A Python Package for Time Series Event Detection

EventDetector

Menouar Azib

Benjamin Renard, Philippe Garnier, Vincent Génot, Nicolas André, Myriam Bouchemit

AKKODIS · FRANCE RELANCE Plan de relance · UNIVERSITÉ TOULOUSE III PAUL SABATIER Université de Toulouse · irap astrophysique & planétologie · CDPP

Fast Machine Learning for Science Workshop 2023, Imperial College London, 27/09, 14:15 - 14:30

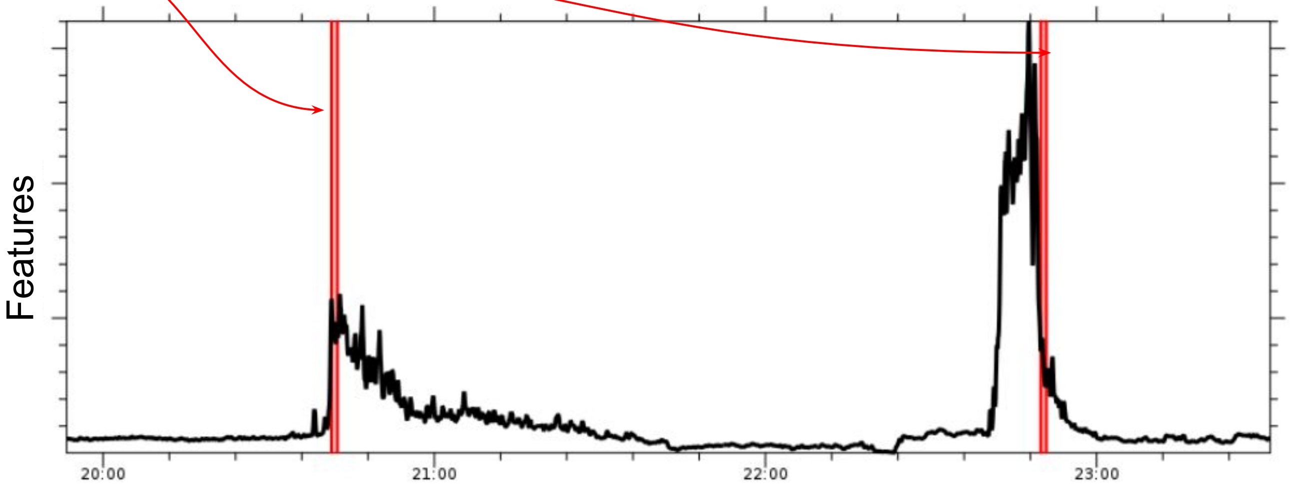
Outline

1. **Introduction**
2. **Review of Existing Literature**
3. **Method**
4. **Python Package**
5. **Usage Examples**
6. **Conclusion**

1. Event Detection in Time Series

- Identifying significant occurrences (events) within time-ordered data (time series).
  - Change Point Detection

- Events:
  - Anomalies
  - Scientific Events
  - Frauds
  - …



2. Review of Existing Literature

Supervised Machine Learning Methods

- A Common Approach: Treating Event Detection as Binary Classification.
- Each time step is labeled as either 0 (non event) or 1 (event).

	Feature 1	Feature 2	Feature 3	Event Label
2023-09-20 07:29:14.123456789	1	4	7	0
2023-09-20 07:29:14.987654321	2	5	8	1
2023-09-20 07:29:15.192837465	3	6	9	0

- Methods include Random Forest, Neural Networks, Naive Bayes, Logistic Regression, SVM, …
- Review: https://doi.org/10.1007/s10115-016-0987-z

3. Method

Combines 4 distinct features

- **Regression-Based Approach**
  - Prediction of continuous values ≠ binary classification.
- **No Need for Time-Step Labeling**
  - No need for labeling each time step ≠ time-consuming.
  - Requires only reference (true) events to be defined as specific time points/intervals of time.

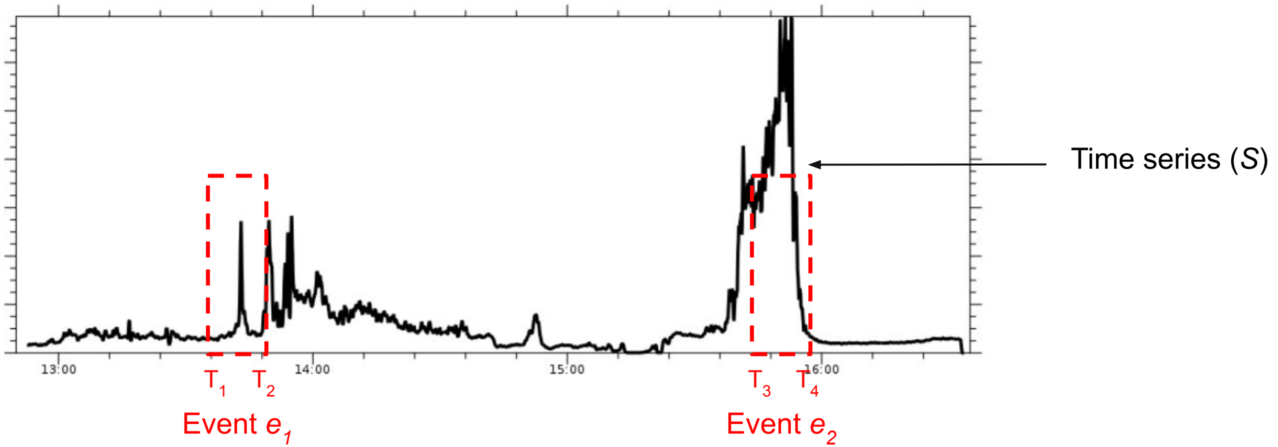
Start Time	Stop Time
2023-01-01 00:00:00	2023-01-01 00:01:00
2023-01-01 00:02:00	2023-01-01 00:03:00
2023-01-01 00:04:00	2023-01-01 00:05:00

- **Stacked Ensemble Learning Meta-Model**
  - Leverages the strengths of multiple base models: robustness.
- **Practical Implementation**
  - Facilitates practical implementation: Python package.

3. Method

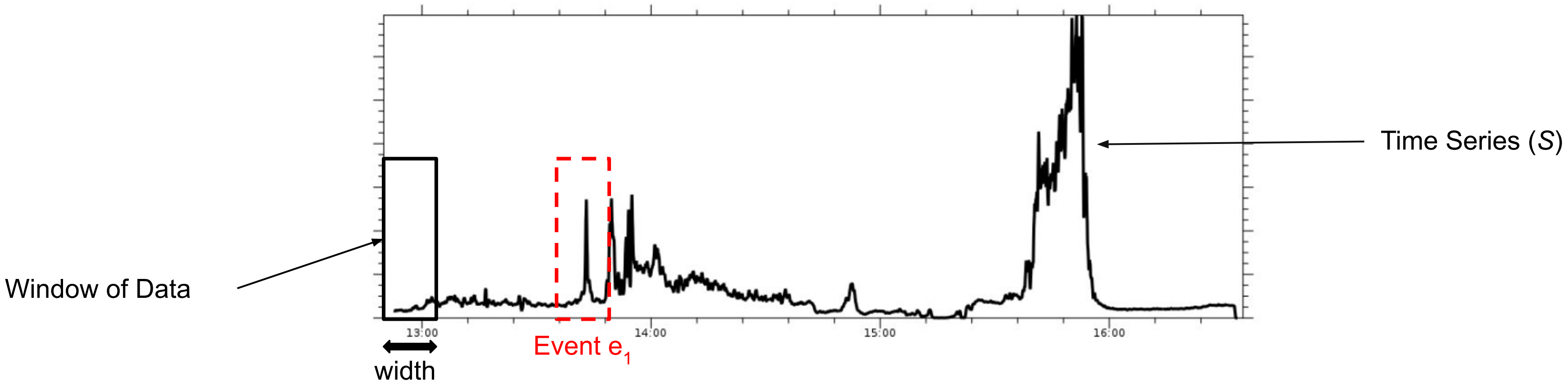
Data

- We require two pieces of data:
  - Time series (S)
    $$S : \mathbb{R} \to \mathbb{R}^f$$
    $$t \mapsto S(t)$$
  - The list of reference events (E)
    $$E = [e_1, e_2, \ldots]$$



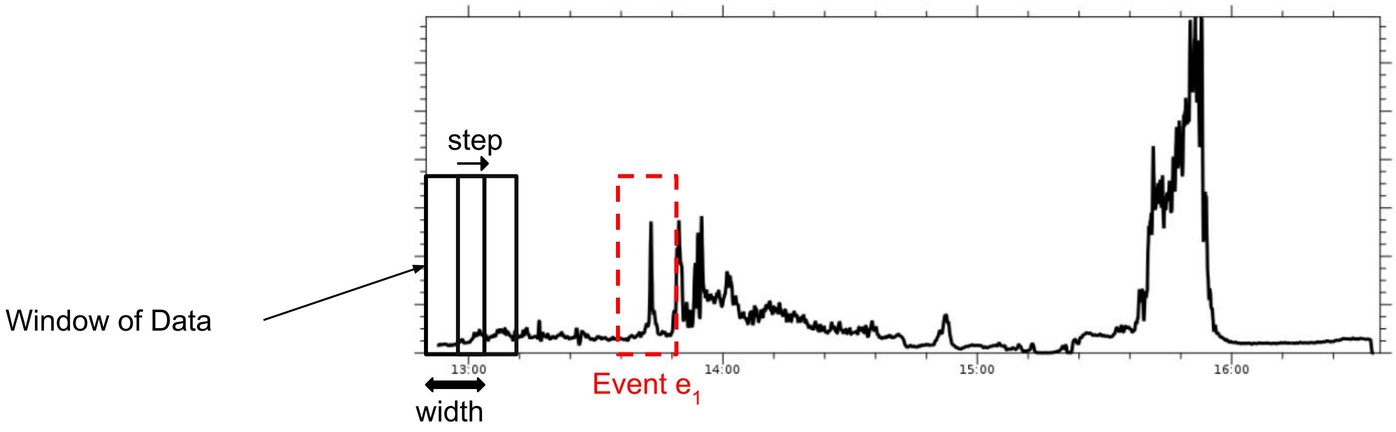
3. Method

Regression-Based Approach: Sliding Windows



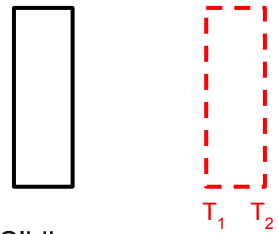
3. Method

Regression-Based Approach: Sliding Windows



3. Method

Regression-Based Approach: Overlapping Parameter (op)



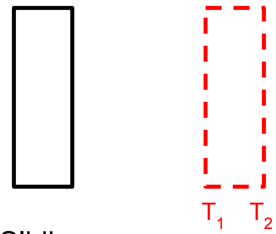
Sliding Window $\boldsymbol{w_i}$

Event $\boldsymbol{e_j}$

$$op(w_i, e_j) = \frac{\text{duration}(w_i \cap e_j)}{\text{duration}(w_i \cup e_j)}$$

3. Method

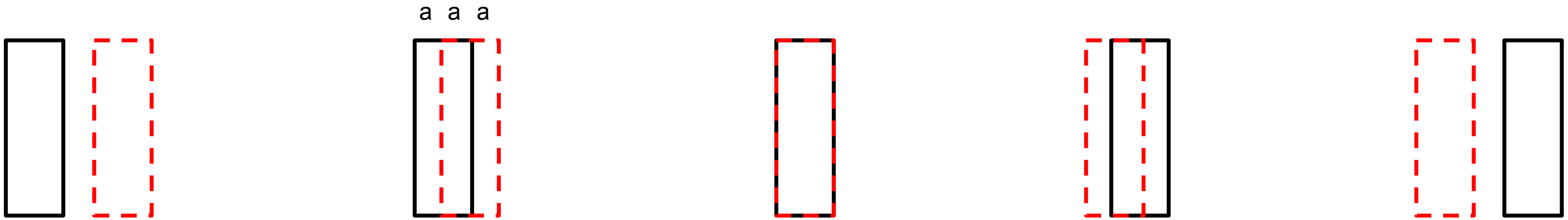
Regression-Based Approach: Overlapping Parameter (op)



Sliding Window $\boldsymbol{w}_i$

Event $\boldsymbol{e}_j$

$$op(w_i, e_j) = \frac{\text{duration}(w_i \cap e_j)}{\text{duration}(w_i \cup e_j)}$$

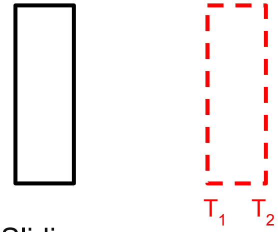


op = *0*

op = *?*

3. Method

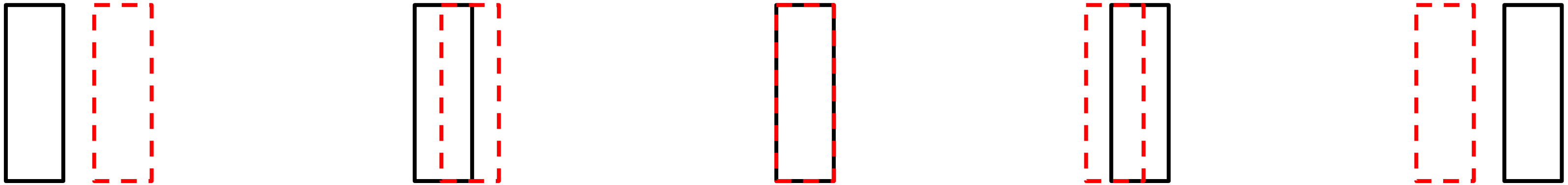
Regression-Based Approach: Overlapping Parameter (op)



Sliding Window $\boldsymbol{w_i}$

Event $\boldsymbol{e_j}$

$$op(w_i, e_j) = \frac{\text{duration}(w_i \cap e_j)}{\text{duration}(w_i \cup e_j)}$$

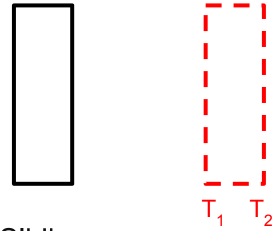


op = *0*

$$\lim_{n\to\infty} n\left(1+\frac{1}{n}\right)^n - n \cdot e + \frac{e}{2} + \frac{1}{3}$$

3. Method

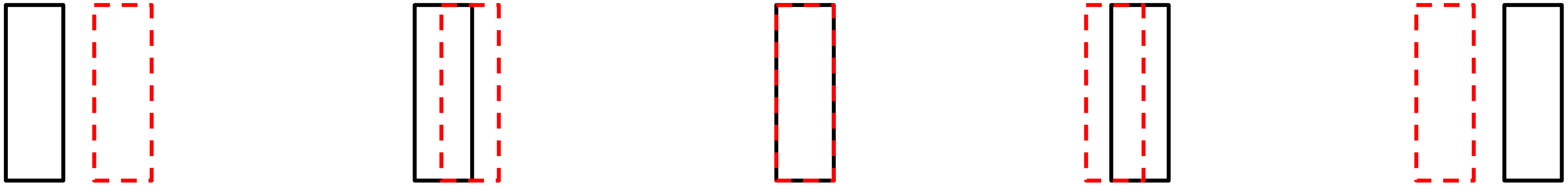
Regression-Based Approach: Overlapping Parameter (op)



Sliding Window $\boldsymbol{w_i}$

Event $\boldsymbol{e_j}$

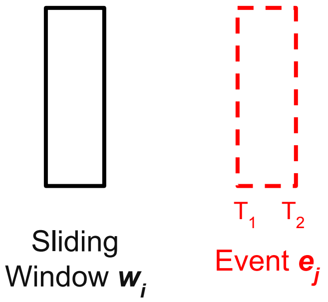
$$op(w_i, e_j) = \frac{\text{duration}(w_i \cap e_j)}{\text{duration}(w_i \cup e_j)}$$



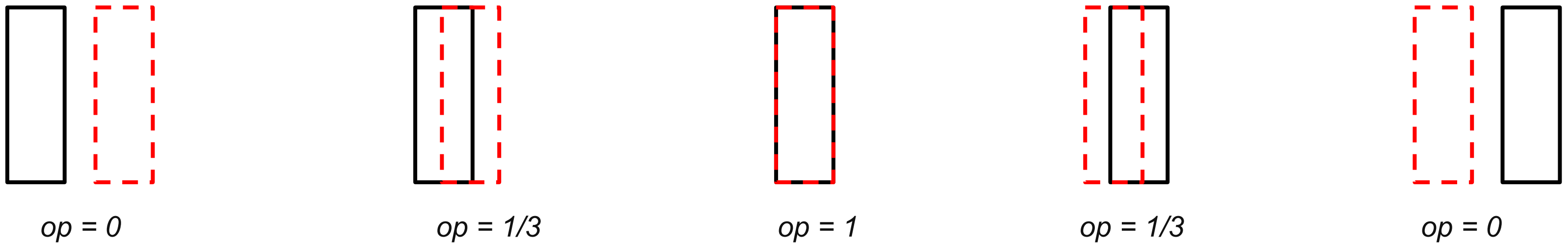
op = 0 *op = 1/3* *op = 1* *op = 1/3* *op = 0*

3. Method

Regression-Based Approach: Overlapping Parameter (op)



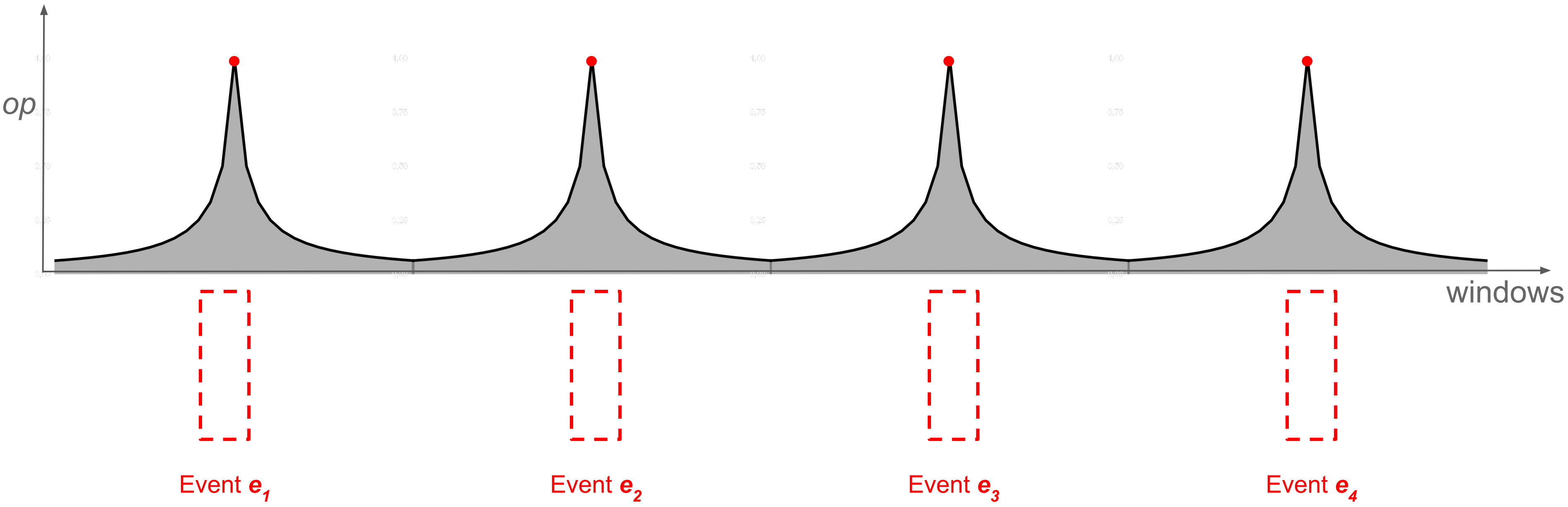
$$op(w_i, e_j) = \frac{\text{duration}(w_i \cap e_j)}{\text{duration}(w_i \cup e_j)}$$



$$op(w_i) = \max_{e_j \in E} op(w_i, e_j)$$

3. Method

Regression-Based Approach: Peaks = Events



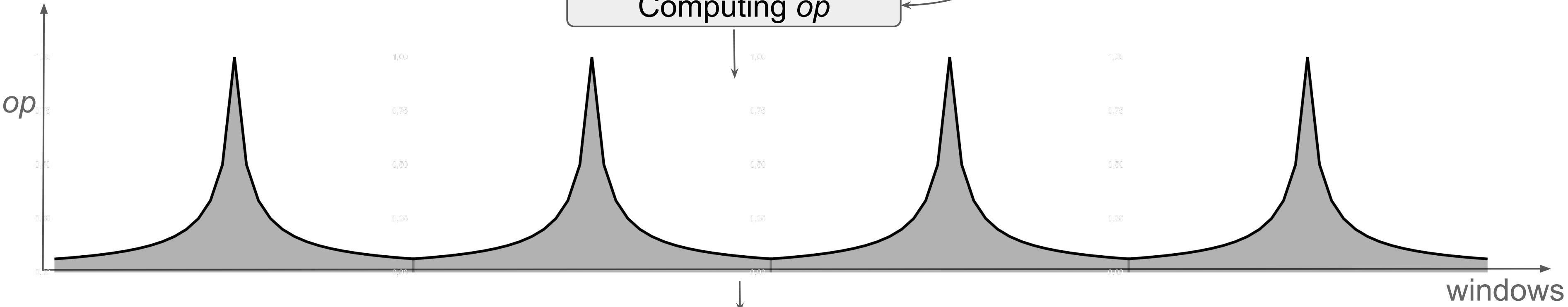
3. Method

Principle of Detection

Time Series → Sliding Windows

Events

Computing *op*



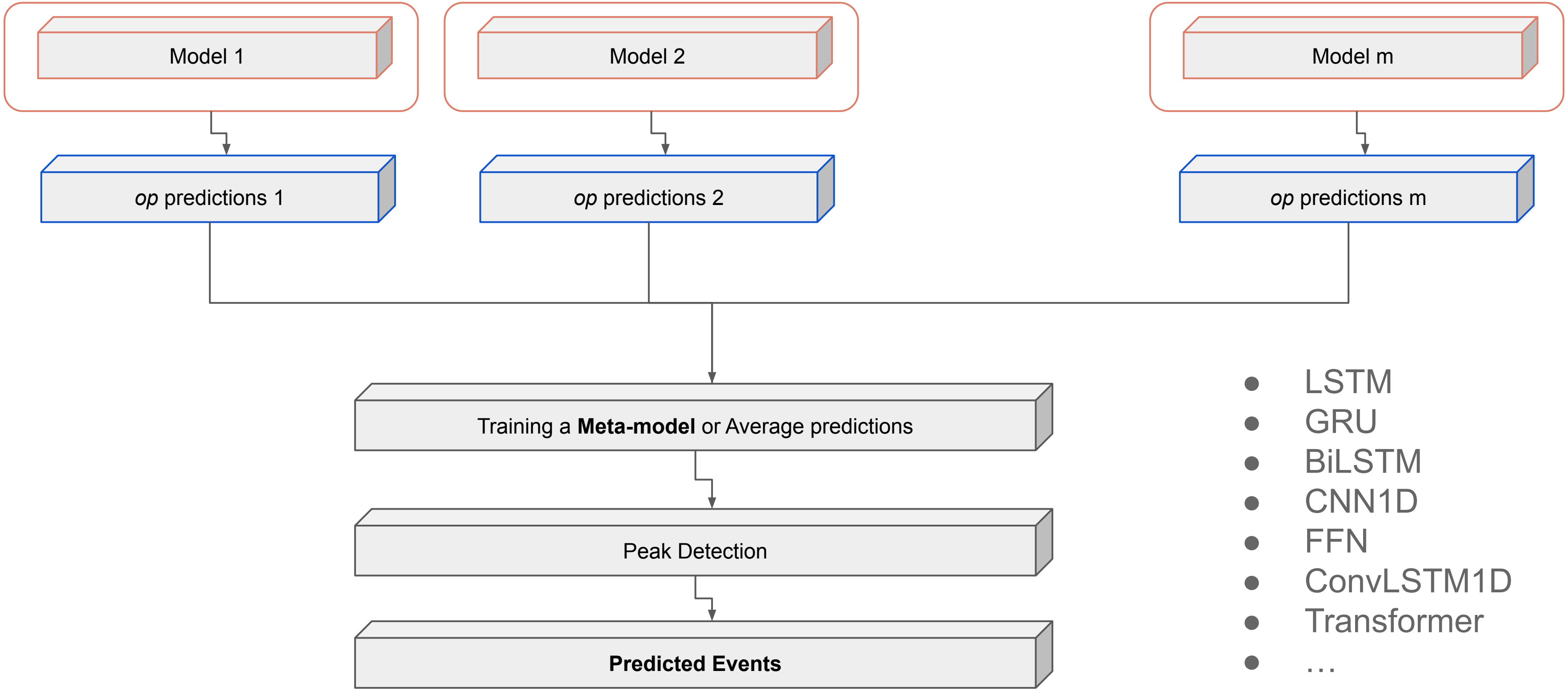
Train a Model to predict *op*

Peak Detection

Predicted Events

3. Method

Stacking Ensemble Learning



- LSTM
- GRU
- BiLSTM
- CNN1D
- FFN
- ConvLSTM1D
- Transformer
- …

3. Method

In-Depth and Theoretical Discussion

Universal Event Detection in Time Series

Menouar Azib[1] MENOUAR.AZIB@AKKODIS.COM
Benjamin Renard[1] BENJAMIN.RENARD@AKKODIS.COM
Philippe Garnier[2] PHILIPPE.GARNIER@IRAP.OMP.EU
Vincent Génot[2] VINCENT.GENOT@IRAP.OMP.EU
Nicolas André[2] NICOLAS.ANDRE@IRAP.OMP.EU

1: *Akkodis, Blagnac, France*

2: *Institut de Recherche en Astrophysique et Planétologie, CNRS, Université de Toulouse, CNES, Toulouse, France*

Abstract

Event detection in time series data is a crucial task spanning various domains, and extensive research has explored methods to achieve this goal. These methods range from traditional threshold-based techniques to more advanced deep learning approaches. However, a comprehensive survey of existing methods reveals that each approach has its limitations, often

Theorem 6 (The Universal Event Approximation Theorem) *If T and T^{-1} are continuous, there exists a feedforward neural network $u \in \Sigma^r(\Psi)$ that utilizes a squashing function Ψ and can approximate the function f_{op} from $\mathcal{Y}$ to $[0,1]$ with arbitrary precision, given a sufficient number of hidden units Q. Here, $\Sigma^r(\Psi)$ represents a set of single hidden layer feedforward neural networks defined as follows:*

$$\{v : \mathbb{R}^r \to \mathbb{R} : v(x) = \sum_{j=1}^{Q} \beta_j \Psi(A_j(x)), x \in \mathbb{R}^r, \beta_j \in \mathbb{R}, A_j \in \mathbf{A}^r\}$$

where $A_j(x) = w_j \cdot x + b_j$, with $w_j \in \mathbb{R}^r$ and $b_j \in \mathbb{R}$. The parameters w_j, b_j, and β_j correspond to the network weights.

Preprint https://doi.org/10.31219/osf.io/uabjg

Submitted to JMLR

4. Python Package

- EventDetector:
  - Github: https://github.com/menouarazib/eventdetector/
  - PyPI: pip install eventdetector-ts
  - Python 3.9+
  - TensorFlow: Already installed

4. Python Package

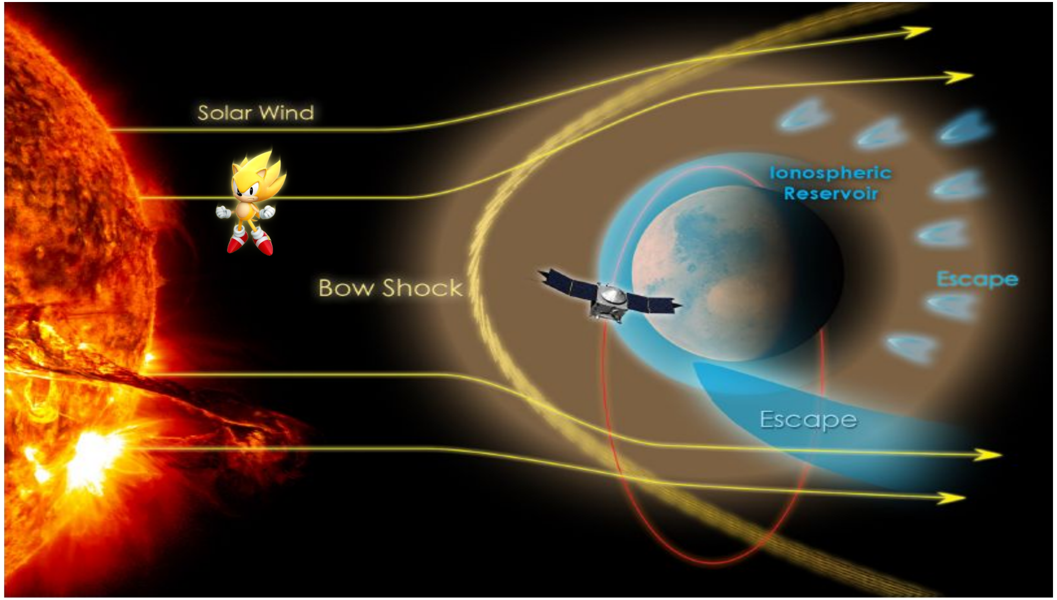
EventDetector

```python
import pandas as pd
from typing import Union

# Time Series
dataset: pd.DataFrame
# Reference Events
events: Union[list, pd.DataFrame]
```

```python
from eventdetector_ts.metamodel.meta_model import MetaModel

meta_model = MetaModel(dataset=, events=, width=,
↪   width_events=, step=, output_dir=)

meta_model.fit()
```

5. Usage Examples

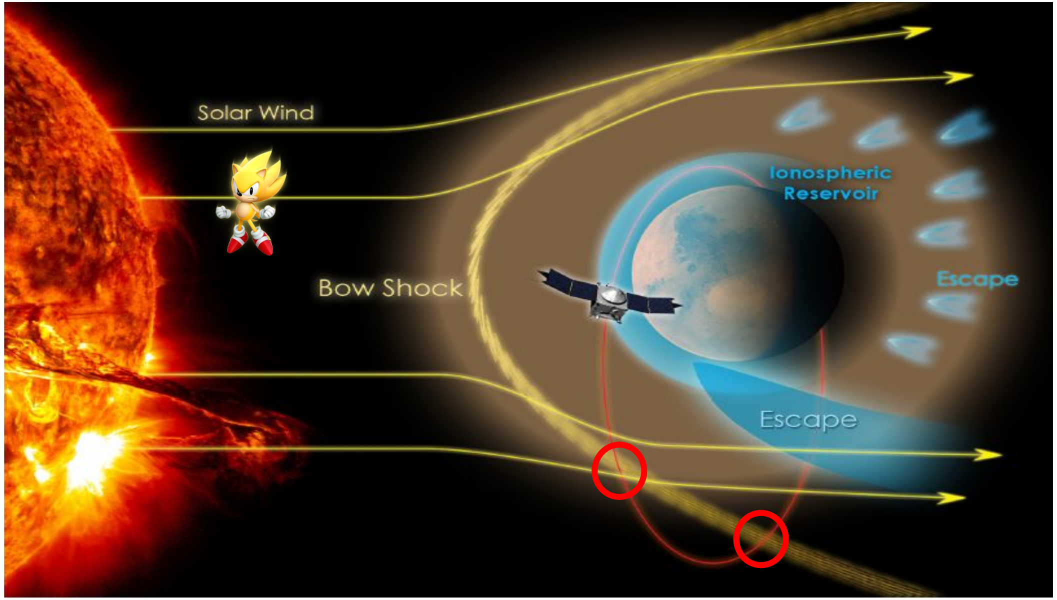
Planetary Science: Martian Bow Shock



Martian bow shock is occured when the supersonic solar wind interacts with the Martian environment, leading to the formation of a shock wave.

5. Usage Examples

Planetary Science: Martian Bow Shock



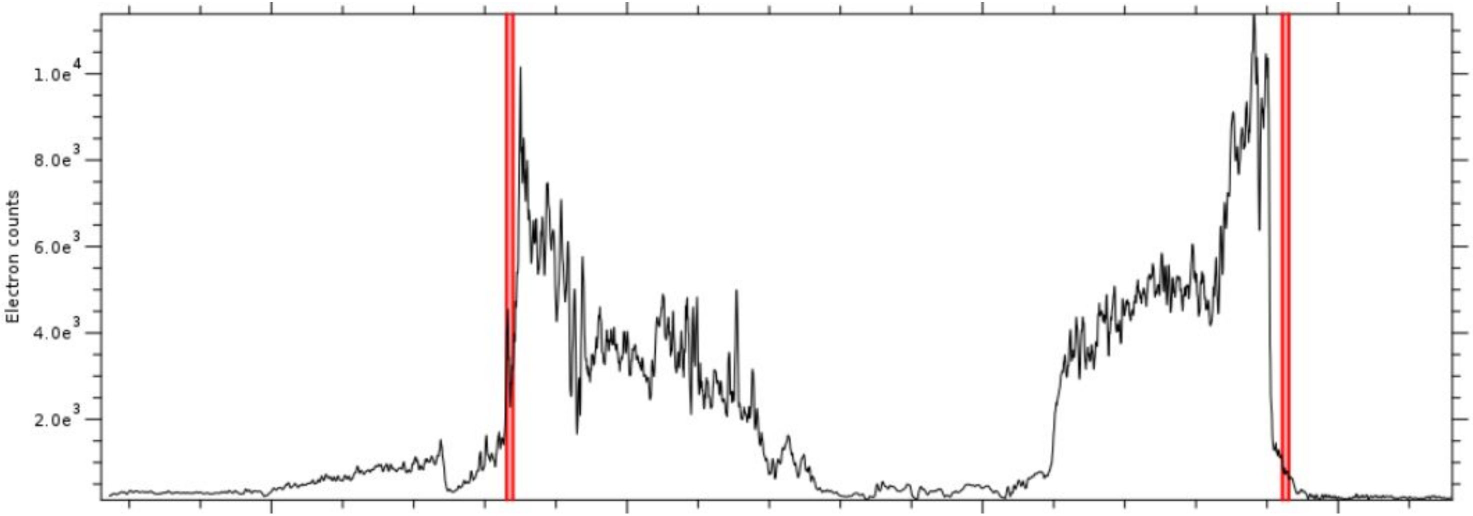
Martian bow shock is occured when the supersonic solar wind interacts with the Martian environment, leading to the formation of a shock wave.

11820 shock crossings by the Mars express spacecraft: https://doi.org/10.1016/B978-012086430-0/50010-5

5. Usage Examples

Planetary Science: Martian Bow Shock

- The dataset represents a time series with a time sampling of 4 second: electron counts, ….



The French national data centre for natural plasmas of the solar system.
http://cdpp.irap.omp.eu/

An on-line database and analysis tool for heliospheric and planetary plasma data.
http://amda.cdpp.eu/

5. Usage Examples

Financial Security: Credit Card Fraud

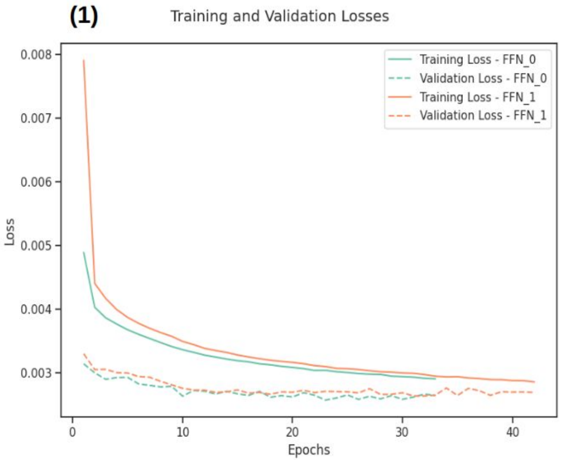
- The dataset represents a time series with a time sampling of 1 second, comprising 492 frauds (events) out of 284,807 transactions.

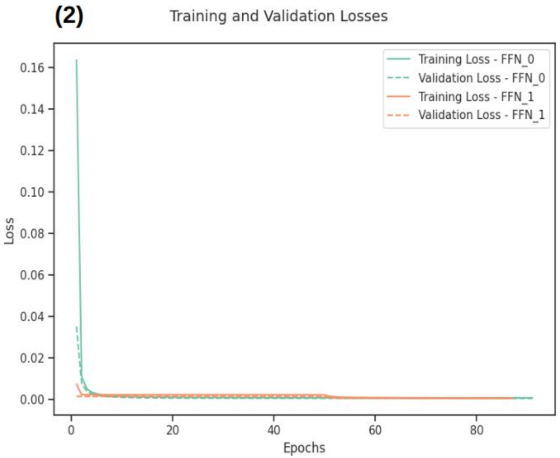
https://www.doi.org/10.1016/j.eswa.2014.02.026

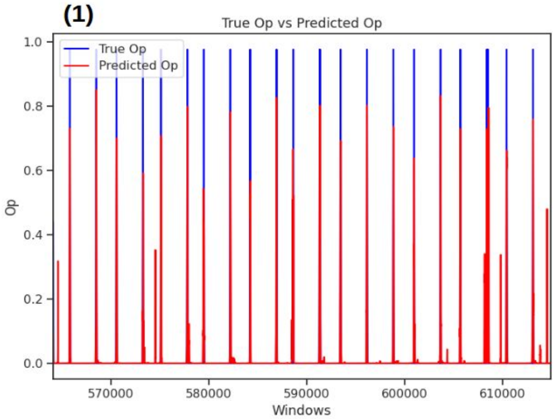
```python
from eventdetector_ts import load_martian_bow_shock
from eventdetector_ts.metamodel.meta_model import MetaModel

dataset, events = load_martian_bow_shock()

meta_model = MetaModel(dataset=dataset, events=events,
↪  width=45, step=1, output_dir="bow_shocks")

meta_model.fit()
```

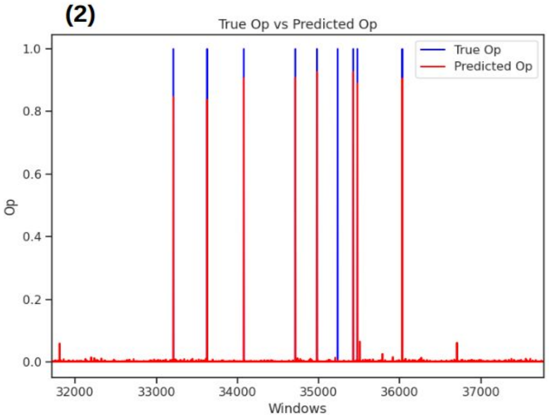
```python
from eventdetector_ts import load_credit_card_fraud
from eventdetector_ts.metamodel.meta_model import MetaModel

dataset, events = load_credit_card_fraud()

meta_model = MetaModel(dataset=dataset, events=events,
↪  width=3, step=1, output_dir='credit_card_fraud')

meta_model.fit()
```

5.1. Losses + op Predictions

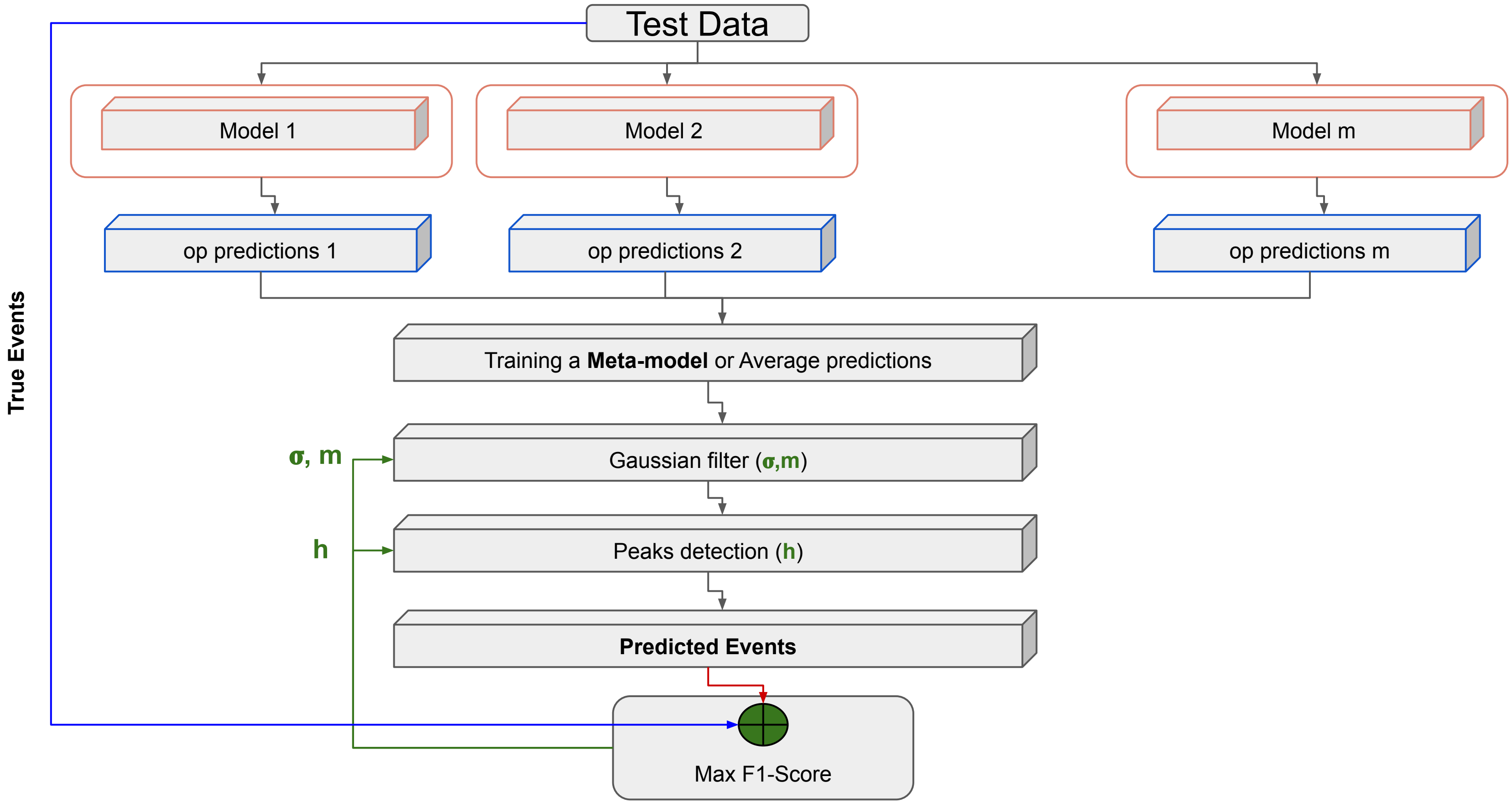


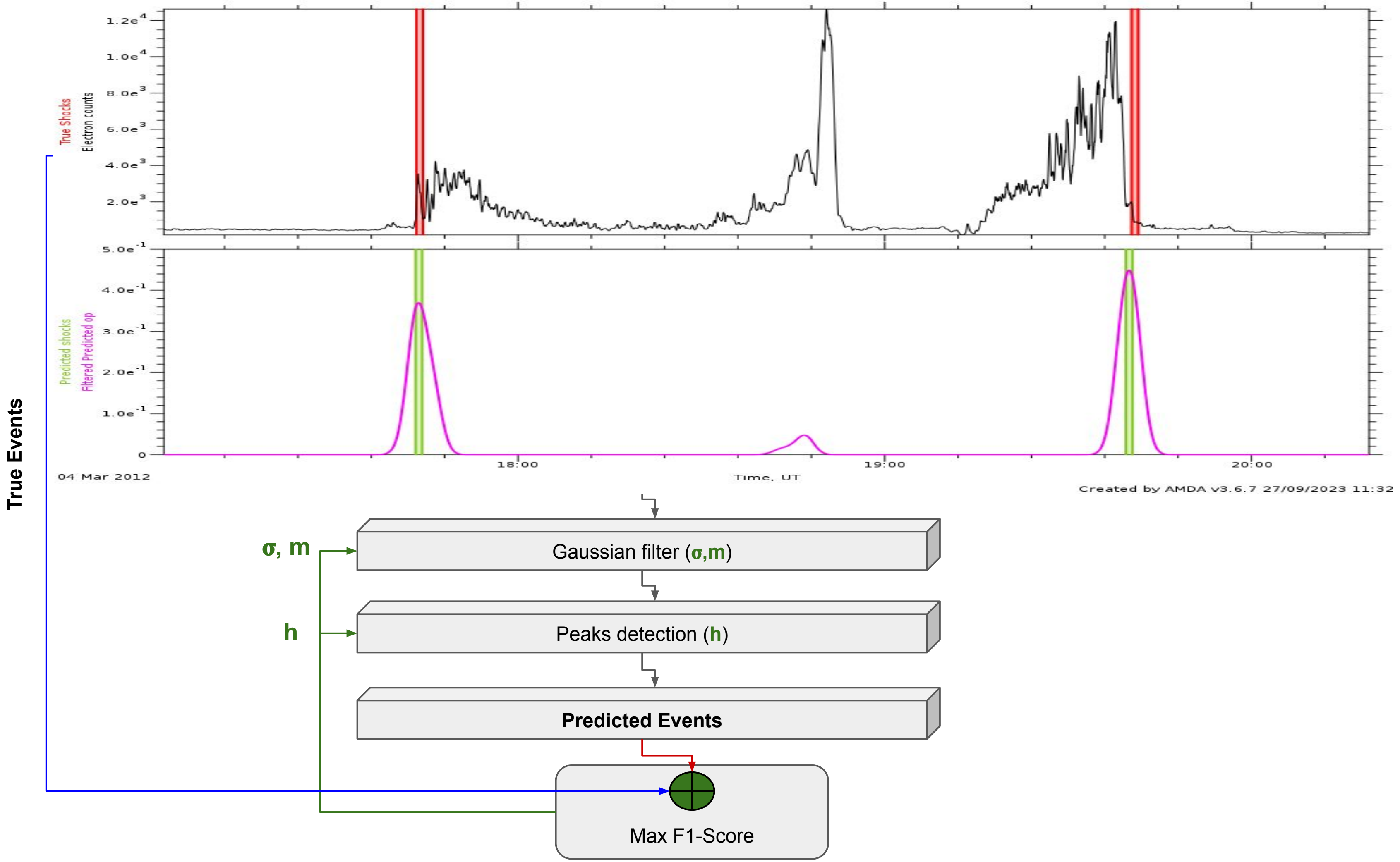






5.2 Optimization




True Shocks
Electron counts
Predicted shocks
Filtered Predicted op
04 Mar 2012
18:00
19:00
20:00
Time, UT
Created by AMDA v3.6.7 27/09/2023 11:32
True Events
σ, m
h
Gaussian filter (σ,m)
Peaks detection (h)
Predicted Events
Max F1-Score

5.3 Results

Data set	F1-Score	Precision	Recall
Martian bow shock	0.9021	0.9455	0.8626
Credit card fraud	0.8372	0.9643	0.7397

- Literature f1 scores:
  - Bow shock
    - f1 scores: [0.90, 0.92]
    - https://doi.org/10.3389/fspas.2022.1016453
  - Credit card fraud
    - f1 scores: [0.80, 0.86]
    - https://doi.org/10.11591/ijeecs.v21.i3.pp1704-1712
- Results:
  - Consistent metrics
  - Represent a **baseline**, and further fine-tuning of the package has the potential to enhance its performance even further

6. Conclusion

- Novel deep-learning supervised method
  - Regression-Based Approach
  - No Need for Time-Step Labeling
  - Stacked Ensemble Learning
  - Practical Implementation

- Python Package
  - Available on PyPI
  - Easy to use
  - Well documented

- Objectives and Results
  - Promising metrics
  - **Primary objective** is not to attain state-of-the-art metrics rather than affirming its adaptability across various domains
  - To obtain state-of-the-art metrics, a deeper exploration of configurations, including stacked models, the meta-model, and sliding windows, is recommended

- In-Depth and Theoretical Discussion
  - Paper: Universal Event Detection in Time Series

Thank you for your attention.
Do you have any questions?

e_j: represents the occurrence of event *j* defined by starttime ***s*** and endtime ***e***

e_j= [***(s+e)/2*** - width_events, ***(s+e)/2*** + width_events]

width_events = width of sliding window as default

t_q: represents the mid-time of the *q*-th peak

$$e_q = \left[t_q - \frac{width\_events}{2}, t_q + \frac{width\_events}{2}\right]$$