



UNIVERSITÀ
DEGLI STUDI
FIRENZE

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

Alessandro Piscopo: *ESPLORAZIONE DI METODI TIME SERIES PER ANOMALY DETECTION: VALUTAZIONE E ANALISI*, Corso di Laurea in Informatica, © 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	15	
3	Metodologia Sperimentale	17		
	3.1	Descrizione Dataset	17	
		3.1.1	Dataset unifi-filtered	18
		3.1.2	Dataset all3	19
	3.2	Preprocessing	20	
	3.3	Metrica Speed Score	21	
	3.4	Strategie	22	
		3.4.1	Approccio Classico	22
		3.4.2	Approccio Time Series con Media Mobile	22
		3.4.3	Approccio Time Series con Differenze	23
	3.5	Window e Shuffle	24	
		3.5.1	Window	24
		3.5.2	Shuffle	24
4	Analisi Risultati	27		
	4.1	Progressione MCC, Error Rate e Speed Score	27	
	4.2	MCC dataset my-all3	33	
	4.3	Progressione MCC al variare della finestra temporale	36	
	4.4	Confronto shuffle True/False	39	
5	Conclusioni	43		
A	Codice Sviluppato	45		

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	14
Figura 8	Esempio Anomalia Puntuale	15
Figura 9	Composizione Dataset unifi-filtered	19
Figura 10	Composizione Dataset my-all3	20
Figura 11	Funzione per preprocessing datasets	21
Figura 12	Funzione per trasformazione dataset con features media mobile	23
Figura 13	Funzione per trasformazione dataset con features differenze	23
Figura 14	Progressione MCC ed Error Rate window=5,shuffle=False	28
Figura 15	Progressione Speed Score window=5,shuffle=False	28
Figura 16	Progressione MCC ed Error Rate window=4,shuffle=False	29
Figura 17	Progressione Speed Score window=4,shuffle=False	29
Figura 18	Progressione MCC ed Error Rate window=3,shuffle=False	30
Figura 19	Progressione Speed Score window=3,shuffle=False	30
Figura 20	Progressione MCC ed Error Rate window=2,shuffle=False	31
Figura 21	Progressione Speed Score window=2,shuffle=False	31
Figura 22	Comparazione MCC unifi e my-all3 window=5,shuffle=False	34
Figura 23	Comparazione MCC unifi e my-all3 window=4,shuffle=False	34
Figura 24	Comparazione MCC unifi e my-all3 window=3,shuffle=False	35
Figura 25	Comparazione MCC unifi e my-all3 window=2,shuffle=False	35
Figura 26	Progressione MCC XGBoost in base alla finestra temporale	36
Figura 27	Progressione MCC Random Forest in base alla fine- stra temporale	37
Figura 28	Progressione MCC Logistic Regression in base alla finestra temporale	37

4 Elenco delle figure

Figura 29 Progressione MCC LDA in base alla finestra temporale 38

Figura 30 Comparazione Progressione MCC shuffle = True/-False 39

Figura 31 Comparazione Feature Importance XGBoost classic window = 5 shuffle = True/False 40

Figura 32 Comparazione MCC modelli con e senza feature inaffidabili 41

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. L'implementazione di algoritmi di ML su dispositivi IoT ed edge risulta ancora in uno stato di sviluppo incompleto, rappresentando un'area di ricerca aperta [19]. L'uso di algoritmi di ML su questi dispositivi può servire a renderli consapevoli del proprio comportamento, ad esempio attraverso l'uso di rilevatori di anomalie, che permettano ai dispositivi di capire quando si ha un comportamento anomalo, agendo di conseguenza in caso di attacco o intrusione. In particolare, il presente lavoro di Tesi fa riferimento al lavoro di ricerca qui citato [19], dove sono stati monitorati degli indicatori di performance da un dispositivo chiamato ARANCINO, che è il nome commerciale per una famiglia di schede IoT e embedded che risiedono sull'omonima architettura. Dal sistema di monitoraggio utilizzato sono stati restituiti dei dati tabulari, ovvero un insieme di righe (osservazioni del set di dati) e di colonne (*features*, in questo caso indicatori di performance), ordinati rispetto al tempo. Lo scopo del lavoro di Tesi è stato confrontare l'approccio classico all'analisi dei dati con 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 algoritmi di machine learning in condizioni diverse: Logistic Regression, Linear Discriminant Analysis, Random Forest, XGBoost. Le condizioni diverse sopracitate sono state date dall'addestra-

mento dei modelli su set di dati differenti in base all'approccio utilizzato, classico o time series. L'analisi sperimentale si è basata sull'analisi di dati provenienti da dispositivi ARANCINO e si è voluto indagare su come un approccio time series possa migliorare o meno le performance dei modelli rispetto all'approccio classico, avendo a disposizione 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. Le principali conclusioni del lavoro di Tesi indicano come l'utilizzo di un approccio time series porti a un significativo miglioramento delle prestazioni dei modelli, con conseguente maggiore efficacia nella rilevazione di anomalie.

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
- Appendice A: codice sviluppato e accesso al link GitHub

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, il ML è un'alternativa alla programmazione tradizionale, in cui a una macchina viene data l'abilità di imparare autonomamente dai dati, senza richiedere istruzioni esplicite [5]. I metodi principali di apprendimento nel ML 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.
- **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 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 algoritmi 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 e rilevazione di anomalie. Sono stati utilizzati in totale 4 algoritmi di classificazione, 2 per ciascuna classe di algoritmi:

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

Algoritmi Statistici

La **Logistic Regression (regressione logistica)** è uno degli algoritmi 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]. L'algoritmo 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]:

$$P(Y = 1|X_1, \dots, X_n) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_n X_n)}} \quad (2.1)$$

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

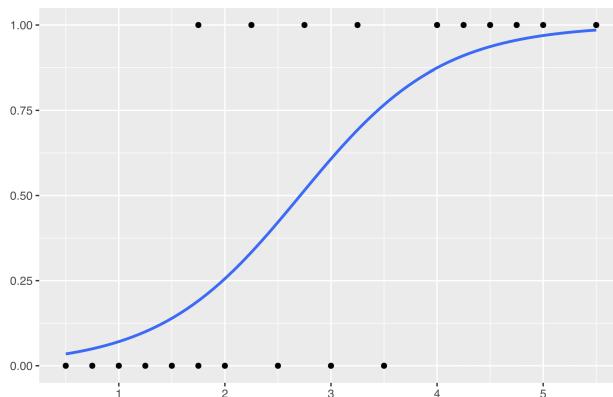


Figura 1.: Logistic Regression

L'altro algoritmo statistico utilizzato è stato la **Linear Discriminant Analysis (LDA)**. Questo algoritmo, 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.

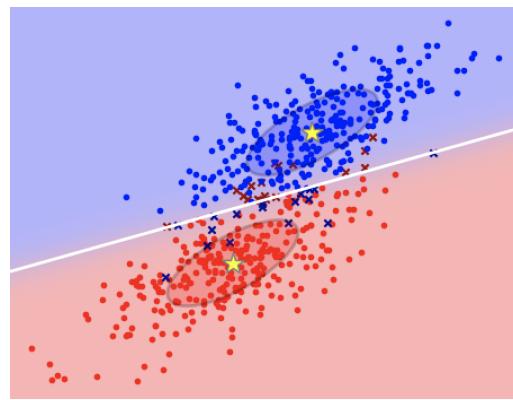


Figura 2.: Linear Discriminant Analysis

Algoritmi basati su Alberi Decisionali

Gli algoritmi non statistici utilizzati sono entrambi basati sugli alberi decisionali, quindi prima di definire i singoli algoritmi, si procederà 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 algoritmo adotta una struttura ad albero gerarchica, caratterizzata da un nodo radice, rami, nodi interni e nodi foglia[13].

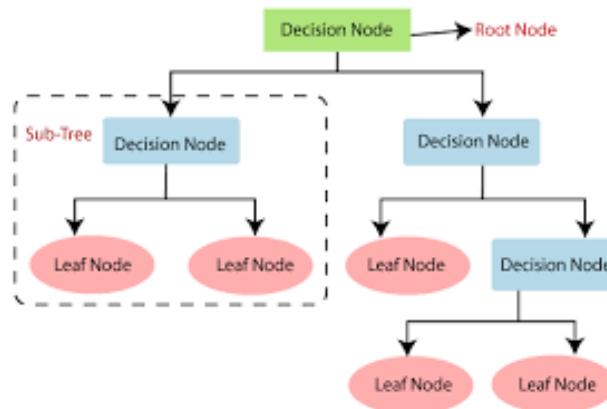


Figura 3.: Schema Albero Decisionale

Come rappresentato nel diagramma sopra, 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 alla 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.

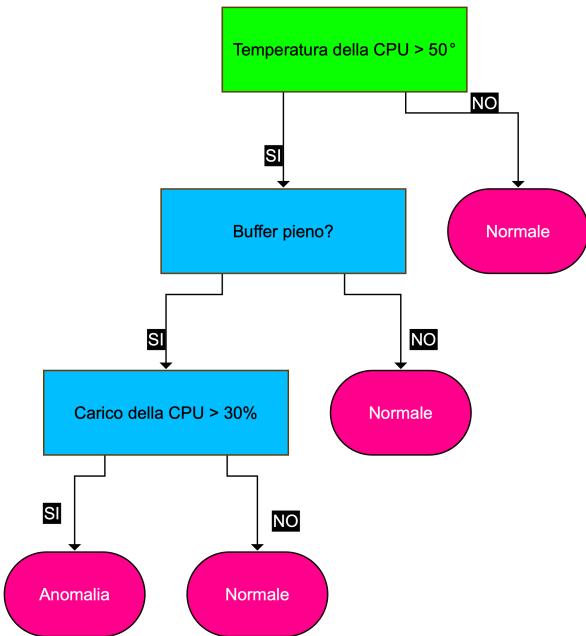


Figura 4.: Esempio Albero Decisionale

Nel nostro caso non sono stati usati direttamente degli alberi decisionali, ma degli algoritmi 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, il modello 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.

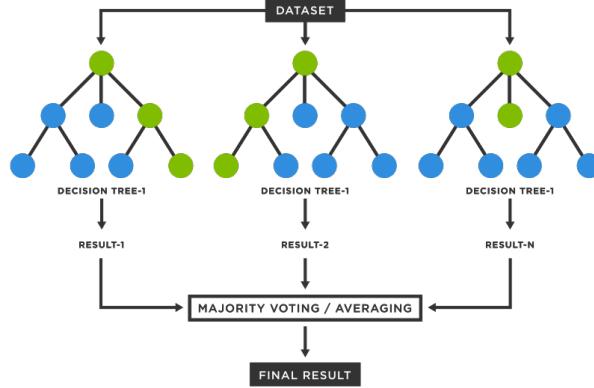


Figura 5.: Schema Random Forest

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 utilizzato 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 analizzare 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 3 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:

$$ACC = \frac{TP + TN}{P + N} \quad (2.2)$$

- **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:

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

Riprendiamo 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.

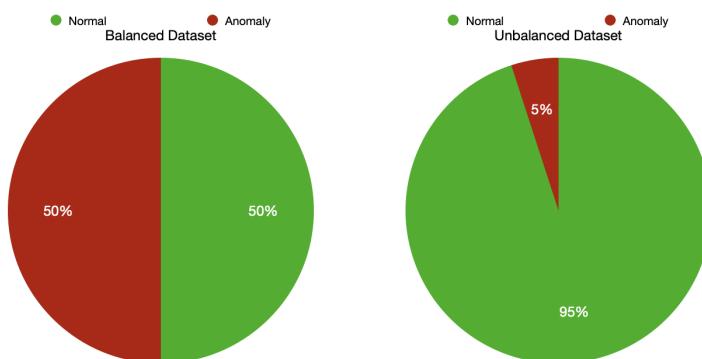


Figura 7.: Balanced e Unbalanced Datasets

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 [8]. Il rilevamento di anomalie è particolarmente importante nel settore della sicurezza informatica, ma anche nei settori quali finanza, medicina e molti altri ancora.

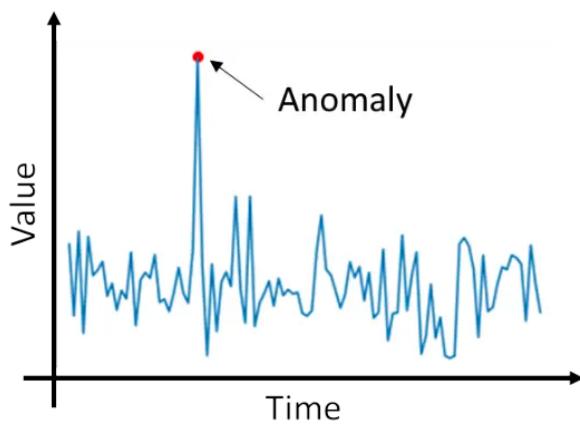


Figura 8.: Esempio Anomalia Puntuale

Un tempo chi si occupava di anomaly detection era solito esaminare 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'.

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 vengono definite 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 storica 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. 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

METODOLOGIA Sperimentale

3.1 DESCRIZIONE DATASET

I dataset utilizzati nel presente lavoro di Tesi sono il risultato della ricerca dell’articolo 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 features 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.

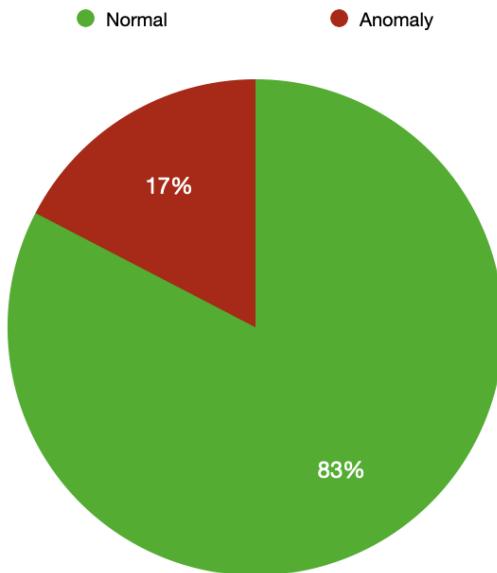


Figura 9.: Composizione Dataset unifi-filtered

3.1.2 Dataset all_3

Il dataset all_3 è stato utilizzato come test set per valutare le performance dei modelli su istanze diverse del problema. In realtà, non è stato utilizzato il dataset all_3 , ma una sua versione modificata. Avendo all_3 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 all_3 chiamata $my-all_3$, 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 $my-all_3$ sono presenti 105700 osservazioni totali, di cui 18245 etichettate come anomalie.

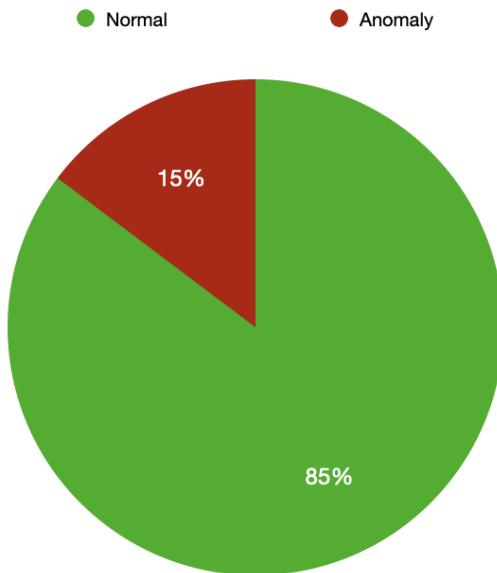


Figura 10.: Composizione Dataset my-all3

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 features 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.

```

def preprocessing_base(csv_path):
    """
    Creazione dataframe 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)
    """
    df = pd.read_csv(csv_path)

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

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

    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 METRICA SPEED SCORE

Per il problema trattato in questo lavoro di Tesi è stato ritenuto opportuno creare una nuova metrica, chiamata **Speed Score (SS)** per misurare le performance dei modelli. Lo scopo di questa metrica è quello di misurare la velocità con cui i modelli rilevano le anomalie. Ovviamente nell’anomaly detection è molto importante che le anomalie vengano rilevate nel minor tempo possibile. Per comprendere la formula del SS basta ricordare che le anomalie sono sempre presenti per 5 istanti di tempo (5 secondi), ovvero 5 righe del dataset. 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.

$$SS = \frac{x_0 \cdot 1 + x_1 \cdot 0.8^2 + x_2 \cdot 0.6^2 + x_3 \cdot 0.4^2 + x_4 \cdot 0.2^2}{\text{TotaleAnomalie}} \quad (3.1)$$

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.

3.4 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. Entrambi gli approcci time series sono stati ottenuti andando a modificare i dataset, aggiungendo delle nuove features che portassero dell’informazione time series con loro. 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*.

3.4.1 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.

3.4.2 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 $t = 1, 2, \dots, T$ contenente i valori osservati di una variabile Y dal tempo 1 al tempo T si definisce media mobile al tempo t il valore [3]:

$$SMA = \frac{1}{k} \sum_{i=t-k+1}^t y_i \quad \text{con } t \geq k \quad (3.2)$$

dove k è il periodo della media mobile, ed equivale al numero degli addendi.

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 features, 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 features siano evoluti nel tempo.

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

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

    df = df.dropna(axis=0)

    df = df.drop(columns=f'label_ma')

    return df
```

Figura 12.: Funzione per trasformazione dataset con features media mobile

3.4.3 Approccio Time Series con Differenze

Un altro modo di portare informazioni time series all'interno del dataset è stato quello di aggiungere delle nuove features che contenessero la differenza tra il valore attuale delle features 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 features che rappresentano la differenza tra il valore attuale all'istante t e i valori all'istante $t-1, t-2, \dots, t-n$.

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

    df = df.dropna(axis=0)

    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

3.5 WINDOW E SHUFFLE

Gli approcci finora descritti sono stati valutati sulla base di 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].

3.5.1 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 features 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.

3.5.2 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

In questo capitolo si analizza in primo luogo quale approccio, tra approccio classico (*classic*), approccio time series con media mobile (*TS-MA*) e approccio time series con differenze (*TS-Diff*), dimostra performance migliori, in termini di MCC, error rate e speed score. In secondo luogo, vengono discussi i risultati delle performance dei modelli sul dataset *my-all3*. Per concludere si commentano i risultati differenti al variare dei parametri *window* e *shuffle*.

4.1 PROGRESSIONE MCC, ERROR RATE E SPEED SCORE

In questa sezione sono presentati i risultati, quindi i grafici, della progressione dell'MCC, dell'Error Rate e dello Speed Score. Come detto nel Capitolo 3, non è appropriato rimescolare i dati quando si tratta di dati time series, quindi per i risultati discussi in questa sezione il parametro *shuffle* è sempre uguale a *False*. A seguire sono illustrati i grafici delle progressioni, partendo da una finestra temporale (*window*) uguale a 5 arrivando fino a 2.

- $window = 5$, $shuffle = False$:

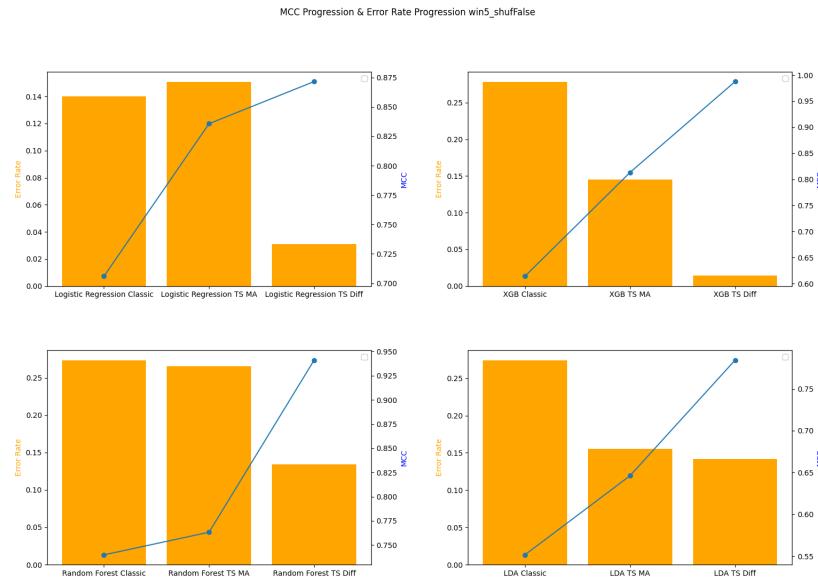


Figura 14.: Progressione MCC ed Error Rate window=5,shuffle=False

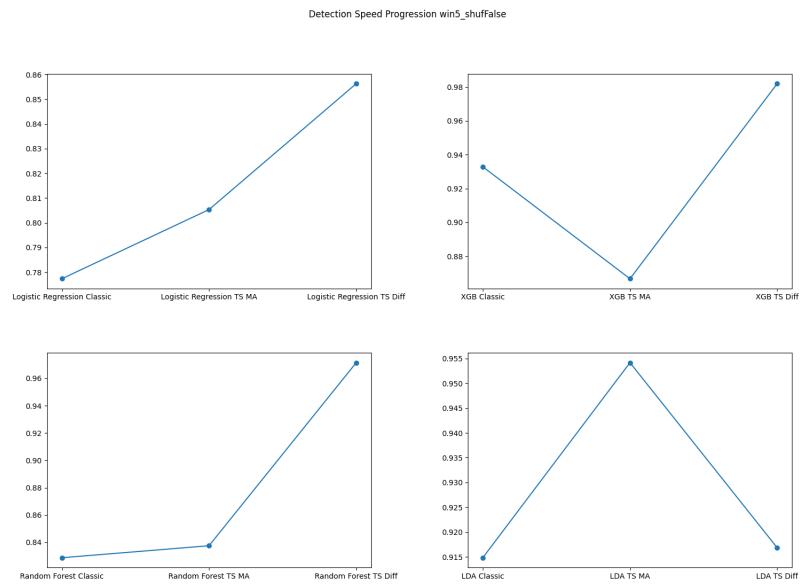


Figura 15.: Progressione Speed Score window=5,shuffle=False

- $window = 4$, $shuffle = False$:

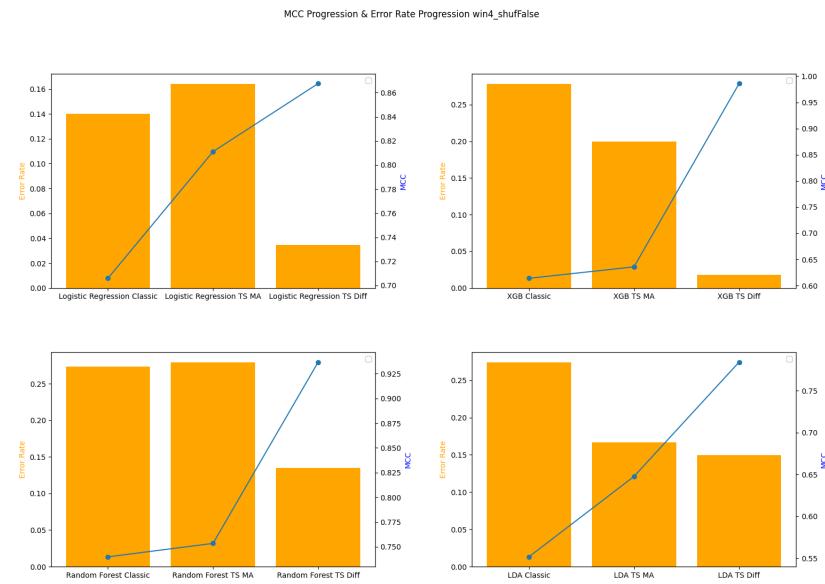


Figura 16.: Progressione MCC ed Error Rate window=4,shuffle=False

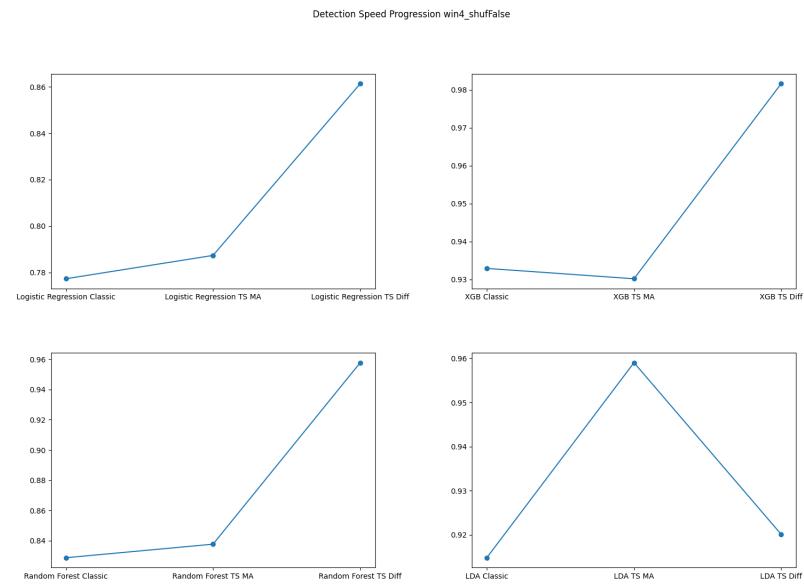


Figura 17.: Progressione Speed Score window=4,shuffle=False

- $window = 3, shuffle = False$:

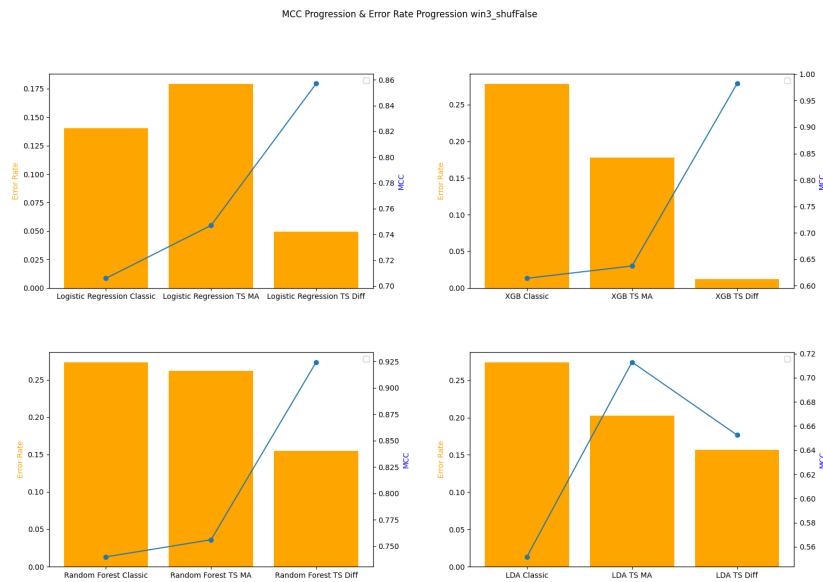


Figura 18.: Progressione MCC ed Error Rate window=3,shuffle=False

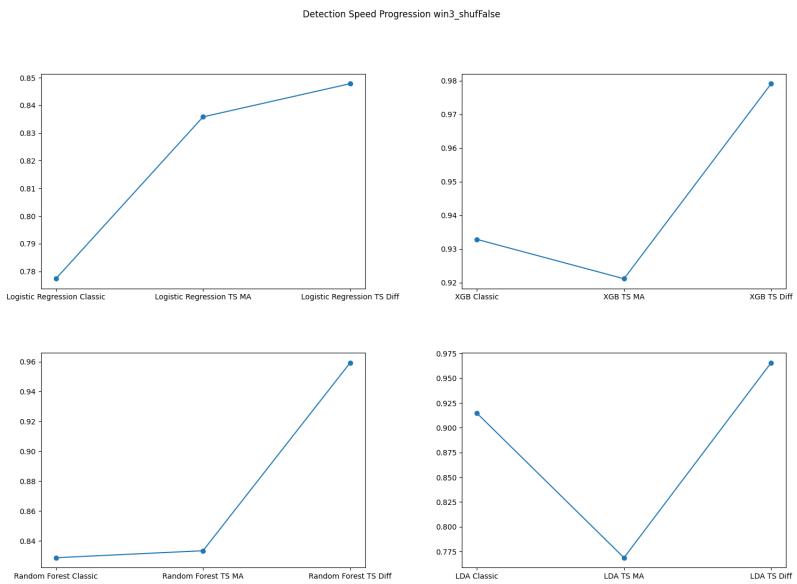


Figura 19.: Progressione Speed Score window=3,shuffle=False

- $window = 2$, $shuffle = False$:

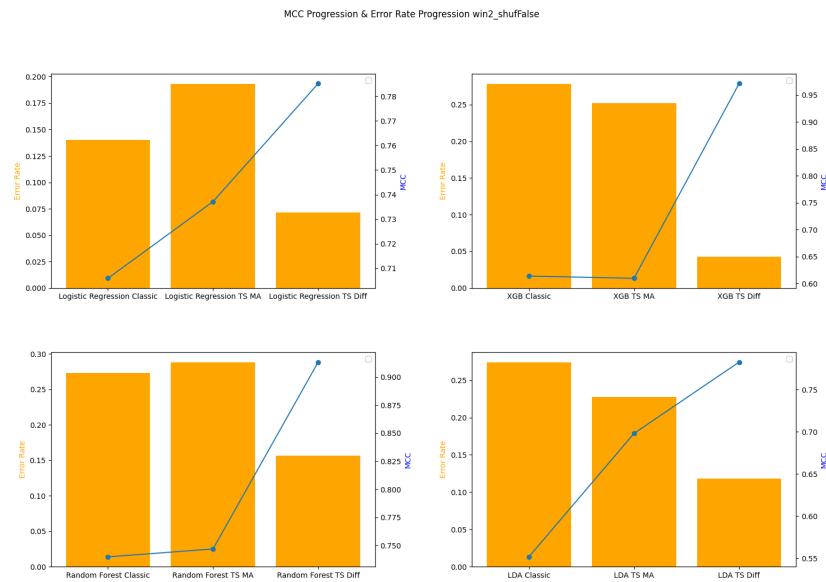


Figura 20.: Progressione MCC ed Error Rate window=2,shuffle=False

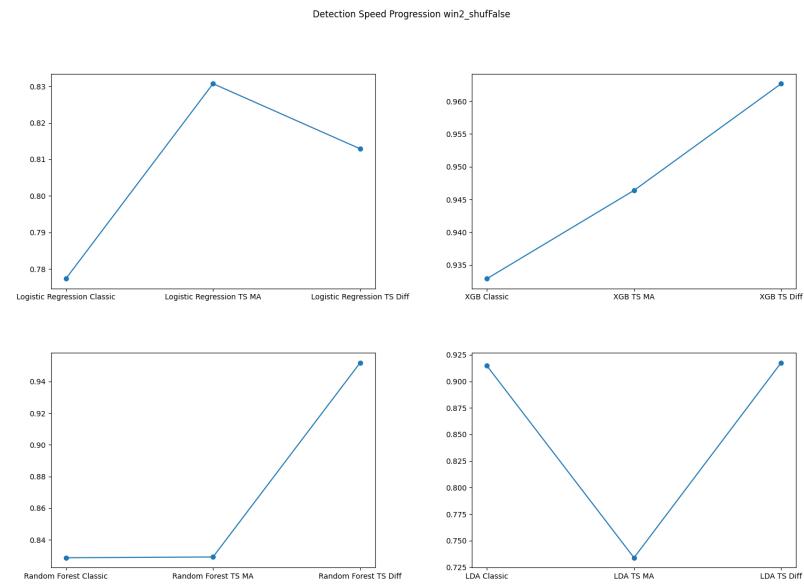


Figura 21.: Progressione Speed Score window=2,shuffle=False

I risultati ottenuti mostrano delle chiare tendenze:

- Nell'87.5% dei modelli (14 su 16) l'MCC è sempre crescente nel seguente ordine: approccio classico (*classic*), approccio time series con media mobile (*TS-MA*) e approccio time series con differenze (*TS-Diff*). In particolare, l'ultimo approccio menzionato, è sempre il migliore in termini di MCC. In media un approccio time series migliora l'MCC, scalato in un intervallo [0, 1], di 0.24 con picchi di miglioramento che variano da 0.16 a 0.38. Quindi, dato il valore medio di MCC per l'approccio *classic* di 0.65 e un'incremento medio di 0.24 con gli approcci time series, abbiamo un miglioramento delle performance in media del 37% in termini di MCC. Si ha quindi un ottimo aumento dei valori di MCC, con un miglioramento medio della metrica principale di più di $\frac{1}{3}$ rispetto ai valori ottenuti con un approccio classico.
- Nel 62.5% dei modelli (10 su 16) l'error rate segue la stessa tendenza dell'MCC, ovvero valori crescenti nell'ordine: *classic*, *TS-MA*, *TS-Diff*. Nel restante 37,5% dei modelli si ha un error rate maggiore con l'approccio *TS-MA* rispetto all'approccio *classic*. Per tutti i modelli, invece, vale sempre che l'error rate minore è dato dall'approccio *TS-Diff*. Il miglioramento medio di error rate da uno dei 2 approcci, *classic* o *TS-MA*, a un approccio *TS-Diff* è di 0.16 con estremi di 0.11 e 0.27, ricordando che l'error rate è compreso in un intervallo [0, 1]. Quindi, considerando per ogni modello l'error rate più alto tra approccio *classic* e *TS-MA*, passare ad un approccio *TS-Diff* porta a un miglioramento delle performance in termini di error rate del 64%, partendo da un error rate medio di 0.25 e avendo una decrescita media di 0.16. In conclusione, passare ad un approccio *TS-Diff* porta a una significativa riduzione dell'error rate, che viene più che dimezzato.
- Per quanto riguarda lo speed score non c'è una chiara tendenza: la tendenza che si ha per l'MCC vale per lo speed score solo nel 50% dei modelli (8 su 16). In particolare, per quanto riguarda questa metrica, la tendenza varia da algoritmo a algoritmo. Random Forest, ad esempio, segue sempre per ogni finestra temporale la tendenza prevalente per l'MCC. Anche Logistic Regression segue la stessa tendenza, a parte per la finestra temporale con *window* = 2 dove si ha in ordine crescente: approccio *classic*, *TS-Diff*, *TS-MA*. Per XGBoost, invece, la tendenza in ordine crescente è la seguente: approccio

TS-MA, classic, TS-Diff a parte per i modelli con $window = 2$ dove la tendenza corrisponde a quella prevalente per l'MCC. Infine, per l'algoritmo LDA, nei modelli addestrati su finestra temporale con $window = 4$ e $window = 5$ si hanno valori crescenti di speed score nel seguente ordine: approccio *classic*, *TS-Diff*, *TS-MA*. Mentre per i modelli con $window = 3$ e $window = 2$ si ha: approccio *TS-MA, classic, TS-Diff*. Questa metrica è sicuramente la meno robusta tra tutte le metriche prese in considerazione perché considera solo la velocità con cui l'anomalia viene rilevata senza prendere in considerazione l'accuratezza della previsione stessa, non essendo presente nella sua formula alcuna forma di penalizzazione in caso di previsione sbagliata. Tale metrica è da considerarsi come benchmark secondario da affiancare a una metrica robusta di riferimento come l'MCC.

4.2 MCC DATASET MY-ALL3

In questa sezione si sono analizzati i risultati che hanno ottenuto i modelli, in termini di MCC, usando come set di test il dataset *my-all3*. Nella prossima pagina sono presenti i grafici comparativi tra la progressione dell'MCC sul test set di *unifi-filtered* e sul dataset *my-all3*.

- $window = 5, shuffle = False$:

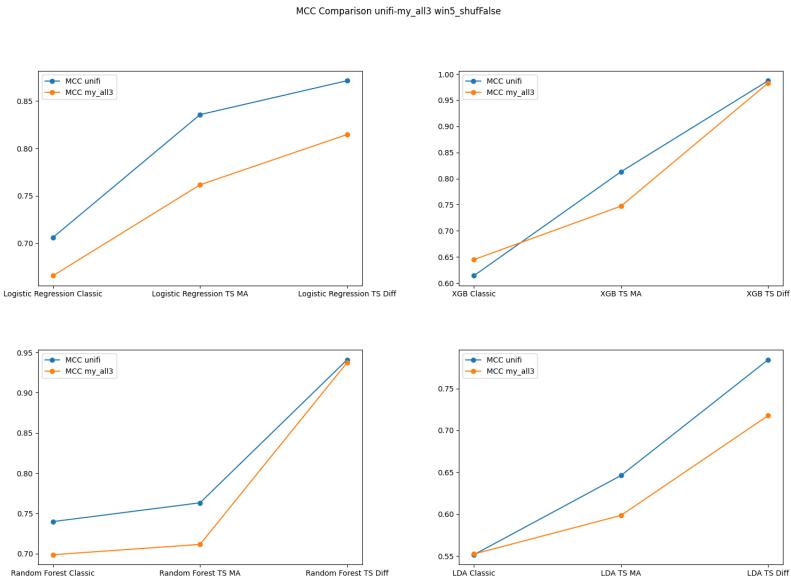


Figura 22.: Comparazione MCC unifi e my-all3 window=5,shuffle=False

- $window = 4, shuffle = False$:

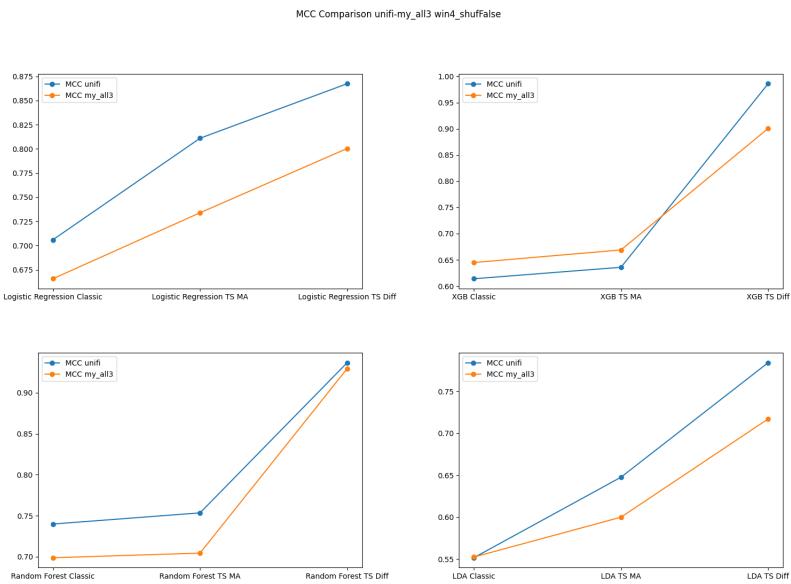


Figura 23.: Comparazione MCC unifi e my-all3 window=4,shuffle=False

- $window = 3, shuffle = False$:

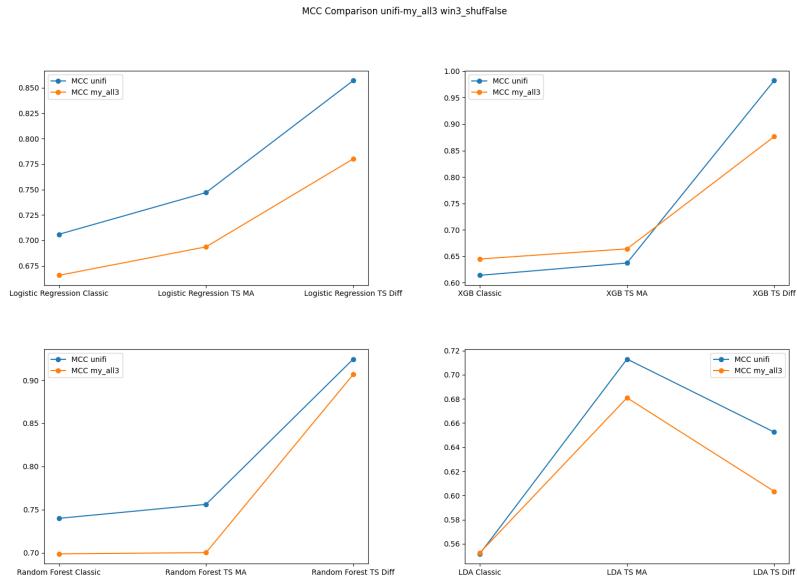


Figura 24.: Comparazione MCC unifi e my-all3 window=3,shuffle=False

- $window = 2, shuffle = False$:

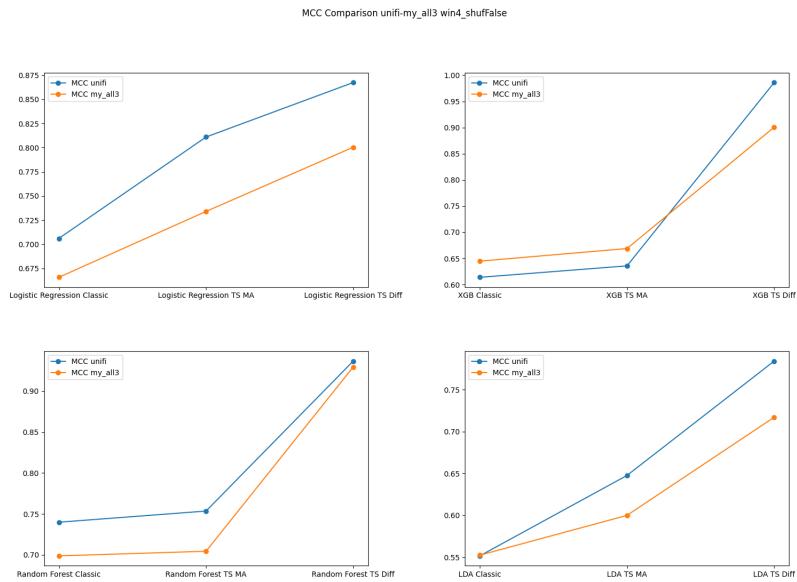


Figura 25.: Comparazione MCC unifi e my-all3 window=2,shuffle=False

Come si può notare dai risultati ottenuti per la maggior parte dei modelli, come ci si può aspettare, le performance sono migliori sul test set di *unifi-filtered*, perché i modelli sono stati addestrati su questa istanza del problema. In altri casi, si hanno performance leggermente migliori sul dataset *my-all3*. In generale si può dire che i modelli hanno generalizzato discretamente, non essendoci grandi differenze di MCC tra i due dataset. Sicuramente possiamo notare come la tendenza nella progressione dell'MCC sia sempre omogenea tra test set di *unifi-filtered* e *my-all3*.

4.3 PROGRESSIONE MCC AL VARIARE DELLA FINESTRA TEMPORALE

Un altro aspetto su cui si è posta l'attenzione è la variazione dell'MCC negli approcci time series al variare della finestra temporale. È un aspetto interessante da analizzare perché ci consente di trovare qual è la finestra temporale ottimale da utilizzare, a seconda delle esigenze. Una finestra temporale più ampia fornisce sicuramente al modello più informazioni da cui apprendere, ma al tempo stesso comporta una moltiplicazione non indifferente delle features nel dataset. È stato quindi rappresentato il grafico della progressione dell'MCC per gli approcci time series al variare della finestra temporale, per tutti i tipi di modelli.

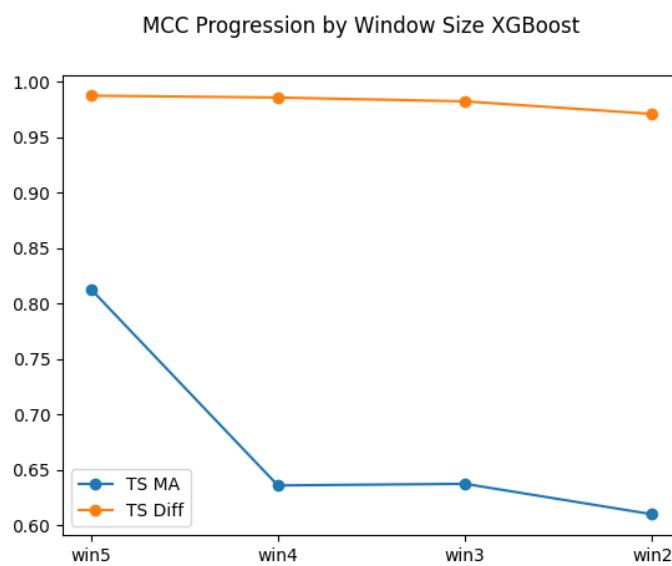


Figura 26.: Progressione MCC XGBoost in base alla finestra temporale

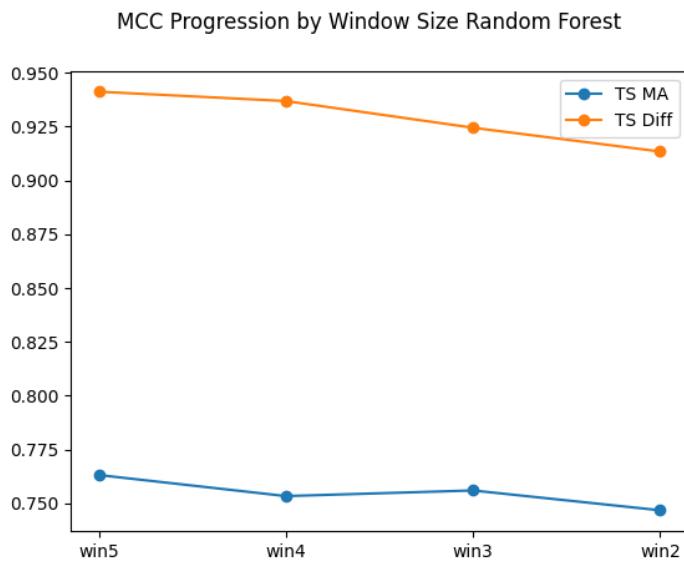


Figura 27.: Progressione MCC Random Forest in base alla finestra temporale

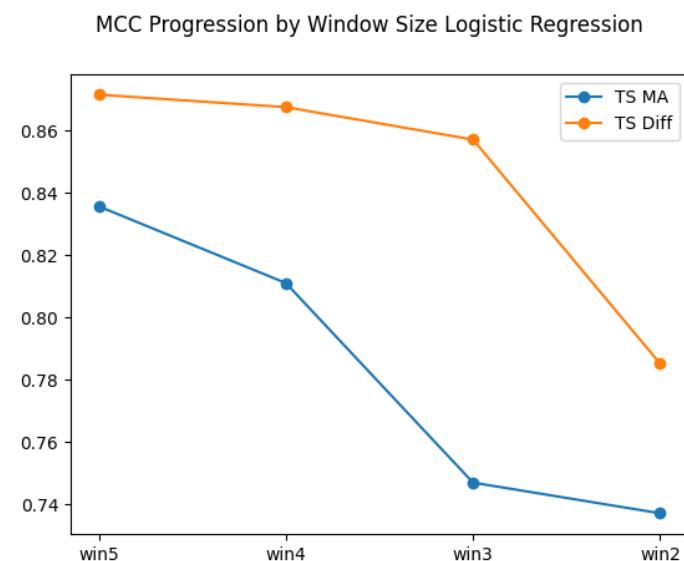


Figura 28.: Progressione MCC Logistic Regression in base alla finestra temporale

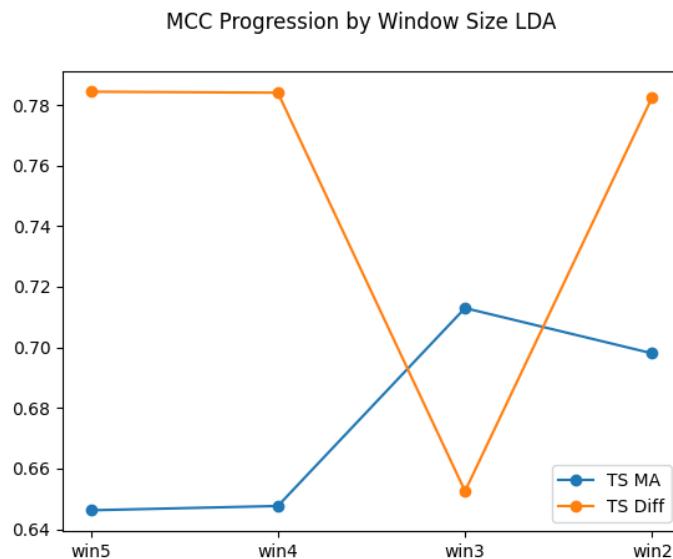


Figura 29.: Progressione MCC LDA in base alla finestra temporale

Dalle progressioni vediamo come le tendenze cambino da algoritmo a algoritmo. In generale, comunque, l'MCC a parte per poche eccezioni, diminuisce al diminuire dell'ampiezza della finestra temporale, avendo il modello meno dati a disposizione durante l'apprendimento.

4.4 CONFRONTO SHUFFLE TRUE / FALSE

Nella presente sezione si sono volute analizzare le performance e le differenze dei modelli addestrati con il parametro $shuffle = True$. Possiamo osservare nei grafici riportati qui sotto la progressione dell'MCC nei modelli addestrati su una finestra temporale $window = 5$ con $shuffle = True$ e $shuffle = False$.

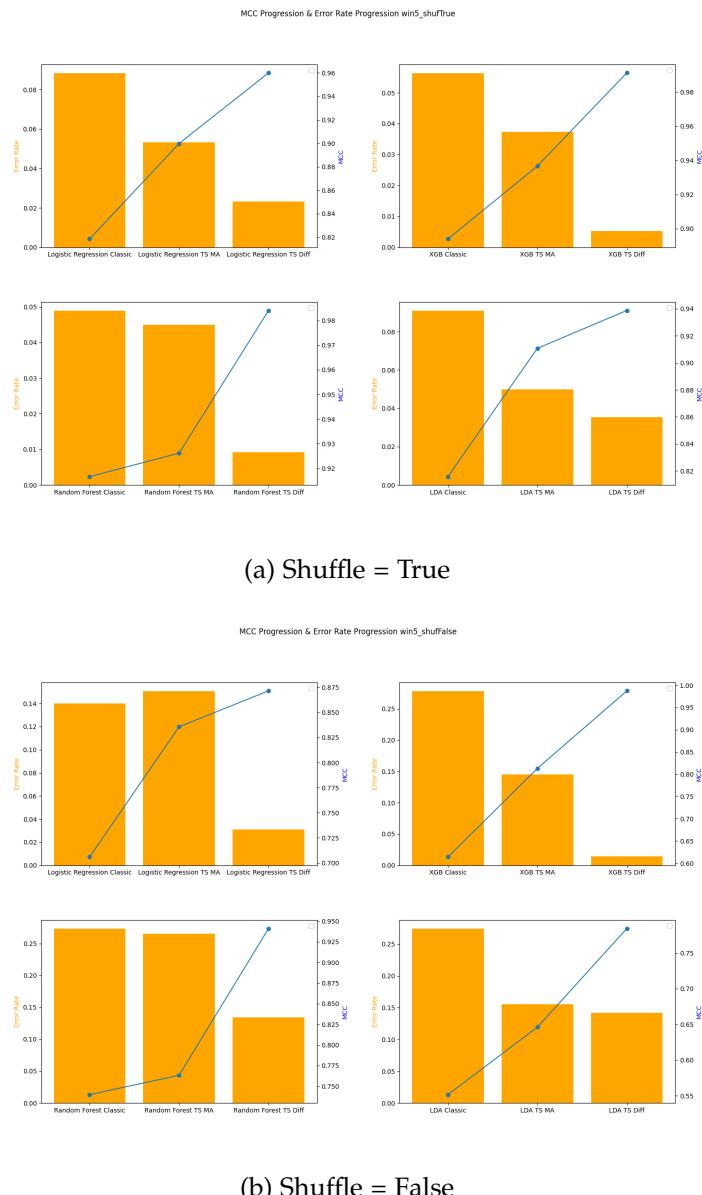


Figura 30.: Comparazione Progressione MCC shuffle = True/False

Come si può notare dai risultati ottenuti, i modelli addestrati con il parametro *shuffle* impostato a *True* ottengono performance migliori in termini di MCC. L'ipotesi da cui siamo partiti per spiegare questo fenomeno è che i modelli addestrati con *shuffle* = *False* dessero troppa importanza a delle features "inaffidabili" e che quindi portassero a un decremento delle prestazioni dei modelli. Per verificare questa ipotesi sono stati rappresentati i grafici delle prime 20 features più importanti per ogni modello, ovvero quelle a cui veniva attribuito maggior peso. A titolo di esempio vengono mostrate le feature importance per i modelli *classic XGBoost* con *window* = 5.

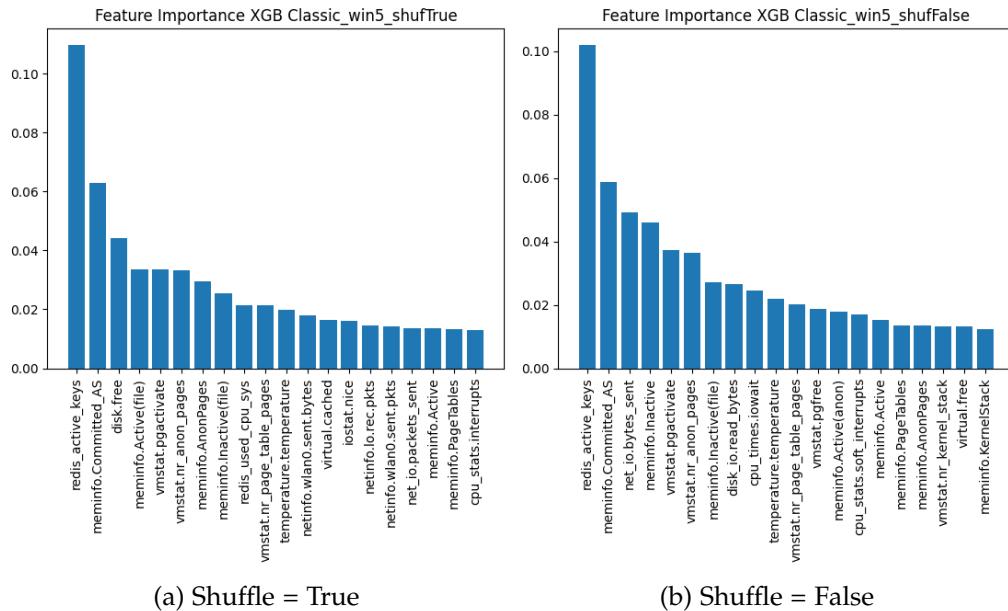


Figura 31.: Comparazione Feature Importance XGBoost classic window = 5
shuffle = True/False

Come volevasi dimostrare, le feature importance dei due modelli sono differenti. Ci sono alcune features in comune nella stessa posizione, altre in posizione diversa e altre ancora sono presenti in una e non nell'altra. Diamo ora la definizione di feature inaffidabile: feature non presente nel grafico (a) e presente nel grafico (b) o, anche se presente nel grafico (a), non in prossimità della posizione individuata nel grafico (b) [31]. Sono state quindi eliminate le feature inaffidabili dal dataset ed è stato eseguito nuovamente il training del modello con il parametro *shuffle* = *False* sul nuovo dataset. Questa operazione di eliminazione delle features

inaffidabili è stata eseguita per ogni modello con approccio *classic*, perché si è dimostrato l'approccio con maggiore differenza in termini di MCC tra modelli addestrati con e senza *shuffle*, come si vede nel grafico 30. L'analisi delle feature inaffidabili è stata svolta solo sui modelli addestrati con approccio *classic*, dato che il miglioramento più sensibile era previsto per questi modelli, ma un miglioramento è atteso anche per i modelli addestrati con approcci diversi. Infine i nuovi modelli sono stati testati sul dataset *my-all3* ed è stata effettuata una comparazione con i vecchi modelli, in termini di MCC.

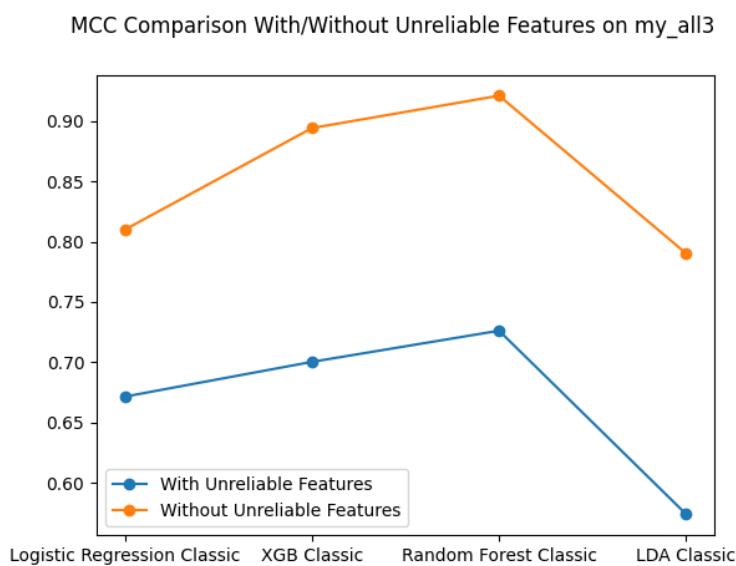


Figura 32.: Comparazione MCC modelli con e senza feature inaffidabili

Come si vede dal grafico riportato qui sopra, c'è un netto miglioramento delle performance da parte dei modelli che sono stati addestrati escludendo le feature inaffidabili.

5

CONCLUSIONI

L’obiettivo del presente lavoro di Tesi è stato quello di esplorare degli approcci time series nel campo dell’anomaly detection e valutare i potenziali benefici dell’applicazione di tali approcci. L’indagine sperimentale ha portato ad ottimi risultati, rivelando che un approccio time series risulta essere molto più efficace di un approccio classico all’analisi dei dati. In particolare possiamo concludere che:

- Un approccio time series determina un aumento delle performance dei modelli. In particolare, si ha un ottimo aumento dei valori di MCC e un error rate più che dimezzato. Tra tutti gli approcci analizzati, l’approccio time series con differenze risulta essere il migliore.
- La validità dei risultati ottenuti vale anche per istanze diverse del problema, come è stato verificato con il dataset *my-all3*, tenendo però in considerazione che la generalità dei modelli è stata testata usando un feature set limitato e sottoposto ad operazioni di preprocessing.
- L’ampiezza della finestra temporale è il parametro da ottimizzare. Una maggiore ampiezza garantisce più informazioni in fase di addestramento dei modelli portando a performance migliori, a discapito però di un maggior numero di features e quindi di un maggior costo computazionale.
- L’introduzione di una metrica come lo Speed Score (SS) nella valutazione di anomaly detectors può essere un buon indice da affiancare a metriche più robuste, come l’MCC.

In conclusione, un approccio time series si è dimostrato essere una valida, ma soprattutto migliore, alternativa all’approccio classico all’analisi dei dati nell’ambito dell’anomaly detection.

A

CODICE SVILUPPATO

Il codice sviluppato per il lavoro di Tesi è presente al seguente link:
https://github.com/xaldvp/Tesi_Triennale, inoltre sono disponibili tutte le risorse prodotte dal codice, come grafici e file csv. Di seguito viene riportato il codice sviluppato.

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import matthews_corrcoef
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
import csv
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```
def preprocessing_base(csv_path):
    """
    Creazione dataframe 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)
    """
    df = pd.read_csv(csv_path)

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

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

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

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

    return X,y,df
```

```
def my_LogisticRegression_pipeline(preprocessor, X_train, y_train):
    # LogisticRegression training
    modelLog = make_pipeline(preprocessor, LogisticRegression(max_iter=1000))
    modelLog.fit(X_train, y_train)

    return modelLog
```

```
def my_LDA_pipeline(preprocessor,X_train,y_train):
    #LDA training
    modelLDA=make_pipeline(preprocessor,LinearDiscriminantAnalysis())
    modelLDA.fit(X_train,y_train)

    return modelLDA
```

```
def my_RandomForest_pipeline(preprocessor, X_train, y_train):
    # RandomForest training
    modelRand = make_pipeline(preprocessor, RandomForestClassifier(random_state=42))
    modelRand.fit(X_train, y_train)

    return modelRand
```

```
def my_XGB_pipeline(preprocessor, X_train, y_train):
    # XGB training
    modelXGB = make_pipeline(preprocessor, xgb.XGBClassifier(random_state=42))
    modelXGB.fit(X_train, y_train)

    return modelXGB
```

```
def is_csv_empty(file_path):
    #Utility per scrivere su file csv
    with open(file_path, 'r') as csvfile:
        csvreader = csv.reader(csvfile)
        try:
            first_row = next(csvreader)
            return False if first_row else True
        except StopIteration:
            return True

def split_array(array, size):
    #Utility per convertire un array in un array bidimensionale composto da array di 5 elementi ciascuno
    split_array = []
    for i in range(0, len(array), size):
        split_array.append(array[i:i+size])
    split_array = [subarray for subarray in split_array if len(subarray) == 5]

    return split_array
```

```
def get_mcc (csv_name,model, X_test, y_test,csv_note=None):
    #Calcola MCC e scrive su file csv
    MCC=(matthews_corrcoef(y_test, model.predict(X_test))+1)/2 # MCC scalato a [0,1]

    if csv_note is not None:
        with open(csv_name, 'a', newline='') as file:
            fieldnames = ['csv_note','MCC']
            writer = csv.DictWriter(file, fieldnames=fieldnames)
            if is_csv_empty(csv_name):
                writer.writeheader()

            writer.writerow({'csv_note': csv_note,'MCC': MCC})

    return MCC
```

```

def get_accuracy_cross_val (csv_name,model, X_train, y_train,csv_note=None):
    #Calcola accuracy ottenuta tramite cross validation e scrive su file csv
    accuracies_lr = cross_val_score(estimator=model, X=X_train, y=y_train)
    avg_accuracy=np.mean(accuracies_lr)

    if csv_note is not None:
        with open(csv_name, 'a', newline='') as file:
            fieldnames = ['csv_note','Accuracy']
            writer = csv.DictWriter(file, fieldnames=fieldnames)
            if is_csv_empty(csv_name):
                writer.writeheader()

            writer.writerow({'csv_note': csv_note,'Accuracy': avg_accuracy})

    return avg_accuracy

```

```

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

    df = df.dropna(axis=0)

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

    return df

```

```

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

    df = df.dropna(axis=0)

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

    return df

```

```

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

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

    df = df.dropna(axis=0)

    df = df.drop(columns=f'label_ma')

    return df

```

```

def speed_detection_score(csv_name,model, X_test: pd.DataFrame, y_test: pd.DataFrame,csv_note=None):
    #Calcolo dello speed detection score come media pesata delle frequenze di rilevamento (decrescita quadratica)
    X_test['label'] = y_test
    X_test=X_test.sort_index()

    X_test_anomalies = X_test[X_test['label'] == 1]
    X_test_anomalies = X_test_anomalies.drop(columns='label')

    predictions = model.predict(X_test_anomalies)
    predictions = split_array(predictions, 5)

    counters={}
    numerator=0
    denominator=0
    score=0

    for i in range(5):
        counters[f'count_{i}']=0

    for x in predictions:
        if (x[0] == 1):
            counters['count_0'] = counters['count_0']+1
        elif (x[1] == 1):
            counters['count_1'] = counters['count_1']+1
        elif (x[2] == 1):
            counters['count_2'] = counters['count_2']+1
        elif (x[3] == 1):
            counters['count_3'] = counters['count_3']+1
        elif (x[4] == 1):
            counters['count_4'] = counters['count_4']+1

    numerator=(counters['count_0']*1+counters['count_1']*0.8**2+counters['count_2']*0.6**2+counters['count_3']*0.4**2+counters['count_4']*0.2**2)
    denominator=(counters['count_0']+counters['count_1']+counters['count_2']+counters['count_3']+counters['count_4']))
    score=numerator/denominator

    if csv_note is not None:
        with open(csv_name, 'a', newline='') as file:
            fieldnames = ['csv_note','speed_0','speed_1','speed_2','speed_3','speed_4']
            writer = csv.DictWriter(file, fieldnames=fieldnames)
            if is_csv_empty(csv_name):
                writer.writeheader()

            writer.writerow({'csv_note': csv_note,'speed_0': counters['count_0'],
                            'speed_1':counters['count_1'],'speed_2': counters['count_2'],
                            'speed_3':counters['count_3'],'speed_4':counters['count_4']})


    return score

```

```

def feature_importance_tree(note, model ,X_train):
    #Calcola la feature importance per i modelli basati su alberi decisionali e ne salva il plot delle prime 20 feature
    feature_importances = model.steps[-1][1].feature_importances_
    features = X_train.columns
    importances_df = pd.DataFrame(data={
        'Feature': features,
        'Importance': feature_importances
    })
    importances_df = importances_df.sort_values(by='Importance', ascending=False).head(20)

    plt.bar(x=importances_df['Feature'], height=importances_df['Importance'])
    plt.title(f'{note}')
    plt.xticks(rotation='vertical')
    plt.savefig(f'{note}',bbox_inches='tight')
    plt.show()

    return importances_df

```

```

def feature_importance_coef(note, model ,X_train):
    #Calcola la feature importance per i modelli statistici e ne salva il plot delle prime 20 feature
    feature_importances = model.steps[-1][1].coef_[0]
    features = X_train.columns
    importances_df = pd.DataFrame(data={
        'Feature': features,
        'Importance': np.abs(feature_importances)
    })
    importances_df = importances_df.sort_values(by='Importance', ascending=False).head(20)

    plt.bar(x=importances_df['Feature'], height=importances_df['Importance'])
    plt.title(f'{note}')
    plt.xticks(rotation='vertical')
    plt.savefig(f'{note}',bbox_inches='tight')
    plt.show()

    return importances_df

```

```

def training_classic(shuffle,X,y,fig_details):
    #Training dei modelli con approccio classico
    preprocessor=StandardScaler()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, shuffle=shuffle)

    modelXGBClassic=my_XGB_pipeline(preprocessor,X_train,y_train)

    modelLogClassic=my_LogisticRegression_pipeline(preprocessor,X_train,y_train)

    modelRandClassic=my_RandomForest_pipeline(preprocessor,X_train,y_train)

    modelLDAClassic=my_LDA_pipeline(preprocessor,X_train,y_train)

    classic_models={

        'XGB':modelXGBClassic,
        'LogisticRegression':modelLogClassic,
        'RandomForest':modelRandClassic,
        'LDA':modelLDAClassic
    }

    feature_importance_tree('Feature Importance XGB Classic_'+fig_details,classic_models['XGB'],X_train)
    feature_importance_coef('Feature Importance Logistic Regression Classic_'+fig_details,classic_models['LogisticRegression'],X_train)
    feature_importance_tree('Feature Importance Random Forest Classic_'+fig_details,classic_models['RandomForest'],X_train)
    feature_importance_coef('Feature Importance LDA Classic_'+fig_details,classic_models['LDA'],X_train)

    return classic_models,X_train,y_train,X_test,y_test

```

```

def map_creation_classic(models,X_train,y_train,X_test,y_test,csv_details):
    #Crea dei dizionari che contengono rispettivamente MCC, accuracy e speed score per ogni modello
    mcc_map={
        'Logistic Regression Classic':get_mcc('mcc_+csv_details+.csv',models['LogisticRegression'],X_test,y_test,'Logistic Regression Classic'),
        'XGB Classic':get_mcc('mcc_+csv_details+.csv',models['XGB'],X_test,y_test,'XGB Classic'),
        'Random Forest Classic':get_mcc('mcc_+csv_details+.csv',models['RandomForest'],X_test,y_test,'Random Forest Classic'),
        'LDA Classic':get_mcc('mcc_+csv_details+.csv',models['LDA'],X_test,y_test,'LDA Classic')
    }

    accuracy_map={
        'Logistic Regression Classic':get_accuracy_cross_val('accuracy_+csv_details+.csv',models['LogisticRegression'],X_train,y_train,'Logistic Regression Classic'),
        'XGB Classic':get_accuracy_cross_val('accuracy_+csv_details+.csv',models['XGB'],X_train,y_train,'XGB Classic'),
        'Random Forest Classic':get_accuracy_cross_val('accuracy_+csv_details+.csv',models['RandomForest'],X_train,y_train,'Random Forest Classic'),
        'LDA Classic':get_accuracy_cross_val('accuracy_+csv_details+.csv',models['LDA'],X_train,y_train,'LDA Classic')
    }

    speed_map={
        'Logistic Regression Classic':speed_detection_score('speed_+csv_details+.csv',models['LogisticRegression'],X_test,y_test,'Logistic Regression Classic'),
        'XGB Classic':speed_detection_score('speed_+csv_details+.csv',models['XGB'],X_test,y_test,'XGB Classic'),
        'Random Forest Classic':speed_detection_score('speed_+csv_details+.csv',models['RandomForest'],X_test,y_test,'Random Forest Classic'),
        'LDA Classic':speed_detection_score('speed_+csv_details+.csv',models['LDA'],X_test,y_test,'LDA Classic')
    }
    return mcc_map,accuracy_map,speed_map

```

```

def training_ts_diff(shuffle,X,y,df,size,fig_details):
    #Training dei modelli con approccio time series con differenze
    new_df_difference = window_input_difference(size, df)
    y = new_df_difference['label']
    X = new_df_difference.drop(columns='label')

    preprocessor=StandardScaler()

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, shuffle=shuffle)

    modelLogTS=my_LogisticRegression_pipeline(preprocessor,X_train,y_train)

    modelRandTS=my_RandomForest_pipeline(preprocessor,X_train,y_train)

    modelXGBTTS=my_XGB_pipeline(preprocessor,X_train,y_train)

    modelLDATS=my_LDA_pipeline(preprocessor,X_train,y_train)

    ts_diff_models={
        'XGB':modelXGBTTS,
        'LogisticRegression':modelLogTS,
        'RandomForest':modelRandTS,
        'LDA':modelLDATS
    }

    feature_importance_tree('Feature Importance XGB TS Diff_+'+fig_details,ts_diff_models['XGB'],X_train)
    feature_importance_coeff('Feature Importance Logistic Regression TS Diff_+'+fig_details,ts_diff_models['LogisticRegression'],X_train)
    feature_importance_tree('Feature Importance Random Forest TS Diff_+'+fig_details,ts_diff_models['RandomForest'],X_train)
    feature_importance_coeff('Feature Importance LDA TS Diff_+'+fig_details,ts_diff_models['LDA'],X_train)

    return ts_diff_models,X_train,y_train,X_test,y_test

```

```

def map_update_TS(models,TS_type,X_train,y_train,X_test,y_test,mcc_map,accuracy_map,speed_map,csv_details):
    #Update dei dizionari
    keys=[None,None,None,None]
    if TS_type=='Diff':
        keys=['Logistic Regression TS Diff','XGB TS Diff','Random Forest TS Diff','LDA TS Diff']
    elif (TS_type=='MA'):
        keys=['Logistic Regression TS MA','XGB TS MA','Random Forest TS MA','LDA TS MA']

    mcc_map.update({
        keys[0]:get_mcc('mcc_'+csv_details+'.csv',models['LogisticRegression'],X_test,y_test,keys[0]),
        keys[1]:get_mcc('mcc_'+csv_details+'.csv',models['XGB'],X_test,y_test,keys[1]),
        keys[2]:get_mcc('mcc_'+csv_details+'.csv',models['RandomForest'],X_test,y_test,keys[2]),
        keys[3]:get_mcc('mcc_'+csv_details+'.csv',models['LDA'],X_test,y_test,keys[3])
    })

    accuracy_map.update({
        keys[0]:get_accuracy_cross_val('accuracy_'+csv_details+'.csv',models['LogisticRegression'],X_train,y_train,keys[0]),
        keys[1]:get_accuracy_cross_val('accuracy_'+csv_details+'.csv',models['XGB'],X_train,y_train,keys[1]),
        keys[2]:get_accuracy_cross_val('accuracy_'+csv_details+'.csv',models['RandomForest'],X_train,y_train,keys[2]),
        keys[3]:get_accuracy_cross_val('accuracy_'+csv_details+'.csv',models['LDA'],X_train,y_train,keys[3])
    })

    speed_map.update({
        keys[0]:speed_detection_score('speed_'+csv_details+'.csv',models['LogisticRegression'],X_test,y_test,keys[0]),
        keys[1]:speed_detection_score('speed_'+csv_details+'.csv',models['XGB'],X_test,y_test,keys[1]),
        keys[2]:speed_detection_score('speed_'+csv_details+'.csv',models['RandomForest'],X_test,y_test,keys[2]),
        keys[3]:speed_detection_score('speed_'+csv_details+'.csv',models['LDA'],X_test,y_test,keys[3])
    })
return mcc_map,accuracy_map,speed_map

```

```

def training_ts_ma(shuffle,X,y,df,size,fig_details):
    #Training dei modelli con approccio time series con media mobile
    new_df_moving_average = window_input_moving_average(size, df)
    y = new_df_moving_average['label']
    X = new_df_moving_average.drop(columns='label')

    preprocessor=StandardScaler()

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, shuffle=shuffle)

    modelLogTSMA=my_LogisticRegression_pipeline(preprocessor,X_train,y_train)

    modelRandTSMA=my_RandomForest_pipeline(preprocessor,X_train,y_train)

    modelXGBTSMa=my_XGB_pipeline(preprocessor,X_train,y_train)

    modelLDA_TSMA=my_LDA_pipeline(preprocessor,X_train,y_train)

    ts_ma_models={

        'XGB':modelXGBTSMa,
        'LogisticRegression':modelLogTSMA,
        'RandomForest':modelRandTSMA,
        'LDA':modelLDA_TSMA
    }

    feature_importance_tree('Feature Importance XGB TS MA_'+fig_details,ts_ma_models['XGB'],X_train)
    feature_importance_coef('Feature Importance Logistic Regression TS MA_'+fig_details,ts_ma_models['LogisticRegression'],X_train)
    feature_importance_tree('Feature Importance Random Forest TS MA_'+fig_details,ts_ma_models['RandomForest'],X_train)
    feature_importance_coef('Feature Importance LDA TS MA_'+fig_details,ts_ma_models['LDA'],X_train)

    return ts_ma_models,X_train,y_train,X_test,y_test

```

```
def progression_plot(ax, map, note, mcc_map=None):
    #Utility per costruire i plot di progressione per MCC,error rate,accuracy e speed score
    x_values = list(map.keys())
    map_values=list(map.values()) #Può essere speed_map o accuracy_map

    if mcc_map==None:
        ax.plot(x_values, map_values, marker='o', linestyle='--', label='Speed Score')
    else:
        err_values=[1-x for x in map_values]
        mcc_values = list(mcc_map.values())
        ax.bar(x_values, err_values,color='orange')
        ax.set_ylabel('Error Rate', color='orange')
        ax2=ax.twinx()
        ax2.plot(x_values, mcc_values, marker='o', linestyle='--')
        ax2.set_ylabel('MCC', color='blue')
        ax.legend()
        ax2.legend()
```

```

def mcc_accuracy_progression(mcc_map,accuracy_map,details):
    #Plot progressione MCC, error rate e accuracy
    logistic_mcc_map={}
    random_forest_mcc_map={}
    xgb_mcc_map={}
    lda_mcc_map={}

    logistic_accuracy_map={}
    random_forest_accuracy_map={}
    xgb_accuracy_map={}
    lda_accuracy_map={}

    for key, value in mcc_map.items():
        if 'Logistic' in key:
            logistic_mcc_map[key] = value
        if 'Random' in key:
            random_forest_mcc_map[key] = value
        if 'XGB' in key:
            xgb_mcc_map[key] = value
        if 'LDA' in key:
            lda_mcc_map[key] = value

    for key, value in accuracy_map.items():
        if 'Logistic' in key:
            logistic_accuracy_map[key] = value
        if 'Random' in key:
            random_forest_accuracy_map[key] = value
        if 'XGB' in key:
            xgb_accuracy_map[key] = value
        if 'LDA' in key:
            lda_accuracy_map[key] = value

    fig, ax = plt.subplots(2, 2, figsize=(18,12))
    fig.suptitle('MCC Progression & Error Rate Progression '+details)
    fig.subplots_adjust(wspace=0.3)
    fig.subplots_adjust(hspace=0.3)
    progression_plot(ax[0,0],logistic_accuracy_map,'Logistic_Regression_MCC_Progression',logistic_mcc_map)
    progression_plot(ax[0,1],xgb_accuracy_map,'XGB_MCC_Progression',xgb_mcc_map)
    progression_plot(ax[1,0],random_forest_accuracy_map,'Random_Forest_MCC_Progression',random_forest_mcc_map)
    progression_plot(ax[1,1],lda_accuracy_map,'LDA_MCC_Progression',lda_mcc_map)
    plt.savefig('MCC_Progression_'+details)

    fig, ax = plt.subplots(2, 2, figsize=(18,12))
    fig.suptitle('Accuracy Progression '+details)
    fig.subplots_adjust(wspace=0.3)
    fig.subplots_adjust(hspace=0.3)
    progression_plot(ax[0,0],logistic_accuracy_map,'Logistic_Regression_Accuracy_Progression')
    progression_plot(ax[0,1],xgb_accuracy_map,'XGB_Accuracy_Progression')
    progression_plot(ax[1,0],random_forest_accuracy_map,'Random_Forest_Accuracy_Progression')
    progression_plot(ax[1,1],lda_accuracy_map,'LDA_Accuracy_Progression')
    plt.savefig('Accuracy_Progression_'+details)

    return logistic_mcc_map,random_forest_mcc_map,xgb_mcc_map,lda_mcc_map

```

```
def speed_progression(speed_map,details):
    #Plot progressionne speed score
    logistic_speed_map={}
    random_forest_speed_map={}
    xgb_speed_map={}
    lda_speed_map={}

    for key, value in speed_map.items():
        if 'Logistic' in key:
            logistic_speed_map[key] = value
        if 'Random' in key:
            random_forest_speed_map[key] = value
        if 'XGB' in key:
            xgb_speed_map[key] = value
        if 'LDA' in key:
            lda_speed_map[key] = value

    fig, ax = plt.subplots(2, 2, figsize=(18,12))
    fig.suptitle('Detection Speed Progression '+details)
    fig.subplots_adjust(wspace=0.3)
    fig.subplots_adjust(hspace=0.3)
    progression_plot(ax[0,0],logistic_speed_map,'Logistic_Regression_Speed_Progression')
    progression_plot(ax[0,1],xgb_speed_map,'XGB_Speed_Progression')
    progression_plot(ax[1,0],random_forest_speed_map,'Random_Forest_Speed_Progression')
    progression_plot(ax[1,1],lda_speed_map,'LDA_Speed_Progression')
    plt.savefig('Speed_Progression_'+details)
```

```

def mcc_all3(classic_models,ts_diff_models,ts_ma_models,size,shuffle):
    #Calcolo MCC per tutti i modelli usando come test set il dataset my_all3
    homenet_df=pd.read_csv('preprocessed_arancino_datasets/homenet_filtered.csv')
    mobile_df=pd.read_csv('preprocessed_arancino_datasets/mobile_filtered.csv')
    unifi_df=pd.read_csv('preprocessed_arancino_datasets/unifi_filtered.csv')
    y=unifi_df['label']
    X=unifi_df.drop(columns='label')
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, shuffle=shuffle)
    my_unifi_df = pd.concat([X_test, y_test], axis=1)

    my_all3_df = pd.concat([homenet_df,mobile_df,my_unifi_df],axis=0)
    my_all3_df.to_csv("preprocessed_arancino_datasets/my_all3.csv", index=False)

    X,y,df=preprocessing_base('preprocessed_arancino_datasets/my_all3.csv')

    mcc_all_map={
        'Logistic Regression Classic':get_mcc('',classic_models['LogisticRegression'],X,y),
        'XGB Classic':get_mcc('',classic_models['XGB'],X,y),
        'Random Forest Classic':get_mcc('',classic_models['RandomForest'],X,y),
        'LDA Classic':get_mcc('',classic_models['LDA'],X,y)
    }

    new_df_moving_average=window_input_moving_average(size,df)
    y = new_df_moving_average['label']
    X = new_df_moving_average.drop(columns='label')

    mcc_all_map.update({
        'Logistic Regression TS MA':get_mcc('',ts_ma_models['LogisticRegression'],X,y),
        'XGB TS MA':get_mcc('',ts_ma_models['XGB'],X,y),
        'Random Forest TS MA':get_mcc('',ts_ma_models['RandomForest'],X,y),
        'LDA TS MA':get_mcc('',ts_ma_models['LDA'],X,y)
    })

    new_df_difference = window_input_difference(size, df)
    y = new_df_difference['label']
    X = new_df_difference.drop(columns='label')

    mcc_all_map.update({
        'Logistic Regression TS Diff':get_mcc('',ts_diff_models['LogisticRegression'],X,y),
        'XGB TS Diff':get_mcc('',ts_diff_models['XGB'],X,y),
        'Random Forest TS Diff':get_mcc('',ts_diff_models['RandomForest'],X,y),
        'LDA TS Diff':get_mcc('',ts_diff_models['LDA'],X,y)
    })

    return mcc_all_map

```

```

def mcc_all3_plot(mcc_all_map, logistic_mcc_map, random_forest_mcc_map, xgb_mcc_map, lda_mcc_map, details):
    #Plot per comparare MCC sul test set di Uni con quello su my_all3
    logistic_mcc_all_map={}
    random_forest_mcc_all_map={}
    xgb_mcc_all_map={}
    lda_mcc_all_map={}

    #Split maps
    for key, value in mcc_all_map.items():
        if 'Logistic' in key:
            logistic_mcc_all_map[key] = value
        if 'Random' in key:
            random_forest_mcc_all_map[key] = value
        if 'XGB' in key:
            xgb_mcc_all_map[key] = value
        if 'LDA' in key:
            lda_mcc_all_map[key] = value

    x_values = list(logistic_mcc_all_map.keys())
    y_values=list(logistic_mcc_map.values())
    y_all_values = list(logistic_mcc_all_map.values())

    fig, ax = plt.subplots(2, 2, figsize=(18, 12))
    ax[0,0].plot(x_values, y_values, marker='o', linestyle='--',label='MCC unifi')
    ax[0,0].plot(x_values, y_all_values, marker='o', linestyle='--',label='MCC my_all3')
    ax[0,0].legend()

    x_values = list(xgb_mcc_map.keys())
    y_values=list(xgb_mcc_map.values())
    y_all_values = list(xgb_mcc_all_map.values())

    ax[0,1].plot(x_values, y_values, marker='o', linestyle='--',label='MCC unifi')
    ax[0,1].plot(x_values, y_all_values, marker='o', linestyle='--',label='MCC my_all3')
    ax[0,1].legend()

    x_values = list(random_forest_mcc_map.keys())
    y_values=list(random_forest_mcc_map.values())
    y_all_values = list(random_forest_mcc_all_map.values())

    ax[1,0].plot(x_values, y_values, marker='o', linestyle='--',label='MCC unifi')
    ax[1,0].plot(x_values, y_all_values, marker='o', linestyle='--',label='MCC my_all3')
    ax[1,0].legend()

    x_values = list(lda_mcc_map.keys())
    y_values=list(lda_mcc_map.values())
    y_all_values = list(lda_mcc_all_map.values())

    ax[1,1].plot(x_values, y_values, marker='o', linestyle='--',label='MCC unifi')
    ax[1,1].plot(x_values, y_all_values, marker='o', linestyle='--',label='MCC my_all3')
    ax[1,1].legend()

    fig.suptitle('MCC Comparison unifi-my_all3 '+details)
    fig.subplots_adjust(wspace=0.3)
    fig.subplots_adjust(hspace=0.3)
    plt.savefig('MCC Comparison Uni-All3 '+details)
    plt.show()

```

```

def main_win5_shufTrue():
    X,y,df=preprocessing_base('preprocessed_arancino_datasets/unifi_filtered.csv')

    shuffle=True
    size=5
    details='win5_shufTrue'

    classic_models,X_train,y_train,X_test,y_test=training_classic(shuffle,X,y,details)
    mcc_map,accuracy_map,speed_map=map_creation_classic(classic_models,X_train,y_train,X_test,y_test,details)

    ts_ma_models,X_train,y_train,X_test,y_test=training_ts_ma(shuffle,X,y,df,size,details)
    mcc_map,accuracy_map,speed_map=map_update_TS(ts_ma_models,'MA',X_train,y_train,X_test,y_test,mcc_map,accuracy_map,speed_map,details)

    ts_diff_models,X_train,y_train,X_test,y_test=training_ts_diff(shuffle,X,y,df,size,details)
    mcc_map,accuracy_map,speed_map=map_update_TS(ts_diff_models,'Diff',X_train,y_train,X_test,y_test,mcc_map,accuracy_map,speed_map,details)

    logistic_mcc_map,random_forest_mcc_map,xgb_mcc_map,lda_mcc_map=mcc_accuracy_progression(mcc_map,accuracy_map,details)
    mcc_all_map=mcc_all3(classic_models,ts_diff_models,ts_ma_models,size,shuffle)
    mcc_all3_plot(mcc_all_map,logistic_mcc_map,random_forest_mcc_map,xgb_mcc_map,lda_mcc_map,details)
    speed_progression(speed_map,details)

    return mcc_all_map

mcc_all_map_main=main_win5_shufTrue()

```

```

def main_win5_shufFalse():
    X,y,df=preprocessing_base('preprocessed_arancino_datasets/unifi_filtered.csv')

    shuffle=False
    size=5
    details='win5_shufFalse'

    classic_models,X_train,y_train,X_test,y_test=training_classic(shuffle,X,y,details)
    mcc_map,accuracy_map,speed_map=map_creation_classic(classic_models,X_train,y_train,X_test,y_test,details)

    ts_ma_models,X_train,y_train,X_test,y_test=training_ts_ma(shuffle,X,y,df,size,details)
    mcc_map,accuracy_map,speed_map=map_update_TS(ts_ma_models,'MA',X_train,y_train,X_test,y_test,mcc_map,accuracy_map,speed_map,details)

    ts_diff_models,X_train,y_train,X_test,y_test=training_ts_diff(shuffle,X,y,df,size,details)
    mcc_map,accuracy_map,speed_map=map_update_TS(ts_diff_models,'Diff',X_train,y_train,X_test,y_test,mcc_map,accuracy_map,speed_map,details)

    logistic_mcc_map,random_forest_mcc_map,xgb_mcc_map,lda_mcc_map=mcc_accuracy_progression(mcc_map,accuracy_map,details)
    mcc_all_map=mcc_all3(classic_models,ts_diff_models,ts_ma_models,size,shuffle)
    mcc_all3_plot(mcc_all_map,logistic_mcc_map,random_forest_mcc_map,xgb_mcc_map,lda_mcc_map,details)
    speed_progression(speed_map,details)

    return xgb_mcc_map,logistic_mcc_map,random_forest_mcc_map,lda_mcc_map

xgb_mcc_map_win5,logistic_mcc_map_win5,random_forest_mcc_map_win5,lda_mcc_map_win5=main_win5_shufFalse()

```

```

def main_win4_shufFalse():
    X,y,df=preprocessing_base('preprocessed_arancino_datasets/unifi_filtered.csv')

    shuffle=False
    size=4
    details='win4_shufFalse'

    classic_models,X_train,y_train,X_test,y_test=training_classic(shuffle,X,y,details)
    mcc_map,accuracy_map,speed_map=map_creation_classic(classic_models,X_train,y_train,X_test,y_test,details)

    ts_ma_models,X_train,y_train,X_test,y_test=training_ts_ma(shuffle,X,y,df,size,details)
    mcc_map,accuracy_map,speed_map=map_update_TS(ts_ma_models,'MA',X_train,y_train,X_test,y_test,mcc_map,accuracy_map,speed_map,details)

    ts_diff_models,X_train,y_train,X_test,y_test=training_ts_diff(shuffle,X,y,df,size,details)
    mcc_map,accuracy_map,speed_map=map_update_TS(ts_diff_models,'Diff',X_train,y_train,X_test,y_test,mcc_map,accuracy_map,speed_map,details)

    logistic_mcc_map,random_forest_mcc_map,xgb_mcc_map,lda_mcc_map=mcc_accuracy_progression(mcc_map,accuracy_map,details)
    mcc_all_map=mcc_all3(classic_models,ts_diff_models,ts_ma_models,size,shuffle)
    mcc_all3_plot(mcc_all_map,logistic_mcc_map,random_forest_mcc_map,xgb_mcc_map,lda_mcc_map,details)
    speed_progression(speed_map,details)

    return xgb_mcc_map,logistic_mcc_map,random_forest_mcc_map,lda_mcc_map

xgb_mcc_map_win4,logistic_mcc_map_win4,random_forest_mcc_map_win4,lda_mcc_map_win4=main_win4_shufFalse()

```

```

def main_win3_shufFalse():
    X,y,df=preprocessing_base('preprocessed_arancino_datasets/unifi_filtered.csv')

    shuffle=False
    size=3
    details='win3_shufFalse'

    classic_models,X_train,y_train,X_test,y_test=training_classic(shuffle,X,y,details)
    mcc_map,accuracy_map,speed_map=map_creation_classic(classic_models,X_train,y_train,X_test,y_test,details)

    ts_ma_models,X_train,y_train,X_test,y_test=training_ts_ma(shuffle,X,y,df,size,details)
    mcc_map,accuracy_map,speed_map=map_update_TS(ts_ma_models,'MA',X_train,y_train,X_test,y_test,mcc_map,accuracy_map,speed_map,details)

    ts_diff_models,X_train,y_train,X_test,y_test=training_ts_diff(shuffle,X,y,df,size,details)
    mcc_map,accuracy_map,speed_map=map_update_TS(ts_diff_models,'Diff',X_train,y_train,X_test,y_test,mcc_map,accuracy_map,speed_map,details)

    logistic_mcc_map,random_forest_mcc_map,xgb_mcc_map,lda_mcc_map=mcc_accuracy_progression(mcc_map,accuracy_map,details)
    mcc_all_map=mcc_all3(classic_models,ts_diff_models,ts_ma_models,size,shuffle)
    mcc_all3_plot(mcc_all_map,logistic_mcc_map,random_forest_mcc_map,xgb_mcc_map,lda_mcc_map,details)
    speed_progression(speed_map,details)

    return xgb_mcc_map,logistic_mcc_map,random_forest_mcc_map,lda_mcc_map

xgb_mcc_map_win3,logistic_mcc_map_win3,random_forest_mcc_map_win3,lda_mcc_map_win3=main_win3_shufFalse()

```

```

def main_win2_shufFalse():
    X,y,df=preprocessing_base('preprocessed_arancino_datasets/unifi_filtered.csv')

    shuffle=False
    size=2
    details='win2_shufFalse'

    classic_models,X_train,y_train,X_test,y_test=training_classic(shuffle,X,y,details)
    mcc_map,accuracy_map,speed_map=map_creation_classic(classic_models,X_train,y_train,X_test,y_test,details)

    ts_ma_models,X_train,y_train,X_test,y_test=training_ts_ma(shuffle,X,y,df,size,details)
    mcc_map,accuracy_map,speed_map=map_update_TS(ts_ma_models,'MA',X_train,y_train,X_test,y_test,mcc_map,accuracy_map,speed_map,details)

    ts_diff_models,X_train,y_train,X_test,y_test=training_ts_diff(shuffle,X,y,df,size,details)
    mcc_map,accuracy_map,speed_map=map_update_TS(ts_diff_models,'Diff',X_train,y_train,X_test,y_test,mcc_map,accuracy_map,speed_map,details)

    logistic_mcc_map,random_forest_mcc_map,xgb_mcc_map,lda_mcc_map=mcc_accuracy_progression(mcc_map,accuracy_map,details)
    mcc_all_map=mcc_all3(classic_models,ts_diff_models,ts_ma_models,size,shuffle)
    mcc_all3_plot(mcc_all_map,logistic_mcc_map,random_forest_mcc_map,xgb_mcc_map,lda_mcc_map,details)
    speed_progression(speed_map,details)

    return xgb_mcc_map,logistic_mcc_map,random_forest_mcc_map,lda_mcc_map

xgb_mcc_map_win2,logistic_mcc_map_win2,random_forest_mcc_map_win2,lda_mcc_map_win2=main_win2_shufFalse()

```

```

def delete_unreliable_features_classic(columns_to_drop,model_name,csv_name):
    #Elimina le feature 'inaffidabili' dai modelli con shuffle=False
    X,y,df=preprocessing_base('preprocessed_arancino_datasets/unifi_filtered.csv')

    X=X.drop(columns=columns_to_drop)
    df=df.drop(columns=columns_to_drop)

    preprocessor=StandardScaler()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, shuffle=False)

    if(model_name=='XGB Classic'):
        model=my_XGB_pipeline(preprocessor,X_train,y_train)
    elif(model_name=='Logistic Regression Classic'):
        model=my_LogisticRegression_pipeline(preprocessor,X_train,y_train)
    elif(model_name=='Random Forest Classic'):
        model=my_RandomForest_pipeline(preprocessor,X_train,y_train)
    elif(model_name=='LDA Classic'):
        model=my_LDA_pipeline(preprocessor,X_train,y_train)

    MCC=(matthews_corrcoef(y_test, model.predict(X_test))+1)/2 # MCC scalato a [0,1]

    with open('mcc_csv/mcc_win5_shuffFalse.csv', 'r') as csvfile:
        csvreader = csv.reader(csvfile)
        for row in csvreader:
            if row[0]==model_name:
                value = row[1]

    with open(csv_name, 'a', newline='') as file:
        fieldnames = ['csv_note','MCC Without Unreliable Features','MCC With Unreliable Features']
        writer = csv.DictWriter(file, fieldnames=fieldnames)
        if is_csv_empty(csv_name):
            writer.writeheader()
        writer.writerow({'csv_note': model_name,'MCC Without Unreliable Features': MCC,'MCC With Unreliable Features': value})

    return model,X,y

```

```

#Elimina Logistic Regression Unreliable Features
columns_to_drop=['meminfo.MemFree','iostat.nice','disk_io.read_time','redis_used_cpu_sys','cpu_times.idle']
model_name='Logistic Regression Classic'
csv_name='mcc_without_unreliable_features.csv'

X,y,df=preprocessing_base('preprocessed_arancino_datasets/my_all3.csv')

model,X,y=delete_unreliable_features_classic(columns_to_drop,model_name,csv_name)
MCC=(matthews_corrcoef(y, model.predict(X))+1)/2 # MCC scalato a [0,1]

mcc_unreliable_map={
    |   model_name:MCC
}

#Elimina XGB Unreliable Features
columns_to_drop=['net_io.bytes_sent','meminfo.Inactive','disk_io.read_bytes','cpu_times.iowait','vmstat.pgfreetime','temperature.temperature']
model_name='XGB Classic'

X,y,df=preprocessing_base('preprocessed_arancino_datasets/my_all3.csv')

model,X,y=delete_unreliable_features_classic(columns_to_drop,model_name,csv_name)
MCC=(matthews_corrcoef(y, model.predict(X))+1)/2 # MCC scalato a [0,1]

mcc_unreliable_map.update(
    |   {model_name:MCC}
)

#Elimina Random Forest Unreliable Features
columns_to_drop=['meminfo.MemAvailable','virtual.available','meminfo.Inactive','virtual.inactive']
model_name='Random Forest Classic'

X,y,df=preprocessing_base('preprocessed_arancino_datasets/my_all3.csv')

model,X,y=delete_unreliable_features_classic(columns_to_drop,model_name,csv_name)
MCC=(matthews_corrcoef(y, model.predict(X))+1)/2 # MCC scalato a [0,1]

mcc_unreliable_map.update(
    |   {model_name:MCC}
)

#Elimina LDA Unreliable Features
columns_to_drop=['cpu_times.idle','cpu_stats.ctx_switches','vmstat.pgreuse','cpu_times.nice','netinfo.lo.sent.pkts']
model_name='LDA Classic'

X,y,df=preprocessing_base('preprocessed_arancino_datasets/my_all3.csv')

model,X,y=delete_unreliable_features_classic(columns_to_drop,model_name,csv_name)
MCC=(matthews_corrcoef(y, model.predict(X))+1)/2 # MCC scalato a [0,1]

mcc_unreliable_map.update(
    |   {model_name:MCC}
)

mcc_plot_all3_unreliable(mcc_all_map_main,mcc_unreliable_map)

```

```

import itertools

def mcc_plot_all3_unreliable(mcc_all_map_main1, mcc_map_without_unreliable_features):
    #Plot per confrontare MCC dei modelli addestrati con e senza feature inaffidabili su my_all3
    x_values = list(itertools.islice(mcc_all_map_main1.keys(), 4))
    mcc_map_with_unreliable_features_values=list(itertools.islice(mcc_all_map_main1.values(), 4))
    mcc_map_without_unreliable_features_values=list(mcc_map_without_unreliable_features.values())

    plt.suptitle('MCC Comparison With/Without Unreliable Features on my_all3 ')
    plt.plot(x_values, mcc_map_with_unreliable_features_values, marker='o', linestyle='-',label='With Unreliable Features')
    plt.plot(x_values, mcc_map_without_unreliable_features_values, marker='o', linestyle='--',label='Without Unreliable Features')
    plt.legend()
    plt.savefig('MCC_Comparison_With_Without_Unreliable_Features_All3')

```

```
def mcc_progression_window(mcc_win5_map,mcc_win4_map,mcc_win3_map,mcc_win2_map,details):
    #Plot per confrontare progressione MCC dei modelli al variare della finestra temporale utilizzata
    x_values = ['win5','win4','win3','win2']
    mcc_values_ma=[mcc_win5_map[details+' TS MA'],mcc_win3_map[details+' TS MA'],mcc_win2_map[details+' TS MA']]
    mcc_values_diff=[mcc_win5_map[details+' TS Diff'],mcc_win4_map[details+' TS Diff'],mcc_win3_map[details+' TS Diff'],mcc_win2_map[details+' TS Diff']]

    if(details=='XGB'):
        details='XGBoost'

    plt.suptitle('MCC Progression by Window Size '+details)
    plt.plot(x_values, mcc_values_ma, marker='o', linestyle='-',label='TS MA')
    plt.plot(x_values, mcc_values_diff, marker='o', linestyle='-',label='TS Diff')
    plt.legend()
    plt.savefig('MCC_Progression_by_Window_Size.'+details)
    plt.clf()

mcc_progression_window(xgb_mcc_map_win5,xgb_mcc_map_win4,xgb_mcc_map_win3,xgb_mcc_map_win2,'XGB')
mcc_progression_window(logistic_mcc_map_win5,logistic_mcc_map_win4,logistic_mcc_map_win3,logistic_mcc_map_win2,'Logistic Regression')
mcc_progression_window(random_forest_mcc_map_win5,random_forest_mcc_map_win4,random_forest_mcc_map_win3,random_forest_mcc_map_win2,'Random Forest')
mcc_progression_window(lda_mcc_map_win5,lda_mcc_map_win4,lda_mcc_map_win3,lda_mcc_map_win2,'LDA')
```

BIBLIOGRAFIA

- [1] Apprendimento automatico. <https://it.wikipedia.org/wiki/ApprendimentoAutomatico>. (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-numpy-6a2f3a2a2a>. (Cited on page [16])
- [3] Media mobile. <https://it.wikipedia.org/wiki/MediaMobile>. (Cited on page [22])
- [4] Standardscaler. <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>. (Cited on page [21])
- [5] train_test_split. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. (Cited on page [24])
- [6] Mohamed Alloghani, Dhiya Al-Jumeily, Jamila Mustafina, Abir Husain, 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 [15])
- [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 [15])
- [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 [17])
- [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] Merlino G. Ceccarelli A. Puliafito A. Bondavalli A. Zoppi, T. Anomaly detectors for self-aware edge and iot devices. *in proceedings at the 23rd IEEE Conference on Software Quality, Reliability, Security (future event) Chang Mai,Thailand*, 2023. (Cited on pages [5] and [17])