Automatic Feature Engineering through Monte Carlo Tree Search

Yiran Huang[1 [0000−0003−3805−1375]] (✉), Yexu Zhou[1 [0000−0002−8866−7998]], Michael Hefenbrock[1 [0000−0002−7583−2376]], Till Riedel[1 [0000−0003−4547−1984]], Likun Fang[1 [0000−0002−5550−3532]], and Michael Beigl[1 [0000−0001−5009−2327]]

Telecooperation Office, Karlsruhe Institute of Technology, Karlsruhe, Germany
{yhuang,zhou,hefenbrock,riedel,fang,michael}@teco.edu

Abstract. The performance of machine learning models depends heavily on the feature space and feature engineering. Although neural networks have made significant progress in learning latent feature spaces from data, compositional feature engineering through nested feature transformations can reduce model complexity and can be particularly desirable for interpretability. To find suitable transformations automatically, state-of-the-art methods model the feature transformation space by graph structures and use heuristics such as ϵ-greedy to search for them. Such search strategies tend to become less efficient over time because they do not consider the sequential information of the candidate sequences and cannot dynamically adjust the heuristic strategy. To address these shortcomings, we propose a reinforcement learning-based automatic feature engineering method, which we call Monte Carlo tree search Automatic Feature Engineering (mCAFE). We employ a surrogate model that can capture the sequential information contained in the transformation sequence and thus can dynamically adjust the exploration strategy. It balances exploration and exploitation by Thompson sampling and uses a Long Short Term Memory (LSTM) based surrogate model to estimate sequences of promising transformations. In our experiments, mCAFE outperformed state-of-the-art automatic feature engineering methods on most common benchmark datasets.

Keywords: data mining, feature engineering · monte carlo tree search · reinforce learning.

1 Introduction

In many applications, the success of machine learning is often attributed to the experience of experts who use not only the best-fitting algorithms but also extensive domain knowledge. This domain knowledge is often reflected in the pre-processing of raw data: it is transformed step-by-step so that it can be optimally processed by an automated machine learning pipeline. Most of this heuristic search performed by an expert is commonly referred to as feature engineering. Due to limited human resources but ever-growing computing capabilities, automating this search process is becoming increasingly attractive.

Feature engineering can be understood as a combinatorial optimization that attempts to maximize the utility of a subsequent optimization step, i.e., fitting the model. By

employing explicit feature engineering, as opposed to deep learning (e.g., Long Short Term Memory (LSTM) [10] in [1]), we obtain a tighter control over the model space.

Furthermore, good feature engineering can increase the robustness (generalizability to unknown data) and interpretability (predictability of decisions based on input features) of the overall machine learning architecture.

However, there are several challenges in automatically searching for useful features made up of sequences of atomic mathematical transformations such as addition (add), logarithm ($\log$), or a sine function ($\sin$). First, the search space grows large very quickly, as the number of possible transformation sequences grows exponentially with their length and the atomic transformations allowed. Second, evaluating a potentially promising transformation sequence can be time-consuming, as it requires training and evaluation of a machine learning model. Both of these features make the search challenging and require methods that search the space efficiently.

To address these challenges, Cognito [6] models the exploration of the transformation space with a transformation tree and explores the tree with some handcrafted heuristic traversal strategies such as depth-first, global traversal, or balanced traversal. Furthermore, the recently proposed reinforcement-based approach [7] applies a Q-learning algorithm and approximates the Q value with linear approximation to automate feature engineering.

While these methods achieve good results, our hypothesis is that they can be significantly improved by addressing two aspects, namely:

- **Choice of search hyperparameters and dynamic adaptation of the heuristic strategy**: A serious problem with an approach that relies entirely on guidance is the tendency to fall into local optima. Strategies like ϵ-greedy and Upper Confidence Bound (UCB) [17] can mitigate this problem, however, both need careful tuning of the initial hyperparameters that also control the dynamic adaptation of their search strategy.
- **Sequential information of the composite transformations**: New features can be transformations of existing features. Such compositions are sensitive to the order in which the atomic transformations are applied. State of the art feature engineering methods approximate the performance of a given transformation sequence with a linear model [7] or a deep convolutional neural network [8], which do not exploit the sequential information (order) contained in the composite transformation.

To address these shortcomings, we present a novel algorithm called Monte Carlo tree search for Automatic Feature Engineering (mCAFE). We choose Thompson sampling as an automatically adjusting selection policy, in combination with an LSTM network to capture the sequential information in the feature transformation sequences, while the main structure follows a Monte Carlo Tree Search (MCTS) [9].

Our contributions can be summarized as 1) we leverage Thompson sampling to guide the exploration, thus avoiding the parameters initialization and strategy dynamic adjustment problem. 2) we utilize sequential information of composed transformations by training an LSTM-based surrogate model for predicting the expected reward of a transformation sequence to a given dataset. 3) we evaluated the algorithm on common benchmark datasets (see [7]) and achieved improvements on most of them.

2 Related work

In recent years, a large number of research results have emerged for domain-specific feature engineering. The work of [2] investigates how to share information through feature engineering in multi-task learning tasks, and [3] tries to find suitable features to improve the class separation. However, less new research has been done on feature engineering applicable to all data types. FCTree, proposed in [12], uses the original and constructed features as the splitting point to partition the data through a decision tree. It constructs local features where the local error is high and the features constructed so far are not well predicted. FEADIS [13] uses a random combination of mathematical functions, including ceiling, modulus, sin, and feature selection methods to construct new features. Of these, features are then selected greedily and added it to the original features. The Data Science Machine (DSM) [4] applies transformations to all features at once. Then, feature selection and model optimization are performed on the generated dataset. A similar procedure was also applied in [5]. In contrast, ExploreKit [14] increases the constructed features iteratively. To overcome the exponential growth of the feature space, ExploreKit uses a novel machine learning-based feature selection approach to predict the usefulness of new candidate features. Similarly, Cognito [6] introduced the notion of a tree-like exploration of the transformation space. Through a few handcrafted heuristics traversal strategies, such as depth-first and global-first strategy, Cognito can efficiently explore the set of available transformations. However, several factors, such as episode budget constraints, are beyond the consideration of the strategy. As an improvement, a reinforcement learning-based feature engineering method was proposed in [7] to explore the available feature engineering choices under a given budget. Finally, LFE [15] considers each feature individually and predicts the best transformation of each feature through the learning-based method. However, none of these methods takes the order of the transformations of the features into account. More recently, a graph-based method was proposed in [8] that guides the exploration of the transformation space with a deep neural network.

3 Methodology

We model the feature engineering problem as a classic episode-based reinforcement learning problem consisting of an agent interacting with the environment. The search starts from the initial state representing the original dataset $D_0 \in \mathcal{D}$, where $\mathcal{D}$ denotes the state space. From D_0, a transformation $t \in \mathcal{T}$ (action) can be chosen to transform the dataset (all the features contained in the dataset) according to t. The new state D' is then obtained by the concatenation of the data of the old state D with $t(D)$, i.e., $D' = [D, t(D)]$. Through this, the new state contains all information (data) from the previous states, which can be seen as a Markov property. Finally, for each state D, a machine learning model can be trained on D to obtain its n fold cross-validation performance. However, since we seek to obtain the best sequence of length L, we further define a feature engineering pipeline, as an ordered sequence $(t_1, \cdots, t_i, \cdots, t_L)$ consisting of L transformations. The i-th entry of the sequence denotes the decision in the i-th step, i.e., the transformation to apply to the data in order to generate new features.

Overall, the environment can be summarized with a 3-tuple ($\mathcal{D}$, $\mathcal{T}$, r), denoting the state space $\mathcal{D}$, the transformation (action) space $\mathcal{T}$ and the rewards $r \in \{0, 1\}$. The reward thereby expresses whether a transformation pipeline of length L improved over the best performing model found so far.

Monte-Carlo Tree Search (MCTS) defines a class sampling-based tree search algorithms used to find optimal decisions in vast search domains and has been successfully applied to related problems like feature subset selection [25]. To deal with huge search spaces, MCTS models the search space as a tree structure and explores the tree iteratively. It gradually favors the most promising regions in the search tree given an arbitrary evaluation function.

Evidently, our search space of feature transformations can span such a tree, which allows the application of MCTS to find a feature set that contains features constructed by an optimal transformation pipeline on the original dataset. We discuss the construction of the tree in the following, alongside the selection policy (Thompson sampling) and a surrogate model-based (LSTM) expansion policy. Finally, we will outline the overall mCAFE algorithm. In contrast to problems like feature subset selection, the ordering of the nodes inside the tree is of critical importance in our case.

3.1 The transformation tree

We illustrate the reinforcement task with a transformation tree of maximum depth L, in which each node represents a state (dataset), each edge represents an action (transformation) and each path from the root to a leaf node represents a feature engineering pipeline. Additionally, each edge in the tree is associated with a distribution, which shows the mean success (reward = 1) probability of taking the action at its parent state. The nodes in the tree are divided into two categories: (1) *root node* D_0 is the initial state for each pipeline and represents the original dataset; (2) *derived nodes* D_i, where $i > 0$, has only one parent node D_j, $i > j \geq 0$ and the connecting edge responds to the action $t \in \mathcal{T}$ applied to the parent node, i.e., $D_i = [D_j, t(D_j)]$. In this way, we translate the feature engineering problem into a problem of exploring the transformation tree to find the node that maximizes the expected reward.

Fig. 1 shows a full transformation tree for a pipeline of $L = 2$ and two available actions $\mathcal{T} = \{\log, \mathrm{add}\}$. Each node in the tree is a candidate dataset for the feature engineering problem. For example, the derived nodes D_4 and D_5, represent

$$D_4 = \{D_0, \mathrm{add}(D_0), \log(D_0), \log(\mathrm{add}(D_0))\}\,,$$
$$D_5 = \{D_0, \log(D_0), \mathrm{add}(D_0), \mathrm{add}(\log(D_0))\}\,.$$

Note that, although the transformations in D_4 and D_5 are the same, the resulting dataset is not identical due to the order in which the transformations are applied.

We can find the optimal node by traversing this tree. However, the complexity of this task grows exponentially as L and the number of available transforms $|\mathcal{T}|$ becomes larger. Since traversing all possible nodes of the tree is prohibitive, mCAFE focuses on optimizing the selection policy π_s and expansion policy π_e to reduce the number of evaluations required to find a good transformation sequence.

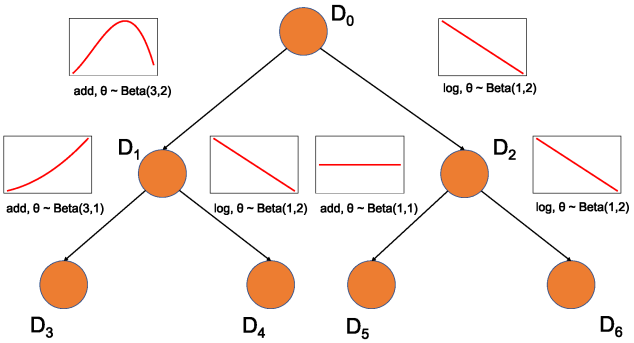

Fig. 1. Representation of feature transformation with a tree structure. Here, each node corresponds to a state D and each edge corresponds to a transformation (action). The distributions on the edges display the distribution over the mean success (reward = 1) probability when taking the action in the parent state.

3.2 The selection policy

The selection policy π_s determines the balance between exploration and exploitation. It guides the selection for known parts of the MCTS. The UCB and ϵ-greedy are the two most commonly used selection policies, for which also strong theoretical guarantees on the regret[1] can be proven. While they have proven successful in various reinforcement learning settings, they are not ideal for the application of feature engineering. This is mainly due to their requirement to explicitly define the exploration and exploitation trade-off through ϵ in ϵ-greedy and λ in the UCB. Additionally, ϵ greedy does not adapt the trade-off dynamically but always pursues ϵ % exploration. To address these problems, we make use of the Thompson sampling as the selection policy. In the following, we introduce Thompson sampling and adapt it to the feature engineering case.

Consider the state space $\mathcal{D}$, the action space $\mathcal{T}$ and rewards $r \in \{0, 1\}$. Thompson sampling selects an action based on the probability of it being the optimal action. Representing the set of N observations $\mathcal{O} = \{(r, t, D)\}^N$, where $D \in \mathcal{D}$, $t \in \mathcal{T}$, we model the probability of different rewards of each action with a parametric likelihood distribution $p(r|t, D, \theta)$ depending on the parameters θ. The prior distribution of these parameters is denoted by $p(\theta)$. Consequently, the posterior distribution given a set of observations $\mathcal{O}$ can be calculated using Bayes rule, i.e., $p(\theta|\mathcal{O}) \propto p(\mathcal{O}|\theta)p(\theta)$. Thompson sampling implements the selection policy π_s by sampling a parameter θ from the posterior distribution $p(\theta|\mathcal{O})$, and taking the action that maximizes the expected reward. Hence,

$$\pi_s(D) = \underset{t \in \mathcal{T}}{\operatorname{argmax}} \quad \mathbb{E}\left[r|t, D, \theta\right] \quad \text{where} \quad \theta \sim p\left(\theta|\mathcal{O}\right). \tag{1}$$

Since, in the case of feature engineering, each state $D \in \mathcal{D}$ satisfies the Markov property, we can simplify the problem of which action to take on state D, to whether

[1] The amount we lose for not selecting optimal action in each state

taking the action $t \in \mathcal{T}$ leads to a performance improvement. This can be modeled as a classic Bernoulli bandit problem, where the variable $\theta = (\theta_1, \theta_2, \cdots)$ denotes the expected values of a Bernoulli random variable expressing the probability of taking the selected action in given a state (and obtaining a reward of one). The distribution of the parameter θ_t can be modeled through a beta distribution

$$p(\theta_t|\alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta_t^{\alpha-1}(1-\theta_t)^{\beta-1},$$

where Γ is the Gamma function. $\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}$ serves as a normalisation constant that ensures the integration of the density function over (0,1) is 1. The parameters α and β control the shape of the distribution and the mean of the distribution is $\frac{\alpha}{\alpha+\beta}$. It denotes the expectation that taking the corresponding action will lead to performance improvement. The higher α, the larger the mean and therefore the probability of the action to be selected. On the other hand, the larger β, the lower the probability.

The beta distribution is conjugate to the Bernoulli distribution (i.e., the posterior distribution $p(\theta|\mathcal{O})$ inherits the functional form the prior distribution $p(\theta)$). Given an observed sample $O = (r, t, D)$, the posterior distribution of the parameters θ is given by

$$\theta_{t'} \sim \mathrm{Beta}\left(\alpha + \mathbb{1}_{r=1,t'=t}, \beta + \mathbb{1}_{r=0,t'=t}\right), \quad t' \in \mathcal{T}. \tag{2}$$

The parameter α is incremented when the action led to an improvement in performance. Otherwise, the parameter β is incremented. In this view, α represents the number of successes in the Bernoulli trial and β represents the number of failures. Furthermore, the support of the beta distribution is $(0, 1)$, independent of the parameterization. This ensures that there is always a nonzero probability for each action to be selected. Consequently, there is always a nonzero probability to take each path in the tree.

Fig. 1 shows an example of the tree representation. Each edge in the tree maintains a beta distribution $\mathrm{Beta}(\alpha, \beta)$. By comparing the two transformations on D_0 with the same β, we can see that the higher the value of α the more the distribution is shifted towards sampling larger values (higher probabilities of success). In each step, an edge is selected based on the sampling result. This ensures the priority of high-quality edges while also allowing inferior edges to be selected occasionally. By using Thompson sampling as the selection policy, we avoid choosing hyperparameters to balance the exploration and exploitation trade-off. In contrast, the trade-off is adjusted dynamically through the posterior distribution of the parameter θ, which is updated along with the observation. Even though α and β represent hyperparameters, their choice is arguably more intuitive as $\alpha = \beta = 1$ describes a uniform distribution.

The requirement to construct and sample from a beta distribution for each action may rise efficiency concerns, as this process is slow compared to an ϵ-greedy selection. However, this is not an issue for feature engineering as in each episode, the selection phase takes little time compared to the other phases of the algorithm. This will be further explored in Section 4 (see Table 1).

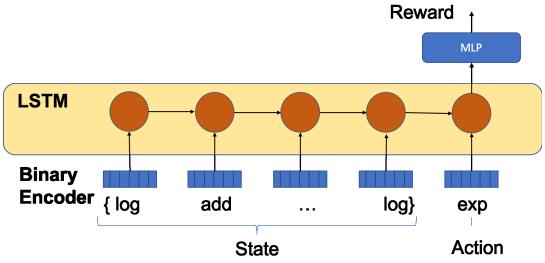

Fig. 2. The Surrogate network consist of 2 LSTM layers of size 32 and a two fully connected layers of size 32 with a ReLU activation function.

3.3 The expansion policy

The selection policy π_s guides the selection of actions in parts of the search space that have been explored. Outside of the explored search space and beyond the leaves of the MCTS, the expansion policy π_e guides the selection of the actions t. It expands the child nodes to the tree and selects the one with the maximum expectation reward (Q value) as the next exploration candidate

$$\begin{aligned} \pi_e(D) &= \underset{t \in \mathcal{T}}{\operatorname{argmax}} \quad \mathbb{E}\left[r|t, D, \theta\right] \\ &= \underset{t \in \mathcal{T}}{\operatorname{argmax}} \quad \hat{Q}(D, t). \end{aligned} \tag{3}$$

Since the state space $\mathcal{D}$ is huge and it is infeasible to calculate the expectation directly, mCAFE models it with a surrogate network $\hat{Q}(D, t)$ as shown in Fig. 2. This network takes the selected action t and the action sequence, which was used to generate the leaf node state D as input, and outputs the expectation reward of taking this action at the state. Considering the characteristic input and order information in the sequence, the surrogate network consists of three parts, namely a binary encoder which takes an action as input and outputs a binary code, one LSTM layer with a hidden size of 32 to deal with different lengths of the input sequence and capture their sequential information, and a fully connected layer with an input size of 32 and ReLU activation function to map the LSTM output to the expectation.

Since each edge in the tree maintains a beta distribution, we collect training data from all the existing edges in the tree and update the surrogate model after each iteration (episode). With the help of the surrogate model $\hat{Q}(D, t)$, the expansion policy can be defined as selecting the action t that maximizes the expectation reward predicted by the surrogate model.

3.4 The mCAFE algorithm

The mCAFE applies MCTS to explore the target space, while the selection policy gradually biases the actions taken towards the more promising regions of search space in

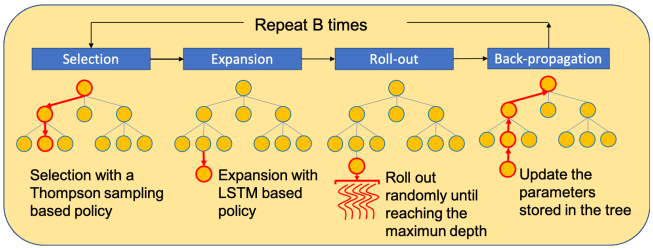

Fig. 3. The mCAFE framework: each iteration (episode) includes four phases: selection, expansion, roll-out, back-propagation. B is the number of iterations.

order to find the optimal sequence of actions. It follows the general MCTS scheme, where the main four phases have been modified as follows (Fig. 3):

Selection Starting from the root node, mCAFE selects the child node according to the selection policy π_s iteratively until it reaches a leaf node.

Expansion In a leaf node of the transformation tree, all the available child nodes are expanded to the tree. One of these nodes is selected to explore according to the expansion policy π_e.

Roll-out Instead of the performance of the current node, we are interested in whether the expectation performance of its descendant nodes has been better than the best performance so far. To achieve this, mCAFE combines the n-folds cross-validation and the general Roll-out process by the following: Assuming that the current node is of depth l in the MCTS tree, mCAFE completes the feature engineering pipeline by sampling $L - l$ transformations from $\mathcal{T}$ randomly with replacement, where L is the predefined length of the pipeline (transformation sequence). This process is repeated n times to get n different pipelines (transformation sequences), where n is the number of iterations in n fold cross validation. A reward of $r = 1$ is returned if the mean evaluation score of the transformation sequences is higher, else $r = 0$.

Back-propagation The reward from the roll-out process is back propagated along the path from the node selected in the expansion process to the root node in the tree, updating the parameters α, β in each edge on the path with the update rule (see Section 3.2).

The algorithm stops after the computational budget is exhausted, e.g. the algorithm stops when the number of episodes reaches 100 in the experiment.

Fig. 4 shows an example of an episode of the mCAFE algorithm. Starting from the root node D_0, it selects explored nodes according to the selection policy π_s until reaching the leaf node D_4. Then an unexplored node D_7 is selected and expanded to the tree according to the expansion policy π_e. If the depth of the current node l (expanded node D_7) is smaller than the predefined pipeline length L (max depth), an action is selected according to the random policy and applied to the current node to create a new node, which is regarded as the new current node. This process is repeated until the depth of the new node l is larger than the pipeline length L. Finally, the current node is evaluated and its reward is back-propagated, updating the parameters of the beta distributions along the path from D_0 to D_7. Since $r = 1$, the α of all edges along

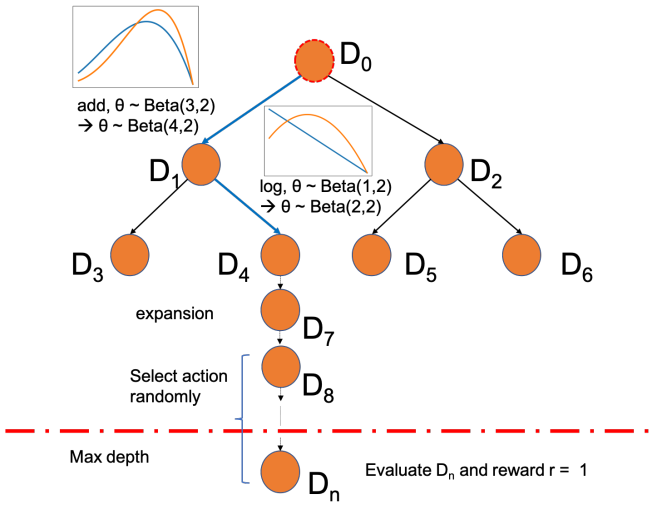

Fig. 4. Example of an episode of mCAFE. The beta distributions of the edges in the selected path are displayed next to the corresponding edge. Blue denotes the distribution before back-propagation and orange after back-propagation.

the path are incremented, while the β remain unchanged. Correspondingly, the beta distribution of each edge in the path is slightly shifted to the right, and the probability of selecting the corresponding actions is increased.

4 Evaluation

In this section, we design six different experiments to address the following questions: 1) How well does the mCAFE approach compare to the state-of-the-art [7]? 2) Is the sampling-based selection policy necessary for the mCAFE algorithm? 3) Is the sequential information of the transformation sequence important for the prediction of the Q value? 4) Is the surrogate-based expansion policy necessary for the mCAFE algorithm? 5) How should the hyperparameter L (pipeline length) be chosen in the mCAFE algorithm? 6) How does mCAFE perform for different predictive models?

For the first five experiments, we use the same benchmarks as [7]. For this, we tried to reproduce this previous work. Some datasets were removed from the experiment since either the results of the base model differ considerably from those in [7], e.g., 'Amazon Employ' and 'Whine Quality Red'. Additionally, 'Wine Quality White', 'Higgs Boson', 'SVMGuide3', 'Bikeshare DC' were removed as they displayed a different dataset size compared to the one cited in [7]. To overcome this problem for future work, we published our code and datasets at https://github.com/HuangYiran/MonteCarlo-AFE.git.

We run the last experiment on the Automatic Machine Learning (AutoML) benchmark datasets [24]. We keep the same hyperparameter setting as in the first four experiments for both our work and the baseline.

In the experiments, we use episode budgets instead of time budgets for the following three reasons. 1) Different from some other optimization tasks, the time spent on candidate evaluation for feature engineering tasks dominates the overall time spent. This time is inevitable for all the evaluation-oriented optimization methods when dealing with feature engineering tasks. Table 1 shows the average percentage of time taken by the mCAFE for each step in the first 20 episodes. The roll-out phase, which consist of random transformation selection and candidate evaluation, takes up an average of 97% of the overall time. 2) The run time varies greatly across datasets. It is influenced by the size of the data and the sensitivity of the data to different transformations. 3) The algorithm implementation and operating environment have a significant impact on the run time.

Table 1. Average percentage of time for each process in the first 20 episodes.

	Size		Time spent in percentage (%)			
Dataset	Rows	Feat.	Selection	Expansion	Roll-out	Back-propagation
SpecFact	267	44	0.01	0.04	96.94	3.01
PimaIndian	768	8	0.02	0.05	97.29	2.66
Lymphography	148	18	0.01	0.04	96.98	2.97
Ionosphere	351	34	0.01	0.06	96.37	3.56
AP-omentum-ovary	275	10936	0.01	0.02	98.57	1.40
SpamBase	4601	57	0.01	0.01	98.74	1.24

For the first five experiments, we use the random forest model of the sklearn package (version 0.24) with default parameters and an episode budget of $B = 100$ as in [7], in order to make the result more comparable.

We set the pipeline length to $L = 4$ according to the result of the third experiment and all the beta distributions are initialized with (1, 1) for a uniform prior. To reduce the computation time, we sub-sample to the dataset with a large number of data points. For the sub-sampling, up to 10^4 data points are considered. To ensure comparability, we did not tune any hyperparameters of the feature engineering algorithms to suit a concrete data set or prediction model (which we changed for the last experiment). Considering the imbalanced datasets, we apply the F1-score to assess the classification performance and use 1 - RAE (Relative Absolute Error) as in [7] as the metric for the regression task. All performances are obtained under 5-folds cross-validation, which also means the parameter n in roll out process is set to 5.

In the experiment, we used the transformation functions $\mathcal{T} = \{$ Log, Exp, Square, Sin, Cos, TanH, Sigmoid, Abs, Negative, Radian, K-term, Difference, Add, Minus, Product, Div, NormalExpansion, Aggregation, Normalization, Binning $\}$.

4.1 Performance of mCAFE

We evaluate the improvement of mCAFE algorithm in comparison with the following methods, namely the *original dataset* (Base), a *Reinforcement-Based Model (RBM)* with discount factor 0.99, learning rate 0.05 and B 100, a *tree-heuristic model (Cognito)* with global search heuristic for 100 nodes, *random selection* selecting a transformation from the available transformation set and applying it to one or more features in the original dataset. If the addition of the new features leads to an improved performance, we keep the new feature. This process is repeated 100 times to get the final dataset.

We summarized the performance of the methods in Table 2. It can be seen that, mCAFE achieves the best score in all the regression datasets against the reinforcement-based model and achieved superior results on most of the classification datasets. However, mCAFE performs worse than the reinforcement based model in two datasets on 'Credit Default' and 'SpamBase'. Among them, the difference on 'SpamBase' is not significant. From these results, we conclude that the proposed method performs better than the state-of-the-art automatic feature engineering approaches.

Table 2. Comparing performance of without feature engineering (Base), reinforcement-based model (RBM) [7], Cognito [6], random selection and mCAFE in 100 episodes using 15 open source datasets. Classification (C) tasks are evaluated with F1-score and regression (R) tasks are evaluated with (1-relative absolute error).

Dataset	C/R	Rows	Feat.	Base	RBM	Cognito	Random	mCAFE
SpecFact	C	267	44	0.686	0.788	0.790	0.748	**0.855 ± 0.036**
PimaIndian	C	768	8	0.721	0.756	0.732	0.709	**0.773 ± 0.026**
German Credit	C	1001	21	0.661	0.724	0.662	0.655	**0.764 ± 0.026**
Lymphography	C	148	18	0.832	0.895	0.849	0.680	**0.967 ± 0.016**
Ionosphere	C	351	34	0.927	0.941	0.941	0.934	**0.962 ± 0.014**
Credit Default	C	30000	25	0.797	**0.831**	0.799	0.766	0.796 ± 0.006
AP-omentum-ovary	C	275	10936	0.615	0.820	0.758	0.710	**0.831 ± 0.036**
SpamBase	C	4601	57	0.955	**0.961**	0.959	0.937	0.953 ± 0.016
Openml_618	R	1000	50	0.428	0.589	0.532	0.428	**0.743 ± 0.015**
Openml_589	R	1000	25	0.542	0.687	0.644	0.571	**0.776 ± 0.018**
Openml_616	R	500	50	0.343	0.559	0.450	0.343	**0.622 ± 0.010**
Openml_607	R	1000	50	0.380	0.647	0.629	0.411	**0.803 ± 0.010**
Opemml_620	R	1000	25	0.524	0.683	0.583	0.524	**0.765 ± 0.012**
Openml_637	R	500	50	0.313	0.585	0.582	0.313	**0.637 ± 0.021**
Openml_586	R	1000	25	0.547	0.704	0.647	0.549	**0.783 ± 0.020**

4.2 Ablation study

The proposed selection strategy and extension strategy are the most important components supporting the performance of the algorithm. To verify their importance, we designed two ablation experiments.

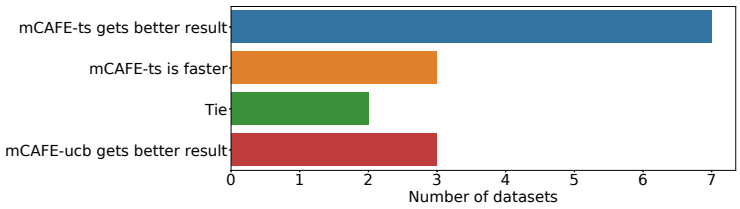

Fig. 5. Comparing the performance between mCAFE-ucb and mCAFE.

Selection policy We apply the traditional UCB with ϵ-greedy policy as selection policy in the mCAFE algorithm (mCAFE-ucb) and compare its performance with the proposed model, which uses Thompson sampling based selection policy (mCAFE-ts). The parameter λ of UCB is set to 1.412 as proposed in [23], the ϵ is set to 0.1 while the mCAFE algorithm keeps the same setting as the last experiment. Performance of the classification task is measured with F1-score and regression task is measured with (1-relative absolute error). Fig. 5 divides the results of the comparison into four categories. 1) mCAFE-ts gets better result: the result performance measured is higher than with mCAFE-ucb. 2) mCAFE-ts is faster: the number of episodes needed to obtain the same result is larger on MCAFE-ucb. 3) Tie: mCAFE-ucb obtains the same result and requires the similar number of epochs (difference smaller than 5). 4) mCAFE-ucb gets better result: mCAFE-ucb obtains the same results and requires a smaller number of episodes than mCAFE). The result in Fig. 5 demonstrates the importance of the selection strategy. mCAFE achieved better performance on 64.7% of the datasets and tied on 13.3%.

Expansion Policy In the expansion process, we use an LSTM neural network to approximate the expectation reward (Q value) of taking an action, since it can capture the sequential information of the transformation sequence. To prove that this information is important for the Q value prediction, we designed an experiment to compare the performance of using MLP and LSTM as the surrogate model in mCAFE.

To make the trained models comparable, the MLP model here contains two 76 dimension hidden layers so that it has a similar number of parameters as the LSTM surrogate model mentioned above. We use the mean absolute error as the evaluation criterion. A smaller value indicates a better model. Both models are trained with 100 epochs. We can see from Fig. 6 that, the LSTM obtains significantly better results than the MLP model in all datasets.

To evaluate its contribution to mCAFE, we compare the performance of the following three models, namely mCAFE with LSTM-based expansion policy, mCAFE with random expansion policy, mCAFE with greedy expansion policy, which always expand the best action explored.

All three models used the same initial parameters as the last experiment. Each model is evaluated 10 times on each dataset. The performance of the models on the regression datasets is displayed with the box plot in Fig. 7. We can see that mCAFE with neural network achieves best performances on all datasets except two, where mCAFE

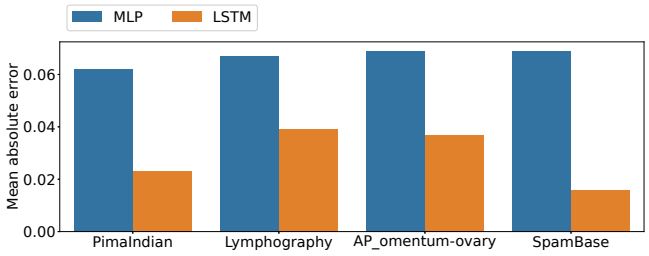

Fig. 6. Comparing the performance of MLP and LSTM model in predicting the Q value.

Fig. 7. Comparing performance of mCAFE with neural network expansion policy (with nn), mCAFE with random expansion policy (with random) and mCAFE with fix expansion policy (with fix) on all the regression dataset. Classification task is evaluated with F1-score and regression task is evaluated with (1-relative absolute error).

with random policy performs better on dataset 'Openml_618' and mCAFE with fixed expansion policy performs better on dataset 'Openml_586'. For all datasets except 'Openml_618' and 'Openml_586', mCAFE with neural network expansion policy also loses to mCAFE with random expansion policy on dataset 'AP-omentum-ovary'.

The main differences between these three expansion approaches are the usages of previous observations and the dispersion of the selected actions. mCAFE with a fixed expansion policy selects actions greedily according to the performance of the actions in the first layer. This selection process is stable, however, hinders the exploration of new transformations, which is likely the reason for its failure in most cases. mCAFE with neural network expansion policy captures the performance information of previous observation and uses it in the prediction of the reward expectation of future actions.

4.3 Length of feature engineering pipeline

The length of the feature engineering pipeline L determines the number of actions selected in each roll-out step, as well as the length of the final transformation sequence. It influences the performance of mCAFE algorithm not only on the final result but also on the time and memory consumption. In general, the larger L, the larger the time and memory consumption and, at the same time, the larger the number of features after the

Table 3. The performances of mCAFE with different predictive models on AutoML benchmark dataset [24]. The improvements brought by the mCAFE are shown in the parentheses. Classification task is evaluated with F1-score and regression task is evaluated with (1-relative absolute error)

	Base performance				Performance with mCafe			
AutoML benchmark datasets	Rbf-svm	Linear-svm	Linear model	Decision tree	Rbf-svm	Linear-svm	Linear model	Decision tree
shuttle	0.901	0.996	0.859	**0.998**	0.996 (0.095)	0.998 (0.002)	0.996 (0.137)	**0.998** (0.001)
phpZLgL9q	0.407	0.432	**0.503**	0.454	0.407 (0.000)	0.432 (0.000)	**0.503** (0.000)	0.454 (0.000)
phpyM5ND4	0.563	0.848	0.775	**0.920**	0.752 (0.189)	0.943 (0.095)	0.893 (0.118)	**0.959** (0.039)
phpvcoG8S	0.471	0.426	0.468	**0.572**	0.561 (0.090)	**0.579 (0.153)**	0.529 (0.061)	0.578 (0.006)
phpQOf0wY	0.320	0.385	0.457	**0.698**	0.620 (0.300)	0.493 (0.108)	0.682 (0.225)	**0.698** (0.000)
phpnBqZGZ	0.016	**0.768**	0.700	0.747	0.467 (0.451)	**0.768** (0.000)	0.760 (0.060)	0.750 (0.003)
phpmPOD5A	**0.919**	0.749	0.869	0.912	**0.919** (0.000)	0.908 (0.159)	0.908 (0.039)	0.913 (0.001)
phpmcGu2X	0.930	**0.953**	0.941	0.854	**0.970** (0.040)	0.953 (0.000)	0.941 (0.000)	0.854 (0.001)
phpMawTba	0.650	0.589	0.716	**0.797**	0.802 (0.152)	0.787 (0.198)	0.787 (0.071)	**0.820** (0.023)
phpkIxskf	0.833	0.767	0.847	**0.877**	0.883 (0.050)	0.883 (0.116)	0.870 (0.023)	**0.892** (0.015)

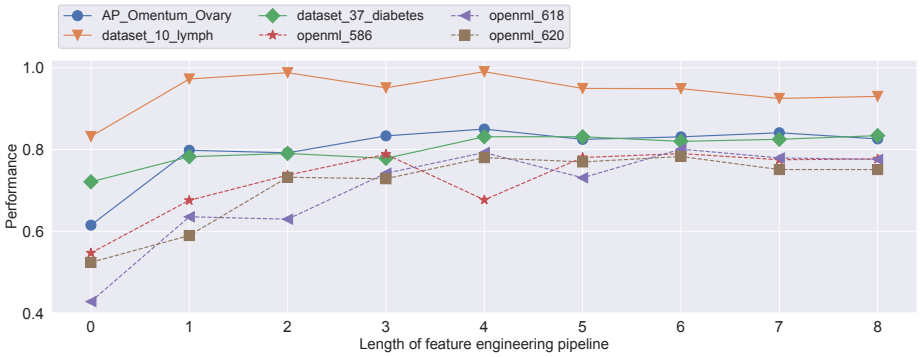

Fig. 8. Comparing performance of mCAFE with different maximum pipeline length on 3 classification datasets (F1-score) and 3 regression datasets (1-relative absolute error).

transformation. To achieve the best results with limited resources, we conducted an experiment to find a suitable parameter L by comparing the performance of the algorithm with different L values.

Fig. 8 shows the relationship between length L and the best performance displayed by the mCAFE algorithm for six datasets. $L = 0$ signifies the performance of the random forest model on the base dataset. We can see that some achieve good results with $L = 1$, however, increasing L can further improve its performance. Most of the datasets reach the maximum performance with $L = 4$, while a small fraction shows a lower performance. This may be due to the random selection in the starting process. The performance on 'Dataset_10_lymph' is worse for higher L, which is probably due to overfitting. From this experiment, we can conclude that the optimal L depends on the dataset. However, $L = 4$ should be a suitable choice in most cases.

4.4 Performances of mCAFE on different predictive models

Different predictive models differ in their performance and sensitivity to mCAFE on the same dataset. To test this conjecture, we tested the performance of the mCAFE with the following predictive models on the AutoML benchmark datasets separately, namely Rbf-svm, Linear-svm, Linear model, Decision tree.

Table 3 summarizes the results of the experiment. We can see that mCAFE brings performance improvements to most of the datasets. The value of feature engineering is more prominent for linear and svm models. It is worth noting that although the performance of each model on the original dataset varies greatly, the performance obtained after mCAFE tends to be close.

5 Conclusion and future work

In this paper, we show that existing automatic feature engineering methods can be significantly improved by building upon two simple observations. Our results suggest that feature engineering should make use of sequence information, incorporating composite transformations into the surrogate model. In addition, a suitable selection policy should be chosen. The proposed novel MCTS-based framework uses an LSTM neural network for the expansion policy to explore the search space efficiently. Furthermore, Thompson sampling is employed to address the trade-off between exploration and exploitation in the selection policy. Through this, we manage to obtain superior results to state-of-the-art methods for automatic feature engineering on the majority of commonly used benchmarks. We believe that further improvements could be made to the algorithms by adding transformations that might also reduce redundant and irrelevant feature during the construction.

6 Acknowledgements

This work was partially funded by the Ministry of The Ministry of Science, Research and the Arts Baden-Wuerttemberg as part of the SDSC-BW and by the German Ministry for Research as well as by Education as part of SDI-C (Grant 01IS19030A)

References

1. Zhou, Y., Hefenbrock, M., Huang, Y., Riedel, T., Beigl, M. Automatic Remaining Useful Life Estimation Framework with Embedded Convolutional LSTM as the Backbone. ECML PKDD 2020.Springer, doi:10.1007/978-3-030-67667-4_28
2. Guo, P., Deng, C., Xu, L., Huang, X., Zhang, Y. Deep Multi-task Augmented Feature Learning via Hierarchical Graph Neural Network. ECML PKDD 2021. Springer, doi:10.1007/978-3-030-86486-6_33
3. Schelling, B., Bauer, L.G.M., Behzadi, S., Plant, C. Utilizing Structure-Rich Features to Improve Clustering. ECML PKDD 2020. Springer, doi:https://doi.org/10.1007/978-3-030-67658-2_6
4. Kanter, J.M., Veeramachaneni, K. Deep feature synthesis: Towards automating data science endeavors. DSAA 2015, IEEE, doi:10.1109/DSAA.2015.7344858

5. Lam, H.T., Thiebaut, J.M., Sinn, M., Chen, B., Mai, T., Alkan, O. One button machine for automating feature engineering in relational databases.arXiv 2017, doi:10.48550/arXiv.1706.00327
6. Khurana, U., Turaga, D., Samulowitz, H., Parthasrathy, S. Cognito: Automated feature engineering for supervised learning. ICDMW 2016,IEEE, doi:10.1109/ICDMW.2016.0190
7. Khurana, U., Samulowitz, H., Turaga, D. Feature engineering for predictive modeling using reinforcement learning. AAAI 2018. PKP, doi:10.1609/aaai.v32i1.11678
8. Zhang, J., Hao, J., Fogelman-Soulié, F., Wang, Z. Automatic feature engineering by deep reinforcement learning.AAMAS 2019, ACM, ACM DL 10.5555/3306127.3332095
9. Coulom, R. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. CG 2006. Springer, doi:10.1007/978-3-540-75538-8_7
10. Graves, A. Long Short-Term Memory. In: Supervised Sequence Labelling with Recurrent Neural Networks. Studies in Computational Intelligence, vol. 385 (2012). Springer, doi:10.1007/978-3-642-24797-2_4
11. Markovitch, S., Rosenstein, D. Feature Generation Using General Constructor Functions. Machine Learning, vol. 49, Springer (2002), doi:10.1023/A:1014046307775
12. Fan, W., Zhong, E., Peng, J., Verscheure, O., Zhang, K., Ren, J., et al. Generalized and heuristic-free feature construction for improved accuracy. SDM 2010, doi:10.1137/1.9781611972801.55
13. Dor, O., Reich, Y. Strengthening learning algorithms by feature discovery. Information Sciences, vol. 189, Elsevier (2012), doi:10.1016/j.ins.2011.11.039
14. Katz, G., Shin, E.C.R., Song, D. Explorekit: Automatic feature generation and selection. ICDM 2016, IEEE, https://doi.org/10.1109/ICDM.2016.0123
15. Nargesian, F., Samulowitz, H., Khurana, U., Khalil, E.B., Turaga, D.S. Learning Feature Engineering for Classification. IJCAI 2017, doi:10.24963/ijcai.2017/352
16. Gelly, S., Silver, D. Monte-Carlo tree search and rapid action value estimation in computer Go. Artificial Intelligence, vol. 175(11), Elsevier, doi:10.1016/j.artint.2011.03.007
17. Auer, P. Using confidence bounds for exploitation-exploration trade-offs. JMLR, vol. 3 (Nov 2002),ACM DL 10.5555/944919.944941
18. Kocsis, L., Szepesvári, C. Bandit Based Monte-Carlo Planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds) Machine Learning: ECML 2006. Springer, doi:10.1007/11871842_29
19. Silver, D., Tesauro, G. Monte-Carlo simulation balancing. In Proceedings of the 26th Annual International Conference on Machine Learning, (2009), doi:10.1145/1553374.1553495
20. Rimmel, A., Teytaud, F. Multiple Overlapping Tiles for Contextual Monte Carlo Tree Search. EvoApplications 2010. Springer, doi:10.1007/978-3-642-12239-2_21
21. Fernández-Delgado, M., Cernadas, E., Barro, S., Amorim, D. Do we need hundreds of classifiers to solve real world classification problems?. JMLR, vol. 15(1), ACM DL 10.5555/2627435.2697065
22. Helmbold, D. P., Parker-Wood, A. All-Moves-As-First Heuristics in Monte-Carlo Go. IC-AI 2009, CiteSeer 10.1.1.183.7924
23. Coquelin, P. A., Munos, R. Bandit algorithms for tree search. arXiv preprint 2007, doi:10.48550/arXiv.cs/0703062
24. Gijsbers, P., LeDell, E., Thomas, J., Poirier, S., Bischl, B., Vanschoren, J. An open source AutoML benchmark. arXiv preprint 2019, doi:10.48550/arXiv.1907.00909
25. Gaudel, R. , Sebag, M. Feature selection as a one-player game, ICML 2010, ACM DL 10.5555/3104322.3104369