ESANN 2023 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 4-6 October 2023, i6doc.com publ., ISBN 978-2-87587-088-9. Available from http://www.i6doc.com/en/.

Learning with Boosting Decision Stumps for Feature Selection in Evolving Data Streams

Daniel Nowak Assis

Independent Researcher

Abstract. Feature selection plays an important role in Machine Learning pipelines, and many challenges emerge for feature selection when data arrives continuously as a stream. In this paper, we extend the Adaptive Boosting for Feature Selection (ABFS) algorithm by (i) using a different Online Boosting strategy and (ii) changing the Boosting scaling factor of instances weighting. Results show that our extended ABFS leveraged the predictive performance of classifiers more than the standard ABFS in the most used monolithic classifiers for stream mining.

1 Introduction

Data stream mining imposes different challenges for machine learning models. Unlike the batch setting, where data is stored and available at any time, in the stream setting, data arrives continuously, one instance at a time, and potentially an infinite amount of instances can arrive. To deal with this scenario, a predictive model must (i) incrementally update with instances, and (ii) process an instance at least as fast as new ones are made available, otherwise data must be discarded, or storing instances will collapse the system due to the lack of memory [1]. Additionally, the assumption that the data is stationary does not hold for the stream setting. Probability properties of data are susceptible to change throughout the stream, a problem known as concept drift [3]. Concept drifts can affect the predictive quality of a model, which must adapt in case of detection.

Feature Selection is a well-known technique in the batch setting, that aims to select a subset of features considered to be relevant, i.e., features that are strongly correlated with the class [4]. In the stream-setting, the set of relevant features can suffer changes throughout the stream, known as feature drift [6] and a streaming feature selection algorithm must track this change to avoid classifiers training with irrelevant features. To cope with this task in the stream-setting, we propose extensions to the Adaptive Boosting for Feature Selection (ABFS) [4] in the boosting training phase, making the instance weighting scaling more sensitive on decision stumps correct and misclassifications.

We make available source codes and additional results on the paper's support website[1].

2 Feature Selection with Boosting Decision Stumps

[1]https://sites.google.com/view/learning-boosting-ds

ESANN 2023 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 4-6 October 2023, i6doc.com publ., ISBN 978-2-87587-088-9. Available from http://www.i6doc.com/en/.

Algorithm 1 ABFS + extensions

```
Input: ensemble size $N$, ensemble of boosting decision stumps $h$, instance $I = (\vec{x}, y)$,
    drift detector $\psi_n$, decision stump candidate $ds_c$, the currently selected subset of
    features $\mathcal{F}'$,
 1: $minPos \leftarrow 1$; $maxPos \leftarrow N$
 2: $correct \leftarrow$ false
 3: $\lambda \leftarrow 1$
 4: $i_{drift} \leftarrow -1$
 5: sort $h$ by $\frac{\lambda_m^{sc}}{\lambda_m^{sc}+\lambda_m^{sw}}$ in ascending order
 6: for $n \leftarrow 1$ to $N$ do
 7:    if $correct$ then
 8:       $pos \leftarrow maxPos$
 9:       $maxPos \leftarrow maxPos - 1$
10:    else
11:       $pos \leftarrow minPos$
12:       $minPos \leftarrow minPos + 1$
13:    end if
14:    if $h_{n/pos}$ $has\ correctly\ classified\ I$ then
15:       $\lambda_m^{sc} \leftarrow \lambda_m^{sc} + \lambda$
16:       $\lambda \leftarrow \lambda(\omega\ \frac{\lambda_m^{sc}+\lambda_m^{sw}}{2 \cdot \lambda_m^{sc}})$
17:       $correct \leftarrow$ true
18:       Update $\psi_n$ with 0
19:    else
20:       $\lambda_m^{sw} \leftarrow \lambda_m^{sw} + \lambda$
21:       $\lambda \leftarrow \lambda(\omega\ \frac{\lambda_m^{sc}+\lambda_m^{sw}}{2 \cdot \lambda_m^{sw}})$
22:       $correct \leftarrow$ false
23:       Update $\psi_n$ with 1
24:    end if
25:    if $\psi_n$ flagged a drift and $i_{drift} = -1$ then
26:       $i_{drift} = n/pos$
27:       break
28:    end if
29:    Remove from $\vec{x}$ the feature selected at $h_{n/pos}$
30: end for
31: if $i_{drift} = -1$ then
32:    Train $ds_c$ with $I$ and weight $\lambda$
33:    if $ds_c$ has selected a feature $f_\alpha$ then
34:       $h \leftarrow h \cup \{ds_c\}$
35:       $ds_c \leftarrow newDecisionStump()$
36:       $\mathcal{F}' \leftarrow \mathcal{F}' \cup \{f_\alpha\}$
37:    end if
38: else
39:    while $N > i_{drift}$ do
40:       Remove from $\mathcal{F}'$ the feature selected in $h_N$
41:       $h \leftarrow h - \{h_N\}$
42:    end while
43: end if
```

Algorithm 1 presents the ABFS algorithm alongside the proposed changes (text with a different color than black). The algorithm adapts the stream Boosting version of [5] and uses stream decision stumps based on [2], being the decision stumps only for feature selection. All line references are related to Algorithm 1.

The rationale of Online Boosting [5] for incremental training of an instance consists of each learner having a weight for the training instance according to a previous base-learner correct or misclassification (lines 14-24, Algorithm 1). The algorithm tracks the sum of the weights of correct and misclassified instances of each learner (λ_m^{sc} and λ_m^{sw}, respectively), and the instance weight λ (line 3) is scaled based on half of the observations made by the learner ($\frac{\lambda_m^{sc}+\lambda_m^{sw}}{2}$, line 16 and 21), making λ_m^{sc} decrease and λ_m^{sw} increase, as desired [5].

For selecting features, the ABFS algorithm initially creates a single decision stump candidate (ds_c) that receives for training the instance with the accumulated training weight λ (line 32). When the decision stump performs a split, the feature used gets added to the subset of selected features, the decision stump becomes a member of the boosting ensemble, and a new ds_c gets instantiated (lines 33-36). A split attempt occurs after the sum of training weights reaches a user-given value [2]. Boosting is a sensitive method for misclassifications and as the number of misclassifications by the decision stumps increases, a split attempt will ensue faster. To track feature drifts, each decision stump in the boosting ensemble has a drift detector (lines 18 and 23). In case of detection, the decision stump and all of the following decision stumps that got a training weight from it are deleted from the ensemble (lines 39-42).

The first extension to the ABFS proposed in this paper is the usage of a different Online Boosting strategy, namely the Boosting Online Learning Ensemble (BOLE) [7]. This extension is highlighted in blue in Algorithm 1.

BOLE does not perform boosting linearly, i.e, a decision stump influences the training weights of the decision stump that was created after him. The decision stumps get sorted by the rate of the correct weights (line 5). The train starts with the worst decision stump regarding correct weight. If the instance is correctly classified, it is assumed that decision stumps with higher correct prediction rates also have a great chance of correctly classifying the instance (an error is unlikely), and the best decision stump not yet trained in the ensemble will receive the weight for training (lines 7-9, Alg. 1). Otherwise, the worst classifier not yet trained will receive weights for voting (lines 10-13, Alg. 1). It is worth noting that only the final accumulated weight λ will influence the ds_c, and BOLE increases the λ more when many decision stumps get misclassifications since the worst decision stumps influence each other until a decision stump makes a right prediction.

The second extension to the ABFS is related to the scaling value of the λ training weight. As mentioned, λ is scaled to half of the observations made by the learner (lines 16 and 21). We propose scaling to higher values of the observations. This extension is marked in red in Algorithm 1 as the parameter ω. Considering the analysis made in [5], the scaling factors will be $f_m^c < \frac{\omega}{2}$ and $f_m^w > \frac{\omega}{2}$, thus still making λ_m^{sc} decrease and λ_m^{sw} increase, as desired. When $\omega > 1$, the accumulated

λ range values increase more with higher misclassifications, forcing dc_s to do a split attempt faster than the standard Online Boosting in misclassifications.

It is worth noting that a pitfall of Boosting methods can be noise data. This is not the focus of this paper and we plan to analyze in future works Boosting methods for feature selection that are more resilient to noise data.

3 Experiments and Results

In this paper, we focus on the multi-class classification problem. We assess predictive performance through accuracy results using a test-then-train evaluation strategy, where every instance is used first for testing and then for training.

All the experiments were held in the MOA[1] framework. We evaluate monolithic classifiers with a Drift Detection Method (DDM)[8] detector (with default parameters) with and without feature selection. The monolithic classifiers evaluated were Naive Bayes, kNN, Hoeffding Tree [2] and the Extremely Fast Decision Tree (EFDT) algorithm [9], all with default parameters from MOA.

In this paper, we focus on real-world datasets because the best ABFS parameters evaluated in [4] differ for real-world and synthetic datasets. All of the datasets configurations and their references are available on the paper's website.

For all the versions of ABFS, we used the best-reported parameters [4]. The minimum sum of training weight for a split attempt (known as Grace Period [2]) was set to 100, the minimum information gain so a split can happen was set to 0.05 and ADaptive WINdowing (ADWIN) [10] drift detector was used.

Datasets	Native	Oza-ABFS $\omega = 1$	Oza-ABFS $\omega = 3$	BOLE-ABFS $\omega = 1$	BOLE-ABFS $\omega = 2$
Outdoor	59.55	66.075	63.45	62.075	63.05
Noaa	70.67	71.661	72.388	73.407	73.11
Elec	81.177	85.94	85.121	87.972	85.836
Nomao	94.441	95.285	95.497	95.398	95.085
Rialto	36.652	45.309	46.965	52.875	47.106
Covtype	88.026	86.624	89.638	88.428	89.058
Spam	74.314	90.894	91.002	90.326	90.691
Avg. Rank	4.857	3	**2.143**	2.286	2.714

Table 1: Naive Bayes accuracy.

Datasets	Native	Oza-ABFS $\omega = 1$	Oza-ABFS $\omega = 3$	BOLE-ABFS $\omega = 1$	BOLE-ABFS $\omega = 3$
Outdoor	59.325	65.975	63.375	61.075	65.025
Noaa	73.638	72.152	71.953	72.912	72.923
Elec	85.414	86.187	85.582	87.983	86.262
Nomao	95.149	95.674	95.697	95.598	95.825
Rialto	40.255	47.836	48.387	54.094	48.501
Covtype	87.355	86.158	89.226	87.812	88.525
Spam	87.066	90.862	91.023	90.133	90.165
Avg. Rank	4.286	3.143	2.714	2.857	**2**

Table 2: Hoeffding Tree accuracy

Datasets	Native	Oza-ABFS $\omega = 1$	Oza-ABFS $\omega = 3$	BOLE-ABFS $\omega = 1$	BOLE-ABFS $\omega = 3$
Outdoor	59.45	63.85	60.95	61.4	64.15
Noaa	74.255	71.529	72.554	72.845	72.823
Elec	86.341	86.619	86.141	88.096	86.52
Nomao	95.784	95.758	95.86	95.607	95.871
Rialto	57.911	54.934	57.154	59.81	54.311
Covtype	86.555	86.177	89.106	87.842	88.501
Spam	89.06	91.216	91.592	89.876	90.401
Avg. Rank	3.429	3.429	2.857	2.714	**2.571**

Table 3: EFDT accuracy.

Datasets	Native	Oza-ABFS $\omega = 1$	Oza-ABFS $\omega = 4$	BOLE-ABFS $\omega = 1$	BOLE-ABFS $\omega = 2$
Outdoor	65.225	49.525	55.7	61.4	53.675
Noaa	76.673	74.288	74.911	72.845	73.974
Elec	76.982	82.131	78.608	88.096	81.841
Nomao	96.405	95.741	96.135	95.607	96.031
Rialto	72.114	67.756	70.637	59.81	68.131
Covtype	87.876	82.492	83.512	87.842	84.608
Spam	84.288	88.438	88.3	89.876	87.784
Avg. Rank	**2.143**	3.571	2.857	3	3.429

Table 4: kNN accuracy.

Tables 1-4 present the accuracy of the monolithic classifiers and bold values indicate the best results. We denote Oza-ABFS as the standard ABFS and BOLE-ABFS as the extended ABFS with BOLE. We also present the algorithms with standard training factor ($\omega = 1$) and the best ω parameter per algorithm (results available on the paper's website).

For decision trees (Tables 2 and 3), all the ABFS extensions proposed leveraged the results more than the decision trees with standard ABFS, and BOLE-ABFS with $\omega = 3$ had the best results reported. For Naive Bayes, BOLE-ABFS with and without the modifications in the scaling factor leveraged Naive Bayes more than the Standard ABFS, but the best-reported results are with Oza-ABFS with $\omega = 3$. For kNN, the parameters reported in [4] did not leverage even the native algorithm, probably because for each data set, the parameters can vary more than the other algorithms. The algorithm that deleveraged kNN the less was Oza-ABFS with $\omega = 4$. Despite these facts, the processing time of kNN decreased drastically with the ABFS algorithm.

We provide a Nemenyi-Friedman posthoc test in the data from Tables 1-4, available on the paper's website. With a critical difference of 2.304, the extended ABFS with Naive Bayes showed to be statistically different compared to Native Naive Bayes, while the standard ABFS is not. The other algorithms were not statistically different from any ABFS version, even for Decision Trees that for all extended versions of ABFS have better average ranking compared to Native Decision Trees and with standard ABFS.

We also provide results regarding Processing Time in the paper's website.

Overall, BOLE-ABFS did not differ so much from Standard-ABFS in terms of average processing time and percentage increase compared to the Native algorithm. But the $\omega > 1$ scaling factor makes ABFS more expensive, probably because split attempts will occur more often and earlier, creating more Decision Stumps.

4 Conclusions and future works

In this paper, we proposed 2 changes in the training phase of the ABFS algorithm, namely, the integration of the BOLE algorithm to ABFS and different scaling factor values of the correct and wrong weights of Boosting. These modifications enhance the standard ABFS with a low overhead of processing time in Naive Bayes and Decision Trees.

In future works we plan to deeply analyze the kNN algorithm with feature selection algorithms, seeking to improve kNN results and make it the most efficient in processing time possible, since kNN complexity can be unfeasible in high-speed data streams that contain high dimensionality. We also plan to analyze Boosting methods for feature selection that are resilient to noise data.

References

[1] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, MOA: Massive online analysis. The Journal of Machine Learning Research, vol. 11, pp. 1601- 1604, 2010

[2] Pedro Domingos and Geoff Hulten. 2000. Mining high-speed data streams. In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '00). Association for Computing Machinery, New York, NY, USA, 71 80.

[3] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama and G. Zhang, "Learning under Concept Drift: A Review," in IEEE Transactions on Knowledge and Data Engineering, vol. 31, no. 12, pp. 2346-2363, 1 Dec. 2019, doi: 10.1109/TKDE.2018.2876857.

[4] Jean Paul Barddal, Fabrício Enembreck, Heitor Murilo Gomes, Albert Bifet, Bernhard Pfahringer, Boosting decision stumps for dynamic feature selection on data streams, *Information Systems*, Volume 83, 2019, Pages 13-29.

[5] Nikunj C. Oza, Stuart J. Russell. Online Bagging and Boosting. Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics, PMLR R3:229-236, 2001.

[6] Jean Paul Barddal, Heitor Murilo Gomes, Fabrício Enembreck, Bernhard Pfahringer, A survey on feature drift adaptation: Definition, benchmark, challenges and future directions, Journal of Systems and Software, Volume 127, 2017, Pages 278-294.

[7] R. S. M. d. Barros, S. Garrido T. de Carvalho Santos and P. M. Goncalves Júnior, "A Boosting-like Online Learning Ensemble," 2016 International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, Canada, 2016, pp. 1871-1878, doi: 10.1109/IJCNN.2016.7727427.

[8] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, Learning with drift detection. in Advances in Artificial Intelligence SBIA 2004, ser. LNCS. Springer, 2004, vol. 3171, pp. 286-295.

[9] C. Manapragada, G. I. Webb, and M. Salehi. 2018. Extremely Fast Decision Tree. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 18). Association for Computing Machinery, New York, NY, USA.

[10] A. Bifet, R. Gavaldà, Adaptive Learning from Evolving Data Streams, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 249-260. doi:10.1007/978-3-642-03915-7 22.