Auto-Lifelong: An AutoML System for Lifelong Machine Learning

Zheng Xiong [*1] xiongz17@mails.tsinghua.edu.cn
Wenpeng Zhang [*2] zhangwenpeng0@gmail.com
Jiyan Jiang [2] jiangjy17@mails.tsinghua.edu.cn
Wenwu Zhu [1,2] wwzhu@tsinghua.edu.cn
([*] Equal Contribution)
[1] *Tsinghua-Berkeley Shenzhen Institute,* [2] *Department of Computer Science and Technology; Tsinghua University, China*

Abstract

AutoML aims at automating the process of designing good machine learning pipelines. Currently, existing AutoML systems are mostly designed for single batch learning. However, in many real-world applications, the data may arrive continuously in sequential batches, even possibly with concept drift. This lifelong machine learning scenario raises a new challenge for AutoML, as most existing AutoML systems cannot effectively evolve over time and adapt to concept drift. In this paper, we propose a novel AutoML system named as Auto-Lifelong for the lifelong machine learning scenario. Using boosting tree as the base model, our system won the second place in the NeurIPS 2018 AutoML Challenge.

1. Introduction

Despite that machine learning has achieved great successes in recent years, it still relies heavily on human experts to preprocess data, extract features, choose learning algorithms, and design neural architectures or set hyperparameters. The whole process is quite complex and thus usually very laborious. AutoML aims to reduce the demand for human efforts in designing machine learning pipelines by automating the process.

Existing AutoML systems, such as Auto-Sklearn (Feurer et al., 2015) and TPOT (Olson et al., 2016), are mainly designed for single batch learning, i.e. an AutoML system only yields one model trained on a single batch of data. However, in many real-world applications, such as recommendation, online advertising, and fraud detection, the data usually arrives continuously in successive batches and its distribution may even change dynamically over time, which is known as suffering concept drift (Gama et al., 2014). This lifelong machine learning scenario (Chen and Liu, 2016) raises a new challenge for AutoML, as most existing AutoML systems cannot evolve over time to effectively utilize the latest information and handle concept drift.

In this paper, to tackle this challenge, we propose a novel AutoML system termed as Auto-Lifelong for lifelong machine learning. Our system can learn continuously, accumulate the knowledge learned in the past, and adapt to concept drift, in changing environments. Using gradient boosting decision tree (GBDT) as the base model, our system won the second place in the NeurIPS 2018 AutoML Challenge (Escalante et al., 2019).

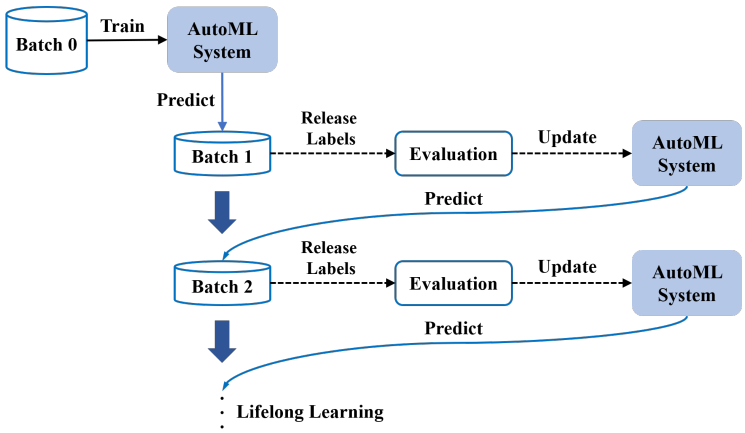

Figure 1: The workflow of an AutoML system under the lifelong machine learning scenario.

2. NeurIPS 2018 AutoML Challenge

In this section, we first introduce the workflow of an AutoML system under lifelong machine learning, and then review the details of the NeurIPS 2018 AutoML Challenge.

As illustrated in Figure 1, the workflow of such an AutoML system is as follows. The system is first trained on a batch of data with true labels. Then new data arrives sequentially in batches. At each subsequent stage, the system first predicts the labels of a new test batch, then the true labels of this batch are revealed to evaluate the prediction results. After evaluation, the current batch can be used as a new training batch to update the AutoML system.

The NeurIPS 2018 AutoML Challenge consists of a feedback phase and a blind test phase. During the feedback phase, five public datasets are provided for the participants to develop their AutoML systems. During the blind test phase, the developed AutoML system is blindly tested on five new datasets without any human intervention. As only binary classification problems are considered in the challenge, normalized AUC (scaling AUC to $[-1, 1]$) is used as the evaluation metric.

Compared to previous challenges, this challenge introduces some new difficulties under the lifelong learning scenario and accordingly raises new requirements for AutoML systems.

- **Varied feature types.** Four different types of features are included, namely numerical, categorical, multi-value categorical and temporal, which requires a more complicated search space for feature engineering, especially for extracting useful information from the interaction between different types of features.
- **High-cardinality categorical features.** Categorical features with high cardinality following a power-law distribution are given, which requires a proper encoding scheme to get a compact and informative representation of categorical features.
- **Concept drift.** The data distribution is changing dynamically over time, which requires the system to evolve adaptively to handle the drifting concept.
- **Scalability.** The datasets in the challenge are 10-100 times larger than those in previous challenges, which requires an efficient and robust design of the system to satisfy the time and memory constraints.

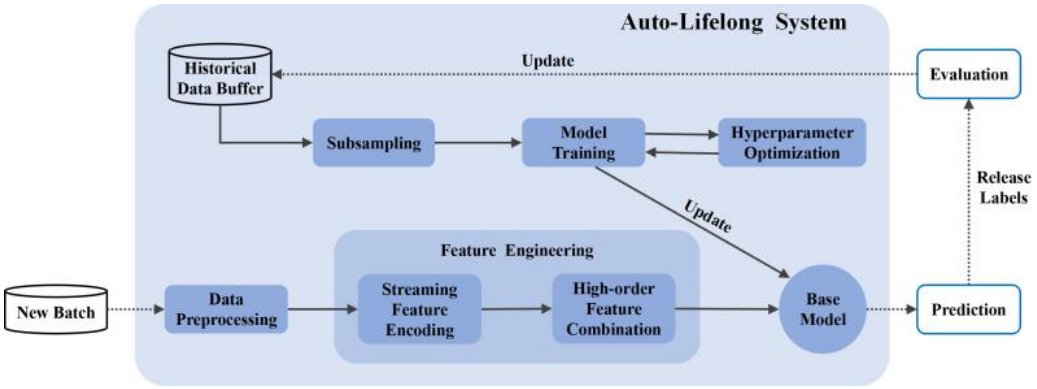

Figure 2: The framework of our proposed AutoML system. The boxes represent operations, the cylinders represent data, and the circle represents the model.

3. Our System

In this section, we first introduce the AutoML system we design for lifelong machine learning, then introduce the detailed realization used in the competition.

3.1 System Overview

The system framework is illustrated in Figure 2. When a new test batch arrives, the system first performs data preprocessing, which may include imputing missing values or removing some low-quality features. Then it comes to feature engineering, which mainly consists of streaming feature encoding and high-order feature combination. Then using the learned model in the previous stage, the system makes predictions for the test batch. After prediction, the true labels of the test batch are revealed to evaluate the performance. Then the batch with labels is put into the historical data buffer. The system updates the base model used for prediction by retraining it with subsampled data, which is contained in a sliding window on the batch sequence saved in the buffer. The optimal configuration of the base model is given by hyperparameter optimization.

3.2 Categorical Feature Encoding

Encoding is essential for categorical features to transform nominal categories into numerical representation which can be handled by machine learning models.

The choice of encoding scheme largely depends on the characteristics of the categorical feature, and has a great influence on model performance (Müller et al., 2016). We highlight two key factors for choosing the proper encoding scheme used in the competition.

First, as mentioned in Section 2, many categorical features in the challenge have high cardinality and follow a power-law distribution. The most commonly used encoding schemes one-hot encoding is therefore infeasible, as it will give an extremely sparse and inefficient representation with very high dimensions (Micci-Barreca, 2001).

Second, we prefer encoding schemes which provide practical meanings in the order of the encoded feature values. This can help construct a more interpretable feature space and possibly improve the model performance.

Taking these factors into consideration, we adopt a frequency encoding scheme by mapping each category X to its frequency in the data batch, which is an empirical estimation of $p(X)$. Frequency encoding can easily handle high cardinality by providing a compact one-dimensional representation. Moreover, it provides an ordinal representation with a clear practical meaning. For example, the frequency of a category may represent the popularity of an item, which can be used as an informative indicator for the target variable. Due to the page limit, we give a full comparison of different encoding schemes we have considered for categorical features in Table 2.

We will further introduce how to improve frequency encoding by streaming encoding in Section 3.4.

3.3 High-order Feature Combination

High-order feature combination aims at extracting useful information from the interaction between different features to improve model performance.

There are two key challenges for high-order feature combination. First, the effect of high-order feature combination varies significantly across datasets, which requires a robust strategy to choose the proper feature combination for different dataset adaptively. Second, the search space of high-order feature combination is extremely large, which requires an efficient strategy to select effective feature transformations within an affordable time budget. We choose the trade-off between robustness and efficiency based on both prior knowledge and automated feature selection.

The search space for high-order feature combination in our system for the competition consists of a set of binary transformations predefined based on prior knowledge, which is much more complicated than the feature processing methods considered in the winning solutions of previous AutoML challenges (Feurer et al., 2015, 2018). Due to the page limit, we give a detailed description of the binary transformations we have used in Appendix B.1.

To search over this feature space and select the proper feature combination for different datasets, we apply each type of transformation on the original feature set and perform automated feature selection in an expansion-reduction fashion (Kanter and Veeramachaneni, 2015; Lam et al., 2017), which includes three key steps.

1. **Pre-selection.** Coarsely select the features used for each type of transformation based on prior knowledge. For example, we discard categorical features with too many or too few unique categories for groupby transformation, as the groups corresponding to these categories usually don't have a clear practical meaning.
2. **Feature generation.** Generate new features with all feasible pairs of the pre-selected features.
3. **Post-selection.** Add all generated features to the current feature set and train a coarse GBDT model. If model performance doesn't improve after adding the new features, discard this type of transformation for the current dataset. Otherwise, sort the new features by feature importance score, use the average score as a threshold, and add the top k generated features above the threshold to the current feature set.

3.4 Concept Drift Adaptation

Concept drift adaptation aims at evolving the AutoML system over time to utilize the latest knowledge and adapt to concept drift in streaming data.

Mathematically, concept drift is defined as the change of joint probability of input variables X and target variable y between different time points t_0 and t_1:

$$\exists X : p_{t_0}(X, y) \neq p_{t_1}(X, y).$$

As a common practice, concept drift can be further classified as real drift and virtual drift (Gama et al., 2014). Real drift refers to the change of $p(X|y)$, which makes previously learned model become obsolete and inaccurate on future test batches. Virtual drift refers to the change of $p(X)$, which makes feature representation inconsistent between batches. We design a windowing strategy and a streaming encoding scheme to tackle these two types of drift respectively.

Windowing strategy tackles real drift by maintaining a predictive model consistent with a set of most recent examples (Widmer and Kubat, 1996; Street and Kim, 2001). Specifically, we apply a sliding window on historical data batches and retrain a new GBDT model with the data in the window for each test batch.

Streaming encoding tackles the distribution drift in categorical feature, which accounts for a major part of virtual drift in the challenge. Specifically, (1) To capture the drifting trends of categorical features, we maintain a streaming frequency encoder and update it in an online fashion. When a new data batch arrives, the information of categorical features in this batch is utilized to update the mapping function of the streaming encoder. Mathematically, let $p_t(X)$ be the mapping function of category X on batch t, then the mapping function on the next batch $p_{t+1}(X)$ is updated as:

$$p_{t+1}(X) = \frac{p_t(X) \cdot N_t + x_{t+1}}{N_t + n_{t+1}},$$

where N_t is the total number of instances in the previous t batches, n_{t+1} is the sample size of batch $(t+1)$, and x_{t+1} is the count of category X in batch $(t+1)$. (2) To achieve a consistent representation of categorical features between the training set and the test set in each test stage, we always encode the training set and the test set simultaneously with the updated encoder in each test stage.

3.5 Other Components

In this subsection, we introduce some other important components in our AutoML system for the competition.

Model selection. Gradient boosting tree (Friedman, 2001) has been proved to be the best choice for many real-world problems with heterogeneous features (Chen and Guestrin, 2016). It has also been reported that using gradient boosting tree alone as the candidate model is sufficient to build an AutoML system with good performance (Feurer et al., 2018; Thomas et al., 2018). Consequently, we use gradient boosting decision tree, implemented by LightGBM (Ke et al., 2017), as the only classification model without explicit model selection, which helps simplify the pipeline and improve the efficiency of our system.

Hyperparameter optimization. To improve the efficiency of our AutoML system, we perform hyperparameter optimization with warm start. We firstly search over a hand-crafted portfolio to find a near optimal configuration, then perform local search around it to find the best hyperparameters, which is implemented by Hyperopt (Bergstra et al., 2013). Details of the portfolio and the local search space are illustrated in Appendix B.2.

Resource management. Resource management aims at automatically adjusting the system configuration to satisfy the resource constraints on different dataset, which consists of time budget control and memory control. For time budget control, the system firstly estimates the computational cost of each component, then automatically adjusts some key configurations to satisfy the time budget, such as whether or not to apply local search in hyperparameter optimization, and the number of iterations in GBDT. For memory control, the system firstly estimates the peak value of memory usage based on data size, then automatically adjusts some key configurations to satisfy the memory constraint, such as the pool size of multiprocessing and the batch number of the historical data buffer.

4. Challenge Results

The NeurIPS 2018 AutoML Challenge attracted more than 300 participants (Escalante et al., 2019), and the final ranking of each team is determined by its average ranking on the five private datasets during the blind test phase. The final ranks of the top 3 teams in the challenge are shown in Table 1. Our team won the second place with an average ranking of 2.4, which validates the effectiveness and generalization of our AutoML system.

Table 1: Final rankings of the top 3 teams in the challenge.

Team Name	Set 1	Set 2	Set 3	Set 4	Set 5	Average
autodidact.ai	2	4	1	2	2	2.2
Meta_Learners	**3**	**1**	**2**	**1**	**5**	**2.4**
GrandMasters	4	6	4	3	4	4.2

We have also conducted further experiments on the five public datasets of the feedback phase to evaluate the novel components proposed in our system. Due to the page limit, please refer to Appendix C for the experimental results.

5. Conclusion

We propose an AutoML system for lifelong machine learning, an important learning scenario which widely exists in many real-world applications but has rarely been considered in previous AutoML systems. Our system can learn continuously, accumulate past learned knowledge and deal with concept drift. For the competition, we adopt GBDT as the base model, introduce frequency encoding, high-order feature combination and streaming encoding, and obtain a good realization of our system. The resulting system won the second place in the NeurIPS 2018 AutoML Challenge. For future work, we plan to add a new memory module to enalbe better continual learning, and enlarge the scope of base models to enable better model selection.

Acknowledgements

The authors thank the organizers of the NeurIPS 2018 AutoML Challenge for their support and other participants of the challenge for their helpful discussion. This work is supported by National Program on Key Basic Research Project No.2015CB352300, National Natural Science Foundation of China Major Project No.U1611461, and Shenzhen Nanshan District Ling-Hang Team Grant No.LHTD20170005.

References

James Bergstra, Dan Yamins, and David D Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, pages 13–20. Citeseer, 2013.

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

Zhiyuan Chen and Bing Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 10(3):1–145, 2016.

Hugo Jair Escalante, Wei-Wei Tu, Isabelle Guyon, Daniel L Silver, Evelyne Viegas, Yuqiang Chen, Wenyuan Dai, and Qiang Yang. Automl@ neurips 2018 challenge: Design and results. *arXiv preprint arXiv:1903.05263*, 2019.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in neural information processing systems*, pages 2962–2970, 2015.

Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Practical automated machine learning for the automl challenge 2018. In *International Workshop on Automatic Machine Learning at ICML*, 2018.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):44, 2014.

James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10. IEEE, 2015.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.

Hoang Thanh Lam, Johann-Michael Thiebaut, Mathieu Sinn, Bei Chen, Tiep Mai, and Oznur Alkan. One button machine for automating feature engineering in relational databases. *arXiv preprint arXiv:1706.00327*, 2017.

Daniele Micci-Barreca. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *ACM SIGKDD Explorations Newsletter*, 3(1): 27–32, 2001.

Andreas C Müller, Sarah Guido, et al. *Introduction to machine learning with Python: a guide for data scientists*. " O'Reilly Media, Inc.", 2016.

Randal S Olson, Ryan J Urbanowicz, Peter C Andrews, Nicole A Lavender, Jason H Moore, et al. Automating biomedical data science through tree-based pipeline optimization. In *European Conference on the Applications of Evolutionary Computation*, pages 123–137. Springer, 2016.

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

Janek Thomas, Stefan Coors, and Bernd Bischl. Automatic gradient boosting. *arXiv preprint arXiv:1807.03873*, 2018.

Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.

Appendix A. Encoding Schemes for Categorical Features

As mentioned in Section 3.2, we considered two key factors for choosing the proper encoding scheme in our system, i.e. whether it can provide a compact representation of high-cardinality categorical features, and whether the encoded feature has practical meaning in its order. We compared four different encoding schemes based on these criterions, as shown in Table 2.

Table 2: Comparison of encoding schemes for categorical features.

Encoding scheme	**Description**	**Compact representation**	**Order with meaning**
One-hot Encoding	Transform a categorical feature with N categories into N binary features, with one of them 1, and all others 0	No	No
Label Encoding	Represent each category by a unique integer between 0 and N, where N is the number of unique categories in the categorical feature	Yes	No
Frequency Encoding	Represent each category X by its frequency, which is an estimation of $p(X)$	Yes	Yes
Target Encoding	Represent each category X by the mean of the target variable in this category, which is an estimation of $p(y\|X)$	Yes	Yes

Both frequency encoding and target encoding satisfy the two criterions. However, target encoding is easy to overfit, as the usage of training labels causes data leakage. Furthermore, real concept drift causes a significant change of $p(X|y)$ between batches, which introduces great noise to target encoding and harms its performance. Consequently, we only adopt frequency encoding in our system.

Appendix B. Configuration Space

B.1 Binary Transformations for Feature Combination

The binary transformations are performed on all feasible pairs of two types of features, represented by operand A and operand B respectively. A detailed description of the transformations we have used is presented in Table 3.

B.2 Search Space for GBDT

To improve the efficiency of hyperparameter optimization, we search over three key hyperparameters which have great influence on the performance of GBDT, i.e. num_leaves (maximum tree leaves for base learners), n_estimators (number of boosting trees to fit) and learning_rate. The hand-crafted portfolio is presented in Table 4. The local search space is presented in Table 5, in which m, n, η are num_leaves, n_estimators and learning_rate of the best configuration in the portfolio respectively.

Table 3: Binary transformations for feature combination.

Transformation	**Operand A**	**Operand B**	**Description**
arithmetic	Numerical	Numerical	$+, -, \times, \div$
num_mean_groupby_cat	Categorical	Numerical	Group B by A, then compute the mean of B in each group as a new feature
cat_cat_combine	Categorical	Categorical	Create a new categorical feature by combining A and B into a tuple (A, B), then perform frequency encoding on (A, B)
cat_nunique_groupby_cat	Categorical	Categorical	Group B by A, then count how many unique categories of B appear in each group as a new feature
time_diff_groupby_cat	Categorical	Temporal	Group B by A, sort the samples in each group by B, then compute the difference of B between adjacent instances in each group as a new feature

Table 4: The hand-crafted portfolio for hyperparameter optimization.

Config No.	**num_leaves**	**n_estimators**	**learning_rate**
1	60	600	0.01
2	80	600	0.01
3	40	1000	0.01
4	60	1000	0.01
5	40	1500	0.01
6	60	1500	0.01

Table 5: The local search space for hyperparameter optimization.

Hyperparameter	**Search space**	**Search step size**
num_leaves	$[m - 20, m + 20]$	5
n_estimators	$[n - 200, n + 200]$	50
learning_rate	$[\eta - 0.005, \eta + 0.005]$	0.001

Appendix C. Experimental Results

To validate the effect of the novel components proposed in our system, we gradually add them to a vanilla model and evaluate the system performance after each one of them is added. We build the vanilla model by retraining a new model with the last N batches for each test batch and encoding categorical features with label encoding. Then we gradually add the following components to the current model: frequency encoding on each batch separately, streaming frequency encoding, high-order feature combination.

We experiment on the five public datasets in the feedback phase of the challenge. We set the batch number of the historical data buffer as $N = 3$ across all datasets. The score on each dataset is computed as the average normalized AUC score over all test batches. The experimental results are illustrated in Table 6, which validate the effect of the proposed components.

Table 6: Average normalized AUC score of the system on the five public datasets with different components added.

Dataset	**Vanilla model**	**Frequency encoding separately**	**Streaming encoding**	**High-order feature combination**
A	0.4598	0.4641	0.5142	0.5488
B	0.1936	0.2995	0.3020	0.3284
C	0.4586	0.5278	0.5114	0.5794
D	0.3428	0.4704	0.4739	0.5968
E	0.6706	0.7275	0.7379	0.7672