Corso di Laurea in Informatica

TESI DI LAUREA

Rilevamento di Anomalie Satellitari con ROCKET: un Approccio Empirico su Dataset Reali

Relatore:
Vincenzo Lomonaco

Candidato:
Francesco Fiaschi

ANNO ACCADEMICO 2024/2025

Sommario

bgnbrtgnbrtgntr

Indice

Elenco delle figure

1 Introduzione

1.1 Rilevamento di Anomalie nei Satelliti

Negli anni si è fatto sempre più presente il bisogno di esplorare lo spazio puntando sempre più lontano, studiando e comprendendo le regole e le strutture dell'universo partendo dall'osservazione di quest'ultimo. Con l'avanzare della tecnologia abbiamo potuto avvicinarci allo spazio tramite i satelliti posizionati in orbita intorno alla Terra; questi sono circa 3500 attivi e si è fatta sempre più importante la richiesta di rilevare le anomalie di tali satelliti e comunicandole alla stazione di controllo, portando un aumento nello scambio di informazioni con un costo non irrilevante data la banda limitata. Prima dell'avvento dell'intelligenza artificiale la rilevazione delle anomalie era lasciata ad un'analisi statica delle componenti, controllo di qualità con regole e soglie fisse, questo ovviamente portava ad avere un'alta probabilità di ricevere tante segnalazioni di anomalie non rilevanti o date da cambiamenti nei riferimenti che creano un diverso ambiente vanificando i controlli statici. Con l'intelligenza artificiale siamo riusciti ad avere un rilevamento adattivo in base all'ambiente che portava a riscontrare un numero minore di false anomalie, ma anche questo approccio portava con se delle problematiche: l'algoritmo di intelligenza artificiale veniva allenato nel centro di controllo su dati reali e poi spedito sul satellite, ogni volta che bisognava modificare i parametri dell'algoritmo, bisognava ripetere tutto il processo dato che i satelliti, all'interno, hanno un processore poco prestante e quindi non sufficiente ad eseguire l'allenamento degli algoritmi, sprecando così tanta banda e molto tempo di trasmissione.

1.2 Contesto di Intelligenza Artificiale

Il machine learning, o apprendimento automatico, è un ramo dell'intelligenza artificiale che permette di imparare dai dati senza essere programmati esplicitamente ed essere eseguiti in maniera statica. Infatti, questi modelli possono migliorare le loro prestazioni identificando schemi nei dati, anche detti pattern, che poi utilizzano per fare previsioni o prendere decisioni in base all'utilizzo di cui abbiamo bisogno.

In relazione al nostro scopo, il machine learning è usato per rilevare anomalie, questa è una missione importante per riconoscere comportamenti imprevisti o anomali in diversi sistemi, come quelli satellitari, nel caso specifico. L'identificazione di anomalie nei satelliti è fondamentale per permettere una gestione rapida e puntuale per risolvere eventuali problemi; questo potrebbe ridurre i costi operativi e aumentare la resilienza del sistema.

Nel machine learning abbiamo tre categorie principali:

- Apprendimento supervisionato (supervised): i modelli di questo tipo sono addestrati su dati etichettati, cioè dove ad ogni esempio è associato il valore o la risposta desiderata, quelli che successivamente chiameremo ground truth;
- Apprendimento non supervisionato (unsupervised): al contrario dei modelli supervised questi apprendono da dati senza etichettatura, cercando di trovare delle strutture ricorrenti, un esempio sono i cluster[1];
- Apprendimento Semi-Supervisionato e rinforzato: questi modelli combinano parti delle due categorie precedenti e si concentrano su interazioni dinamiche con l'ambiente.

[1]I cluster sono insiemi di dati che hanno almeno un elemento comune

1.3 Panoramica su Rilevamento di Anomalie

Quando parliamo di anomalie intendiamo un parametro o un osservazione che si distacca in modo marcato dalla normalità; di conseguenza il rilevamento delle anomalie è l'identificazione delle stesse che possono differire in tre tipologie:

- Point Anomalies o anomalie puntuali: sono singoli punti che si discostano dal resto del dataset;
- Contextual Anomalies o anomalie contestuali: sono dati che in un contesto specifico risultano anomali, un esempio può essere un intervallo temporale;
- Collective Anomalies: sono gruppi di dati o osservazioni che prese tutte insieme rappresentano un comportamento anomalo.

Oltre alle difficoltà nell'identificazione delle anomalie data dalle diverse situazioni e i diversi tipi di anomalie che possono presentarsi, sono presenti anche problematiche per quanto riguarda i dati. Quest'ultimi, che vengono raccolti durante le missioni spaziali, non sono uniformi, dato che le anomalie rappresentano una minoranza nel totale. Un altra problematica è legata all'etichettatura di questi dati che potrebbe essere costosa da fare e in alcuni casi può non essere presente (come nel dataset NASA che vedremo). L'ultima sfida è adattarsi ai dati in continuo cambiamento nell'arco della missione.

1.4 NASA ed ESA

Negli anni sia la NASA (National Aeronautics and Space Administration) che l'ESA (Agenzia Spaziale Europea) hanno pubblicato molteplici dataset contenenti dati reali di missioni e relativi benchmark sull'esecuzione di algoritmi di intelligenza artificiale per trovare il giusto compromesso tra efficacia e efficienza di un algoritmo, focalizzandosi maggiormente sul trovare l'algoritmo migliore per il rilevamento delle anomalie.

I dataset sono stati resi pubblici per incentivare la comunità a contribuire alla ricerca di nuove tecniche di monitoraggio e rilevamento delle anomalie, potendo usare dati reali.

1.5 Obiettivo

Partendo dai dati proposti nel più semplice dataset OPS_SAT e successivamente comprendendo il benchmark di NASA, vogliamo analizzare quali test sono stati effettuati per capire come poter fare un ulteriore passo in avanti nel rilevamento continuo di anomalie, cercando di rendere più efficiente un algoritmo già efficace, in modo da poterlo allenare direttamente sul satellite, limitando così lo scambio di comunicazioni e restringendo ancora di più il rilevamento di false anomalie.

Tutto questo processo è dedito a trovare un algoritmo che abbia un giusto compromesso tra efficienza ed efficacia, così che possa rilevare in modo corretto la maggioranza delle anomalie senza però avere un costo molto alto in termini di consumo di banda e di risorse del satellite.

In primo luogo procederemo a scendere più nello specifico aggiungendo o aggiustando i parametri di alcuni algoritmi tra cui XGBOD. Per validare l'implementazione fatta portiamo a sostegno risultati ottenuti effettuando test e confronti tra un'implementazione standard e la nostra proposta mettendo a paragone le metriche di valutazione.

In secondo luogo vogliamo proporre un'implementazione di due algoritmi, ROCKET e ROCKAD, sui quali, recentemente, sono stati pubblicati articoli dove si afferma la loro bassa complessità e le ottime performance sull'analisi delle timeseries. Per questi motivi abbiamo adattato questi algoritmi per il rilevamento delle anomalie su timeseries, applicando preprocessing ai dati estratti dai dataset. Partendo dal dataset OPS_SAT tramite vari test abbiamo estrapolato vari risultati di test e metriche corrispondenti, per poi spostarci su NASA, a cui accediamo tramite SpaceAI [1], al quale abbiamo collaborato per integrare i nostri contributi. Anche su questo dataset sono state riportate metriche di valutazione, allo scopo di poter confrontare gli algoritmi in termini di efficacia ed efficienza. Tutti questi

test portano ad un confronto e una verifica ulteriore dell'effettiva possibilità di applicazione di questo nuovo algoritmo, ancora non usato nel contesto della rilevazione delle anomalie; proponendo riflessioni e ragionamenti avvalorati da dati riscontrati nei test effettuati.

2 Metodologie Applicate

In questo capitolo viene esposto il flusso che abbiamo seguito e le scelte effettuate per la realizzazione quest'analisi empirica.

2.1 Esplorazione dei Dataset

Il primo approccio avviene tramite un approfondita esplorazione dei dataset reali disponibili, concentrandosi maggiormente su OPS_SAT e NASA, entrambi con caratteristiche e peculiarità uniche, di interesse per il contesto satellitare.

Questi dataset sono stati presi in esempio, non solo per avere un confronto diretto con il repository di SpaceAI[1], ma anche per le sfide nell'ambito della rilevazione di anomalie, come la scarsa quantità di anomalie, i problemi relativi all'etichettatura e i dati che a volte possono mancare.

OPS_SAT, invece, permette un primo approccio molto più tranquillo, portando una struttura semplice, in modo che sia facile da comprendere e da utilizzare per lo sviluppo di nuovi algoritmi. NASA, d'altro canto, è il dataset che rappresenta di più la realtà, portando un dataset più complesso, che sarà utilizzato come banco di prova più realistico.

2.2 La soluzione: ROCKET

ROCKET (Random Convolutional Kernel Transform), ancora non utilizzato per il rilevamento delle anomalie, è stato selezionato come nuovo algoritmo, per la sua grande capacità di estrarre caratteristiche importanti dalle serie temporali e per l'efficienza di utilizzo. Tutto questo è reso possibile grazie anche ai kernel casuali

che non necessitano di grande capacità di calcolo, rispetto ad altri algoritmi più complessi.

Ciò che garantisce la robustezza di ROCKET sono i kernel casuali e le tecniche di pooling che vi si possono applicare, portando così a poter eliminare la necessità di ottimizzare gli iperparametri. ROCKET viene descritto come efficiente e scalabile anche su grandi dataset, oltre ad un fattore di adattabilità ai dataset con caratteristiche variabili, molto elevato.

L'obbiettivo principale è utilizzare ROCKET per il rilevamento delle anomalie direttamente a bordo dei satelliti, riducendo la necessità di scambiare informazioni con la stazione centrale e minimizzando i falsi positivi.

2.2.1 ROCKAD

Dopo l'uscita di ROCKET e valutata la sua efficacia è stato introdotto ROCKAD (Random Convolutional Kernel Transform Anomaly Detector), questo è un estensione di ROCKET progettata appositamente per il rilevamento delle anomalie.

Secondo gli sviluppatori, ROCKAD permette di ottenere un rilevamento di anomalie più preciso e mirato, sfruttando le caratteristiche estratte da ROCKET. ROCKAD trova maggiore spazio in contesti in cui l'etichettatura dei dati è limitata o assente (come vedremo per NASA), consentendo una classificazione basata su punteggi di anomalia.

2.3 Problematiche

Nonostante le caratteristiche ed i vantaggi di ROCKET e ROCKAD, rimangono sempre alcune problematiche che possono essere riscontrate. Nel caso di ROCKET abbiamo la ridondanza di caratteristiche che porta ad un aumento di probabilità di avere un overfitting su dataset di piccole dimensioni. Per ROCKAD, dato il modello aggiuntivo KNN, richiede più potenza di calcolo per gestirlo, ma anche la trasformazione delle caratteristiche che avviene internamente richiede più

risorse. Per finire entrambi gli algoritmi rendono complessa l'analisi dettagliata dei risultati data la limitata accessibilità delle caratteristiche estratte.

2.4 Flusso di Lavoro

Il primo approccio, come abbiamo spiegato precedentemente, avviene con il dataset OPS_SAT, dal quale, dopo aver validato tutti i risultati ottenuti dai modelli del relativo paper, abbiamo cercato di migliorare XGBOD, uno dei migliori modelli per capacità di rilevare le anomalie e velocità di esecuzione. Abbiamo effettuato test per trovare i miglior iperparametri basandoci sulle metriche e sul tempo di esecuzione.

Su OPS_SAT siamo poi passati all'implementazione di ROCKET, apportando modifiche ai dati estratti per renderli compatibili ed utilizzabili con ROCKET. Data la natura di ROCKET, che utilizza le serie temporali, siamo riusciti ad utilizzarlo come estrattore di caratteristiche per poi utilizzarle con un classificatore per ottenere i risultati voluti. Come classificatori abbiamo utilizzato vari modelli, sia unsupervised che supervised, tra cui l'uso di una threshold in percentuale, utilizzando il 95° percentile dei punteggi di anomalia; ulteriori prove sono state effettuate con KNN, LogisticRegression, Ridge e RidgeClassifierCV che troviamo utilizzati anche nel paper di ROCKET[2]. Lo stesso processo è stato effettuato con ROCKAD utilizzando però come classificatore il modello proposto nel paper di riferimento[3], NearestNeighborOCC.

Partendo da OPS_SAT siamo arrivati a NASA per ottenere una conferma dell'efficienza ed efficacia degli algoritmi. In questo dataset al contrario di OPS_SAT l'etichettatura dei dati di training non è presente potendo così utilizzare solo modelli di classificazione unsupervised. Anche per questo dataset c'è stata la necessità di rendere i dati compatibili per l'utilizzo.

Tutti i risultati ottenuti sono stati proposti in tabelle ordinate e analizzati per trarne le conclusioni, in rifermento soprattutto a ROCKET e ROCKAD i quali sono stati utilizzati per la prima volta come metodi per il rilevamento delle anomalie, valutandone così la vera potenza e efficienza.

3 Dati di Riferimento Disponibili

Procediamo ora a descrivere i dataset contenenti dati di telemetrie registrati negli anni e resi pubblici in modo da poter sperimentare e migliorare gli algoritmi già presenti.

3.1 Dataset ESA

Il dataset di riferimento più importante per la rilevazione delle anomalie è quello fornito dall'ESA (European Space Agency) che conta dati di tre missioni. Di queste solo i dati di due vengono utilizzati per creare il benchmark, per le caratteristiche dell'insieme di dati; infatti in *"Mission 3"* abbiamo: poche anomalie e per lo più banali, un gran numero di buchi di comunicazione e segmenti non validi.

Nell'articolo in questione i dati satellitari grezzi di *"Mission 1"* e *"Mission 2"* vengono preprocessati per renderli uniformi e quindi utilizzabili con la maggior parte degli algoritmi.

In questa fase utilizziamo lo strumento OXI[4] per l'etichettatura collaborativa dei dati da cui si è potuto estrarre delle telemetrie che rappresentano periodi di funzionamento nominale e anomalo. Tutti i dati sono stati sottoposti ad una doppia fase di etichettatura e controllo.

I canali sono divisi tra "target" e "non target" dove quest'ultimi sono usati per gli algoritmi solo come informazioni addizionali. Un canale target, quello usato per rilevare le anomalie, è diviso in gruppi di dati con caratteristiche simili così da rendere più facile per l'algoritmo processarli ed interpretarli e, nel caso, allenarlo solo sul quel gruppo specifico.

	Mission 1	**Mission 2**
Channels	76	100
Target/Non target	58/18	47/53
Telecommands	698	123
Annotate Events	200	644
Anomalies	118	31
Rare Nominal Events	78	613
Univariate/Multivariate	32/164	9/635

Tabella 3.1: Configurazione Dataset ESA

Possiamo osservare dalla Tabella 3.1 che la densità di anomalie in termini di punti di dati annotati, varia tra $0,57\%$ per la *"Mission 2"* e $1,80\%$ per la *"Mission 1"* questo va a confermare un'impronta più realistica rispetto ai dataset meno recenti che avevano una densità di anomalie estremamente più alta ed irrealistica.

3.2 Dataset NASA

Il dataset NASA riporta i dati provenienti da missioni reali effettuate negli anni, questi coprono più aspetti relativi al contesto spaziale con il conseguente aumento di complessità.

Il dataset contiene serie temporali con numerosi parametri di telemetria, tra cui eventi normali e anomali. Le anomalie sono rare, infatti possiamo riscontrare canali che non ne possiedono nemmeno una, rappresentando una sfida significativa, insieme anche al fatto che non tutti i dati sono etichettati.

Elenchiamo alcune particolarità e soprattutto sfide:

- Rarità: le anomalie presenti nel dataset sono in quantità ridotta, rendendo l'addestramento dei modelli più difficile;
- Etichettatura: le etichettature dei dati sono incomplete, complicando ulteriormente l'identificazione;
- Rumorosità: i dati presenti, a causa anche della provenienza da diverse missioni, contengono lacune e sono spesso rumorosi a causa di fattori esterni;
- Variabilità: i parametri variano molto;

- Veridicità: i dati rispecchiano la realtà, portando così a poter testare i modelli in un contesto simile ai dati reali.

Utilizziamo questo dataset perché permette di utilizzare dati realistici, valutando in maniera più accurata gli algoritmi proposti. Inoltre, utilizzare un dataset con molteplici sperimentazioni e test effettuati, permette di avere un confronto diretto dei modelli, potendo anche validare i risultati ottenuti, aumentando la credibilità e avvalorando le analisi proposte da altri o la possibilità che altri validino le nostre.

3.3 Motivazioni

Il dataset ESA precedentemente descritto porta alla risoluzione di vari problemi noti nell'ambito della rilevazione di anomalie. Il primo problema è la loro struttura, come si evidenzia nei dataset *NASA Soil Moisture Active Passive* (SMAP) e *Mars Science Laboratory* (MSL), i quali offrono brevi frammenti di segnali e comandi correlati da, rispettivamente, 55 e 27 parametri di telemetria, con un totale di 105 anomalie annotate; infatti abbiamo una densità di anomalie irrealistica, alto numero di anomalie banali, verità di base etichettate in maniera errate ed una mancanza di correlazione tra comandi e canali. Come conseguenza di questo è stato deciso che questi dataset non andrebbero usati per il benchmarking del rilevamento delle anomalie. Il secondo problema invece rappresenta la mancanza di annotazione di eventi anomali, alcuni esempi sono gli insiemi di dati di *Mars Express6* e *NASA WebTCAD7*.

Il dataset ESA risolve i problemi elencati, ma nasce come dataset di missioni su larga scala, le quali sono molto complesse e stabili, portando con sé problemi non più relativi alla distribuzione delle anomalie, ma improntati verso difficoltà di esecuzione e potenza di calcolo.

Il dataset che introdurremo successivamente è concettualmente diverso, infatti affronta una missione ESA OPS_SAT molto semplificata, al fine di rendere più accessibile l'uso, oltre ad essere di dimensione considerevolmente minore.

Questo processo di apertura è reso anche tramite la sostituzione giornaliera dell'intero sistema software fino al sistema operativo del satellite, così da consentire alle persone che sperimentano di caricare il proprio software a bordo e a terra. Per implementare questa funzionalità venne usato FDIR[5].

Le telemetrie grezze sono caratterizzate da molte lacune nei dati ed altre imprecisioni, queste vengono curate da ingegneri ed esperti nei modelli di apprendimento automatico per rendere l'insieme di dati fruibile alla creazione di tecniche di rilevamento delle anomalie basate su di essi. Una di queste modifiche è la seguente:

> Selezione e annotazione di verità di base (ground_truth) di 2123 brevi frammenti di telemetria a canale singolo, ossia serie temporali univariate[6], registrate in 9 canali di telemetria.

Infine questo dataset, con il relativo benchmark e tutti i dati disponibili al suo interno, è stato messo a disposizione per aiutare la comunità nella ricerca di nuovi approcci per la rilevazione delle anomalie, confrontandosi tra di loro in modo imparziale ed equo, così da affrontare anche il problema della riproducibilità nell'ambiente dell'apprendimento automatico (dato che i dataset sono uniformati, l'ambiente di esecuzione potrebbe essere diverso e sono state utilizzate varie metriche).

Tutto ciò che è stato sviluppato con l'aiuto del dataset verrà convalidato con il lancio alla fine del 2025 del satellite successore OPS_SAT VOLT (dopo aver distribuito i vari modelli a bordo del satellite).

3.4 OP-SAT Dataset

Nel dataset OPS_SAT sono contenuti i dati delle telemetrie del satellite OPS_SAT dell'ESA (Figura 3.1). Questo satellite di tipo CubeSat aveva dimensione 3 unità (3U dove 1U=10cm^3); esso ormai non è più in orbita, era stato lanciato a Dicembre del 2019 con lo scopo di dimostrare l'elaborazione dei dati in orbita e di generare dati utili come immagini satellitari e telemetrie.

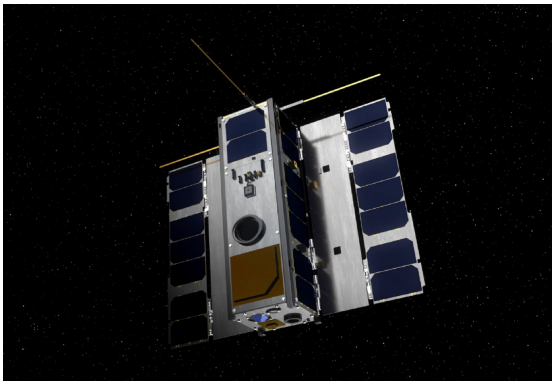

Figura 3.1: Satellite OPS_SAT. Fonte: European Space Agency

Come nel dataset precedente anche qui i dati hanno bisogno di essere preprocessati per renderli fruibili alla maggior parte degli algoritmi (OXI[4]). In questo caso sono state progettate 18 caratteristiche per l'attività di rilevamento delle anomalie, ossia sono stati estratti tratti significativi dai dati. Questo processo è chiamato Feature Extraction e serve per ridurre la complessità dei dati in ingresso, rendendoli più significativi.

Tutti i segmenti ricavati rappresentano le sfide che dobbiamo affrontare con i dati delle telemetrie, ognuno di questi è composto da:

- ⟨`timestamp`⟩: rappresenta il marcatore temporale nel momento della registrazione;
- ⟨`channel`⟩: è il nome del canale;
- ⟨`value`⟩: è il valore del segnale acquisito;
- ⟨`label`⟩: rappresentano le annotazioni certe, quelle annotate manualmente;
- ⟨`segment`⟩: rappresenta il numero consecutivo del segmento;
- ⟨`sampling`⟩: è il tasso di campionamento;
- ⟨`train`⟩: indica se il segmento appartiene al training set.

Il dataset è stato diviso in una parte di allenamento, Training Set (T), ed una di test, Test Set (Ψ). Questi rappresentano rispettivamente i dati usati per

l'allenamento del modello e i dati usati per fare le valutazioni delle performance dell'algoritmo.

Classi	**Training Set** (T)	**Test Set** (Ψ)	**Total**
Nominali	1273	416	1689
Anomalie	321	529	434

Tabella 3.2: Composizione Dataset OPS_SAT

Le classi mostrate in Tabella 3.2, nominali e anomalie, rappresentano rispettivamente segmenti che rispettano valori normali o attesi e segmenti invece che superano questi valori o discordano dai valori attesi. Partendo dai benchmark effettuati su questi due dataset, analizzeremo le metriche e le prestazioni utilizzando in modo più specifico il secondo dataset (OPS_SAT), che contiene dati più rilevanti per il nostro obiettivo.

4 Misure di Valutazione

Analizzeremo ora le metriche che ci permettono di valutare le prestazioni di un algoritmo allenato sui dati del training set e utilizzato come classificatore per il rilevamento delle anomalie quindi con i relativi pesi calcolati.

Utilizziamo le seguenti metriche, dalle quali trarremo le conclusioni sulle capacità di ciascun algoritmo:

- **Accuratezza**: rappresenta la percentuale di previsioni corrette (TP - True Positive) su tutti i casi possibili, tiene in considerazione anche i veri negativi (TN). Questa misura ci indica quanto ci stiamo avvicinando ai dati di allenamento.

  Formula:

  $$\frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

- **Precisione**: rappresenta la percentuale di anomalie vere rilevate in confronto a tutte le anomalie segnalate, quindi, la percentuale di veri positivi.

  Formula:

  $$\frac{TP}{TP + FP} \tag{4.2}$$

  Nel nostro caso questa metrica è particolarmente importante, dato che un valore troppo basso significherebbe un'alta probabilità di avere falsi positivi, portando quindi uno spreco di banda ed energia per mandare i messaggi ed un allarme che richiede un intervento non necessario.

- **Richiamo**: rappresenta la capacità di rilevare tutte le anomalie vere tenendo quindi in considerazione anche delle anomalie non rilevate (FN). Questa misura è anche detta sensibilità.

Formula:

$$\frac{TP}{TP + FN} \tag{4.3}$$

Nel contesto dei satelliti questa metrica è cruciale dato che, con un valore basso avremo un grande numero di falsi negativi (FN), che senza intervento potrebbero portare a conseguenze molto gravi.

- ***F*$_1$ score**: rappresenta una media armonica tra precisione e recupero dove il massimo si ottiene con il valore uno e il minimo a zero.

  Formula:

  $$F_1 = \frac{2 * TP}{2 * TP + FP + FN} \tag{4.4}$$

  Questa metrica nel nostro caso è importante, infatti cerchiamo un buon compromesso tra precisione e richiamo, per non avere un alto numero, né di falsi negativi, né di falsi positivi. Questo porta ad un equilibrio tra precisione e capacità di rilevazione.

- **Coefficiente di correlazione di Matthews (MCC)**[7]: rappresenta una misura della qualità del modello con dati molto variabili.

  Formula:

  $$\frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + TP) \cdot (TN + FN)}} \tag{4.5}$$

  Questa misura è particolarmente indicata per il rilevamento delle anomalie, dato che concede la stessa importanza a veri positivi, falsi positivi, veri negativi e falsi negativi.

- **L'area sottesa alla curva ROC (AUC$_{\mathbf{ROC}}$**[8][9]): rappresenta il rapporto tra il tasso di veri positivi e il tasso di falsi positivi (Figura: 4.1). Questa permette di osservare come varia il richiamo in funzione della metrica di precisione. Può anche essere usato per scegliere il modello migliore tra due, guardando semplicemente l'area sottesa al grafico: quella con l'area più grande è generalmente quello migliore.

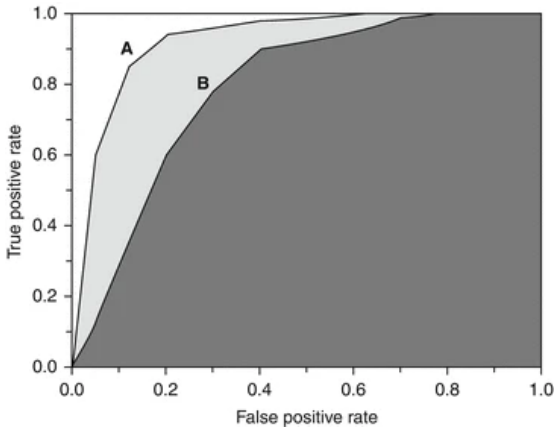

Figura 4.1: Curva ROC

- **Area sottesa alla curva Precisione-Richiamo (AUC_PR)**: rappresenta un semplice modo per sintetizzare le prestazioni generali di un modello, più alto è il valore più alto sarà il numero di predizioni. Non vengono considerati i veri positivi nella curva precisione-richiamo.

In queste formule per il calcolo delle metrica abbiamo usato: TP per rappresentare i veri positivi, ossia i segmenti delle telemetrie correttamente identificati come anomalie; TN per i veri negativi, cioè segmenti correttamente identificati come nominali ossia regolari; FP per i falsi positivi, ossia i segmenti erratamente classificati anomalie e FN per i falsi negativi che rappresentano i segmenti erratamente classificati come non anomalie. Tutte le metriche descritte possono assumere valori tra $[0, 1]$, tranne MCC che assume valori tra $[-1, 1]$. Tutte le metriche però più si avvicinano ad uno e più il modello testato risulta migliore, rispetto ad uno con valori inferiori delle stesse.

Le metriche appena viste ci serviranno successivamente per valutare le capacità degli algoritmi avendo un metro di paragone unico per tutti i modelli. Tra gli obbiettivi resta comunque cercare di non sovraccaricare il processore del satellite, mantenendo quindi la complessità del modello abbastanza bassa. Nell'eventualità di una piccola perdita di prestazioni, ma con un grande risparmio in complessità del modello, viene ovviamente premiata l'efficienza.

5 Validazione Paper OPS_SAT

In questa sezione vogliamo verificare le prestazioni di tutti gli algoritmi presenti nel paper relativo ad OPS_SAT, questo per avere un confronto con le metriche risultanti degli algoritmi che vedremo successivamente.

I dati che utilizziamo sono estratti tramite le funzioni rese disponibili dal paper, partendo quindi già con i dati elaborati e divisi in `X_train, X_test, y_train, y_test`. Tramite la funzione `StandardScaler()` rendiamo ancora più compatibili i dati, ottenendo le metriche desiderate, riportate nella tabella sottostante.

Modello	Accuracy	Precision	Recall	F1	MCC	AUC-PR	AUC-ROC	N-Score
Supervised								
LinearSVC	0.926	0.911	0.726	0.808	0.771	0.949	0.976	0.867
LogisticRegression	0.924	0.92	0.708	0.8	0.764	0.949	0.976	0.867
FCNN	0.96	0.926	0.885	0.905	0.88	0.963	0.982	0.903
AdaBoost	0.934	0.89	0.788	0.836	0.797	0.923	0.962	0.841
RF+ICCS	0.964	0.98	0.85	0.91	0.891	0.949	0.976	0.867
Linear+L2	0.902	0.969	0.558	0.708	0.69	0.889	0.95	0.814
XGBOD	0.968	0.953	0.894	0.922	0.903	0.969	0.99	0.912
Unsupervised								
MO-GAAL	0.896	0.939	0.549	0.693	0.669	0.771	0.849	0.699
AnoGAN	0.594	0.296	0.655	0.408	0.19	0.403	0.651	0.239
SO-GAAL	0.89	0.937	0.522	0.67	0.649	0.858	0.919	0.761
IForest	0.701	0.297	0.292	0.295	0.105	0.347	0.635	0.301
KNN	0.849	0.78	0.407	0.535	0.489	0.658	0.852	0.593
OCSVM	0.837	0.721	0.389	0.506	0.447	0.659	0.788	0.655
ABOD	0.845	0.782	0.381	0.512	0.472	0.644	0.843	0.584
INNE	0.83	0.689	0.372	0.483	0.418	0.624	0.801	0.646
ALAD	0.819	0.667	0.301	0.415	0.361	0.537	0.7	0.451
LMDD	0.822	1.0	0.168	0.288	0.37	0.624	0.765	0.663
SOD	0.826	0.611	0.513	0.558	0.453	0.621	0.797	0.549
COF	0.834	0.667	0.442	0.532	0.449	0.603	0.774	0.593
LODA	0.83	0.689	0.372	0.483	0.418	0.549	0.692	0.522
LUNAR	0.819	0.743	0.23	0.351	0.313	0.541	0.797	0.46
CBLOF	0.802	0.569	0.292	0.386	0.304	0.45	0.574	0.372
DIF	0.788	1.0	0.009	0.018	0.084	0.494	0.805	0.522
VAE	0.794	0.532	0.292	0.377	0.283	0.446	0.687	0.513
GMM	0.783	0.482	0.239	0.32	0.225	0.426	0.713	0.389
DeepSVDD	0.788	0.509	0.239	0.325	0.241	0.344	0.55	0.336
PCA	0.779	0.464	0.23	0.308	0.21	0.373	0.612	0.363
COPOD	0.767	0.4	0.177	0.245	0.147	0.328	0.627	0.257
SOS	0.758	0.364	0.177	0.238	0.125	0.308	0.524	0.274
ECOD	0.767	0.396	0.168	0.236	0.14	0.34	0.637	0.345

Tabella 5.1: risultati esecuzione di tutti gli algoritmi di OPS_SAT

6 Valutazione Modelli

In questo capitolo andremo ad analizzare gli algoritmi di nostro interesse, ossia XGBOD, ROCKET e ROCKAD. Questi algoritmi lavorano con le timeseries, le quali sono una sequenza di dati registrati ad intervalli di tempo consecutivi. Ogni punto della serie è associato ad un timestamp il quale indica il momento in cui è stato registrato.

6.1 XGBOD

XGBOD (eXtreme Gradient Boosting for Outlier Detection) è una struttura composta da tre fasi:

1. Generazione di nuove rappresentazioni di dati: vengono applicati vari metodi di rilevamento di anomalie non supervisionati ai dati originali, per ottenere punteggi di anomalie, questi rappresentano la nuova vista dei dati;

2. Seleziona i punteggi rilevanti: i punteggi ottenuti nella fase precedente vengono filtrati per usare sollo quelli utili, quest'ultimi sono combinati con le caratteristiche iniziali creando un nuovo spazio delle caratteristiche arricchito;

3. Addestramento del modello XGBoost: viene addestrato il modello XGBoost su questo nuovo spazio delle caratteristiche e le previsioni che otteniamo determinano se ogni dato è un'anomalia o no.

Utilizziamo XGBOD invece che XGBoost direttamente perché quest'ultimo, essendo un modello supervisionato, ha bisogno di dati etichettati e soprattutto con anomalie rare, non facili da etichettare.

XGBOD aggiunge una parte di preprocessing, aumenta le informazioni del set di dati con punteggi di anomalie ed utilizza metodi di rilevamento non supervisionato come Isolation Forest, Local Outlier Factor, ecc..

6.1.1 XGBoost

Il modello XGBoost di tipo supervisionato, si sviluppa con un processo iterativo di addestramento di alberi decisionali deboli (alberi decisionali poco profondi e quindi poco accurati), questi vengono combinati tra di loro portando un miglioramento progressivo delle prestazioni del modello.

XGBoost è composto da pochi passi ma ripetuti iterativamente: come primo passo vengono calcolati i residui, la differenza tra le previsioni iniziali ed i valori reali; questi sono i valori che vogliamo ridurre. Con questi valori il modello addestra un insieme di alberi decisionali deboli, dove ognuno cerca di correggere questi valori migliorando le previsioni del modello precedente. Tutti gli alberi vengono aggiunti al modello complessivo di XGBoost, che aggiorna le sue previsioni combinando tutti gli alberi precedentemente costruiti.

Per regolare tutto questo processo, sono applicate internamente tecniche di limitazione e regolazione per evitare un overfitting del modello. All'interno di XGBoost è presente anche una metrica chiamata *tasso di apprendimento*, che permette di decidere quanto un albero incide sul risultato finale, minimizzando così gli errori di percorso.

6.1.2 Risultati ottenuti

Qui sono elencati i risultati ottenuti effettuando varie prove con parametri diversi per ottimizzare XGBOD ed ottenere il miglior compromesso tra efficienza ed efficacia.

Modello	Accuracy	Precision	Recall	F1	MCC	AUC-PR	AUC-ROC	Nscore
M+P	0.97	0.945	0.912	0.928	0.909	0.973	0.992	0.92
Early	0.97	0.971	0.885	0.926	0.909	0.969	0.99	0.912
+M	0.968	0.944	0.903	0.923	0.903	0.974	0.91	0.92
P	0.964	0.943	0.885	0.913	0.891	0.972	0.991	0.912
Senza P	0.962	0.935	0.885	0.909	0.886	0.977	0.992	0.912
Grid	0.947	0.989	0.761	0.86	0.839	0.898	0.945	0.969

Tabella 6.1: Prove esecuzione di XGBOD

LEGENDA:

- M+P: significa più modelli e parametri;
- Grid: gridsearch con modelli;
- EarlyStop: viene utilizzato un meccanismo di EarlyStop che ferma l'esecuzione dell'algoritmo quando gli iperparametri non migliorano più per un numero definito di cicli.

Dalla Tabella 6.1 possiamo vedere che il miglior risultato è quello che utilizza più modelli per l'addestramento ed i parametri modificati al fine di efficientare l'esecuzione. Oltre ad avere degli ottimi risultati, l'esecuzione rimane praticamente istantanea sul nostro dataset di esempio OPS_SAT.

6.2 ROCKET

ROCKET è un algoritmo convoluzionale, questi sono anche detti reti neurali convoluzionali (CNN). Questi algoritmi lavorano su dati con una struttura a griglia come immagini e serie temporali e sono progettati per riconoscere pattern all'interno dei dati tramite operazioni di convoluzione. Vengono utilizzati i kernel[1], che operando sui dati in input estraggono caratteristiche locali dei dati (features); in questa fase possono essere applicate tecniche di centratura, ovvero aggiungere dei bordi attorno all'input, così da mantenere le dimensioni dopo aver applicato il kernel, questo si chiama padding.

[1]Matrice di pesi usata per eseguire operazioni di filtraggio per estrarre caratteristiche specifiche, opera tramite moltiplicazioni

Ci sono tecniche applicabili a ROCKET come il pooling, che ha lo scopo di ridurre la dimensione e quindi di rendere l'algoritmo meno sensibile alle traslazioni. In merito a questo vediamo le due tecniche più utilizzate:

- Max Pooling: prende solo il valore massimo in una finestra specifica riducendo la dimensione dell'input;
- Avarege Pooling: riduce la dimensione prendendo la media dei valori in una finestra.

In conclusione abbiamo il passaggio per gli strati: i dati vengono appiattiti (flattering) fatti passare attraverso uno o più strati completamente connessi per la classificazione o la regressione, in modo da ottenere il risultato desiderato.

ROCKET in particolare utilizza i kernel convoluzionali casuali, generandone un gran numero con parametri scelti casualmente, come lunghezza del kernel e pesi. Questi kernel filtrano i dati delle serie temporali producendo una serie di features. Utilizzando tecniche di pooling viste in precedenza, otteniamo statistiche riassuntive da queste features. La procedura viene ripetuta per tutti i kernel, portando ad un enorme quantità di caratteristiche e quindi una maggiore possibilità di estrarre tutti i pattern comuni.

Le caratteristiche estratte vengono poi date in input ad algoritmi di classificazione, tipicamente una regressione logistica data la sua scalabilità e velocità su grandi dataset. L'algoritmo viene addestrato su queste caratteristiche per effettuare la classificazione delle serie temporali.

Per rendere più chiara la logica di funzionamento prendiamo in considerazione la Figura 6.1, presa dal paper di ROCKAD[3]. Come possiamo osservare, in ingresso ROCKET prende la timeseries di una specifica features T, e tramite moltiplicazioni tra matrici, chiamate filtri (kernel), otteniamo un array di caratteristiche di $2K$ features. Questo processo viene ripetuto per ogni features del dataset, portando ad avere un numero di caratteristiche pari a 10.000 volte il numero di features iniziali (nel caso di kernel di default).

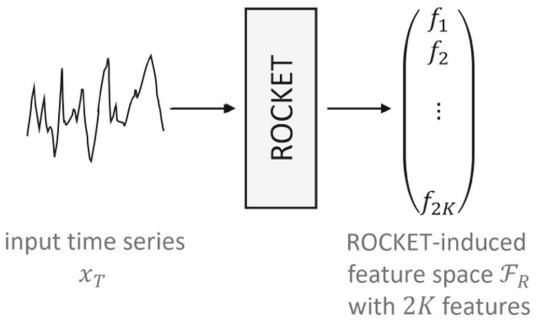

Figura 6.1:

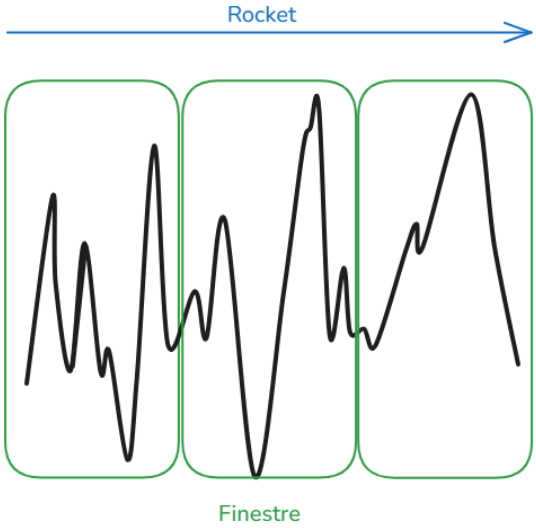

Figura 6.2: Finestre ROCKET su timeseries

ROCKET quindi scorre la timeseries con delle finestre di dimensione uguale tra loro, come possiamo vedere dalla Figura 6.2. Le caratteristiche estratte da ciascuna finestra vengono successivamente utilizzate da un classificatore, che elabora tali informazioni per fornire una predizione sull'intera serie temporale relativa alla feature originale.

6.2.1 Aspetti Tecnici

Per implementare ROCKET abbiamo usato la libreria utilizzata anche dal paper di ROCKAD tramite:

```
1 from sktime.transformations.panel.rocket import Rocket
```

I parametri più importanti sono:

- Numero di kernel (num_kernels): rappresenta il numero di kernel casuali da generare, il valore predefinito è 10.000, un numero maggiore di kernel tende a migliorare l'accuratezza della classificazione, di conseguenza aumenta la complessità e quindi il tempo di calcolo;
- Lunghezza del kernel (input_length): rappresenta la lunghezza del singolo kernel, questa è casuale e determina quanto, della serie temporale, viene considerato durante la convoluzione;
- Dilatazione: è un parametro che controlla la distanza tra i punti analizzati nel kernel; ROCKET usa varie dilatazioni per catturare caratteristiche di diverse scale temporali;
- Padding: determina come vengono gestiti i bordi delle serie temporali durante la fase di convoluzione.

I vantaggi di ROCKET sono:

- Efficienza computazionale elevata: è progettato per essere estremamente veloce e scalabile in modo che possa gestire grandi dataset in tempi ristretti;
- Robustezza: dato che si basa su kernel casuali, consente di generalizzare bene su nuovi problemi, senza la necessità di perfezionare gli iperparametri;
- Semplicità: ROCKET permette di usare un solo iperparametro, ossia il numero di kernel, riducendo così la complessità associata al perfezionamento rispetto ad altri metodi di classificazione delle serie temporali.

6.2.2 Metodologia Applicata

Nel paper di ROCKET[2], esso viene applicato, in particolare per la classificazione delle serie temporali, portando a sostegno molteplici valutazioni su dataset di BakeOff. Partendo da queste analisi e valutazioni del modello sulle serie temporali vogliamo riprodurre tali dati, molto promettenti, per la prima volta utilizzando ROCKET con lo scopo di classificazione delle anomalie sulle serie temporali.

Un aspetto da considerare è la formattazione dei dati che utilizziamo, infatti estraendo i dati dal dataset grezzo di OPS_SAT e elaborandoli, come vedremo meglio nella parte relativa a ROCKAD, in modo che siano compatibili con ROCKET.

6.2.3 Test Effettuati

Come primo aspetto abbiamo eseguito in locale gli esperimenti del paper di ROCKET per avere una validazione empirica dell'algoritmo, ottenendo risultati compatibili con quelli della tabella del paper[2]; successivamente abbiamo testato la nostra implementazione sui medesimi dataset ottenendo i risultati della Tabella 6.2.

Dataset	**Accuracy**
Coffee	1.0
Computers	0.64
Adiac	0.634
ArrowHead	0.788
BeetleFly	0.8
CinCECGTorso	0.7898
CBF	0.971
ChlorineConcentration	0.590
GunPoint	0.9866
Ham	0.771
HandOutlines	0.935
InlineSkate	0.367
Lightning2	0.623
Mallat	0.928
MiddlePhalanxTW	0.558

Tabella 6.2: Esecuzione di ROCKET su dataset "BakeOff"

Nei risultati che osserveremo in seguito nella Tabella 6.3, sono state effettuate molte prove con varie modalità. La prima è unsupervised, ovvero utilizziamo ROCKET per estrarre una grande quantità di caratteristiche e, tramite una threshold, fare la classificazione senza aver effettuato un training con le etichette dei risultati. Una threshold è una soglia, che indica il valore di riferimento per decidere quando una predizione deve essere classificata come positiva o negativa, in questo caso segnala quando un valore deve essere considerato o no anomalo. La seconda modalità che abbiamo osservato è supervised, effettuando quindi il training sulle features trovate da ROCKET passando però le etichette delle classificazioni. Per questo abbiamo utilizzato vari algoritmi supervised per analizzare le varie performance e trovare il migliore. Abbiamo anche osservato una modalità ibrida, ossia al posto dell'utilizzo di una threshold o un algoritmo supervised abbiamo puntato su uno unsupervised (KNN) facendo il training sulle features e restituendo delle prediction e delle classificazioni.

`RidgeClassifierCV`: è il modello di classificazione usato come standard nel paper di ROCKET; nel nostro caso utilizzandolo con il dataset OPS_SAT abbiamo riscontrato metriche migliori in tutti i testn.

Modalità	Accuracy	Precision	Recall	F1	MCC	AUC-PR	AUC-ROC	NScore
Unsupervised								
Threshold	0.546	1.0	0.048	0.092	0.161	0.46	0.468	0.419
Supervised								
RidgeCV	0.777	0.704	0.919	0.797	0.584	0.787	0.829	0.726
LogisticReg	0.777	0.704	0.919	0.797	0.584	0.773	0.837	0.774
RidgeReg	0.538	1.0	0.032	0.062	0.131	0.779	0.844	0.758
Ibrid Unsupervised								
KNN	0.531	0.6	0.048	0.09	0.049	0.407	0.294	0.29

Tabella 6.3: Algoritmi eseguiti su OPS_SAT

6.2.4 Implementazione

Riportiamo qui il codice dell'implementazione degli algoritmi più importanti. Il primo rappresenta ROCKET con la threshold come decision function:

```
from sktime.transformations.panel.rocket import Rocket
import numpy as np

def detect_anomalies_with_threshold(scores, threshold):
    return (scores > threshold).astype(int)

# Genera kernel convoluzionali casuali
input_length = X_train.shape[1]
num_kernels = 10000

rocket_transformer = Rocket(num_kernels = num_kernels,
    n_jobs=-1)

# Applica i kernel alle serie temporali
features_train = rocket_transformer.fit_transform(X_train)
features_test = rocket_transformer.transform(X_test)

# Sintesi delle caratteristiche per esempio
anomaly_scores_train = np.mean(features_train, axis=1)
anomaly_scores_test = np.mean(features_test, axis=1)

# Rilevamento delle anomalie
threshold = np.percentile(anomaly_scores_train , 95)
anomaly_labels_train = detect_anomalies_with_threshold(
    anomaly_scores_train , threshold)
anomaly_labels_test = detect_anomalies_with_threshold(
    anomaly_scores_test , threshold)
```

La seconda implementazione è quella relativa a ROCKET con RidgeClassifierCV:

```
import numpy as np
from sklearn.linear_model import RidgeClassifierCV
from scipy.special import softmax

# Genera kernel convoluzionali casuali
input_length = X_train.shape[1]
num_kernels = 10000
rocket_transformer = Rocket(num_kernels = num_kernels,
    n_jobs=-1)

# Applica i kernel alle serie temporali
features_train = rocket_transformer.fit_transform(X_train)
features_test = rocket_transformer.transform(X_test)

# Addestramento del modello supervisionato
model = RidgeClassifierCV(alphas = np.logspace(-3, 3, 10))
model.fit(features_train, y_train)

# Predizione delle anomalie nei dati di test
y_pred = model.predict(features_test)

# Per separare multiclasse o monoclasse
if  len(np.unique(y_test)) > 2:
    y_proba = softmax(model.decision_function(features_test)
    , axis=1)
else:
    y_proba = softmax(model.decision_function(features_test)
    , axis=0)
```

6.3 ROCKAD

ROCKAD (Random Convolutional Kernel Transform Anomaly Detector) è un algoritmo basato su ROCKET per la classificazione di anomalie su serie temporali.

6.3.1 Funzionamento

La logica dietro a questo algoritmo è spezzata in due parti che sfruttano due modelli: nella prima è utilizzato ROCKET come estrattore di caratteristiche unsupervised; nella seconda viene addestrato un singolo KNN o un insieme combinato di più KNN (detto ensemble di KNN).

In modo più dettagliato ROCKAD è diviso in tre passaggi fondamentali:

1. Estrazione di caratteristiche: tramite ROCKET ricaviamo le caratteristiche delle timeseries;

2. Trasformazione: le caratteristiche estratte vengono poi trasformate utilizzando un power trasformer;

3. Rilevamento anomalie: per concludere viene addestrato un insieme di KNN sulle caratteristiche estratte per calcolare dei punteggi di anomalia (score) per le serie temporali che serviranno successivamente per ottenere le predizioni.

I parametri principali di ROCKAD sono tre: il numero di kernel convoluzionali (il valore predefinito è 10.000), il numero di estimatori di tipo KNN (il valore predefinito è 10) e il numero di vicini che vengono utilizzati per calcolare il punteggio di anomalia (il valore predefinito è 5).

Oltre alla classe ROCKAD è necessario importare anche la classe NearestNeighborOCC, che implementa un classificatore di anomalie basato sul nodo prossimo più vicino. Questo è un metodo aggiuntivo per il rilevamento delle anomalie, il quale ha un funzionamento diverso dal KNN classico, dato che, invece di utilizzare la distanza media dai k vicini più prossimi, NearestNeighborOCC calcola un punteggio di anomalia basato sul rapporto tra due distanze:

1. la distanza tra la timeseries analizzata e il suo vicino prossimo;

2. la distanza tra il vicino più prossimo e il suo vicino più prossimo.

Se il risultato di questo rapporto è inferiore o uguale ad 1, la timeseries viene classificata come normale, altrimenti come anomala.

In conclusione NearestNeighborOCC aggiunge un ulteriore passaggio di analisi per la classificazione, il quale permette di aumentare la robustezza e l'accuratezza nel rilevamento delle anomalie.

6.3.2 Validazione

Nel paper relativo a ROCKAD[3] sono paragonati i risultati ottenuti con il suo utilizzo e il confronto con gli altri algoritmi. Nel nostro caso, come per ROCKET, validiamo l'algoritmo con i dati presenti nella Tabella 6.4, per poi concentrarci sulla sua applicazione sui nostri dataset di riferimento.

Dataset	**AUC-ROC**
GunPoint	0.9792
CBF	1.0
BME	1.0
ArrowHead	0.852
Computers	0.808
ChlorineConcentration	0.655
Ham	0.486
InlineSkate	0.637
Lightning2	0.623
Mallat	1.0
CricketX	0.760
Crop	0.999
CricketX	0.760
Fish	0.933

Tabella 6.4: Esecuzione locale ROCKAD

6.3.3 Preprocessing

Come prima cosa, per poter ottenere i risultati sul datsaet OPS_SAT, bisogna manipolare i dati per renderli compatibili con ROCKAD. I dati accettati dai metodi `fit` e `predict_proba` sono di tipo `numpy.array`.

Siamo partiti estraendo i dati grezzi provenienti da OPS-SAT nel file `segments.csv`, eseguendo un preprocessing per strutturarli in modo tale che abbiano una forma del tipo `(numero esempi, numero di features, lunghezza sequenza)`. Nel nostro caso abbiamo fissato la lunghezza della sequenza a 250 e il numero di features ad 1 per avere la compatibilità con ROCKAD, quindi eseguendo la funzione `.shape` dovrebbe risultare $(347, 1, 250)$.

I dati processati vengono poi passati a ROCKAD, suddivisi in due parti: una per il fitting del modello e l'altra per calcolare gli *score_train* e gli *score_test*. Questi punteggi vengono successivamente utilizzati rispettivamente per il training e la predizione dell'algoritmo NearestNeighborOCC, al fine di calcolare le predizioni e confrontarle con i *ground_truth*.

Il codice del preprocessing è diviso in due:

- la parte riguardante i dati di training

```
X_train_final = []
dfSegment = pd.read_csv("data/segments.csv", index_col="timestamp")

for channel in dfSegment["channel"].unique():
    # Itera su ogni segmento unico per il canale corrente
    for segment in dfSegment[dfSegment["channel"] == channel]["segment"].unique():
        mask = (dfSegment["train"] == 1) & (dfSegment["channel"] == channel) & (dfSegment["segment"] == segment)

        # Filtra i dati in base alla maschera
        X_trainS = dfSegment.loc[mask, "value"]
```

```
11
12          # Suddividi in sottoliste di STEP elementi
13          for i in range(0, len(X_trainS) - STEP + 1, STEP):
14              sublist = X_trainS[i:i + STEP]
15              X_train_final.append(sublist)
16
17  # Converti la lista in un numpy array
18  X_train = np.array(X_train_final)
19
20  # Reshape per ottenere la shape desiderata
21  X_train = X_train.reshape(X_train.shape[0], X_train.shape
        [1], 1)
22  X_train = X_train.transpose(0, 2, 1)
```

- la parte corrispondente ai dati di test

```
1  X_test_final = []
2  y_test_final = []
3
4  for channel in dfSegment["channel"].unique():
5      for segment in test_data[test_data["channel"] == channel
       ]["segment"].unique():
6
7          mask = (test_data["channel"] == channel) & (
       test_data["segment"] == segment)
8          X_testS = test_data.loc[mask, "value"]
9          y_testS = test_data.loc[mask, "anomaly"]
10
11         for i in range(0, len(X_testS) - STEP + 1, STEP):
12             X_test_final.append(X_testS[i:i + STEP])
13             y_test_final.append(y_testS[i])
14
15 X_test = np.array(X_test_final)
16 X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
17 X_test = X_test.transpose(0, 2, 1)
```

```
18 y_test = np.array(y_test_final)
```

6.3.4 Test Effettuati su OPS_SAT

Una volta effettuata la parte di preprocessing possiamo utilizzare i dati appena modificati per allenare ROCKAD ed effettuare la classificazione con NearestNeighborOCC. Come prima cosa inizializziamo il modello importando le classi ROCKAD e NearestNeighborOCC estratte dall'implementazione fornita con il paper ROCKAD[3]. Una volta effettuato possiamo passare a ROCKAD i dati di training per il fitting e calcoliamo con `predict_proba` gli score relativi al training set. Gli score sono poi passati a NearestNeighborOCC come dati di input, calcolando così anche gli score di test, sui quali per finire eseguiamo la `predict` con la quale otteniamo i valori risultati che ci indicano quali sotto sequenze della timeseries sono anomale e quali invece no.

Effettuando vari test abbiamo osservato che il miglior numero per `n-neighbors` è quello di default, ossia uguale a 5.

Per il numero di estimatori sono stati effettuati molteplici test riportando i risultati nella Tabella 6.5, dove il primo valore della tupla indica il numero di estimatori ed il secondo quello dei kernel.

Modalità	**Accuracy**	**Precision**	**Recall**	**F1**	**MCC**	**AUC-PR**	**AUC-ROC**	**N-Scores**
(10, 1000)	0.492	0.476	0.645	0.548	-0.002	0.755	0.702	0.677
(10, 5000)	0.538	0.514	0.613	0.559	0.084	0.76	0.705	0.677
(10,10.000)	0.546	0.515	0.823	0.634	0.137	0.757	0.704	0.677
(20,10.000)	0.454	0.447	0.613	0.517	-0.082	0.758	0.705	0.677
(30,10.000)	0.508	0.489	0.71	0.579	0.036	0.762	0.709	0.677
(35,10.000)	0.515	0.494	0.71	0.583	0.052	0.762	0.708	0.677
(40,10.000)	0.531	0.506	0.71	0.591	0.082	0.762	0.707	0.677
(10,20.000)	0.538	0.513	0.645	0.571	0.088	0.758	0.705	0.677

Tabella 6.5: Test con numero di estimatori variabile

6.4 Test NASA

Per verificare in maniera più completa le performance di ROCKET e ROCKAD, utilizziamo anche il dataset NASA. L'integrazione di questi algoritmi con il dataset, avviene in parte tramite l'uso di codice nel repository GitHub di SpaceAI[1]: come prima cosa vengono scaricati i dati, se non già presenti, da una risorsa remota tramite due modi diversi, chiamati "prediction" e "anomaly", che rispettivamente si usano per estrarre i dati delle serie temporali e per prendere i valori anomali nella timeseries. Successivamente dobbiamo spezzare la timeseries in sotto sequenze, così da poterla utilizzare con i kernel casuali di ROCKET per generare le features.

Effettuiamo gli stessi passaggi per l'estrazione e l'elaborazione dei dati di test e per i valori delle predizioni attese, ossia i *ground truth*; questi devono essere creati, dato che l'estrazione tramite la funzione è sotto forma di insieme, dove si evidenziano solo gli estremi tra i quali la timeseries è anomala. Questa lista viene spezzata in sottoliste di 250 elementi, che possono essere 0 o 1; per ottenere una lista monodimensionale è necessario applicare un metodo per scegliere se una sottolista è anomala o no, in base al numero di elementi uguali ad 1. Nel nostro caso abbiamo utilizzato una soglia del 25%, cioè 25 elementi uguali ad 1 in una sottolista.

Qui sotto è stato riportata l'implementazione della parte di codice relativa all'estrazione dei dati e la relativa formattazione per rendere compatibili i dati.

```
1 import numpy as np
2 import pandas as pd
3 from NASA.nasa import NASA
4
5 for channel_id in NASA.channel_ids:
6     if channel_id == "T-10":
7         continue
8     print(f"Processing channel: {channel_id}")
9
```

```
    # Lista per memorizzare i segmenti di training
    X_train_final = []

    # Uso del dataset NASA per tutti i canali
    dataset = NASA("./datasets", channel_id, mode="anomaly")
    data = dataset.data
    train = []
    for i in range(0, data.shape[0] - STEP +1, OFFSET):
        train.append(data[i:i+STEP])

    train = np.stack(train)

    # Estrazione anomalie
    dataset = NASA("./datasets", channel_id, mode="anomaly",
    train=False)
    data = dataset.data
    Test = []
    output = []
    o = np.zeros(data.shape[0])
    for start,end in dataset.anomalies:
        o[start:end] = 1
    for i in range(0, data.shape[0] - STEP +1, OFFSET):
        Test.append(data[i:i+STEP])
        output.append(o[i:i+STEP])

    output = np.stack(output)
    Test = np.stack(Test)
```

Arrivati a questo punto abbiamo tutti i dati che ci servono per utilizzare ROCKET e ROCKAD, che però dovranno essere applicati ciclicamente a tutti i canali del dataset; per ognuno di questi vengono valutate le metriche che serviranno per calcolare le metriche effettive di tutto il dataset, facendo una media tra tutti i risultati dei canali.

Poniamo particolare attenzione all'utilizzo dell'OFFSET introdotto per utiliz-

zare al meglio ROCKAD e ROCKET sul dataset NASA, dato che le timeseries di alcuni canali sono molto corte e non permettono di utilizzare il KNN come classificatore mantenendo il numero di nodi vicini mantenendo il valore di default.

Tramite l'uso dell'OFFSET in combinazione con la variabile STEP, permette di prendere finestre della stessa dimensione spostandosi solo del valore indicato nell'OFFSET, portando così ad avere overlapping, ossia due finestre che condividono alcuni elementi. Tutto ciò porta ad avere un maggior numero di finestre con lo stesso numero di esempi di partenza (Figura 6.3).

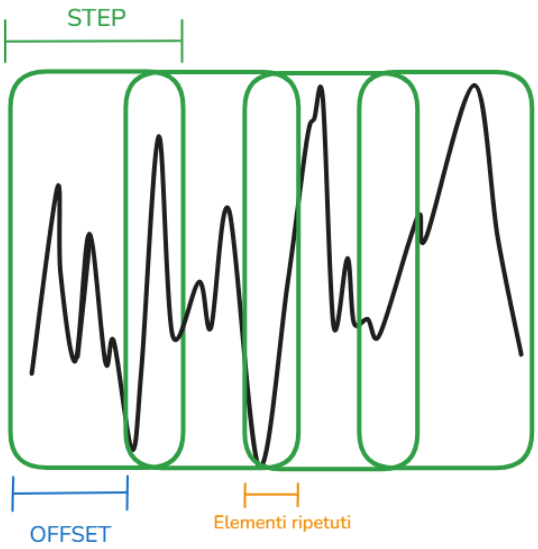

Figura 6.3: Finestre ROCKET su timeseries

6.4.1 Metriche Rilevate

In questo paragrafo riportiamo le metriche riscontrate dell'esecuzione di ROCKET e ROCKAD con diversi valori di: numero di kernel, valore di OFFSET e numero di n_neighbors.

Le metriche, data la conformazione del dataset, sono divise per canale, questo porta ad una scelta che bisogna effettuare, ossia come calcolare le metriche. Per uniformare i valori delle metriche, in particolare dell'F1, scegliamo di conservare in un file *.json* tutti i valori necessari (TP, TN, FP, FN) per calcolare le metriche dopo l'analisi di tutti i channel, ritardando così il calcolo. Riportiamo queste metriche nella Tabella 6.6. Un'altra possibile soluzione prevede di calcolare subito

le metriche di ogni canale e poi fare la media alla fine di tutte le metriche, questo calcolo è meno accurato e tende a produrre metriche minori, riportiamo comunque i risultati ottenuti nella Tabella 6.7.

Accuracy	**Precision**	**Recall**	**F1**	**MCC**	**AUC-PR**	**AUC-ROC**	**NScore**
ROCKET							
kernel=10.000, n_neighbors=1 e OFFSET=50							
0.735	0.299	0.471	0.366	0.232	0.286	0.652	0.378
kernel=10.000, n_neighbors=2 e OFFSET=30							
0.719	0.284	0.482	0.357	0.219	0.28	0.647	0.371
kernel=1000, n_neighbors=1 e OFFSET=50							
0.731	0.296	0.478	0.367	0.231	0.288	0.654	0.384
kernel=1000, n_neighbors=2 e OFFSET=30							
0.721	0.284	0.473	0.355	0.217	0.281	0.647	0.37
kernel=1000, n_neighbors=5 e OFFSET=10							
0.730	0.295	0.477	0.365	0.230	0.278	0.648	0.382
ROCKAD							
kernel=10.000, n_neighbors=1 e OFFSET=50							
0.593	0.106	0.19	0.136	-0.129	0.287	0.651	0.378
kernel=10.000, n_neighbors=2 e OFFSET=30							
0.628	0.131	0.207	0.16	-0.076	0.29	0.646	0.379
kernel=1000, n_neighbors=2 e OFFSET=50							
0.6	0.106	0.193	0.137	-0.131	0.286	0.651	0.378
ROCKAD con 1000 kernel, n_neighbors=2 e OFFSET=30							
0.614	0.129	0.218	0.162	-0.083	0.29	0.646	0.381
kernel=1000, n_neighbors=5 e OFFSET=10							
0.453	0.093	0.25	0.136	0.26	0.292	0.65	0.381

Tabella 6.6: Calcolo delle metriche posticipato

Accuracy	Precision	Recall	F1	MCC	AUC-PR	AUC-ROC	NScore
ROCKET							
kernel=10.000, n_neighbors=1 e OFFSET=50							
0.723	0.357	0.499	0.328	0.0	0.736	0.514	0.0
kernel=10.000, n_neighbors=2 e OFFSET=30							
0.709	0.341	0.519	0.337	0.0	0.736	0.517	0.0
kernel=1000, n_neighbors=1 e OFFSET=50							
0.722	0.354	0.501	0.320	0.0	0.738	0.510	0.0
kernel=1000, n_neighbors=2 e OFFSET=30							
0.712	0.327	0.499	0.319	0.0	0.737	0.509	0.0
kernel=1000, n_neighbors=5 e OFFSET=10							
0.719	0.355	0.504	0.332	0.0	0.741	0.505	0.0
ROCKAD							
ROCKAD con 10.000, n_neighbors=2 e OFFSET=50							
0.593	0.119	0.167	0.09	-0.111	0.729	0.516	0.454
kernel=10.000, n_neighbors=2 e OFFSET=30							
0.622	0.114	0.191	0.108	-0.079	0.728	0.514	0.473
ROCKAD con 1000 kernel, n_neighbors=2 e OFFSET=50							
0.594	0.107	0.16	0.088	-0.109	0.725	0.510	0.455
kernel=1000, n_neighbors=2 e OFFSET=30							
0.612	0.128	0.192	0.115	-0.09	0.723	0.503	0.448
kernel=1000, n_neighbors=5 e OFFSET=10							
0.476	0.127	0.291	0.14	0.136	0.728	0.499	0.467

Tabella 6.7: Metriche calcolate con la media

Per avere un valutazione più approfondita abbiamo effettuato test anche con altri classificatori, oltre a NearestNeighborOCC. Abbiamo preso in considerazione tre algoritmi, presenti all'interno della librerie fornite da `scikit-learn`[10], questi sono:

- OneClassSVM: sfrutta SVM, un algoritmo di tipo supervisionato che cerca di massimizzare il margine tra le classi, per delineare una frontiera che divide i valori anomali da quelli normali;
- Isolation Forest: sceglie una caratteristica casualmente e, tramite quella, isola le osservazioni in base ad una soglia, sempre scelta casualmente nell'intervallo di valori della caratteristica; questo processo avviene in modo ricorsivo, portando ad evidenziare le anomalie come i percorsi più brevi;
- Local Outlier Factor (LOF): si basa sul calcolo di un punteggio di anomalia per ogni osservazione, questo avviene tramite la misurazione della deviazione

della densità locale rispetto ad i nodi vicini.

Le metriche riportate nella tabella sono calcolate dopo aver effettuato la somma di tutti i valori di TP, TN, FP e FN, così da uniformare la valutazione. La colonna parametri che troviamo nella Tabella 6.8 e nella Tabella 6.9 corrispondono ripetitivamente a (numero di kernel, OFFSET) e (numero di kernel, n_neighbors, OFFSET).

Parametri	Accuracy	Precision	Recall	F1	MCC	AUC-PR	AUC-ROC	N-Scores
OneClassSVM								
(10.000, 250)	0.748	0.167	0.148	0.157	0.009	0.166	0.356	0.167
(10.000, 150)	0.66	0.109	0.152	0.127	-0.088	0.165	0.341	0.149
(1000, 50)	0.526	0.079	0.181	0.11	-0.218	0.164	0.334	0.122
(1000, 10)	0.5	0.083	0.205	0.118	-0.23	0.161	0.324	0.12
Isolation Forest								
(10.000, 50)	0.271	0.124	0.575	0.204	-0.321	0.129	0.379	0.093
(1000, 50)	0.285	0.125	0.568	0.205	-0.291	0.129	0.38	0.116
(1000, 10)	0.211	0.120	0.611	0.201	-0.492	0.125	0.365	0.096
Local Outlier Factor								
(10.000, 50)	0.19	0.14	0.773	0.237	-0.384	0.139	0.406	0.15
(1000, 50)	0.19	0.14	0.773	0.237	-0.383	0.137	0.404	0.132
(1000, 10)	0.194	0.143	0.793	0.242	-0.331	0.127	0.364	0.136

Tabella 6.8: Test con classificatori diversi - ROCKET

Parametri	Accuracy	Precision	Recall	F1	MCC	AUC-PR	AUC-ROC	N-Scores
OneClassSVM								
(10.000, 2, 50)	0.726	0.09	0.069	0.078	-0.086	0.287	0.651	0.378
(1000, 2, 50)	0.722	0.09	0.72	0.08	-0.087	0.286	0.651	0.378
(1000, 5, 10)	0.654	0.141	0.201	0.166	-0.05	0.292	0.65	0.381
Isolation Forest								
(10.000, 2, 50)	0.706	0.092	0.084	0.088	-0.094	0.287	0.651	0.378
(1000, 2, 50)	0.701	0.094	0.89	0.091	-0.094	0.286	0.651	0.378
(1000, 5, 10)	0.6	0.113	0.196	0.144	-0.117	0.292	0.65	0.381
Local Outlier Factor								
(10.000, 2, 50)	0.203	0.150	0.803	0.253	-0.29	0.287	0.651	0.378
(1000, 2, 50)	0.201	0.150	0.799	0.252	-0.303	0.286	0.651	0.378
(1000, 5, 10)	0.206	0.155	0.82	0.261	-0.256	0.292	0.65	0.381

Tabella 6.9: Test con classificatori diversi - ROKCAD

6.4.2 Panoramica su OPS_SAT con Algoritmi

In questa sezione riportiamo i risultati dei test effettuati su OPS_SAT, con i modelli appena testati su NASA. Questo ci permette di avere una visione di insieme del comportamento di questi modelli, in relazione anche al dataset utilizzato. Tutti i test sono effettuati utilizzando i migliori parametri riscontrati per ROCKAD, ossia 10 estimatori e 10.000 kernel, come possiamo vedere nella Tabella 6.5; per ROCKET invece non abbiamo molti parametri da modificare, ma manteniamo il numero di kernel di default pari a 10.000.

Algoritmo	Accuracy	Precision	Recall	F1	MCC	AUC-PR	AUC-ROC	N-Scores
ROCKET								
OneClassSVM	0.192	0.126	0.134	0.13	-0.1	0.463	0.392	0.371
Isolation Forest	0.385	0.144	0.269	0.187	-0.132	0.442	0.336	0.339
Local Outlier Factor	0.4	0.148	0.28	0.194	-0.098	0.441	0.334	0.371
ROCKAD								
OneClassSVM	0.3	0.3	0.3	0.3	-0.4	0.757	0.704	0.677
Isolation Forest	0.277	0.234	0.287	0.247	-0.476	0.757	0.704	0.677
Local Outlier Factor	0.408	0.295	0.426	0.31	-0.246	0.757	0.704	0.677

Tabella 6.10: Algoritmi diversi a confronto, ROCKET e ROCKAD su OPS_SAT

6.4.3 Analisi dei risultati NASA e Problematiche dei Modelli

Nel paragrafo successivo andremo ad analizzare i risultati elencati in tabella discutendo le migliori soluzioni.

La metrica più significativa tra tutte quelle calcolate, nel contesto del dataset NASA, è sicuramente il valore di *F1 Score*. Considerando quindi come metrica principale questa e il tempo impiegato per l'esecuzione, oltre a tutte le metriche secondarie riportate in tabella, possiamo osservare che nel caso di ROCKET i miglior valori dei parametri testati sono due:

- kernel=1000, n_neighbors=1 e OFFSET=50: questi valori garantiscono metriche buone ed una velocità di esecuzione molto elevata, data la semplicità del KNN e dell'OFFSET abbastanza alto, portando ad una scarsa espressività e performance peggiori con dataset rumorosi. Nel nostro caso però potrebbe tornare utile per la velocità di esecuzione, che con un processore non troppo potente è un parametro importante;

- kernel=10.000, n_neighbors=1 e OFFSET=50: in questa soluzione utilizziamo il valore standard dei kernel lasciando invariati, rispetto al modello sopra, gli altri valori. Un maggior numero di kernel porta ad un maggior numero di caratteristiche e quindi di informazioni per le classificazioni, che però sono semplificate dalla semplicità del KNN e dell'OFFSET. Il problema principale di questi parametri è il tempo di esecuzione prolungato ma non in modo eccessivo;

- kernel=1000, n_neighbors=5 e OFFSET=10: anche in questo caso le metriche sono comparabili al precedente, ma la velocità di esecuzione qui aumenta notevolmente, insieme ad un aumento dell'espressività del KNN e dal numero basso di OFFSET che rende l'algoritmo più robusto e stabile.

Per quanto riguarda ROCKAD, avremo anche qui due possibili soluzioni con metriche simili: la prima con 10.000 kernel, n_neighbors=2 e OFFSET=30; la seconda con 1000 kernel, n_neighbors=2 e OFFSET=30. Rispetto a ROCKET però, il tempo di esecuzione cambia in modo ancora più marcato, portando all'esclusione della prima soluzione, che utilizza il numero di kernel standard per ROCKET (10.000).

Test con Classificatori Diversi

I test effettuati con algoritmi diversi, rispetto a quelli utilizzati nei paper o nelle analisi precedenti (KNN e NearestNeighborOCC), ci permettono di capire meglio se i modelli scelti in partenza per l'analisi e quelli di default di ROCKET e ROCKAD sono i migliori. Questo porta alla ricerca di alternative più efficienti, tramite confronti e riflessioni.

I modelli alternativi sono stati scelti sulla base del dataset NASA, quindi sono tutti algoritmi unsepervised, questi permettono di superare il problema della lunghezza troppo breve delle timeseries per il KNN, che non aveva abbastanza nodi per la classificazione in alcuni canali.

Nel caso di ROCKET, gli algoritmi Isolation Forest, OneClassSVM e Local Outlier Factor si comportano male, avendo un F1 minore in tutti i casi, rispetto ai

modelli standard, e comportandosi addirittura peggio di un classificatore casuale, come possiamo notare dal valore di MCC negativo (**??**).

Confrontando comunque Local Outlier Factor (LOF) con il miglior risultato ottenuto con KNN, osserviamo che LOF ha un valore molto alto di recall, ossia permette di rilevare un maggior numero di anomalie ma questo a discapito della precisione, che invece rimane molto bassa. Questo rapporto ci permette di dire che LOF classifica molti falsi positivi che nel nostro caso è molto negativo.

In generale quindi possiamo affermare che con ROCKET, il miglior algoritmo è KNN, dato che LOF ha metriche peggiori praticamente su tutto.

Modello	**Accuracy**	**Precision**	**Recall**	**F1**	**MCC**	**AUC-PR**	**AUC-ROC**	**NScore**
LOF	0.194	0.143	0.793	0.242	-0.331	0.127	0.364	0.136
KNN	0.730	0.295	0.477	0.365	0.230	0.278	0.648	0.382

Tabella 6.11: Confronto modelli con ROCKET

Anche per ROCKAD il miglior compromesso tra gli algoritmi diversi testati rimane LOF, portando con se il problema dei troppi falsi positivi classificati. A differenza però di ROCKET i risultati ottenuti da LOF qui sono più promettenti, infatti effettuando un confronti metrica per metrica notiamo Precision, Recall, AUC-ROC e F1 risultano superiori con LOF, questo indica una migliore capacità di classificare le anomalie pagando il prezzo, come detto in precedenza, di molti falsi positivi. Con NearestNeighborOCC (NN_OCC) invece abbiamo valori maggiori delle metriche; Accuracy, MCC, AUC-PR e N-Score, che descrivono un modello che si comporta meglio di un algoritmo casuale e risulta più bilanciato.

La scelta tra i due modelli di classificazione dipende dall'utilizzo, LOF è sicuramente preferibile nel caso in cui volessimo trovare il maggior numero di anomalie anche a costo di avere molti falsi positivi, invece come per il nostro caso, NearestNeighborOCC rappresenta una scelta più sicura e meno costosa in termini di controlli di falsi positivi.

Modello	**Accuracy**	**Precision**	**Recall**	**F1**	**MCC**	**AUC-PR**	**AUC-ROC**	**NScore**
LOF	0.0.206	0.155	0.82	0.261	-0.256	0.292	0.65	0.381
NN_OCC	0.476	0.127	0.291	0.14	0.136	0.728	0.499	0.467

Tabella 6.12: Confronto modelli con ROCKAD

7 Analisi dei Risultai

Dall'analisi dei valori notiamo che le migliori soluzioni escludono, per la maggior parte, l'utilizzo di 10.000 kernel se non con un valore di OFFSET abbastanza alto e di conseguenza un KNN molto semplificato. Contrariamente a quanto affermato nei paper, relativi ad entrambi i modelli, ROCKET e ROCKAD non hanno un efficienza così elevata. Prendendo, infatti, in considerazione il dataset OPS_SAT, in caso di un valore basso di STEP, o nel caso del dataset NASA prendendo un valore basso di OFFSET, riscontriamo un efficienza non ottimale.

Nella fase di test non è stato possibile estrarre le metriche con un valore di STEP inferiore a 250, dato che portava ad un esecuzione che non terminava mai. Per il dataset NASA, invece, questa situazione si verificava con un numero di kernel pari a 10.000 ed un valore di OFFSET inferiore a 50, questo valore di OFFSET portava con sé anche il problema di non poter aumentare il numero di nodi vicini del KNN, per mancanza di vicini, data la struttura del dataset.

Relativamente alle metriche in questione analizziamo il grafico in Figura 7.1, che mostra come cambia la metrica fondamentale, ossia l' F1, con l'aumentare dei kernel. Nel caso di un numero di kernel pari a 1000 o 10.000 il valore di F1 è pressoché identico, mentre in tutti gli altri valori, la metrica diminuisce significativamente. Da considerare nell'analisi dobbiamo evidenziare anche il tempo di esecuzione, come mostrato dal grafico nella Figura 7.2, che riporta i tempi di esecuzione dei vari valori dei kernel.

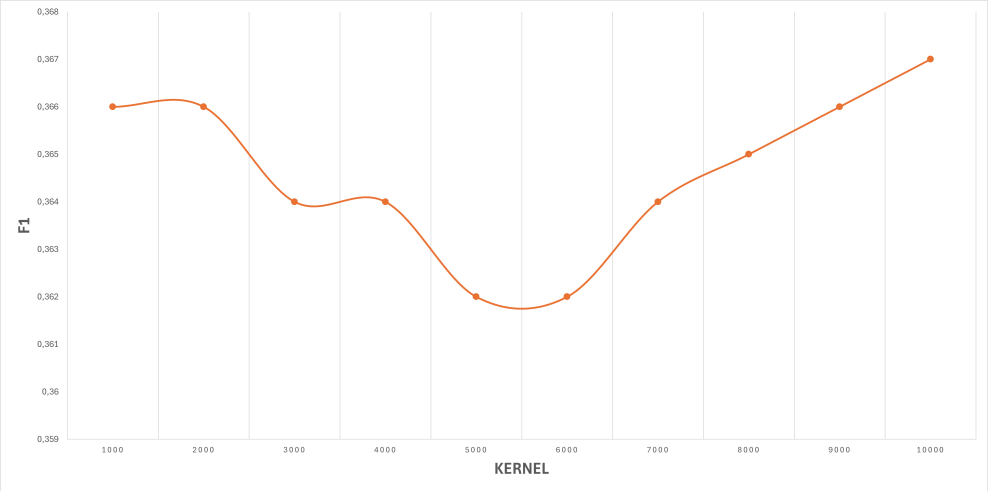

Figura 7.1: Relazione tra F1 e Numero di Kernel

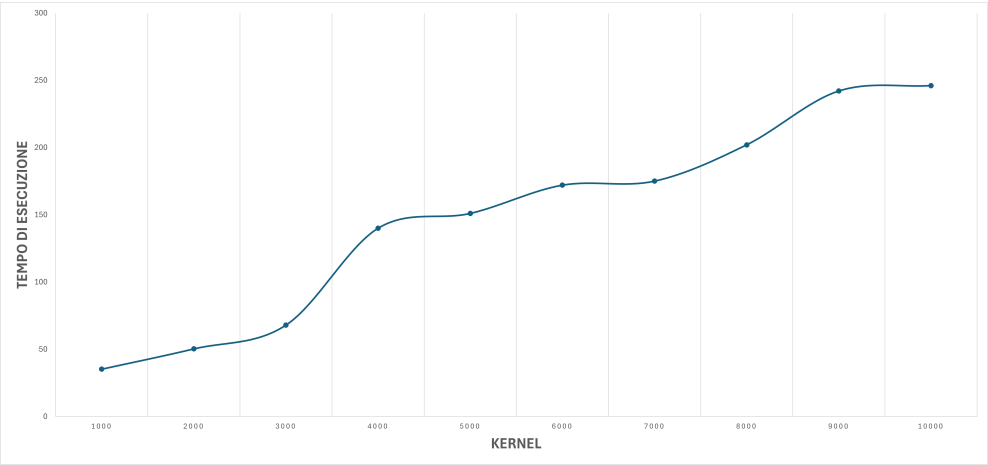

Figura 7.2: Relazione tra Tempi di Esecuzione e Numero di Kernel

Da questi grafici possiamo osservare come il tempo di esecuzione aumenti in maniera significativa all'aumentare del numero di kernel, lasciando invariati tutti gli altri parametri. Da questo, evidenziamo meglio che la scelta migliore per il numero di kernel, praticamente a parità di valore di F1, è indirizzata verso un valore pari a 1000, dove abbiamo un tempo di esecuzione quasi 7 volte minore rispetto a 10.000.

Il problema che abbiamo nell'uso di ROCKET, soprattutto su NASA, è proprio quello evidenziato sopra, dato che fissando STEP a 250 l'algoritmo funziona senza troppi problemi e ad una velocità accettabile, ma al diminuire di questo valore il numero di caratteristiche aumenta esponenzialmente portando ad impiegare troppo tempo per fare training e test, non riuscendo a concludere la classificazione.

Questa riflessione ci permette di capire che il collo di bottiglia di questa metodologia non è direttamente l'applicazione di ROCKET, ma la consecutiva applicazione di un algoritmo di classificazione, che avendo una quantità di features enorme, non riesce a gestirli in tempi normali.

7.1 ROCKAD

In merito a ROCKAD, l'analisi della metrica F1 con un numero di kernel variabile tra 1000 e 10.000 è esposta nel grafico in Figura 7.3. Queste misurazione sono state estratte mantenendo fissati il numero di n_neighbors a 2 e OFFSET a 50; notiamo che questo ha un andamento diverso da quanto visto per ROCKET, portando a premiare un numero di kernel compreso tra 3000 e 5000, dove riscontriamo valori più alti di F1.

Il grafico è stato proposto con questi parametri a causa dei problemi relativi ai tempi di esecuzione, infatti nella Tabella 6.6 possiamo vedere come i parametri migliori hanno un OFFSET uguale a 30, ma provando l'esecuzione con quest'ultimo valore, l'algoritmo non terminava in tempi normali. Infatti a conferma di quanto appena detto, come possiamo vedere dal grafico in Figura 7.4, i tempi di esecuzione, aumentano molto rapidamente fino ad arrivare, nel nostro caso, a più di dieci minuti; tutto questo praticamente a paragone dei valori di F1.

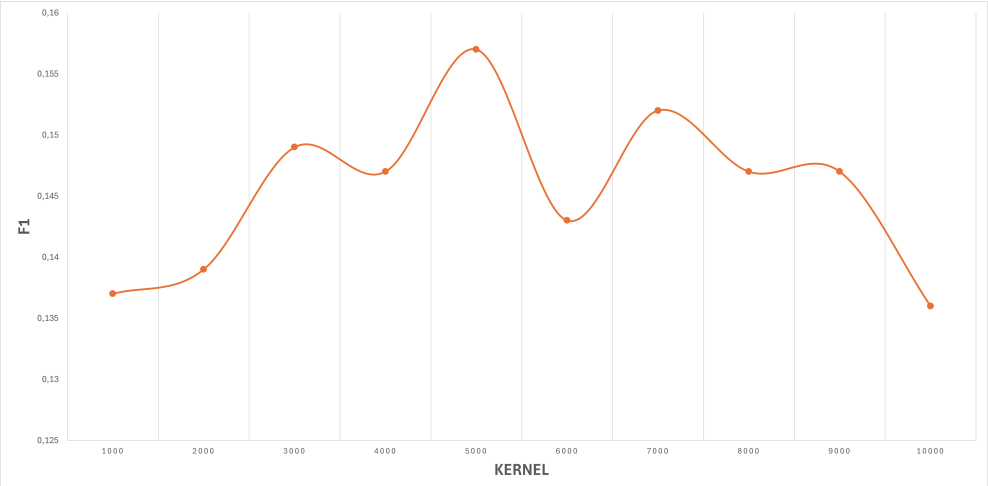

Figura 7.3: Relazione tra F1 e Numero di Kernel

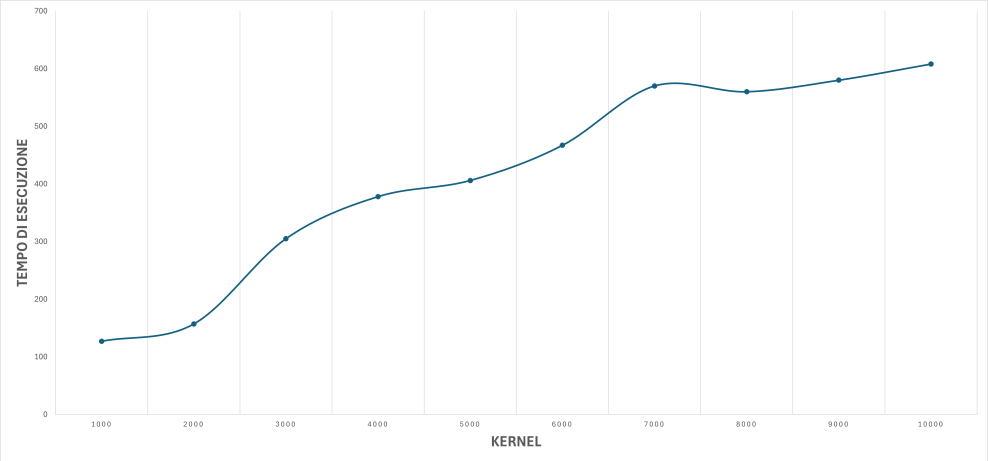

Figura 7.4: Relazione tra Tempi di Esecuzione e Numero di Kernel

Analizzando i dati raccolti fino a questo momento, ROCKET si comporta in maniera migliore in tutti i test effettuati, questo lo possiamo osservare dalle metriche, sul dataset OPS_SAT, nelle tabelle 6.3 e 6.5. Prendendo come metrica di riferimento sempre F1, vediamo che ROCKET ottiene un punteggio di 0.797, invece ROCKAD totalizza 0.634 come picco massimo.

7.2 Riflessioni

Nonostante tutto ciò che è stato discusso precedentemente l'utilizzo di ROCKET e ROCKAD per il rilevamento delle anomalie risulta un opzione più che valida da poter percorrere, anche se ancora troppo immatura per essere al pari di modelli molto più usati e con varie iterazioni di miglioramenti che gli sono stati apportati, come nel caso di XGBOD che abbiamo visto.

Soprattutto ROCKET, che permette di essere utilizzato in un ambiente senza etichettatura dei dati, si rivela interessante per i dataset come NASA o più in generale in un contesto reale con poche e rare anomalie, spesso non etichettate.

Come già evidenziato in precedenza, ROCKET ha un basso costo computazionale a confronto con modelli più complessi, questo, soprattutto nel contesto satellitare, è un punto critico date le risorse hardware limitate.

Purtroppo però l'applicazione di ROCKAD in un contesto reale risulta ancora poco efficiente, dato soprattutto il suo utilizzo del modello NearestNeighborOCC,

portando ad una computazione maggiore ed un rischio per la scarsa capacità hardware.

7.2.1 Possibili Sviluppi

Per rendere ROCKET e ROCKAD preferibili ai già consolidati modelli, è necessario apportare ulteriori efficientamenti, così da uguagliare o addirittura superare quelli già esistenti. Ovviamente questo serve anche per migliorare le attuali performance, già buone in alcuni campi, come l'uso di ROCKET con dataset senza etichettatura. Elenchiamo di seguito delle possibili proposte per migliorare questi modelli:

- Complessità del Modello: ridurre la complessità del modello, data soprattutto dalla generazione di un gran numero di kernel e quindi di caratteristiche; questo può avvenire tramite tecniche di pruning per eliminare le caratteristiche ridondanti o non utili, riducendo la complessità del modello e la possibilità di cadere in overfitting;

- Compressione dei Dati: comprimere i dati permette di usarli in modo più efficiente per l'elaborazione, riducendo così anche l'uso della memoria del dispositivo satellitare;

- Maggiori Informazioni: data la natura casuale di ROCKET, i pattern non sono analizzabili esternamente portando ad un limite nell'interpretazione e comprensione delle caratteristiche; migliorando questo aspetto, tramite ad esempio strumenti di visualizzazione, potrebbe portare ad un miglioramento nell'analisi dei risultati;

- Validazione: l'ultimo aspetto, dopo aver effettuato tutte le migliorie possibili, sarebbe l'applicazione di questi modelli in un contesto reale per validarne l'efficacia e l'efficienza, questo potrebbe avvenire, ad esempio, sulla nuova missione OPS_SAT programmata per la fine di questo anno.

8 Panoramica sull'Efficientamento

Efficientare un algoritmo significa renderlo eseguibile in maniera ottimizzata anche con macchine con caratteristiche molto restringenti, come nel caso dei satelliti, con poche risorse hardware come ad esempio la CPU poco performante. Nel nostro caso effettuiamo un Transfer Learning, ossia il riuso di un modello già addestrato su un dataset, riadattandolo al nuovo compito o dataset che vogliamo utilizzare. Questo processo può essere effettuato in diversi modi:

- Estrazione delle Caratteristiche: consiste nell'estrarre caratteristiche utili dai dati senza modificare i pesi calcolati nel training precedente;
- Fine-Tuning:
  - Addestramento sull'intera rete: in questo caso aggiorniamo i pesi eseguendo di nuovo il training sul nuovo dataset;
  - Addestramento del Classificatore Finale: aggiorniamo solo gli stati più profondi (finali) della rete mentre i pesi dei primi rimangono congelati;
  - Addestramento a Blocchi: i pesi dei blocchi vengono aggiornati singolarmente effettuando l'addestramento blocco per blocco.

Per il nostro scopo possiamo effettuare diversi tipi di fine-tuning:

1. Precisione dei Pesi: i pesi derivati dall'addestramento della rete hanno una precisione, ossia il numero di bit che vengono usati per rappresentare il numero, diminuendola impiegheremo meno memoria necessaria e velocizzeremo la velocità di calcolo;

2. Riducendo la Complessità del Modello: possiamo eliminare nodi poco significativi tramite la tecnica detta pruning, diminuendo anche la quantità di operazioni necessarie;

3. Compromesso Concorrenza Aggiornamento pesi: il batch size ossia il numero di pesi che si attende prima di aggiornarli evitando di farlo ogni volta per non sprecare risorse di calcolo, incentivando così la concorrenza;

4. Ridurre tempo di addestramento: riducendo il numero di epoche, ossia il numero di volte in cui il dataset viene passato attraverso il modello durante la fase di allenamento.

A Questa è un'appendice

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Questa è un'altra sezione, ma non viene inserita nell'indice

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Bibliografia

[1] ContinualIST. Spaceai. *https://github.com/continualist/space-ai*.

[2] Angus Dempster, François Petitjean, and Geoffrey I Webb. Rocket: Exceptionally fast and accurate time classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 2020.

[3] Andreas Theissler, Manuel Wengert, and Felix Gerschner. Rockad: Transferring rocket to whole time series anomaly detection. In Bruno Crémilleux, Sibylle Hess, and Siegfried Nijssen, editors, *Advances in Intelligent Data Analysis XXI*, pages 419–432, Cham, 2023. Springer Nature Switzerland.

[4] J. Andrzejewski C. Haskamp B. Ruszczak, K. Kotowski and J. Nalepa. Oxi: An online tool for visualization and annotation of satellite time series data. Jul. 2023.

[5] D. J. Evans. Ops-sat: Fdir design on a mission that expects bugs - and lots of them. *SpaceOps Conferences*, SpaceOps 2016 Conference, American Institute of Aeronautics and Astronautics, 2016.

[6] Bogdan Ruszczak, Krzysztof Kotowski, Jacek Andrzejewski, Alicja Musiał, David Evans, Vladimir Zelenevskiy, Sam Bammens, Rodrigo Laurinovics, and Jakub Nalepa. Machine learning detects anomalies in ops-sat telemetry. In Jiří Mikyška, Clélia de Mulatier, Maciej Paszynski, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M.A. Sloot, editors, *Computational Science – ICCS 2023*, pages 295–306, Cham, 2023. Springer Nature Switzerland.

[7] Brunak S. Chauvin Y. Andersen C. A. F. Nielsen H. Baldi, P. Assessing the accuracy of prediction algorithms for classification: an overview. *https://academic.oup.com/bioinformatics/article-pdf/16/5/412/48836094/bioinformatics*$_1$6$_{54}$12*.pdf*, $Bioinformatics16, 2000.$

[8] Google. Classificazione: Roc e auc. *https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc*.

[9] McNeil B. J. Hanley, J. A. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143, 1982.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.