Datastream project: weather and battles in Ukraine with Kafka

Dimitri Henrard-Iratchet, Tristan Waddington & Fabien Lagnieu

23 January 2025

Abstract

The aim of our project was to determine whether it be possible to predict the number of incidents on the Ukrainian battlefields, based solely on a stream of weather forecasts for the region. We used a Kafka topic to interrogate WeatherAPI and then feed the meteorological data to a handful of CapyMOA models. [1][2][3][4]

Our conclusion is that the number of battles can probably not be inferred from such data. Indeed, our best models only achieved slightly better results than our statistical baseline. The present report provides details about our approach and discusses its results. Our code can be found on GitHub. [5]

1 Description of our approach

1.1 Aim, features and initial target

The weather plays a crucial role in warfare. By influencing visibility, manoeuvrability or the morale of soldiers, it can partly decide the outcome of battles. Hence, weather forecasts play an equally crucial role in military planning. But to what extent does that make battlefield events predictable? If a clear pattern could emerge, it could in turn be used by opposing forces to improve their tactical thinking.

In order to answer this question, we interrogated, besides WeatherAPI, the database on political violence maintained by ACLED. [6]

WeatherAPI provides a whole range of features on a hourly basis, making it particularly suitable for use with Kafka. A small sample can be seen on figure 1. We chose to focus on those features most relevant to warfare:

Feature	**Metric**	**Relevance**
Temperature	°C	Mobility and morale
Wind	kph	Availability of air support, precision of artillery
Precipitation	mm	Mobility and morale
Snow	cm	Mobility and morale
Visibility	km	Efficiency of surveillance and long-range weapons
Moonlight	%	Possibility of night battle
Sunrise	time	Daytime available for battle
Sunset	time	””

The last two features, we merged into a single `daylength` one (expressed in hours).

As for the target, ACLED’s data is event-based and provides a whole range of different types of events, from violent protests to exactions against civilians. We

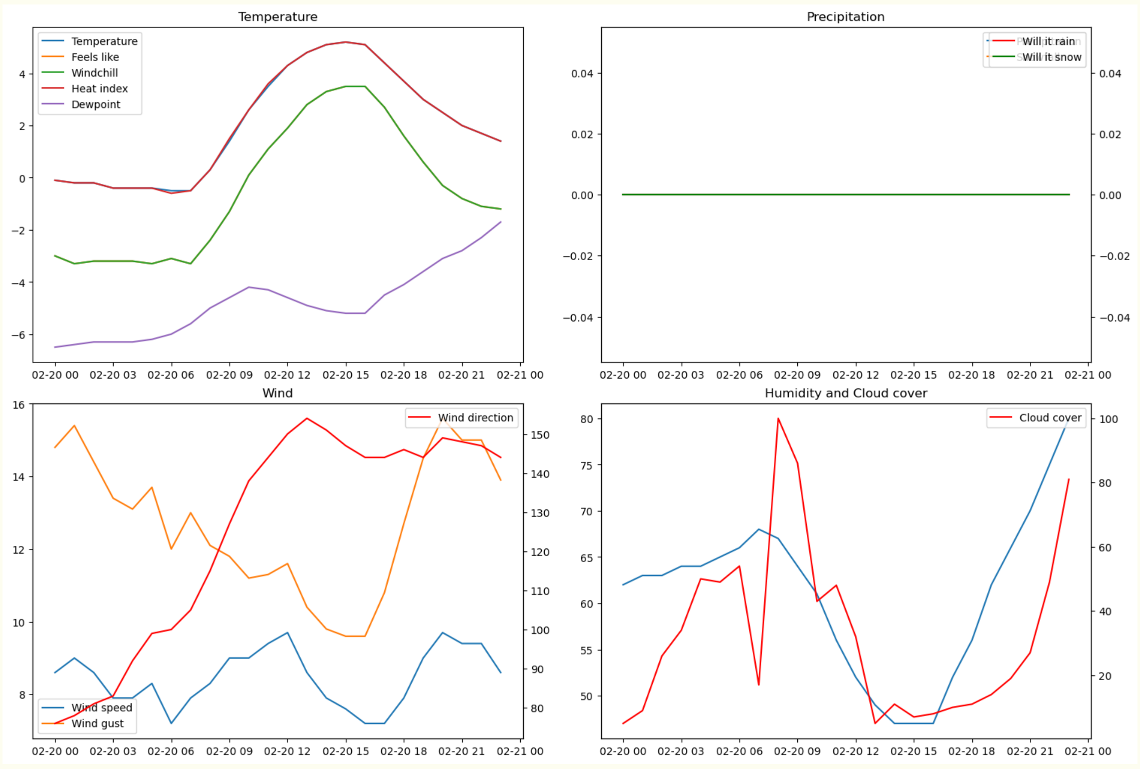


Figure 1: A small sample of WeatherAPI numerous features: different temporal patterns can be observed.

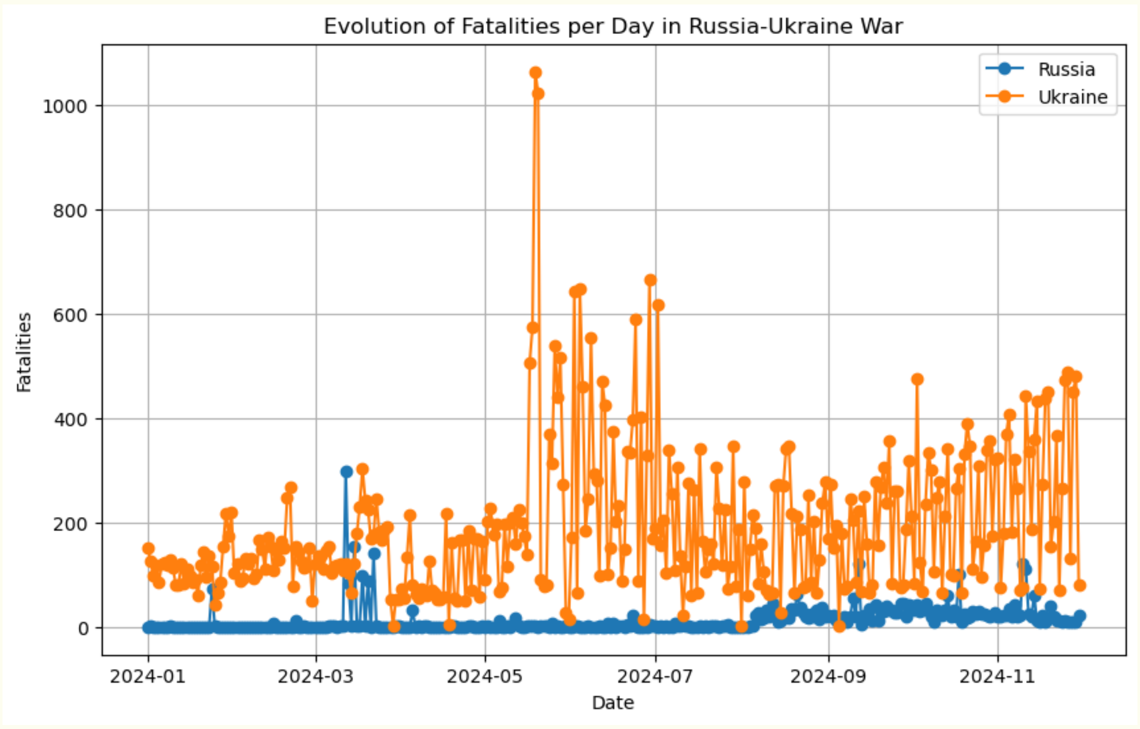


Figure 2: Estimated fatalities in 2024: the Russian losses are notoriously hard to estimate.

chose to focus solely on ground battles in Ukraine, i.e. direct clashes between Russian and Ukrainian troops. We ruled out using the data relative to shelling and bombing, as it is inherently less localised, hence impossible to predict through local weather forecasts. Moreover, as can be seen in figure 2, the estimations of fatalities are highly unreliable, especially regarding Russian forces. Hence we targeted the number of battle events themselves, rather than the estimated fatalities.

1.2 Limitations in the databases and refinement of features and target

Although WeatherAPI proposes hourly data, ACLED's database only provides the date of the events. Therefore we decided to work with daily features.

This led to a second limitation, on WeatherAPI side: our basic plan didn't allow us to interrogate the database more than 365 days in the past. Our initial approach to work on a macro level, summing up battles in Ukraine and neighbouring Russia and using multiple features from different weather stations in the region, was thus unusable for lack of data. Hence we switched to working oblast by oblast[1]. We picked the five oblasts presenting the most battles during the period:

Oblast	**Total number of battles**
Donetsk	11,974
Kharkiv	2,974
Kherson	772
Luhansk	1,208
Zaporizhzhya	1,080

This led to a third and final adaptation: as there is a huge imbalance between Donetsk and the other oblasts, we changed our approach, from predicting the

[1]'Oblast' is the designation of Ukraine's high-level administrative subdivisions.

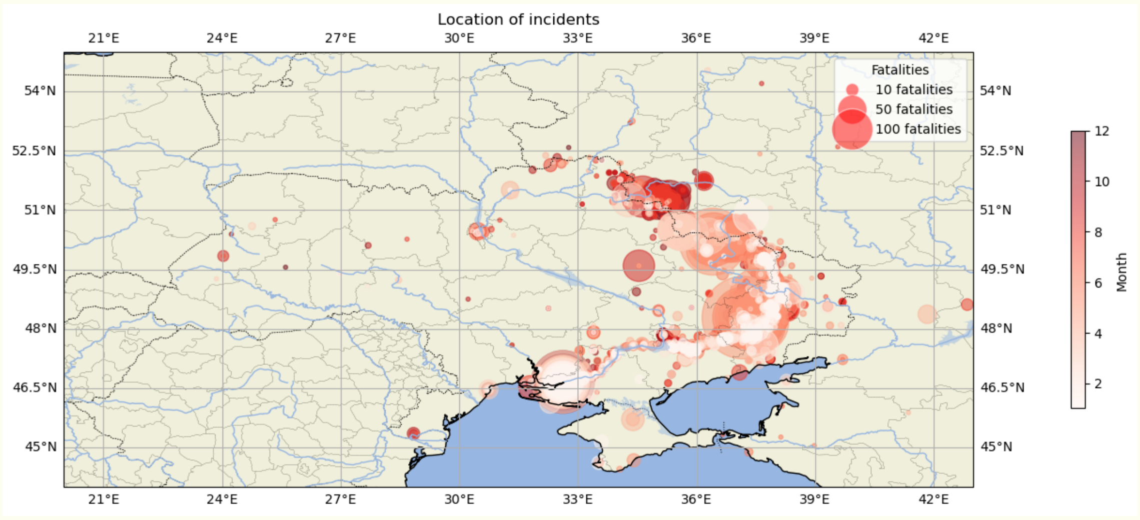


Figure 3: Battle events in 2024: they are mostly concentrated in five eastern oblasts.

raw number of battles in any given oblast on any given day, to predicting the proportion of that oblast's yearly battles taking place on that day. In other words, for every oblast and every day, we divided the sum of battle events taking place on this day by the yearly total provided in the table above. This constitutes the target our models were trained on.

It could be argued that such an approach would lead to data leakage, if we were to fine-tune our models' hyper-parameters **after** putting their predictions back to scale by multiplying them by the oblasts' totals. However, even though we did do just that in order to analyse our results, we didn't perform any subsequent fine-tuning. Had we chosen to, we could have avoided data leakage by using the grand totals of the previous year instead.

1.3 Pipeline

We wrote a dedicated script[2] to interrogate WeatherAPI, process the data, format it and send it through a Kafka producer. Since the data isn't actually real-time, the question arose: in which order should WeatherAPI be interrogated? Several options existed:

[2]`ingest.py` in the GitHub.

- Pick randomly among the 5 oblasts and 365 days.
- Request all 5 oblasts for the same day, then move on to the next day.
- Request all 365 for the same oblast, then move on to the next oblast.

We chose the third option for its preserving the natural time-dependency of the data and reducing the number and scale of concept drifts: only every 365 days, and from one oblast in January to another in the same season.

In the end, the Kafka producer's messages were formatted as such:

```
{'oblast': 'Donetsk',
'target': 0.001837314180724904,
'features': {'temperature': 13.7, 'wind': 28.4,
             'precipitation': 0.0, 'snow': 0.0,
             'visibility': 10.0, 'moon': 31,
             'daylength': 13.05}}
```

A Kafka consumer then fetched the messages which were processed into `X`, `y` and `oblast` (`X` being a dictionary at this point). Working in python, we had the choice between River and CapyMOA to perform the learning itself. As our attempts with the former consistently gave inconsistent results, we opted for the latter. However, we still used two crucial tools of River.

The scaler. Meteorological data is expressed in a broad range of very different metrics and scales, which can gravely hamper the learning process. `river.preprocessing` provides a `StandardScaler` suited for streams. We used it at this point of the pipeline. [7]

The baseline. `river.dummy` provides a `StatisticRegressor` which, in effect, merely computes the rolling average of the target. We used it as a baseline. [8]

Since CapyMOA doesn't yet provide seamless integration with Kafka, at this point we built custom instances which we fed to the regression models we had chosen: stochastic gradient descent (SGD), Fast Incremental Model Tree with Drift Detection (FIMTDD), Online Regression Tree with Options (ORTO) and Self-Optimising K-Nearest Leaves (SOKNL and SOKNLBT)[3]. Each model was used with its standard hyper-parameters. 1825 instances were fed to them, as we didn't want to feed them twice the same data and provoke overfitting.

2 Results analysis

2.1 Results processing and visualisation

The recourse to custom instances prevented us from using CapyMOA's `prequential_evaluation_multiple_learners` and similar tools. Moreover, for better interpretability, we opted for rescaling the predictions as explained above. This implied the need to separate the predictions oblast by oblast, thus reducing the value-added of CapyMOA's built-in evaluation tools anyway.

As is apparent on figure 4, the models performed very differently, with ORTO and FIMTDD varying widely in their predictions, and both SOKNL and SOKNLBT being much more conservative.

We used absolute error (AE) to determine the quality of the predictions, for this metric is directly interpretable. Moreover, we compared each model's AE with that of the baseline (we computed the ratio). As can be seen on figure 5, the scale of the peaks vary greatly from one model to another, with SOKNL and SOKNLBT performing the best.

Finally, we computed the mean absolute error (MAE) for each model and each oblast. We also computed the standard deviation of the numbers of battles in

[3]We also experimented with Passive Aggressive and Shrubs, which produced aberrant results.

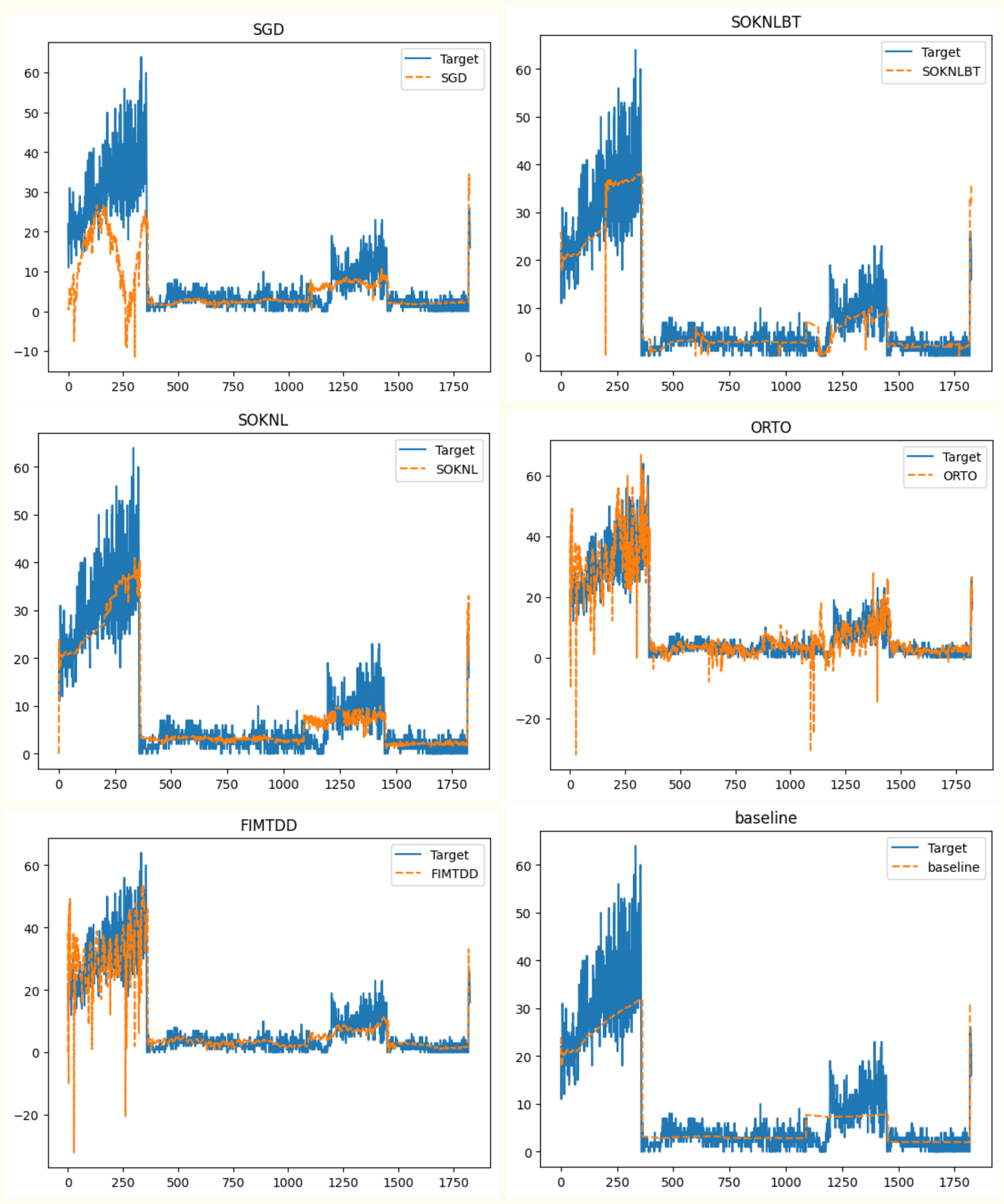


Figure 4: Model predictions vs target: some models may be too conservative, but the others seem erratic.

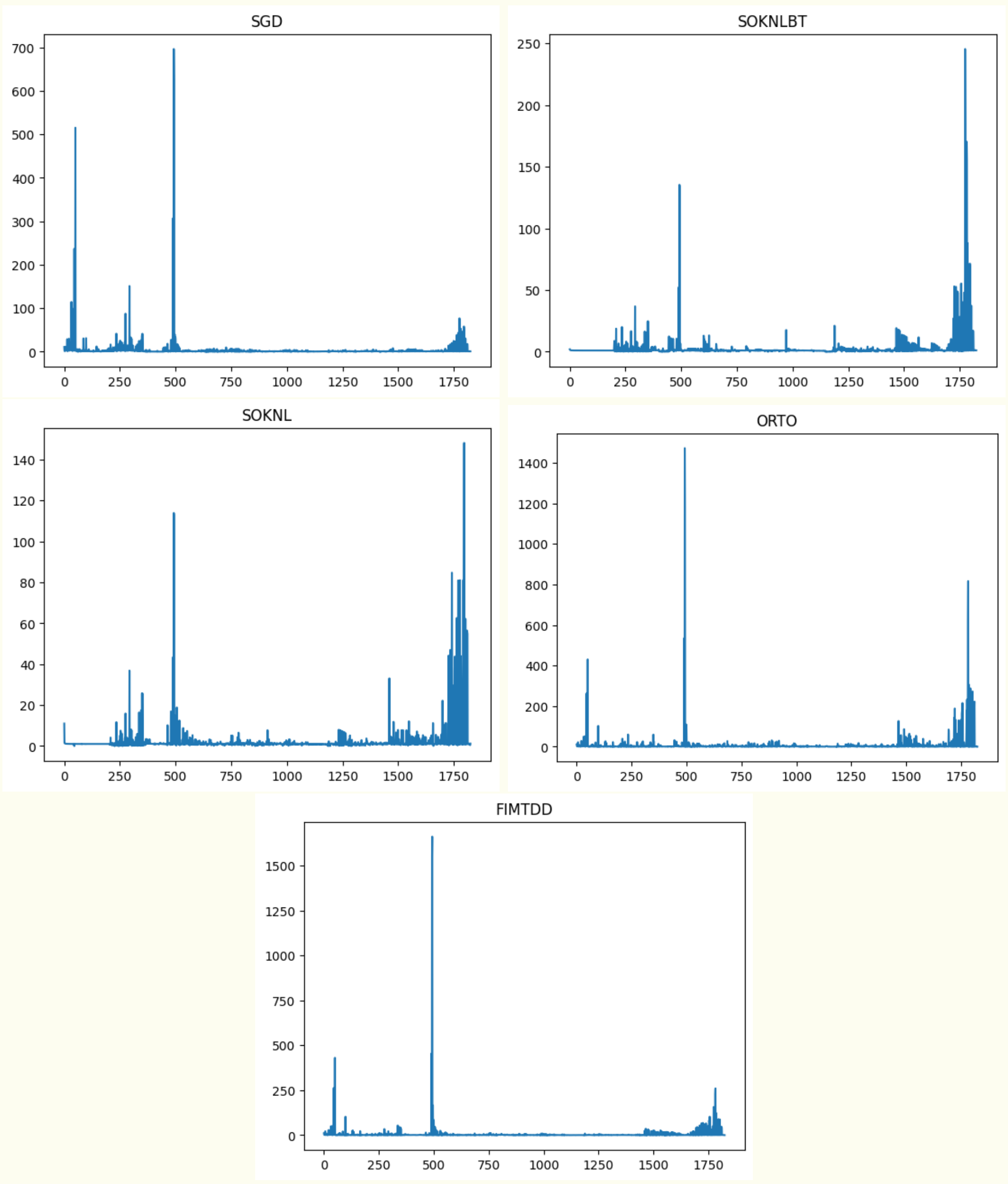
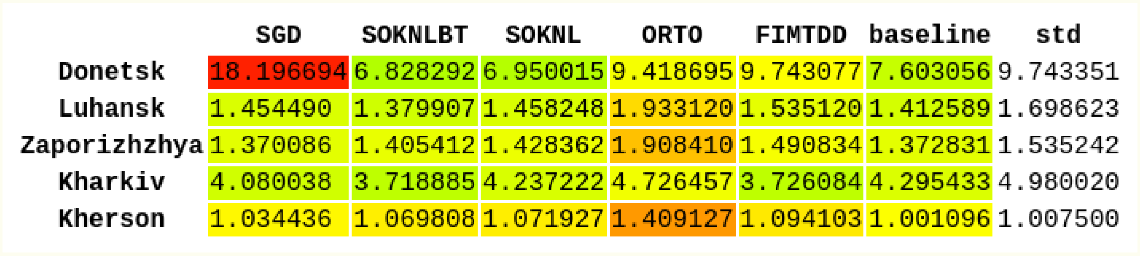


Figure 5: Ratio model-AE / baseline-AE: ORTO has the worst AE.



	SGD	SOKNLBT	SOKNL	ORTO	FIMTDD	baseline	std
Donetsk	18.196694	6.828292	6.950015	9.418695	9.743077	7.603056	9.743351
Luhansk	1.454490	1.379907	1.458248	1.933120	1.535120	1.412589	1.698623
Zaporizhzhya	1.370086	1.405412	1.428362	1.908410	1.490834	1.372831	1.535242
Kharkiv	4.080038	3.718885	4.237222	4.726457	3.726084	4.295433	4.980020
Kherson	1.034436	1.069808	1.071927	1.409127	1.094103	1.001096	1.007500

Figure 6: Overall performance of the models: SOKNL and SOKNLBT are the only ones on par with the baseline.

each oblast. Figure 6 shows how ORTO was the worst-performing overall, while SGD had a hard time dealing with Donetsk in particular[4]. Only SOKNL and SOKNLBT managed to perform slightly better than our rolling average baseline.

2.2 Conclusion and further research

Such poor predictive performances do not suggest that there was a pattern to be discovered in the first place. Hence our conclusion is that the number of battles can probably not be inferred from the weather, despite the latter's importance in combat. We propose two main explanations to that fact:

- The use of different equipments and tactics adapted to different meteorological conditions, allowing different types of battle but with a constant intensity.
- The attempts, on both sides, to use presumably unfavourable weather to their advantage, either by provoking surprise or by exploiting the enemy's assessed inferior adaptability to adapt to such weather.

However, if one would like to investigate further, we propose the following avenues of approach:

- Explore different theatres, such as Gaza, Sudan, etc.

[4]We repeated the training several times and systematically obtained a similar result.

- Invest in a better paid plan on WeatherAPI in order to access data more than one-year old.
- Methodically fine-tune hyper-parameters of the models.

References

[1] URL: https://kafka.apache.org/.

[2] URL: https://www.weatherapi.com/.

[3] URL: https://capymoa.org/.

[4] URL: https://github.com/adaptive-machine-learning/CapyMOA.

[5] *Datastream project: weather and battles in Ukraine with Kafka*. URL: https://github.com/diratche/datastream/tree/main.

[6] *Armed Conflict Location and Event Data*. URL: https://acleddata.com/.

[7] URL: https://riverml.xyz/dev/api/preprocessing/StandardScaler/.

[8] URL: https://riverml.xyz/dev/api/dummy/StatisticRegressor/.

A Regression models used in this paper

The descriptions below are taken from CapyMOA's documentation.

A.1 SGD

https://capymoa.org/api/modules/capymoa.regressor.SGDRegressor.html

Streaming stochastic gradient descent regressor.

This wraps sklearn.linear_model.SGDRegressor for ease of use in the streaming context. Some options are missing because they are not relevant in the streaming context. Furthermore, the learning rate is constant.

A.2 FIMTDD

https://capymoa.org/api/modules/capymoa.regressor.FIMTDD.html

Implementation of the FIMT-DD tree as described by Ikonomovska et al.

Fast Incremental Model Tree with Drift Detection is the regression version for the famous Hoeffding Tree for data stream learning.

FIMT-DD is implemented in MOA (Massive Online Analysis) and provides several parameters for customization.

Reference:

Ikonomovska, Elena, João Gama, and Sašo Džeroski. Learning model trees from evolving data streams. Data mining and knowledge discovery 23.1 (2011): 128-168.

A.3 ORTO

https://capymoa.org/api/modules/capymoa.regressor.ORTO.html

Implementation of the Online Regression Tree with Options (ORTO).
ORTO is an extension to FIMT-DD that allows options during the tree's growth and splits.

Reference:
Ikonomovska, Elena, Joao Gama, Bernard Zenko, and Saso Dzeroski. "Speeding-up hoeffding-based regression trees with options." In Proceedings of the 28th International Conference on Machine Learning (ICML-11), pp. 537-544. 2011.

A.4 SOKNL

https://capymoa.org/api/modules/capymoa.regressor.SOKNL.html

Self-Optimising K-Nearest Leaves (SOKNL) Implementation.
SOKNL extends the AdaptiveRandomForestRegressor by limiting the number of base trees involved in predicting a given instance. This approach overrides the aggregation strategy used for voting, leading to more accurate prediction in general.
Specifically, each leaf in the forest stores the sum of each feature and builds the "centroid" upon request. The centroids then are used to calculate the Euclidean distance between the incoming instance and the leaf. The incoming instance gets the aggregation from k trees with closer leaves as the final prediction. The performances of all possible k value are accessed over time and next prediction takes the best k so far.

Reference:
Sun, Yibin, Bernhard Pfahringer, Heitor Murilo Gomes, and Albert Bifet. "SOKNL: A novel way of integrating K-nearest neighbours with adaptive random forest regression for data streams." Data Mining and Knowledge Discovery 36, no. 5

(2022): 2006-2032.

A.5 SOKNLBT

https://capymoa.org/api/modules/capymoa.regressor.SOKNLBT.html

The base tree for Self-Optimising K Nearest Leaves as distribed by Sun. at el. SOKNLBT modifies the FIMT-DD algorithm to store information at the leaves that allows SOKNL to calculate the distance between a given instance and a leaf.

Reference:

Sun, Yibin, Bernhard Pfahringer, Heitor Murilo Gomes, and Albert Bifet. “SOKNL: A novel way of integrating K-nearest neighbours with adaptive random forest regression for data streams.” Data Mining and Knowledge Discovery 36, no. 5 (2022): 2006-2032.