Lab 1 - Data Stream Processing

Godefroy Du Chalard, Guillaume Philippe

December 2022

1 Introduction

For streaming data, classical machine learning approaches do not work well. Thus, it was necessary to develop specific models for this data. In 2021, a Python library, named River [5], is introduced with the objective of providing an intuitive API to train models with streaming data.

The objective of this lab is to test different classifiers implemented in River on an artificial data stream. We used the Forest Covertype dataset, which we have described below.

2 Dataset

The Forest Covertype dataset was developed by Jock A. Blackard and Colorado State University. It contains the forest cover type for 30 x 30 meter cells obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. It contains 581,012 instances and 54 attributes, and it has been used in several papers on data stream classification.

It has 54 attributes corresponding to 10 quantitative variables, 4 binary wilderness areas and 40 binary soil type variables.

3 Metrics

In this section, we present the metrics that we will use for the experiments.

Accuracy is very common metric to measure the performance of a model. It reports the number of correct predictions (i.e. true positive and true negative predictions) over the total number of predictions (i.e true positive, true negative, false positive and false negative predictions) of the classifier. The formula is given below :

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

The second metric is *Cohen Kappa*, it's a statistic that is used to measure inter-rater reliability for qualitative (categorical) items. It is generally thought to be a more robust measure than simple percent agreement calculation, as it's takes into account the possibility of the agreement occurring by chance.

Name	Data Type	Measurement	Description
Elevation	quantitative	meters	Elevation in meters
Aspect	quantitative	azimuth	Aspect in degrees azimuth
Slope	quantitative	degrees	Slope in degrees
Horizontal_Distance_To_Hydrology	quantitative	meters	Horz Dist to nearest surface water features
Vertical_Distance_To_Hydrology	quantitative	meters	Vert Dist to nearest surface water features
Horizontal_Distance_To_Roadways	quantitative	meters	Horz Dist to nearest roadway
Hillshade_9am	quantitative	0 to 255 index	Hillshade index at 9am, summer solstice
Hillshade_Noon	quantitative	0 to 255 index	Hillshade index at noon, summer soltice
Hillshade_3pm	quantitative	0 to 255 index	Hillshade index at 3pm, summer solstice
Horizontal_Distance_To_Fire_Points	quantitative	meters	Horz Dist to nearest wildfire ignition points
Wilderness_Area (4 binary columns)	qualitative	0 or 1	Wilderness area designation
Soil_Type (40 binary columns)	qualitative	0 or 1	Soil Type designation
Cover_Type (7 types)	integer	1 to 7	Forest Cover Type designation

Table 1: Forest Covertype attributes (for qualitative data type, 0 and 1 mean presence and absence respectively)

Next, *Rolling* is a wrapper metric that allows you to apply a metric over a window of observations. Under the hood, a buffer with the window size most recent pairs of ($\hat{y}$, y) is memorised. When the buffer is full, the oldest pair is removed and the revert method of the metric is called with said pair. In the script, *Rolling* is apply with the *Accuracy* and *Cohen Kappa* metrics over a window size of 50000. It is important to remark that we will report *Rolling* metrics only over the window size, it means that each rolling are mutually exclusive (no overlapping).

4 Predict & Train

In this section, we present our method to evaluate and train the model.

First and foremost, we need a function to iter our dataset to create an artificial data stream. Hopefully, River already implement one, named *iter_pandas*, which able us to iter a dataframe from pandas. Thus, we were able to iterate each sample in the dataset one by one.

Secondly, we use our model in order to predict the class of the sample. The prediction $\hat{y}$ is compared to the true value y, and the set of metrics presented in the section above is updated. Occasionally, we save the score of the metrics in an array and use it for the figures at the end.

Thirdly, the sample is used to train the model. We call the *learn_one* function, that train our model on one sample and which update the model's state.

Fourth, back at the second step, the process is repeated for each sample. When all samples have been processed, the model training is complete and we can return the final metrics.

In short, the model is trained sample by sample. Moreover, each sample is used to predict and train the model. The prediction is of course done before the training. This is important because otherwise the model result would be biased because it would have already seen the sample. Below the pseudocode of the predict & train function :

```
Data: samples
for samples do
    predict_one(sample)
    update metrics
    learn_one(sample)
end
```

Algorithm 1: Predict and Train

5 Classifiers & Results

5.1 KNNClassifier

The K-Nearest Neighbors algorithm (KNN) considers the K closest observed data from the training set to decide on a class for an unlabeled data. K is a parameter chosen by the user.

For our experiments, we use default hyperparameters, including the number of nearest neighbors, K, to 5. Documentation of the model on River is available here.

Below, the evolution of the metrics as a function of the number of iterations for this model. We obtain very good scores for each metric around 0.9.

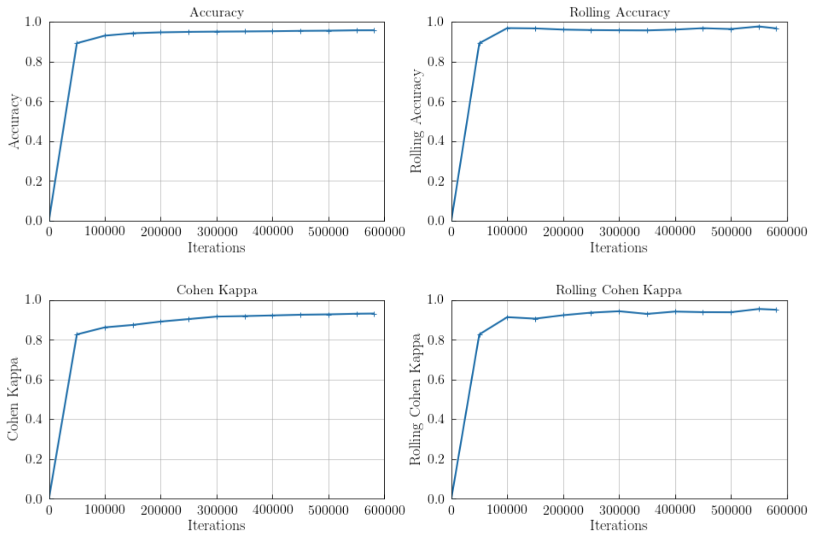

Figure 1: KNNClassifier results

5.2 GaussianNB

Any naive bayes approach including Gaussian Naive Bayes depends on the Bayes Theorem. Bayes theorem gives us the probability of an event, given that we have some extra knowledge about that event.

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)} \tag{Bayes' Formula}$$

In River, there are no hyperparameters for this model. The documentation of the River's implementation is also available here.

Below, the evolution of the metrics as a function of the number of iterations for this model. Results for *Rolling Accuracy* are almost stable around 0.4. Unfortunately, results for *Cohen Kappa* are zero, this means that it attributes all match predictions to chance. This method is rather basic, so it is not surprising to get bad results.

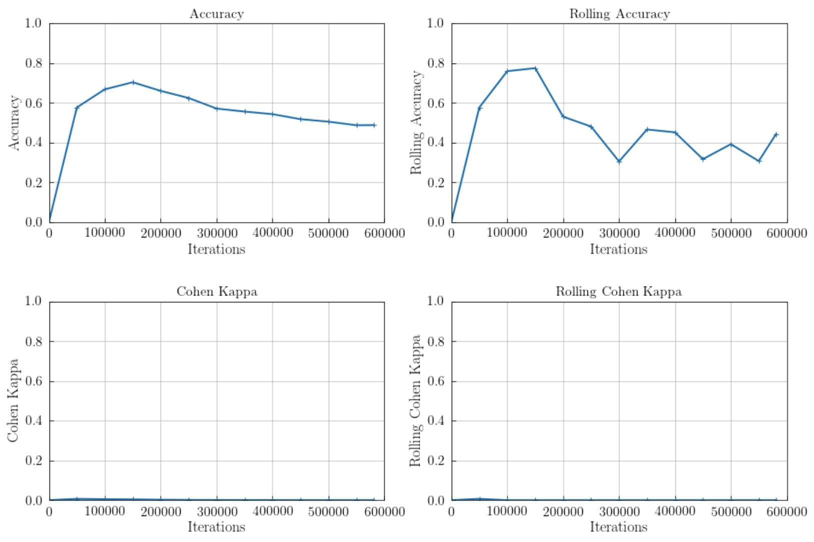

Figure 2: GaussianNB results

5.3 HoeffdingTreeClassifier

HoeffdingTreeClassifier ([4], [1]) is a decision tree learning method for stream data classification. It uses the Hoeffding bound to determine, with high probability, the smallest number, N, of examples needed at a node when selecting a splitting attribute. The Hoeffding bound is independent of the probability distribution, unlike most other bound equations. This is desirable, as it may be impossible to know the probability distribution of the information gain, or whichever attribute selection measure is used.

In order to speed up the training, we decide to set some hyperparameters of these models. Specially, we decide to set the maximum depth a tree can reach to 7 (defaults to None) and the number of instances a leaf should observe between split attempts to 2000 (defaults to 200). Documentation concerning this model is available here.

Below, the evolution of the metrics as a function of the number of iterations for this model. We remark that *Accuracy* and *Cohen Kappa* tend to decrease with the number of iterations, which can be problematic with streaming data. *Rolling Accuracy* looks stable through time, but *Rolling Cohen Kappa* is highly unstable, on the last few windows, the metric is close to 0.1. At 300000 iterations, the metric was at 0.6.

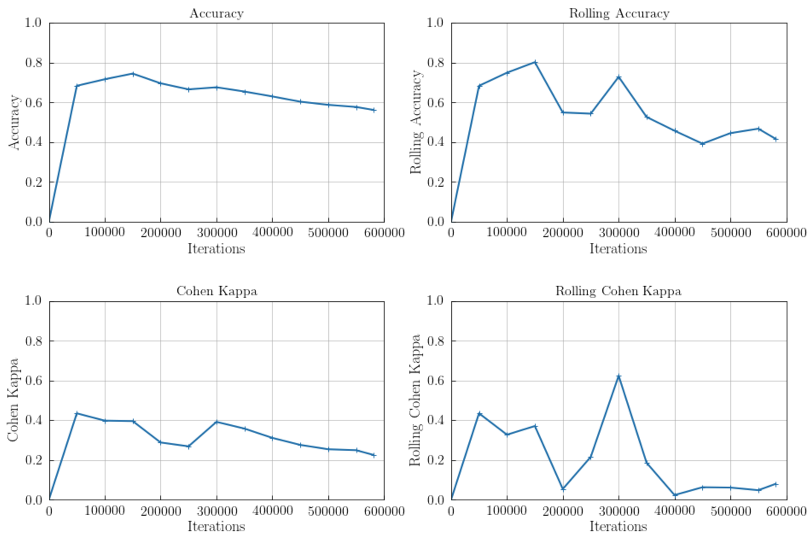

Figure 3: HoeffdingTreeClassifier results

5.4 AdaptiveRandomForestClassifier

AdaptiveRandomForestClassifier (ARFC) [3] is an adaptation of the original Random Forest algorithm, which has been successfully applied to a multitude of machine learning tasks. In layman's terms, the original Random Forest algorithm is an ensemble of decision trees, which are trained using bagging and where the node splits are limited to a random subset of the original set of features. The "Adaptive" part of ARFC comes from its mechanisms to adapt to different kinds of concept drifts, given the same hyperparameters.

Specifically, the 3 most important aspects of the ARFC algorithm are:

- diversity through resampling ("bagging");
- diversity through randomly selecting subsets of features for node splits;
- one drift and warning detector per base tree, which cause selective resets in response to drifts.

As HoeffdingTreeClassifier, we decide to change some defaults hyperparameters. We set the number of trees in the ensemble to 5 (defaults to 10) and the max number of attributes for each node split to logarithm 2 of the number of features (defaults to square root). Documentation can be find here.

Below, the evolution of the metrics as a function of the number of iterations for this model. Overall, this classifier performs very well, even after changing some of its hyperparameters.

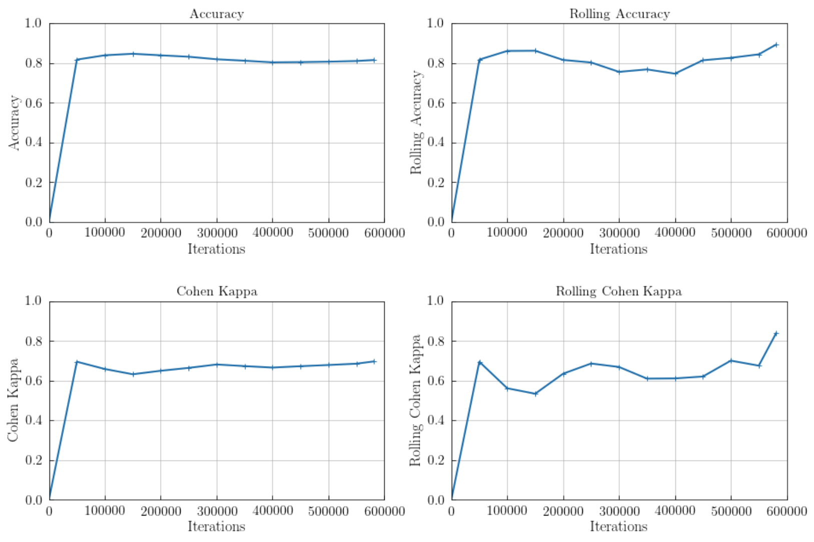

Figure 4: AdaptiveRandomForestClassifier results

5.5 AdaBoostClassifier

AdaBoostClassifier is an ensemble method. The algorithm first trains a HoeffdingTreeClassifier and uses it to make predictions on the streaming data. Then, it increases the relative weights of misclassified instances. Then, it trains a second HoeffdingTreeClassifier using the updated weights, and again make predictions on streaming data, and increases the relative weights of misclassified instances, and so on. The explanations are based on [2].

Also here, we finetune some hyperparameters. We limit the number of trees in the ensemble to 5 (defaults to 10). Moreover, it is require to provide a classifier method, we decide to use an HoeffdingTreeClassifier with the number of instances a leaf should observe between split attempts to 2000, a max depth of 7 and attempt to minimise Gini impurity (default to Information Gain). API interface of this model in River is available here.

Below, the evolution of the metrics as a function of the number of iterations for this model. The results obtained are correct. However, it can be seen that *Accuracy* and *Rolling Accuracy* tend to weaken over time. For the *Rolling Cohen Kappa* metric varies strongly over time, on some windows it is close to 0 and on others close to 0.5.

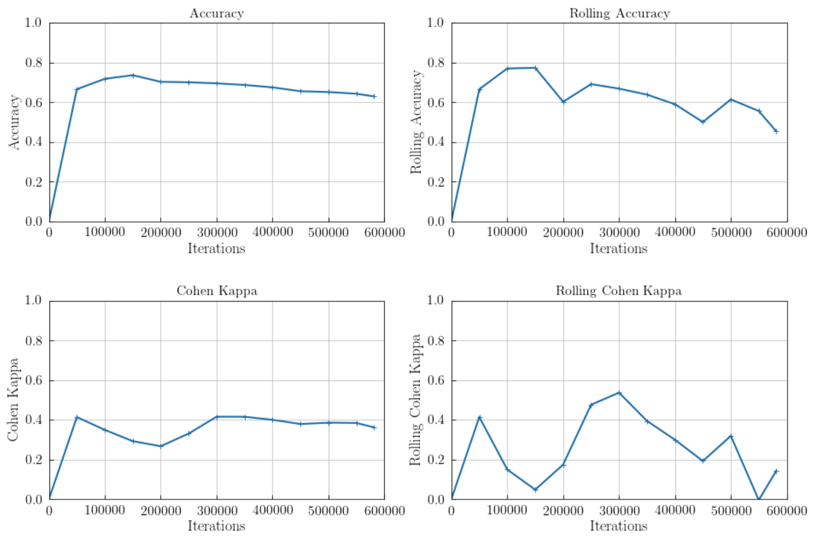

Figure 5: AdaBoostClassifier results

6 Summary

In this section, we briefly show all our final results obtained for the metrics with our models.

Model	*Accuracy*	*Rolling Acc.*	*Kappa*	*Rolling Kappa*
KNNClassifier	**95.74**	**96.70**	**93.16**	**95.09**
GaussianNB	48.76	44.31	0.03	0.00
HoeffdingTreeClassifier	56.17	41.60	22.50	7.95
AdaptiveRandomForestClassifier	79.91	87.31	66.97	80.87
AdaBoostClassifier	57.81	50.95	24.83	16.63

Table 2: Model results with *Accuracy*, *Rolling Accuracy*, *Cohen Kappa*, *Rolling Cohen Kappa* metrics (with a window size of 50000 for *Rolling* metrics).

It is important to remark that we start predicting from the first sample. At the beginning, the model is not trained, therefore metrics are very low. At the end, *Accuracy* and *Cohen Kappa* over all samples are decreased by the predictions at the beginning. This is why it is more representative to choose a rolling metric as it is calculated over windows. Another solution could be to start calculating metrics only after 10000 samples, it will avoid the "cold start" we are encountering.

7 Conclusion

Through this lab, we have seen that River is particularly suited to training models on data streams. The library implements methods ranging from the simplest to the most complex.

On the study of the Cover Dataset, we notice that our models obtain very different results. A "simple" model like K-Nearest Neighbors obtains the best results with a *Rolling Accuracy* of 96.70% and a *Rolling Cohen Kappa* of 95.09% at the end. The many dimensions of the dataset do not seem to disturb this method. However, the Gaussian Naive Bayes method obtains the worst results and the *Cohen Kappa* and *Rolling Cohen Kappa* metrics are both close to 0. As far as the ensemble methods are concerned, the results are not as good as expected given the relative complexity of the methods. This is due to the different hyperparameters that we have set in order to obtain a more efficient training.

An important aspect is the tradeoff between results and performance. In a stream context, the performance of the model is important because we cannot store the data and let it accumulate. Unfortunately, to improve performance we are often forced to reduce the complexity of the model and possibly reduce its performance. We have chosen to focus on performance by using hyperparameters to lighten the training

References

[1] Albert Bifet et al. "MOA: Massive Online Analysis". In: *J. Mach. Learn. Res.* 11 (Aug. 2010), pp. 1601–1604. ISSN: 1532-4435.

[2] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems.* Sebastopol, CA: O'Reilly Media, 2017. ISBN: 978-1491962299.

[3] Heitor M. Gomes et al. "Adaptive Random Forests for Evolving Data Stream Classification". In: *Mach. Learn.* 106.9–10 (Oct. 2017), pp. 1469–1495. ISSN: 0885-6125. DOI: 10.1007/s10994-017-5642-8. URL: https://doi.org/10.1007/s10994-017-5642-8.

[4] Geoff Hulten, Laurie Spencer, and Pedro Domingos. "Mining Time-Changing Data Streams". In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* KDD '01. San Francisco, California: Association for Computing Machinery, 2001, pp. 97–106. ISBN: 158113391X. DOI: 10.1145/502512.502529. URL: https://doi.org/10.1145/502512.502529.

[5] Jacob Montiel et al. "River: machine learning for streaming data in Python". In: (2021).