Corso di Laurea in Informatica

TESI DI LAUREA

Nuovi approcci metodologici per il rilevamento di anomalie: una Validazione Empirica

Relatore:
Vincenzo Lomonaco

Correlatore:
Lampei Li

Candidato:
Francesco Fiaschi

ANNO ACCADEMICO 2024/2025

Sommario

bgnbrtgnbrtgntr

Indice

Elenco delle figure

1 Aquisizione dei Dati

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 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.

1.3 Obbiettivo

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, al quale accediamo tramite SpaceAI [1] riportando anche qui metriche di valutazione, allo scopo di poter confrontare gli algoritmi in termini di efficacia.

2 Misure di Valutazione

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.

2.1 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[2] 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 2.1: Configurazione Dataset ESA

Possiamo osservare dalla Tabella 2.1 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.

2.2 Motivazioni

Il dataset ESA pecedentemente 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 55 e 27 parametri di telemetria, rispettivamente, 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. Quindi è stato deciso che questi dataset non andrebbero usati per il benchamrking 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[3]

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 sui dati. 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[4], registrate in 9 canali di telemetria.

Infine questo dataset con il relativo benchmark e tutti i dati al suo interno disponibili 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 averli distribuiti a bordo del satellite).

2.3 OP-SAT Dataset

Nel dataset OPS_SAT sono contenuti i dati delle telemetrie del satellite OPS_SAT dell'ESA (Figura 2.1). 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 Di-

cembre del 2019 con lo scopo di dimostrare l'elaborazione dei dati in orbita e di generare dati utili come immagini satellitari e telemetrie.

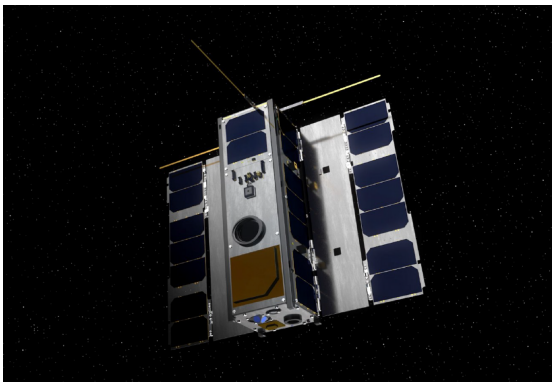

Figura 2.1: Satellite OPS_SAT. Fonte: European Space Agency

Come nel dataset precedente anche qui i dati hanno bisogno di essere preprocessati per renderli fruibili alla maggior parte degli algoritmi (OXI[2]). 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 2.2: Composizione Dataset OPS_SAT

Le classi mostrate in Tabella 2.2, 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 obbiettivo.

3 Valutazione Modelli

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:

  $$\frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

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

  Formula:

  $$\frac{TP}{TP + FP} \tag{3.2}$$

  Nel nostro caso questa metrica è particolarmente importante dati 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:

$$\frac{TP}{TP + FN} \tag{3.3}$$

Nel contesto dei satelliti questa metrica è cruciale dato che con un valore basso avremmo un grande numero di falsi negativi (FN) che quindi 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:

  $$F_1 = \frac{2 * TP}{2 * TP + FP + FN} \tag{3.4}$$

  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)**[5]: rappresenta una misura della qualità del modello con dati molto variabili.

  Formula:

  $$\frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + TP) \cdot (TN + FN)}} \tag{3.5}$$

  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 sotto la curva ROC ($\mathbf{AUC_{ROC}}$**[6][7]): rappresenta il rapporto tra il tasso di veri positivi e il tasso di falsi positivi (Figura: 3.1) 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 sotto al grafico, quella con l'area più grande è generalmente quello migliore.

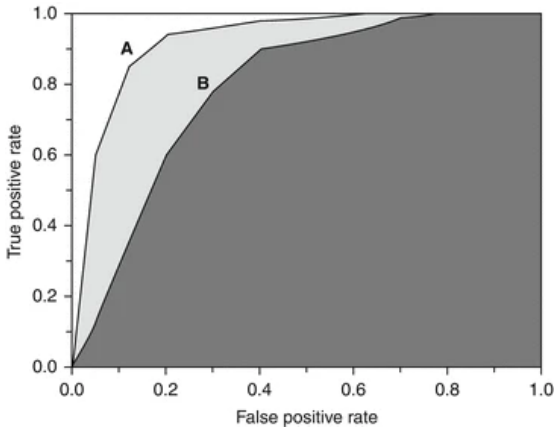

Figura 3.1: Curva ROC

- **Area sotto la curva Precisione-Richiamo (AUC_PR)**: rappresenta un semplice modo per sintetizzare le prestazioni generali di un modello, più alto è il valore più avrà un numero di predizioni alto. 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.

4 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.

4.1 XGBOD

XGBOD (eXtreme Gradient Boosting for Outlier Detection) è una struttura composta in 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 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..

4.1.1 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.

4.1.2 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	Prec.	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 4.1: Prove esecuzione di XGBOD

LEGENDA:

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

Dalla Tabella 4.1.2 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 dei risultati ottimi l'esecuzione rimane praticamente istantanea sul nostro dataset di esempio OPS_SAT.

4.2 Rocket

Rocket è un algoritmo convoluzionale, anche detti reti neurali convoluzionali (CNN), questi 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 applicabile a rocket come il pooling che ha lo scopo di ridurre la dimensione e quindi a 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. Questa 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.

4.2.1 Aspetti Tecnici

Per implementare Rocket abbiamo usato le funzioni che il relativo paper[8] aveva fornito:

```
1 def generate_kernels(input_length, num_kernels)
2 def apply_kernel(X,weights, length, bias, dilation, padding)
3 def apply_kernels(X, kernels)
```

I parametri più importanti sono:

- Numero di kernel (num_kernels): rappresenta il numero di kernel casuali da generare, il valore predefinito è 10000, un numero maggiore di kernel tende a migliorare l'accuratezza della classificazione ma 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 a 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.

4.2.2 Metodologia Applicata

Nel paper di rocket[8] viene usato in particolare per la classificazione delle serie temporali portando a sostegno molte 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.

4.2.3 Test Effettuati

Come primo aspetto abbiamo eseguito in locale gli esperimenti del paper di ROCKET per avere una validazione empirica dell'algoritmo ottenendo gli stessi risultati della tabella del paper[8], successivamente abbiamo provato la nostra implementazione sui medesimi dataset ottenendo i risultati della Tabella 4.2.3. Nei

Dataset	**Accuracy**	**Precision**	**Recall**	**F1**	**MCC**	**auc-pr**	**auc-roc**
Coffee	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Computers	0.704	0.713	0.704	0.701	0.417	0.358	0.753
Adiac	0.777	0.823	0.777	0.761	0.773	0.79	0.978
ArrowHead	0.771	0.809	0.771	0.756	0.69	0.88	0.942
BeetleFly	0.85	0.885	0.85	0.847	0.834	0.331	1.0
CinCECGTorso	0.823	0.831	0.823	0.822	0.767	0.914	0.963
CBF	0.992	0.992	0.992	0.992	0.988	1.0	1.0
ChConcentration	0.782	0.778	0.782	0.774	0.633	0.813	0.902
GunPoint	1.0	1.0	1.0	1.0	1.0	0.315	1.0
Ham	0.657	0.658	0.657	0.655	0.314	0.37	0.734
HandOutlines	0.916	0.917	0.916	0.915	0.817	0.963	0.96
InlineSkate	0.705	0.708	0.705	0.7	0.404	0.865	0.838
Lightning2	0.738	0.743	0.738	0.733	0.473	0.829	0.805
Mallat	0.943	0.951	0.943	0.944	0.936	0.977	0.994
PhalanxTW	0.552	0.516	0.552	0.523	0.419	0.438	0.822

Tabella 4.2: Esecuzione di rocket su dataset "BakeOff"

risultati che osserveremo successivamente nella Tabella 4.3, sono state effettuate molte prove con varie modalità, una unsepervised quindi usando 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 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 trining sulle feaures e restituendo delle prediction e delle classificazioni.

`RidgeClassifierCV`: è il modello usato nella demo del paper di rocket ed abbiamo riscontrato metriche migliori anche nei nostri test.

Modalità	**Acc.**	**Prec.**	**Recall**	**F1**	**MCC**	**auc-pr**	**auc-roc**	**NScore**
Unsupervised								
Threshold	0.834	0.963	0.23	0.371	0.424	0.726	0.772	0.646
Supervised								
RidgeCV	0.977	0.972	0.92	0.945	0.932	0.962	0.984	0.929
LogisticReg	0.974	0.963	0.912	0.936	0.92	0.964	0.986	0.92
RidgeReg	0.864	0.936	0.389	0.55	0.554	0.871	0.94	0.92
Ibrid Unsupervised								
KNN	0.834	0.963	0.23	0.371	0.424	0.726	0.772	0.646

Tabella 4.3: Algoritmi eseguiti su OPS_SAT

4.2.4 Implementazione

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

```
from rocket_functions import generate_kernels, apply_kernels

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
kernels = generate_kernels(input_length, num_kernels)

# Applica i kernel alle serie temporali
features_train = apply_kernels(X_train2, kernels)
features_test = apply_kernels(X_test2, kernels)

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

# 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
```

```
import pandas as pd
from sklearn.linear_model import RidgeClassifierCV
from scipy.special import softmax

from rocket_functions import generate_kernels, apply_kernels

# Genera kernel convoluzionali casuali
input_length = X_train.shape[1]
num_kernels = 1000
kernels = generate_kernels(input_length, num_kernels)

# Applica i kernel alle serie temporali
features_train = apply_kernels(X_train2, kernels)
features_test = apply_kernels(X_test2, kernels)


# 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)
```

4.3 ROCKAD

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

4.3.1 Funzionamento

La logica dietro a questo algoritmo è l'utilizzo di ROCKET come estrattore di caratteristiche non supervisionato e addestrare 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, spiegato precedentemente, 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 estrarre dei punteggi di anomalia (score) per le serie temporali che serviranno successivamente per calcolare le predizioni.

I parametri principali sono tre: il numero di kernel convoluzionali (il valore predefinito è 10000), il numero di estimatori di tipo KNN (valore predefinito 10) e il numero di vicini che vengono utilizzati per calcolare il punteggio di anomalia (valore predefinito 5).

Oltre alla classe ROCKAD è necessario importare anche la classe NearestNeighborOCC, che implementa un classificatore di anomalie basato su il nodo prossimo più vicino. Questo è un metodo aggiuntivo per il rilevamento delle anomalie, il quale ha un funzionamento diverso da KNN 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: la distanza tra la timeseries analizzata e il suo vicino prossimo e la distanza tra il vicino più prossimo e il suo vicino più prossimo. Se il risultato di questo rapporto fosse

inferiore o uguale ad 1, la timeseries viene classificata come normale, altrimenti viene classificata come anomala.

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

4.3.2 Validazione

Nel paper relativo a ROCKAD[9] sono paragonati i risultati ottenuti con il suo uso e il confronto con gli altri algoritmi. Nel nostro caso, come per ROCKET, validiamo l'algoritmo con i dati presenti nella Tabella 4.3.2, 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 4.4: Esecuzione locale ROCKAD

4.3.3 Test Effettuati

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 da estrarre 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 riguardande 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"]

        # Suddividi in sottoliste di STEP elementi
        for i in range(0, len(X_trainS) - STEP + 1, STEP):
            sublist = X_trainS[i:i + STEP]
            X_train_final.append(sublist)

# Converti la lista in un numpy array
X_train = np.array(X_train_final)

```

```
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)
```

e 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)
```

5 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] 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.

[3] 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.

[4] 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.

[5] 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*$_1 6_{54} 12.pdf, Bioinformatics 16, 2000$.

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

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

[8] 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.

[9] 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.