Introduction Online Learning

Fatiha BARRADE, Fatima NOUTFI

Keywords. *Online Learning; Self-Adaptive Systems; Incremental Learning; Stochastic Gradient Descent; Adaptive Learning*

1 Overview of Adaptive Learning Methods

Adaptive learning in machine learning is a dynamic approach that allows algorithms and models to evolve and improve autonomously as they encounter new data or changing conditions. Unlike traditional static models, adaptive learning systems continuously adjust their behavior, parameters, and structure based on real-time feedback and environmental cues. This process enables them to adapt to shifting data distributions, emerging patterns, and evolving user preferences, thereby enhancing their performance, accuracy, and relevance over time.

One key aspect of adaptive learning is its ability to leverage online learning techniques, where models update themselves incrementally with each new data point, rather than relying on batch processing. This enables them to capture and respond to changes in the underlying data distribution promptly, ensuring that their predictions remain up-to-date and accurate. Additionally, adaptive learning often incorporates feedback loops that allow the system to assess its own performance and make adjustments accordingly. This feedback mechanism enables the model to learn from its mistakes, refine its predictions, and continuously improve its performance.

Moreover, adaptive learning encompasses dynamic model architecture adjustments, where the structure and parameters of the model are modified based on observed performance and user feedback. This flexibility allows adaptive models to tailor their behavior to specific tasks, adapt to changing requirements, and optimize performance in diverse contexts. For example, in recommendation systems, adaptive learning algorithms can personalize recommendations based on user interactions and feedback, leading to more relevant and engaging content suggestions over time.

Despite its advantages, adaptive learning poses several challenges that need to be addressed. These include ensuring the quality and reliability of incoming data, interpreting and explaining the decisions made by adaptive models, managing computational resources efficiently, and addressing ethical considerations related to privacy, fairness, and transparency. Nevertheless, with careful design, rigorous evaluation, and responsible deployment, adaptive learning techniques have the potential to revolutionize machine learning applications across various domains, driving innovation and improving user experiences.

2 Self Adaptive Learning with MAPE-K

Self-Adaptive Systems (SASs) are systems that monitor and adapt their behavior autonomously in response to dynamic state and environmental conditions. A typical architecture of SASs is constituted of a Manager (Autonomic) Sub-System that controls a Managed Sub-System. A well-known architecture of the Autonomic Sub-System is the MAPE-K model. It is constituted of the **Monitor**, **the Analysis**, **the Plan**, and **the Execution stages** and **the Knowledge Base**. The major challenge of SAS is that all the stages are subject to uncertainty, significantly impacting the adaptation quality. Currently, uncertainty is considered a first-class concern in constructing Self-Adaptive Systems. However, few detailed works have been done about uncertainty in the MAPE-K Control Loop.

The architecture for the MAPE-K loop, as depicted in the following figure, represents a fundamental framework for self-adaptive systems (SASs) .

In the Monitor stage, sensors collect data reflecting the state of the system, while in the Analyze stage, this data is analyzed to understand the system's behavior and performance. Based on the analysis, the Plan stage formulates appropriate adaptation strategies to address any identified issues or changes. Finally, in the Execute stage, these strategies are implemented, and effectors apply changes to the managed subsystem. Throughout this process, the Knowledge Base facilitates information exchange between the components, enabling informed decision-making and adaptive behavior in response to dynamic conditions. This architecture provides a structured approach to autonomously monitor, analyze, plan, and execute adaptations, thereby enhancing the resilience and performance of self-adaptive systems.

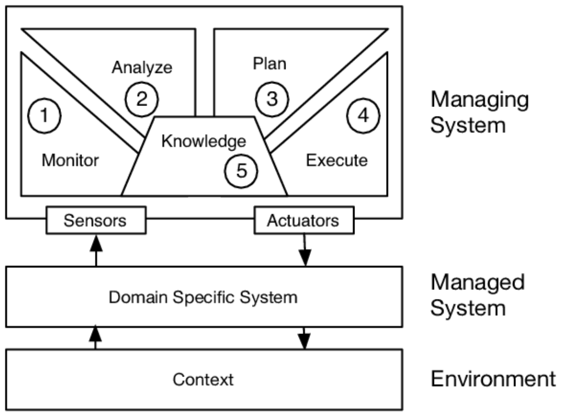

Figure 1: : MAPE-K reference model for self-adaptive systems

3 Logistic Regression

3.1 Binary Logistic Regression

Binary logistic regression aims to classify input observations into one of two categories, typically represented as 1 (positive class) or 0 (negative class). This approach is commonly used in various domains, such as sentiment analysis, where decisions are binary, such as positive or negative sentiment.

3.1.1 Model Representation

- Each input observation x is represented by a vector of features $[x_1, x_2, ..., x_n]$, where n denotes the number of features.
- The classifier output y indicates whether the observation belongs to the positive class (1) or negative class (0).
- Logistic regression learns from a training set, determining a vector of weights w and a bias term b.
- The weight w_i associated with each feature x_i represents its importance in the classification decision, with positive weights supporting the positive class and negative weights supporting the negative class.
- Additionally, the bias term b is a constant added to the weighted sum of features, allowing for flexibility in the decision boundary.

3.1.2 Decision Making Process

To classify a new instance, the classifier calculates a weighted sum of the features and adds the bias term. The resulting value, denoted as z, represents the evidence for the positive class. However, z is not constrained to the range $[0, 1]$, posing a challenge for interpretation.

3.2 The Sigmoid Function

To transform the weighted sum into a valid probability estimate, logistic regression employs the sigmoid function, denoted as $\sigma(z)$. The sigmoid function maps the real-valued input z to the range $(0, 1)$, ensuring that the output represents a probability. It has the mathematical form

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

and exhibits desirable properties, including near-linearity around 0 and the ability to compress extreme values towards the ends.

3.2.1 Probability Estimation

The sigmoid function yields the probability $P(y = 1)$ of the positive class and $P(y = 0)$ of the negative class. By ensuring that $P(y = 1) + P(y = 0) = 1$, logistic regression generates well-calibrated probability estimates for binary classification tasks. The input to the sigmoid function, the logit z, captures the log-odds of the positive class, providing a convenient interpretation of the classification process.

In summary, binary logistic regression leverages the sigmoid function to transform weighted inputs into valid probability estimates, facilitating binary decision-making in classification tasks.

3.3 Multinomial Logistic Regression

Sometimes, binary classification is not sufficient, and we need to classify observations into more than two classes. For instance, in sentiment analysis, we might want to classify text into positive, negative, or neutral sentiment. In such cases, we utilize multinomial logistic regression, also known as softmax regression or maxent classifier in older NLP literature.

3.3.1 Model Representation

In multinomial logistic regression, the goal is to assign each observation to one class from a set of K classes, where $K > 2$. This approach ensures that each observation is assigned only one class, known as hard classification. We represent the output y for each input x as a vector of length K. Specifically, if class c is the correct class, we set $y_c = 1$, and all other elements of y are set to 0. This vector y with one value equal to 1 and the rest 0 is termed a one-hot vector. The classifier's task is to produce an estimate vector $\hat{y}$, where $\hat{y}_k$ represents the classifier's estimate of the probability $p(y_k = 1|x)$ for each class k.

3.3.2 Softmax Function

The multinomial logistic classifier employs a generalization of the sigmoid function, known as the softmax function, to compute $p(y_k = 1|x)$. The softmax function maps a vector $z = [z_1, z_2, ..., z_K]$ of K arbitrary values to a probability distribution, ensuring that each value is in the range $[0, 1]$ and the values sum up to 1. Similar to the sigmoid function, the softmax function is an exponential function.

For a vector z of dimensionality K, the softmax is defined as:

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}, \quad 1 \leq i \leq K$$

The softmax of an input vector $z = [z_1, z_2, ..., z_K]$ is itself a vector:

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{K} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{K} \exp(z_i)}, ..., \frac{\exp(z_K)}{\sum_{i=1}^{K} \exp(z_i)} \right]$$

The denominator $\sum_{i=1}^{K} \exp(z_i)$ is used to normalize all the values into probabilities. Consequently, for a given vector z, the softmax function ensures that the resulting probabilities sum up to 1.

The softmax function has the property of squashing values towards 0 or 1, similar to the sigmoid function. Therefore, larger inputs tend to have higher probabilities, while smaller inputs are suppressed.

3.3.3 Applying Softmax in Logistic Regression

In multinomial logistic regression, we apply the softmax function to the weighted sum of inputs, similar to how we use the sigmoid function in binary logistic regression. However, in multinomial logistic regression, we require separate weight vectors w_k and biases b_k for each of the K classes.

The probability of each output class $\hat{y}_k$ can be computed as:

$$p(y_k = 1|x) = \frac{\exp(w_k \cdot x + b_k)}{\sum_{j=1}^{K} \exp(w_j \cdot x + b_j)}$$

To facilitate efficient computation, we often represent the set of weight vectors as a weight matrix W and a bias vector b. Each row k of W corresponds to the vector of weights w_k, resulting in W having the shape $[K \times f]$, where f denotes the number of input features. The bias vector b has one value for each of the K output classes.

Using this matrix representation, we can compute the vector $\hat{y}$ of output probabilities for each of the K classes using a single equation:

$$\hat{y} = \text{softmax}(Wx + b)$$

This equation efficiently computes the estimated probabilities for all classes simultaneously.

3.4 Offline Learning

The training in logistic regression is achieved by minimizing the cross-entropy loss function using an optimization algorithm such as gradient descent. The cross-entropy loss function measures how well the classifier's output matches the true labels. It is derived from the principle of maximizing the log probability of the true labels given the observations. Mathematically, it is expressed as:

$\mathcal{L}_{CE} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(p_{i,c})$

In this equation:

- $\mathcal{L}_{CE}$ represents the Cross Entropy Loss.
- N is the number of samples.

- C is the number of classes.
- $y_{i,c}$ is the ground truth label for sample i and class c, typically encoded as a one-hot vector.
- $p_{i,c}$ is the predicted probability of sample i belonging to class c, often obtained from a softmax function.

Gradient descent is then employed to minimize this loss function by iteratively adjusting the weights w and b in the direction opposite to the gradient. The update rule for each weight θ_j is given by:

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial L}{\partial \theta_j}$$

where α is the learning rate. By iteratively updating the weights using this rule, we aim to find the set of weights that minimize the average loss across all training examples.

4 Incremental Learning

In traditional batch learning, the machine learning model is trained on the entirety of the dataset at once. However, incremental learning follows a different approach. It learns from new data points as they become available, updating its model parameters incrementally, which is a stark contrast to batch learning's all-at-once methodology.

For instance, consider a spam email filter. With batch learning, the filter is trained with a large set of emails at once and then applied to future emails. If the nature of spam emails changes, the filter might start failing unless retrained on a new batch of emails, which includes the updated spam characteristics.

On the other hand, an incremental learning-based spam filter would adapt itself as new emails arrive, progressively updating its understanding of what constitutes spam. If spam strategies change, this type of filter could potentially learn to recognize new spam styles without needing a whole new batch of training data.

4.0.1 Benefits of Incremental Learning

- Efficient use of resources: Incremental learning models need to store less data at a time, which can lead to significant memory savings.

- Real-time adaptation: These models can adapt to changes in real-time, allowing for dynamic adjustments based on evolving data.
- Efficient learning: Breaking a task into smaller parts can enhance the model's ability to learn new tasks quickly and effectively.
- Learning from non-stationary data: Incremental learning models are valuable in scenarios where data evolves rapidly, allowing for continuous adaptation and improvement.

4.0.2 Limits of Incremental Learning with Offline Learning

Offline learning in logistic regression, particularly, cannot facilitate incremental learning due to its inherent characteristics and limitations. Logistic regression typically involves training on a fixed dataset where the model learns from all available data at once. Once trained, the model parameters, such as weights and biases, are determined based on this static dataset and remain unchanged unless the model is retrained with updated data. This static nature of offline learning prevents logistic regression models from adapting to new observations or changes in the data distribution without retraining from scratch.

Moreover, logistic regression models trained offline are susceptible to catastrophic forgetting, a phenomenon where the model tends to forget previously learned information as it learns new data. Since offline learning does not allow for incremental updates, the model lacks the ability to retain knowledge from past observations while incorporating new information. Additionally, logistic regression models trained offline may struggle to handle concept drift, which refers to abrupt changes in data trends over time. Without the capability to adapt in real-time, offline logistic regression models cannot adjust their parameters to accommodate shifting data distributions or evolving environments.

Furthermore, offline logistic regression may be prone to overfitting, where the model becomes too closely aligned with the training data and fails to generalize well to unseen instances. Incremental updates, which are common in online learning settings, help mitigate overfitting by allowing the model to adjust gradually to new observations and prevent drastic changes in parameter values. However, offline learning lacks this adaptive mechanism, making it challenging to maintain a balance between model complexity and generalization.

5 the FOLKLORE Algorithm

In this section, we introduce the FOLKLORE (Fast Online Learning with Large Online Regression) algorithm for online multiclass logistic regression. FOLKLORE is designed to address the limitations of existing algorithms, providing a practical solution that runs significantly faster while maintaining competitive performance guarantees.

5.1 Key Ideas

The FOLKLORE algorithm leverages the following key ideas:

- **Quadratic Complexity**: Unlike previous algorithms with polynomial complexity, FOLKLORE achieves a quadratic time complexity per iteration. This significant reduction in computational cost enables efficient processing of large-scale datasets in real-time.
- **Norm-dependent Regret Bound**: FOLKLORE's regret bound exhibits a linear dependence on the norm of the predictors, allowing for faster convergence without sacrificing performance guarantees. This balance between computational efficiency and regret minimization is crucial for practical applications.
- **Online-to-Batch Conversion**: FOLKLORE includes an online-to-batch conversion mechanism, enabling seamless integration with batch learning frameworks. This flexibility allows practitioners to leverage the benefits of online learning while retaining the option to perform offline analysis when necessary.

5.1.1 Mathematical Background

The FOLKLORE algorithm is based on the principles of online convex optimization and logistic regression. At each iteration, the algorithm predicts class probabilities using the logistic function and computes the loss function to measure the deviation between predicted and true labels. The gradient of the loss function with respect to the model parameters is then used to update the model using gradient descent.

The regret bound of the FOLKLORE algorithm is derived using techniques from online convex optimization and concentration inequalities. By analyzing the algorithm's performance in terms of regret, we can quantify its effectiveness in minimizing prediction errors over time.

5.2 Pseudo Code

Below is the pseudo code for the FOLKLORE algorithm:

```
**Algorithm 1** FOLKLORE
**input**: regularization $\lambda$, improper regularization $\phi_t$
Set $A_0 = \lambda \cdot \mathbf{I}_{Kd}, G_0 = 0$.
**for** $t = 1, \ldots, T$ **do**
    Receive $x_t$ and play $z_t = W_t x_t$ where
        $$W_t = \underset{W \in \mathbb{R}^{d \times k}}{\operatorname{argmin}} \lambda \|W\|_F^2 + \sum_{s=1}^{t-1} \hat{\ell}_t(W) + \phi_t(W) = \underset{W \in \mathbb{R}^{d \times k}}{\operatorname{argmin}} \|\overrightarrow{W}\|_{A_{t-1}}^2 + \langle \overrightarrow{W}, G_{t-1} \rangle + \phi_t(W). \quad (3.3)$$
    Receive $y_t$ and suffer loss $\ell_t$.
    Update $A_t = A_{t-1} + \frac{1}{BR + \ln(K)/2} \nabla^2 \ell_t(W_t)$.
    Update $G_t = G_{t-1} + \nabla \ell_t(W_t)$.
```

Figure 2: Pseudo Code FOLKLORE

5.3 Why FOLKLORE Allows Incremental Learning

The FOLKLORE algorithm, designed for online multiclass logistic regression, embodies key features that enable incremental learning, making it particularly adept at handling dynamic and evolving datasets.

Firstly, FOLKLORE's quadratic complexity stands out as a fundamental advantage. Unlike previous algorithms burdened by polynomial complexity, FOLKLORE achieves a quadratic time complexity per iteration. This efficiency enables the algorithm to process large-scale datasets in real-time, crucial for incremental learning where data arrives continuously.

Secondly, FOLKLORE's norm-dependent regret bound contributes to its effectiveness in incremental learning. By exhibiting a linear dependence on the norm of the predictors, FOLKLORE ensures faster convergence without sacrificing performance guarantees. This balance between computational efficiency and regret minimization is essential for incremental learning, where rapid adaptation to new data is paramount.

Moreover, FOLKLORE's incorporation of an online-to-batch conversion mechanism enhances its adaptability and flexibility. This feature enables seamless integration with batch learning frameworks, allowing practitioners to leverage the benefits of online learning while retaining the option for offline analysis when necessary.

In practical terms, these features collectively empower FOLKLORE to handle streaming data effectively. The algorithm's ability to update model parameters incrementally, coupled with its efficient processing of large-scale datasets, facilitates adaptive learning in real-time. By continuously updating the model based on incoming data, FOLKLORE captures changes in the data distribution and adjusts its predictions accordingly. This adaptability ensures improved performance and robustness in dynamic environments, making FOLKLORE a powerful tool for incremental learning in logistic regression.

Overall, FOLKLORE's unique blend of computational efficiency, regret minimization, and adaptability positions it as an ideal choice for incremental learning, especially in scenarios where data arrives continuously and must be processed in real-time.

Conclusion

The report offers a comprehensive overview of adaptive learning methods, logistic regression, offline learning, and incremental learning. These topics delve into various aspects of machine learning, emphasizing the importance of adaptability, efficiency, and performance in handling dynamic datasets.

The adaptive learning methods highlighted in the report underscore a personalized and data-driven approach, ensuring tailored experiences in diverse contexts. Logistic regression, both in binary and multinomial forms, emerges as a powerful tool for classification tasks, providing insights into model representation, decision-making processes, and probability estimation. The comparison between offline and incremental learning in logistic regression reveals nuanced trade-offs and benefits, particularly in managing evolving datasets.

The introduction of the FOLKLORE algorithm presents a novel solution for online multiclass logistic regression, addressing the challenges of computational complexity, regret minimization, and adaptability. By leveraging quadratic complexity, norm-dependent regret bounds, and an online-to-batch conversion mechanism, FOLKLORE enables efficient processing of streaming data while maintaining competitive performance guarantees.

In conclusion, the report underscores the significance of adaptability and efficiency in machine learning algorithms, particularly in the context of handling dynamic and evolving datasets. The FOLKLORE algorithm exemplifies these principles, offering a practical solution for incremental learning in logistic regression and demonstrating the potential for improved performance and robustness in dynamic environments.

References

[1] Maria Casimiro, Paolo Romano, David Garlan, Mark Klein. *Self-adaptive Machine Learning Systems: Research Challenges and Opportunities*, August 2022. In: *Software Architecture*. https://doi.org/10.1007/978-3-031-15116-$3_7 10.1007/978-3-031-15116-3\_7$

[2] IEEE, F. (2015). Modeling and Analyzing MAPE-K Feedback Loops for Self-adaptation *10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* . Retrieved from Springerlink.com.

[3] Ambrosio, L., Gigli, N., & Savaré, G. (2008). *Self-Adaptation for Machine Learning Based Systems* .`https://ceur-ws.org/Vol-2978/saml-paper6.pdf`

[4] Biyani, K., & Kulkarni, S. (2007). Mixed-Mode Adaptation in Distributed Systems: A Case Study. In: *Software Engineering for Adaptive and Self-Managing Systems*.

[5] Brown, G., et al. (2006). Goal-oriented specification of adaptation requirements engineering in adaptive systems. In: *Software Engineering for Adaptive and Self-Managing Systems*.

[6] Abeywickrama, D. B., Hoch, N., & Zambonelli, F. (2013). SimSOTA: engineering and simulating feedback loops for self-adaptive systems. In: *International C* Conference on Computer Science & Software Engineering, C3S2E13, Porto, Portugal - July 10 - 12, 2013*, ACM, pp. 67–76.