DOI: 10.48011/asba.v2i1.1491

An Incremental Learning approach using Long Short-Term Memory Neural Networks

Álvaro C. Lemos Neto * **Rodrigo A. Coelho** *
Cristiano L. de Castro *

* *Graduate Program in Electrical Engineering - Universidade Federal de Minas Gerais - Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, MG, Brazil (e-mail: alvarolemos@ufmg.br; toluenotnt@gmail.com; crislcastro@ufmg.br)*

Abstract: Due to Big Data and the Internet of Things, Machine Learning algorithms targeted specifically to model evolving data streams had gained attention from both academia and industry. Many Incremental Learning models had been successful in doing so, but most of them have one thing in common: they are complex variants of batch learning algorithms, which is a problem, since in a streaming setting, less complexity and more performance is desired. This paper proposes the Incremental LSTM model, which is a variant of the original LSTM with minor changes, that can tackle evolving data streams problems such as concept drift and the elasticity-plasticity dilemma without neither needing a dedicated drift detector, nor a memory management system. It obtained promising results that shows it reacts fast to concept drifts and that is also robust to noise data.

Keywords: Machine Learning, Incremental Learning, Neural Networks, Long Short-Term Memory.

1. INTRODUCTION

In recent years Big Data changed from just a *buzzword* to become a real thing, or a real problem for many companies. The increase of processing and memory capabilities of computers, more sensors generating data, the rise of the *Internet of Things*, gave businesses the opportunity to generate more insights by understanding their products and clients better, but also brought a burden to IT departments. More data means need for more storage, more processing power and the need of techniques to process this data in a distributed manner, because in many cases they don't fit into a single machine (Mehta et al., 2017; De Francisci Morales et al., 2016; Žliobaitė et al., 2016).

This problem reflects for the Machine Learning field as well, because normally Machine Learning models are designed to handle finite data sets, so that the learning process involves solving complex optimization problems, such as Support Vector Machines (SVMs) (Awad and Khanna, 2015). The other thing is that most Machine Learning models are trained with chunks of data that are supposed to be stationary, which in the real world it is rarely the case. Instead, information is always coming in an infinite stream of data that may change over time, so the Machine Learning researchers and developers should take that into account (Ditzler et al., 2015).

This paper tackles this problem of supervised learning from streaming data by proposing a novel Incremental Learning technique based on the Long Short-Term Memory (LSTM) Neural Networks (Hochreiter and Schmidhuber, 1997). The main idea is to use the LSTM's intrinsic features to be able to detect *concept drifts* and to learn new concepts (of course, keeping the old relevant ones) in an automatic manner using minimal memory resources.

The rest of this paper is organized as follows: in Section 2, both Incremental Learning and LSTMs are defined; Section 3 introduces the proposed method; In Section 4 the experimental methodology is presented; Section 5 show the results and Section 6 discusses some conclusions.

2. THEORY

A Machine Learning algorithm can be trained in two distinguished learning modes: *offline learning* and *incremental* or *online learning*. In the former, the whole dataset is available at the time of training, whereas in the latter, the model process data as they come in a real time stream that may be infinite. In such case, there are two main problems that it faces (Gama et al., 2014):

(1) If the dataset is infinite, the system hosting the model should have infinite memory to accommodate it, what is unreal
(2) Even if we were able to keep a long history of past data, the data distribution may change as time passes by (due to *concept drifts*), what would make past data stale regarding to the current data distribution

So an incremental learning algorithm is subject to a trade-off where it has to keep past information so the model doesn't learn outliers, but don't keep too much of it, because the system has memory constraints and also needs to be able to learn new concepts (Gama et al., 2014).

2.1 Concept Drifts & the Stability-Plasticity Dilemma

One of the major concerns of Incremental Learning methods is the *Concept Drift*, which primarily refers to an online supervised learning scenario when the relation between the input data and the target variable changes over time (Gama et al., 2014). How an algorithm adapts to a concept drift is also a challenge: react too quick, old information is lost; wait too long, concept drifts are not caught at all. This tradeoff is known as the *stability-plasticity dilemma* (Mermillod et al., 2013).

Although these are the most evident challenges faced by incremental learning methods, there are more, as pointed out by Gepperth and Hammer (2016). Many approaches have been proposed to address such challenges, which can easily be divided into two major categories: **active approaches**, which explicitly detect concept drifts, and **passive approaches**, those that continuously adapt themselves without explicit awareness of occurring drifts (Losing et al., 2016).

2.2 State of the Art

Most methods that follow the active approach are agnostic to the drift detection mechanism, meaning that one could try many flavours of a single method with different drift detectors. This is the case of many state of the art methods, like the *Oza Bagging ADWIN Classifier*, which is a mixture of the Online Bagging algorithm (Oza, 2005) (an online version of the original Bagging (Breiman, 1996)), with the ADaptive sliding Window (ADWIN) (Bifet and Gavalda, 2007) drift detection mechanism.

Another method that follows that same structure is the *Adaptive Random Forests* (Gomes et al., 2017), which is a online version of the original Random Forest algorithm (Breiman, 2001). For every incoming sample from a data stream, it runs a drift detection mechanism for each tree of the ensemble and, depending on the value obtained, it either warns a possible drift (which triggers the training of a background tree), or detects one, in which case the trained background tree is used to replace the existing one. A similar recent work is the *Adaptive XGBoost* (Montiel et al., 2020), which adapts the *XGBoost* algorithm (Chen and Guestrin, 2016) for evolving data streams. Both methods achieved the best results with the ADWIN drift detector.

For the passive approach methods, one that has achieved great results is the *Self Adjusting Memory k Nearest Neighbours* (SAM-kNN) (Losing et al., 2016). It's an ensemble of two models, one trained by the most recent samples from the stream, called the *Short Term Memory*, and another one trained with past samples, called *Long Term Memory*. The passive way that the drift detection takes place is by training the Short Term Memory model with different subsets of a window of recent samples, like a hyper parameter tuning taking place for every incoming sample.

One characteristic that all those methods have in common is that they are all structured in a complex way around classic supervised learning batch algorithms. In another hand, *Long Short-Term Memory* (LSTM) neural networks seem to fit perfectly for streaming scenarios: its *hidden state* and more importantly, the *memory cell*, was built to deal exactly with *stability-plasticity dilemma* and avoid *catastrophic forgetting* (Gepperth and Hammer, 2016). By applying minor changes in its architecture, we propose the Incremental LSTM, an incremental learning algorithm that doesn't need neither explicit drift detection, nor memory management.

2.3 Long Short-Term Neural Networks

Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) is a type of *Recurrent Neural Networks* (RNNs) and, as such, it is suited to learn sequence problems by tracking dependencies along the time. It does that by feeding back the hidden state of a layer, from a time step t, as an input to the next step $t+1$, along with the input for that step. Its weights are trained with the Backpropagation Through Time optimization algorithm (Goodfellow et al., 2016).

The problem with the simple RNNs arises when they are used to learn sequence problems that have long term dependencies. This happens due to a phenomenon known as *vanishing gradients* (Pascanu et al., 2013), where the network's weight updates in the beginning of the sequence tend to lose their contribution due to small values of gradient. To surpass this, the LSTM introduces the *memory cell* in its architecture, so it not only track dependencies at time step t with the hidden state h_{t-1} from the previous step $t-1$, but it also has a *memory cell* c_{t-1}, which doesn't vanish during the Backpropagation Through Time (Goodfellow et al., 2016).

The computations performed by an LSTM cell are listed in Equation 1, which are better understood along with the following Figure 1.

$$
\begin{aligned}
\Gamma_{f,t} &= \sigma(\boldsymbol{W_f}[\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + b_f) \\
\Gamma_{u,t} &= \sigma(\boldsymbol{W_u}[\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + b_u) \\
\widetilde{\boldsymbol{c}}_{\boldsymbol{t}} &= \tanh(\boldsymbol{W_c}[\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + b_c) \\
\boldsymbol{c_t} &= \Gamma_{f,t} \odot \boldsymbol{c_{t-1}} + \Gamma_{u,t} \odot \widetilde{\boldsymbol{c}}_{\boldsymbol{t}} \\
\Gamma_{o,t} &= \sigma(\boldsymbol{W_o}[\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + b_o) \\
\boldsymbol{h_t} &= \Gamma_{o,t} \odot \tanh(\boldsymbol{c_t}) \\
\hat{\boldsymbol{y}}_{\boldsymbol{t}} &= softmax(\boldsymbol{W_y h_t} + b_y)
\end{aligned}
\tag{1}
$$

In (1), the hidden state $\boldsymbol{h_{t-1}}$ from the previous time step $t-1$ is concatenated with the current step t input, $\boldsymbol{x_t}$, which is represented by $[\boldsymbol{a_{t-1}}, \boldsymbol{x_t}]$. It is weighted by $\boldsymbol{W_f}$, added with b_f and then applied to a sigmoid function $\sigma(\cdot)$, resulting in the *forgetting gate* $\Gamma_{f,t}$, which is ranged between 0 and 1, as shown in (2).

In (3), the *forgetting gate* $\Gamma_{f,t}$ weights the previous step's cell memory, $\boldsymbol{c_{t-1}}$. Its role is to forget or not this cell memory, depending if its value is closer to 0 than to 1.

In (4), the current time step's cell memory's contribution $\widetilde{\boldsymbol{c}}_{\boldsymbol{t}}$ is computed. The amount that it will contribute to the next step's cell memory is calculated in (6) and will depend on the *update gate* $\Gamma_{u,t}$, which is computed the same way as the *forgetting gate*, but with its own weights $\boldsymbol{W_u}$ and bias b_u, as shown in (5).

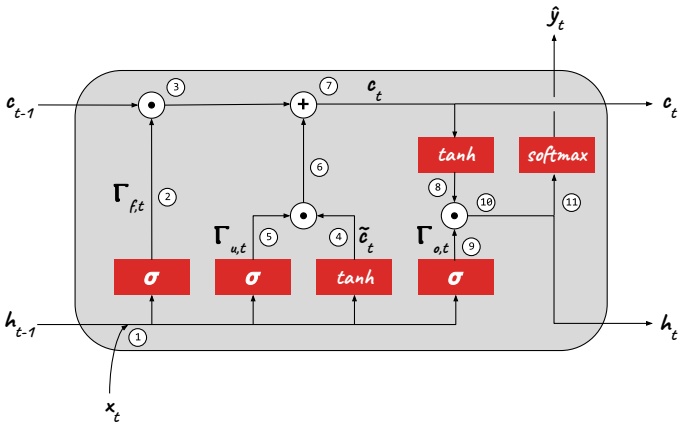

Figure 1. An LSTM cell
Adapted from Olah (2015)

The network's memory cell c_t is updated in (7) based on the previous and current steps' memory cell ($\boldsymbol{c_{t-1}}$ and $\Gamma_{u,t} \cdot \widetilde{\boldsymbol{c}}_{\boldsymbol{t}}$, respectively). Finally, it is weighted by the *output gate* $\Gamma_{o,t}$ in (10), which provides the *hidden state* $\boldsymbol{h_t}$, which is weighted by $\boldsymbol{W_y}$, followed by an element-wise addition of b_y, then fed to a *softmax* activation [1] in (11), providing the prediction $\hat{\boldsymbol{y}}_{\boldsymbol{t}}$ of the current step.

Figure 1 depicts a single cell of an LSTM layer, which is made of a sequence of cells. Also, LSTM layers can be connected, by feeding the hidden state as the input of the next layer, as shown in Figure 2.

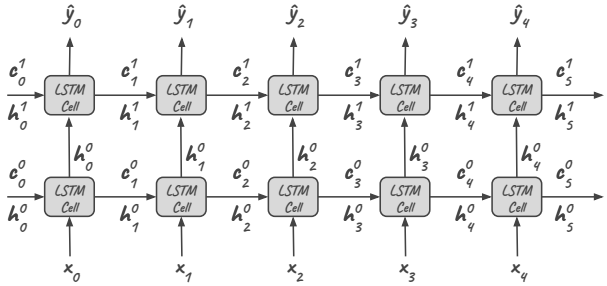

Figure 2. A neural network composed of two LSTM layers design for sequences with five time steps

It is important to note that both initial hidden states ($\boldsymbol{h_0^0}$ and $\boldsymbol{h_0^1}$) and initial memory cells ($\boldsymbol{c_0^0}$ and $\boldsymbol{c_0^1}$) are assigned randomly. In other words, the context carried by both memory cells and hidden states are exclusive for a single sequence.

This is the property that the proposed method focuses on, as described next.

3. INCREMENTAL LSTM

Before going into the details of learning algorithm, it is important to define how the proposed method prepares the received data into sequences suited for LSTMs.

[1] This is the case for a Neural Network designed to address a multiclass classification task, otherwise, other activation function should be used. Also, if this is not the output layer, the hidden state is passed to the next layer - i.e. without the need of step (11).

As will be seen in mentioned in Section 4, batches with a specific number of samples are accumulated in the data stream and then provided to the model, so it can predict, and then train. In the other hand, LSTMs expect a batch of sequences as input for training.

In order to transform a batch of samples into a batch of sequences, the batch is ordered by the time they were received. Then, it is broken down into multiple sub-batches in a sliding window manner. Figure 3 illustrates this. In blue, we have the current batch, with length 10, which is split into sub-batches of length 5 (in red), each of which is presented to the model as a sample sequence. So, with that example in mind, a batch of ten samples became a batch of six sample sub-batches.

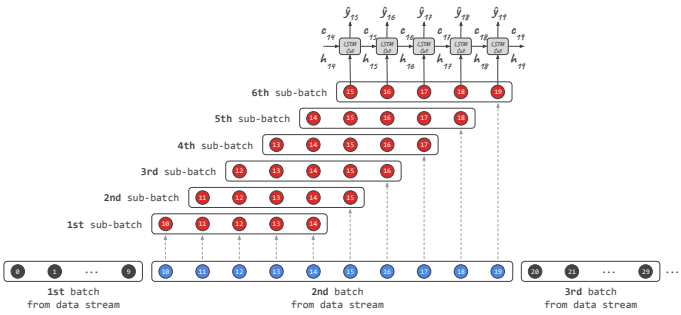

Figure 3. Data stream sub-batching

With the data preprocessing defined, the incremental learning process here proposed relies on using LSTM's own components to determine when to forget past concepts and when to learn new ones.

As mentioned earlier, the hidden state and memory cell are reset for each sample sequence received by the model. So, in batch B_t, we have $\boldsymbol{c_0} = \boldsymbol{0}$ and $\boldsymbol{s_0} = \boldsymbol{0}$. The trick here is to keep the last value of both $\boldsymbol{c}$ e $\boldsymbol{t}$, so that at the next batch, they are provided, instead of vectors of zeros.

By doing so, the following problems are solved:

(1) No need to keep a history of past sample that may violate the system's memory constraints, since only the LSTM cell's weights ($\boldsymbol{W_f}$, $\boldsymbol{W_u}$, $\boldsymbol{W_c}$, $\boldsymbol{W_o}$ and $\boldsymbol{W_y}$), biases (b_f, b_u, b_c, b_o and b_y) and the last values (meaning, at time t) of the cell and hidden state (c_t and a_t) should be kept in memory
(2) No need to implement a specific mechanism to detect a *concept drift*, since they should be captured automatically by the *memory cell*

This approach can be applied to distinct supervised Machine Learning problems, such as *regression*, *classification* and *forecasting*.

4. EXPERIMENTAL METHODOLOGY

To evaluate the proposed method, experiments were conducted comparing state of the art models using Prequential Evaluation. In the setting, data comes in batches over time. The initial batch, usually bigger than the subsequent ones, is used to pre-train the model. In the following ones, the batch data is first presented to the model without labels, which makes predictions. After that, a prequential

error is calculated using a metric best suited for the given experiment.

For all the experiments, the initial batch had a window size of 300 samples, while the subsequent batches had 100 samples. The prequential error evaluation metric used was accuracy, since only classification datasets with reasonable balancing were used. All experiments were conducted using the *Scikit-multiflow* framework [2] (Montiel et al., 2018) and were repeated five times, to increase statistical representativity.

4.1 Datasets

Four datasets were used for the experiments, which are:

- **Electricity** (Zliobaite, 2013): formed from data collected in the Australian electricity market. In this market, prices fluctuate according to the demand and data was sampled every five minutes. It has 45312 instances, 8 dimensions and 2 classes. There is no information regarding when concept drifts occur.
- **SEA** (Street and Kim, 2001): a synthetic dataset with 60000 instances, 3 dimensions and 2 classes. It contains two unbalanced classes with 22384 and 37616 instances each. The dataset presents four different concepts, with 15000 instances each and 10% of the data are noisy.
- **SINE 1** (Gama et al., 2004): has a total of 10000 instances, without noise, 2 dimensions and two different concepts with 5000 instances each and an abrupt change of concept. In the first concept, all points below the curve $y = \sin(x)$ are classified as positive and after the concept drift, the classification is reversed.
- **SINE 2** (Gama et al., 2004): has a total of 10000 instances, of 4 dimensions, two of which are noisy values, and two different concepts, 5000 instances each, with gradual change. The gradual change of concept was carried out by a sigmoid function $f(t) = 1/(1 + e^{4(tp)/w})$, where $p = 5000$ is the position that happens the change and $w = 500$ the width of the transition.

4.2 Models

The proposed algorithm was compared with the following incremental learning models, configured with their default hyperparameters from their implementation on *Scikit-multiflow*:

- **Oza Bagging ADWIN** (Oza, 2005)
- **Adaptive Random Forest** (Gomes et al., 2017)
- **SAM kNN** (Losing et al., 2016)

The proposed method was configured equally for all experiments:

- **LSTM layers**: 3 layers, with 150, 100 and 50 units respectively, all of them using the *hyperbolic tangent* activation function
- **Number of epochs**: 300 for the initial batch, 60 for the subsequent ones
- **Sub-batch size**: 60
- **Optimizer**: Adam, with $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $epsilon = 1e^{-7}$
- **Loss function**: Binary cross-entropy

[2] Available online on: `scikit-multiflow.github.io`

5. RESULTS

A summary for experimental results is presented in Table 1, whereas a more detailed behavior of the models' accuracy over time can be seen in Figures 4, 5, 6 and 7. In such Figures, we have a thick gray line representing the average accuracy for each training batch (across the five experiments), in lighter gray, the accuracy average standard deviation and in dashed red, the timestamps where a concept drift occurred. As pointed out early, there is no information regarding drifts for the *Electricity* dataset.

Dataset	Model	Accuracy (%)
SEA	ILSTM	84.68 ±3.83
	SAM-KNN	87.09 ±2.82
	ARF	**87.35 ± 3.78**
	OB-ADWIN	85.66 ±5.88
Sine 1	**ILSTM**	**97.34 ± 8.63**
	SAM-KNN	96.80 ±13.72
	ARF	97.01 ±8.74
	OB-ADWIN	87.76 ±23.54
Sine 2	**ILSTM**	**95.16 ± 8.75**
	SAM-KNN	86.01 ±17.52
	ARF	89.39 ±9.75
	OB-ADWIN	81.89 ±17.70
Electricity	ILSTM	59.47 ±12.27
	SAM-KNN	64.20 ±10.27
	ARF	**79.77 ± 9.66**
	OB-ADWIN	67.22 ±10.10

Table 1. Experiments overall results

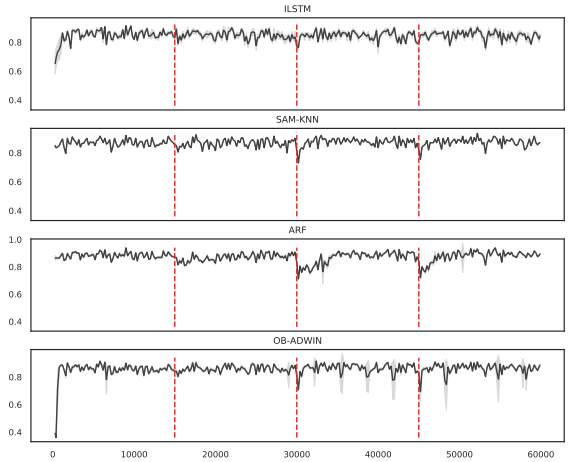

Figure 4. Accuracy evolution over time for the *SEA* dataset

The proposed method achieved competitive performance in comparison with the state-of-the-art methods. As can be observed in Figures 4, 5, 6 and 7, the accuracy drawdown that usually occurs after a concept drift tends to be smaller for ILTSM than for the other methods.

Nevertheless, those are preliminary results, and a more deep understanding of the Incremental LSTM's properties

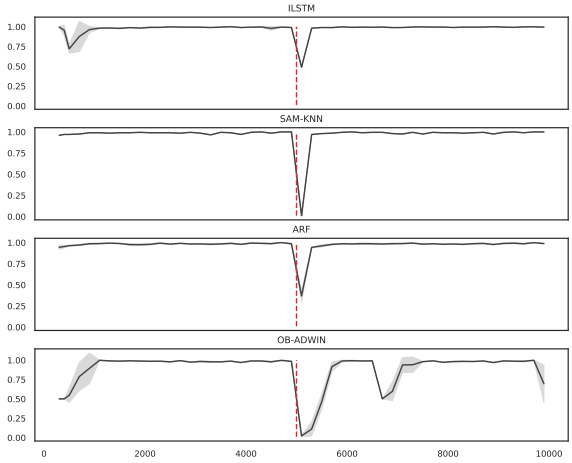

Figure 5. Accuracy evolution over time for the *Sine 1* dataset

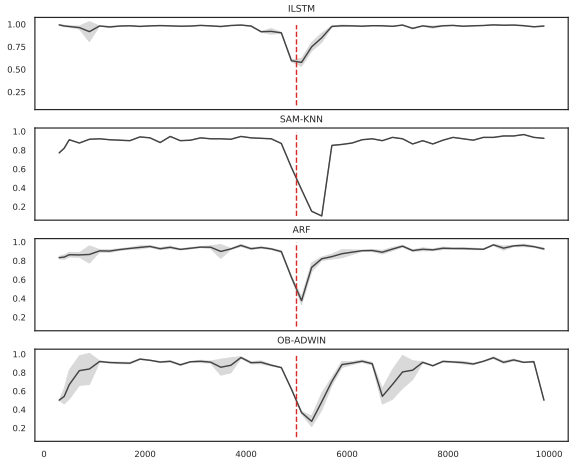

Figure 6. Accuracy evolution over time for the *Sine 2* dataset

and its sensitiveness to hyperparameter tuning is necessary. The method seems to be robust to noise, as shown by the results with SEA and SINE 2 datasets. However, the performance achieved for the Electricity was poor compared to the other methods.

6. CONCLUSION

The proposed method achieved promising results, having its simplicity as main strength. In contrast to other online learning algorithms, it does not need a complex memory structure, nor explicit mechanisms for detecting drifts or forgetfulness. Instead, it uses the ability to deal with the stability-plasticity dilemma of LSTM networks, through a simple but effective adaptation.

This is still a work in progress. The authors believe that experimenting with different architectural setups, as well as other recurrent neural network variants, like

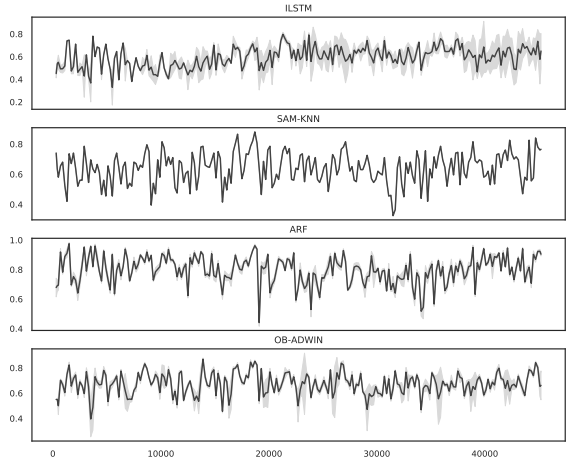

Figure 7. Accuracy evolution over time for the *Electricity* dataset

the Gated Recurrent Units (GRU) (Chung et al., 2014), performance could be improved. Another future effort would be to experiment with datasets for different tasks, such as regression and forecasting.

REFERENCES

Awad, M. and Khanna, R. (2015). Support vector machines for classification. In *Efficient Learning Machines*, 39–66. Springer.

Bifet, A. and Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*, 443–448. SIAM.

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2), 123–140.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1), 5–32.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

De Francisci Morales, G., Bifet, A., Khan, L., Gama, J., and Fan, W. (2016). Iot big data stream mining. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2119–2120.

Ditzler, G., Roveri, M., Alippi, C., and Polikar, R. (2015). Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4), 12–25.

Gama, J., Medas, P., Castillo, G., and Rodrigues, P. (2004). Learning with drift detection. In *Brazilian symposium on artificial intelligence*, 286–295. Springer.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4), 1–37.

Gepperth, A. and Hammer, B. (2016). Incremental learning algorithms and applications.

Gomes, H.M., Bifet, A., Read, J., Barddal, J.P., Enembreck, F., Pfharinger, B., Holmes, G., and Abdessalem, T. (2017). Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9-10), 1469–1495.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735–1780.

Losing, V., Hammer, B., and Wersing, H. (2016). Knn classifier with self adjusting memory for heterogeneous concept drift. In *2016 IEEE 16th international conference on data mining (ICDM)*, 291–300. IEEE.

Mehta, S. et al. (2017). Concept drift in streaming data classification: Algorithms, platforms and issues. *Procedia computer science*, 122, 804–811.

Mermillod, M., Bugaiska, A., and Bonin, P. (2013). The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in psychology*, 4, 504.

Montiel, J., Mitchell, R., Frank, E., Pfahringer, B., Abdessalem, T., and Bifet, A. (2020). Adaptive xgboost for evolving data streams. *arXiv preprint arXiv:2005.07353*.

Montiel, J., Read, J., Bifet, A., and Abdessalem, T. (2018). Scikit-multiflow: A multi-output streaming framework. *The Journal of Machine Learning Research*, 19(1), 2915–2914.

Olah, C. (2015). Understanding lstm networks.

Oza, N.C. (2005). Online bagging and boosting. In *2005 IEEE international conference on systems, man and cybernetics*, volume 3, 2340–2345. Ieee.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning*, 1310–1318.

Street, W.N. and Kim, Y. (2001). A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 377–382.

Zliobaite, I. (2013). How good is the electricity benchmark for evaluating concept drift adaptation. *arXiv preprint arXiv:1301.3524*.

Žliobaitė, I., Pechenizkiy, M., and Gama, J. (2016). An overview of concept drift applications. In *Big data analysis: new algorithms for a new society*, 91–114. Springer.