per ogni modello è stato calcolato l'MCC, lo speed score e infine l'accuracy tramite cross-validation. Inoltre è stata graficata per tutti i modelli la progressione delle metriche appena menzionate rispetto agli approcci utilizzati. È stato graficato anche un confronto tra la progressione dei valori di MCC calcolati sul test set di *unifi-filtered* e quella su *my-all3*.

3.3.1 *Approccio Classico*

Nell'approccio classico è stato effettuato il training dei 4 modelli, che ricordiamo essere Logistic Regression, Linear Discriminant Analysis,

Random Forest e XGBoost, sul dataset *unifi-filtered* senza alcuna modifica, a parte le operazioni di preprocessing.

3.3.2 *Approccio Time Series con Media Mobile*

La media mobile semplice (SMA, *simple moving average*) è un indicatore molto utilizzato nell'analisi di serie storiche, questo perché ogni punto della media mobile porta con sé l'informazione di n istanti di tempo precedenti. Diamo la definizione di media mobile semplice: data una serie storica y_t con $t = 1, 2, ...T$ contenente i valori osservati di una variabile Y dal tempo 1 al tempo T si definisce media mobile al tempo t il valore [3]:

$$SMA = \frac{1}{k} \sum_{i=-m_1}^{m_2} y_{i+1} \tag{3.1}$$

dove $k = m_1 + m_2 + 1$ è il periodo della media mobile, ed equivale al numero degli addendi.

Entrambi gli approcci time series, sia con media mobile che con differenze, sono stati ottenuti andando a modificare i dataset, aggiungendo delle nuove features che portassero dell'informazione time series con loro. Il primo approccio sperimentato è stato quello di creare per ogni feature presente nel dataset una feature aggiuntiva che contenesse il valore della media mobile semplice (SMA) di n istanti precedenti. In questo modo i modelli sono stati addestrati sul doppio delle feature, dato che per ogni *feature* è stata creata la corrispondente feature *feature-media-mobile*, con a disposizione le importanti informazioni su come i valori delle feature siano evoluti nel tempo.

```python
#Trasformazione del dataset in TS con media mobile
def window_input_moving_average(window_length: int, data: pd.DataFrame) -> pd.DataFrame:

    df = data.copy()

    for x in data.columns:
        df[f'{x}_ma']=df[f'{x}'].rolling(window_length).mean()

    # Drop delle righe dove è presente valore NaN
    df = df.dropna(axis=0)

    # Drop labels
    df = df.drop(columns=f'label_ma')

    return df
```

Figura 12: Funzione per trasformazione dataset con features media mobile

3.3.3 *Approccio Time Series con Differenze*

Un altro modo di portare informazioni time series all'interno del dataset è quello di aggiungere delle nuove feature che contengano la differenza tra il valore attuale delle feature e il loro valore precedente. Anche in questo caso, come per la media mobile, le differenze sono state calcolate su n istanti di tempo, quindi per ogni feature sono presenti altre n feature che rappresentano la differenza tra il valore attuale all'istante t e i valori all'istante $t-1, t-2, ...t-n$.

```python
#Trasformazione del dataset in TS con differenze
def window_input_difference(window_length: int, data: pd.DataFrame) -> pd.DataFrame:

    df = data.copy()

    i = 1
    while i < window_length:
        for x in data.columns:
            df[f'{x}_{i}'] =data[f'{x}']-data[f'{x}'].shift(i)
        i = i + 1

    # Drop delle righe dove è presente valore NaN
    df = df.dropna(axis=0)

    # Drop labels
    for i in range(1, window_length):
        df = df.drop(columns=f'label_{i}')

    return df
```

Figura 13: Funzione per trasformazione dataset con features differenze

3.4 WINDOW E SHUFFLE

Gli approcci finora descritti sono stati valutati insieme a due parametri molto importanti: *window* (nel codice *window_length*) e *shuffle*. *Window* è il nome dato al parametro che definisce la finestra temporale, ovvero il numero di istanti di tempo, su cui vengono calcolate la media mobile e le differenze negli approcci time series. La funzione *train_test_split* è stata utilizzata per dividere il dataset in set di training e set di test e il parametro *shuffle* è un parametro booleano che stabilisce se rimescolare o meno i dati prima della suddivisione in train set e test set [5].

3.4.1 *Window*

Negli approcci time series sono stati utilizzati diversi valori per il parametro *window* e quindi per la finestra temporale. Per i valori da assegnare a

window è stato scelto un intervallo $[2, 5]$. Per chiarezza, con $window = 2$ nel caso delle differenze viene aggiunta al dataset 1 feature per ogni feature presente con la differenza tra il valore all'istante t e il valore all'istante $t - 1$, mentre nel caso della media mobile la media sarà calcolata tra i valori agli istanti di tempo t e $t - 1$. Con $window = 5$ nel caso delle differenze verranno aggiunte al dataset 4 feature per ogni feature presente con le differenze tra il valore all'istante t e il valore gli istanti $t - 1, t - 2, t - 3, t - 4$, mentre per la media mobile la media sarà calcolata tra i valori agli istanti $t, t - 1, t - 2, t - 3, t - 4$. L'intervallo $[2, 5]$ è stato scelto perché 2 rappresenta la finestra temporale minima e 5 è il numero di secondi per cui le anomalie si protraggono all'interno del sistema. Lo scopo è stato quello di verificare come le performance dei modelli andassero a cambiare al variare della finestra temporale.

3.4.2 *Shuffle*

In generale non è appropriato rimescolare i dati quando si tratta di dati time series. Questo perché ciò che contraddistingue i dati time series dagli altri tipi di dati è che i dati sono raccolti nel tempo e c'è correlazione tra osservazioni adiacenti, quindi preservare l'ordine dei dati è importante. Per questo di default durante i vari training dei modelli il parametro *shuffle* è sempre stato impostato a *False*. Abbiamo comunque voluto verificare quali fossero le differenze di performance dei modelli con il parametro $shuffle = True$. Da notare che con un valore diverso di *shuffle* cambiano i dati nel set di training del modello e quindi si hanno performance e feature importance diverse.

4

ANALISI RISULTATI

4.1 PROGRESSIONE MCC, ERROR RATE E SPEED SCORE

4.2 MCC DATASET ALL3

4.3 PROGRESSIONE MCC AL VARIARE DELLA FINESTRA TEMPORALE

4.4 CONFRONTO SHUFFLE TRUE/FALSE

5

CONCLUSIONI

NULL

BIBLIOGRAFIA

[1] Apprendimento automatico. https://it.wikipedia.org/wiki/Apprendimento_automatico. (Cited on page 7.)

[2] The complete guide to time series forecasting using sklearn pandas and numpy. https://towardsdatascience.com/the-complete-guide-to-time-series-forecasting-using-sklearn-pandas-and-n (Cited on page 17.)

[3] Media mobile. https://it.wikipedia.org/wiki/Media_mobile. (Cited on page 24.)

[4] Standardscaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. (Cited on page 23.)

[5] train_test_split. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. (Cited on page 25.)

[6] Mohamed Alloghani, Dhiya Al-Jumeily, Jamila Mustafina, Abir Hussain, and Ahmed J Aljaaf. A systematic review on supervised and unsupervised machine learning algorithms for data science. *Supervised and unsupervised learning for data science*, pages 3–21, 2020. (Cited on page 7.)

[7] Gérard Biau and Erwan Scornet. A random forest guided tour. *Test*, 25:197–227, 2016. (Cited on page 11.)

[8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009. (Cited on page 16.)

[9] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016. (Cited on page 12.)

[10] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21(1):1–13, 2020. (Cited on page 13.)

[11] Philippe Esling and Carlos Agon. Time-series data mining. *ACM Computing Surveys (CSUR)*, 45(1):1–34, 2012. (Cited on pages 5 and 16.)

[12] Maurizio Giacobbe, Francesco Alessi, Angelo Zaia, and Antonio Puliafito. Arancino. cctm: an open hardware platform for urban regeneration. *International Journal of Simulation and Process Modelling*, 15(4):343–357, 2020. (Cited on page 19.)

[13] Carl Kingsford and Steven L Salzberg. What are decision trees? *Nature biotechnology*, 26(9):1011–1013, 2008. (Cited on page 10.)

[14] Kaitlin Kirasich, Trace Smith, and Bivin Sadler. Random forest vs logistic regression: binary classification for heterogeneous datasets. *SMU Data Science Review*, 1(3):9, 2018. (Cited on page 8.)

[15] Andrea De Mauro. *Big data analytics : guida per iniziare a classificare e interpretare dati con il machine learning*. Apogeo, 2019. (Cited on page 7.)

[16] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7:21, 2013. (Cited on page 12.)

[17] James H Stock, Mark W Watson, et al. *Introduction to econometrics*, volume 104. Addison Wesley Boston, 2003. (Cited on page 8.)

[18] Petros Xanthopoulos, Panos M Pardalos, Theodore B Trafalis, Petros Xanthopoulos, Panos M Pardalos, and Theodore B Trafalis. Linear discriminant analysis. *Robust data mining*, pages 27–33, 2013. (Cited on page 9.)

[19] Tommaso Zoppi, Giovanni Merlino, Andrea Ceccarelli, Antonio Puliafito, and Andrea Bondavalli. Anomaly detectors for self-aware edge and iot devices. (Cited on page 19.)