1. Simplification of Feature Optimization Problem -- Approximating the Optimal Feature

Given a dataset D with an original feature set F_0, a downstream ML model M, and an operator set O, There exists a new optimal feature set F^* constructed from F_0 and O, which can lead to the maximal improvement in the performance of the downstream model M. In an ideal scenario, **if the representational power of the operator set O is unlimited, this set may contain an optimal feature f^*, which directly describes the relationship between the original feature set F_0 and the target label, thereby reducing the prediction error of the downstream ML model M to near 0.**

2. Feature Similarity Measurement Based on Abstract Syntax Trees -- Maximum Common Subtree Size

In this simplified scenario, given a new feature f constructed from F_0 and O, the optimization goal for feature f is to maximize its similarity to the optimal feature f^*. For convenience, we measure similarity by comparing the syntactic structures of f and f^*. Specifically, **we define the similarity $S(f, f^*)$ of two features as their largest common subtree size**, i.e., the number of nodes in their largest common subtree: if f and f^* are identical, their similarity S reaches its maximum value. Thus, the optimization objective can be written as:

Maximize $S(f, f^*)$

Minimize $\text{size}(f)$

(The reason for minimizing $\text{size}(f)$ is that if f is too large, the downstream ML model needs to reverse-engineer the effective part from it.)

When $S(f, f^*) = \text{size}(f^*)$ and $\text{size}(f) = \text{size}(f^*)$, then $f = f^*$, and we have obtained the optimal feature.

3. Feature Quality Verifier Based on Downstream ML Model Scores with Error

After determining the optimization objective, before analyzing optimization algorithms, we first need to analyze the feature quality verifier V used by the LLM-based automated feature engineering method, which relies on changes in ML model scores. As mentioned earlier, feature quality has already been measured using $S(f, f^*)$. Therefore, the function of this feature quality verifier can be defined as **evaluating the similarity between feature f and the optimal feature f^* by inspecting the downstream model score. Considering factors such as the risk of overfitting, this verifier may have some error.** That is, when the verifier compares the quality of a feature f with the feature f' obtained by executing a certain feature optimization operation on f (i.e., comparing $S(f, f^*)$ and $S(f', f^*)$), it may give an incorrect answer with some probability. **In this setting, the optimization algorithm must have a high optimization efficiency**, meaning that with each round of optimization, it should have a high probability of discovering a better feature f' than f. Otherwise, due to the verifier's error, there is a significant chance that it will accept a new feature that is worse than the original feature.

4. Qualitative Comparison Between AST-based Local Optimization Algorithm and Black-box Optimization Algorithm in High-dimensional Feature Space

Based on the analysis of the verifier above, the quality of a feature generation algorithm largely depends on the probability of discovering better features in each round of optimization. In practical optimization, for the proposed AST+GLS algorithm (afe-master), since we improve features by using an LLM to traverse each subtree of the AST and combine the LLM's semantic knowledge for reasoning, proposing locally optimized solutions for these subtrees, this method has two advantages: **(1) the LLM will definitely traverse the largest common subtree between f and f^*, as well as it's subtrees, and propose improvement options to these effective structures**, and **(2) the local optimization solutions typically have a smaller edit distance, so they do not significantly disrupt the largest common subtree of f and f^*.** Since the proposed local optimization solutions are derived from the LLM's reasoning based on both domain knowledge and feature engineering knowledge, they generally have strong rationality and a high probability of making effective improvements, thereby enlarging the common subtree between the optimized feature f and f^*.

Therefore, each round of optimization with afe-master has a high probability of proposing improvements that can expand the common subtree between the optimized feature and the optimal feature f^*, i.e., increasing $S(f, f^*)$. The verifier V will likely accept these valid solutions that expand the common subtree. **Even if the verifier occasionally makes an error and accepts some invalid solutions, because the edit distance of the local optimization is small, these invalid changes will not cause significant damage to the effective subtrees in the original feature.** In this way, we can iteratively improve the feature f in a positive direction. A concrete example can be found in this anonymous link.

In contrast, if we use a black-box optimization algorithm, since the LLM does not transform the feature's linear expression into a syntax tree and does not analyze each improvement plan for the syntax tree individually, the model does not perform much explicit textual reasoning. Instead, **it may tend to randomly add or update operands from the original feature set F_0 and operators from the operator set O into the feature f's linear expression.** In this case, the edit distance of the optimization for the feature's syntax tree is highly uncontrollable. When f has higher order, due to the complex feature structure and lack of effective decomposition and detailed analysis, the algorithm is more likely to make random changes to the valid subtrees. **Considering that the search space for feature engineering grows exponentially with the size of the feature, this optimization method is unlikely to accidentally expand the size of the effective subtree.** Effective improvements are rare, and the verifier V, under the influence of errors, will likely accept many invalid or negative feature improvements. Furthermore, **due to the uncontrollable edit distance, a single negative optimization step may cause significant damage to the valid subtree.** Therefore, black-box optimization methods face significant difficulties in performing effective optimization in high-dimensional feature space and are more likely to generate negative or ineffective high-order features, with a high risk of overfitting.