FEATPILOT: Automatic Feature Augmentation on Tabular Data

Jiaming Liang*
University of Pennsylvania
liangjm@seas.upenn.edu

Chuan Lei, Xiao Qin, Jiani Zhang, Asterios Katsifodimos,
Christos Faloutsos, Huzefa Rangwala
Amazon Web Services
chuanlei, drxqin, zhajiani, akatsifo, faloutso, rhuzefa@amazon.com

***Abstract*—Tabular data within enterprises or open data repositories provide a huge opportunity for feature augmentation. Using these data sources to augment training data often boosts model performance, which is crucial in data-centric AutoML systems. Recent works on automatic feature augmentation have limited capabilities in utilizing useful features that cannot be joined with the base table without connecting through intermediate tables. We present FEATPILOT, a novel framework that explores and integrates high-quality features in tabular data for ML models. FEATPILOT evaluates a candidate feature from two aspects: (1) the efficacy of a join path connecting the feature to the base table and (2) the intrinsic value of a feature towards an ML task. FEATPILOT efficiently identifies high-quality features and their optimized join paths to augment the base table. Our experimental results show that FEATPILOT achieves up to a 10.27% improvement in ML model performance compared to state-of-the-art solutions across six public datasets.**

I. INTRODUCTION

Feature augmentation on tabular data is crucial for improving machine learning (ML) model performance [1]–[5]. With additional features, models can capture more complex patterns and relationships within the data, leading to better prediction and generalization results. By exploring and integrating features [6], [7] from open tabular data repositories (e.g., Google Dataset Search[1], Kaggle[2], and OpenML[3]), ML models [8] can avoid overfitting [9] and improve performance on unseen data [3]–[5].

Given a base table with an ML task defined on it and a set of candidate tables that can be joined with the base table via one-hop or multi-hop paths, feature augmentation is the task of selecting feature columns from the candidate tables that can maximize the performance of the ML task. This process typically involves identifying useful features that can reveal additional insights or patterns for the given ML task, and determining the optimal paths to join these candidate features with the base table to enrich the initial training data.

Motivating Example. An ML engineer is developing a classification model to predict whether a product should be recommended or not. The base table `product` contains only a limited set of features for building a high-quality model (shown in Fig. 1). Fortunately, useful features exist in related tables such as `address` in the `customer` table. To augment the base table, the ML engineer needs to find a high-quality join path to augment the base table. Among three join paths in Fig. 1, `customer` ⋈ `order` ⋈ `product` (indicated by the thick red line) provides more feature instances (i.e., `address`) compared to paths involving the `rating` or `review` table, as customers may not provide ratings or reviews for every purchased product. As a result, the ML engineer can exploit this augmented table to improve the classification model.

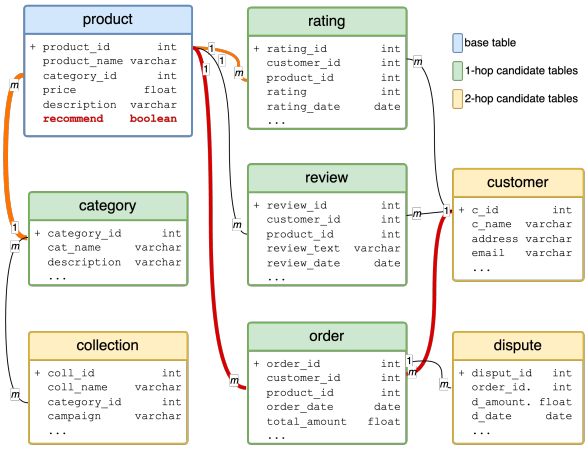


Fig. 1: Example of feature augmentation. The base table, `product`, is in blue. The task is to predict the values in the Boolean type column 'recommend'. The tables in green and yellow are 1-hop and 2-hop joinable with the base table.

Challenges. In this paper, we address the following three key challenges in feature augmentation.

(1) *Multiple Join Paths Selection.* In real-world open data settings, a candidate feature can be integrated with the base table via multiple join paths, resulting in different integration quality as explained in the motivating example. Hence, it is critical to identify the optimal join path to achieve the full potential of a candidate feature.

(2) *Expensive Join Operations.* To find out the actual benefit of augmenting the base table with a candidate feature, expensive joins need to be executed, which is computationally expensive and time consuming, especially when a long join path involves multiple joins. Candidate features in open data repositories often have similar join paths. Therefore, sharing join operations is crucial for efficient feature augmentation.

*The work was done when Jiaming Ling was with Amazon Web Services.
[1]`https://datasetsearch.research.google.com/`
[2]`https://www.kaggle.com/datasets`
[3]`https://www.openml.org/`

(3) *Search Complexity.* The search space of all candidate tables and their join paths is vast and intractable, making it prohibitively expensive to find the optimal set of candidate tables. To address this challenge, we need effective strategies to reduce the search space while preserving optimality.

State-of-the-art Approaches. Prior approaches [3], [10] augment the base table by joining other tables using either their relevancy to the base table or the improvement of the candidate features bring to the ML models. Recently, ALITE [11] and DIALITE [12] are introduced to integrate tables based on joinability or unionability, which can improve the quality of downstream ML models. However, these approaches assume that the features and their respective relevance to the base table are specified. Hence, they fail to leverage all possible candidate features and consequently do not achieve optimal model performance. Reinforcement learning (RL) [13], [14] has emerged as an effective approach to balance between feature exploration and exploitation. Several studies [4], [5], [15] leverage RL techniques such as multi-armed bandits and Deep Q-Networks (DQNs) to explore the large search space of candidate tables. Although these RL-based methods have achieved promising results, they still face critical challenges. Specifically, METAM [5] is limited to directly joinable candidate tables and AutoFeature [4] does not provide fine-grained optimization on integration quality. Moreover, RL often requires extensive training data and tends to overfit, reducing its effectiveness in real-world applications.

Proposed Solution. To cope with the above mentioned challenges, we propose FEATPILOT, an efficient and effective framework for automatic feature augmentation. Given a base table, a set of candidate tables and an ML task, FEATPILOT first selects a subset of candidate tables using a clustering algorithm based on table characteristics. FEATPILOT further chooses representative join paths from these candidate tables to the base table. This allows FEATPILOT to discover the actual performance gain led by the candidate tables following the join paths (*Challenge 1*). Consequently, FEATPILOT learns an integration quality model and a feature importance estimator to calculate the performance gain of any unexplored candidate tables and their respective join paths (*Challenge 2*). Lastly, FEATPILOT's join-path search algorithm prunes the search space based on both integration quality and feature importance, and finds the optimized integration paths for the selected candidate tables in polynomial time (*Challenge 3*).

Contributions. Our main contributions are the following.

(1) We introduce FEATPILOT, a novel feature augmentation framework for ML tasks over tabular data repositories, by decomposing the intractable problem of feature augmentation into feature importance and integration quality.

(2) We design an effective feature exploration module consisting of a clustering-based feature importance estimation and a learning-based integration quality model.

(3) We propose an efficient feature path search algorithm that exploits both estimated feature importance and integration quality to identify high-quality features and their optimized join paths for integration.

(4) Our experiments show that FEATPILOT achieves up to a 10.27% improvement in ML model performance compared to state-of-the-art solutions across six public datasets.

Reproducibility. The source code of FEATPILOT is available at `https://anonymous.4open.science/r/FeaturePilot-C308`.

II. Preliminaries and System Overview

A. Basic Notions

Definition 1 (Base Table): The base table, T_{base} = $\{c_1, c_2, \ldots, c_n, c_l\}$, refers to the original table that includes the initial set of features and the label column before any augmentation. Each c_i ($i \in [1, n]$) represents a column and c_l denotes the label column, which stores the prediction target for each row.

Definition 2 (Candidate Tables): The candidate tables, $\mathcal{T}$ = $\{T_1, T_2, \ldots, T_m\}$, are m joinable tables that can be used to augment a base table T_{base}. Each table is denoted by T_j = $\{f_{j,1}, f_{j,2}, \ldots, f_{j,k}\}$ ($j \in [1, m]$), where $f_{j,k}$ denotes the k-th column (a.k.a feature)[4] in table T_j.

Definition 3 (Join Path): Given a set of tables $T_1, T_2, \ldots, T_n$, a join path P can be defined as a sequence of join operations between pairs of tables. A join operation between any two tables T_i and T_j can be denoted as $T_i \bowtie T_j$. The join path P can be represented as: $P = T_1 \bowtie T_2 \bowtie T_3 \ldots \bowtie T_n$.

Without loss of generality, we allow different types of join operators between T_{base} and $T_j \in \mathcal{T}$, including inner join, left outer join, right outer join, etc.

Definition 4 (Augmented Table): The augmented table $T_{aug} = T_{base} \bowtie_P T_j$ is the result of joining the base table T_{base} and a candidate table T_j following a join path P.

Note that there can be multiple join paths between the base table and a candidate table. Hence the augmented table could be different depending on the chosen join path.

Definition 5 (Machine Learning Task): A machine learning (ML) task is defined as a function f that maps an input space $\mathcal{X}$ to an output space $\mathcal{Y}$. Given a dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, the objective is to learn the function f that best approximates the true underlying relationship from the inputs to the outputs.

Definition 6 (Utility Gain): Given a ML task, the utility score, $US(T)$, is defined as the performance metric (e.g., accuracy, precision, recall, etc.) of the task on a table T achieved by a specific ML model[5]. Let $US(T_{base})$ be the performance metric on the base table, and $US(T_{aug}) = US(T_{base} \bowtie_P T_j)$ be the performance metric on the augmented table. The utility gain UG is defined as the improvement in the performance metric: $UG = US(T_{aug}) - US(T_{base})$.

Problem Statement. Given a base table T_{base}, a set of candidate tables $\mathcal{T}$, and a ML task, our goal is to select features from these candidate tables $\mathcal{T}$ and to identify the best join paths to augment the base table T_{base} with the selected

[4]We use two terms, column and feature, alternately in this paper.

[5]$US(T)$ uses the negation of a metric value when smaller values are preferred (e.g., mean squared error (MSE), mean absolute error (MAE), etc.).

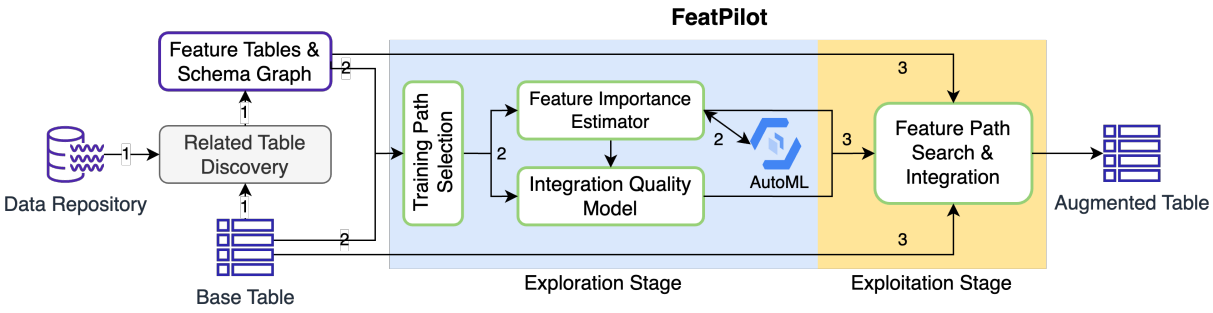


Fig. 2: FEATPILOT overview.

features, such that the utility gain UG of the ML task on T_{aug} is maximized. Formally,

$$\max_{\mathcal{S}\subseteq\mathcal{T},\mathcal{P}} UG = US(T_{aug}) - US(T_{base}) \quad \text{subject to } T_{aug} = T_{base} \bowtie_{\mathcal{P}} \mathcal{S} \tag{1}$$

where $\mathcal{S} \subseteq \mathcal{T}$ is the subset of selected candidate tables from $\mathcal{T}$, $\mathcal{P}$ is the set of join paths used to augment T_{base} with $\mathcal{S}$, and T_{aug} is the augmented table obtained by joining T_{base} with the selected tables $\mathcal{S}$ following the join paths $\mathcal{P}$.

Note that we treat the ML model (e.g., linear regression, deep neural network, XGBoost etc.) and the corresponding task as a black box such that the user can leverage our framework to augment features regardless of any particular model, and the model parameters can be updated iteratively.

The complexity of feature augmentation problem generally increases exponentially with the number of candidate features and all possible join paths among the base and candidate tables. The combination of these two exponential search spaces makes the problem NP-hard. Even with various heuristic and optimization techniques [16], [17] to reduce the complexity, the cost of feature augmentation can be still expensive.

B. FEATPILOT System Overview

For a given ML task, FEATPILOT takes input as a base table with a prediction task on the target column and a tabular data repository consisting of candidate tables with features. As depicted in Fig. 2, FEATPILOT leverages data discovery systems such as Aurum [18] or NYU Auctus [19] to identify the candidate tables within k-hop joins to the base tables. FEATPILOT then triggers its utility gain model in two stages, i.e., exploration and exploitation.

FEATPILOT's exploration stage tackles *Challenge 1* and *Challenge 2* mentioned in Section I. It takes the following four steps. (1) FEATPILOT groups the identified features into clusters based on their representations (embeddings, data distributions, data types, etc.). (2) FEATPILOT employs a path selection strategy to choose a set of features and candidate join paths. (3) FEATPILOT then follows the selected join paths to integrate the candidate features into the base table and runs AutoML (e.g., AutoGluon[6]) to compute the utility gain over the model trained on the base table. (4) Using the data collected in (3), FEATPILOT trains an integration quality model and assigns the actual feature importance to the selected features in the clusters. The importance of the other features can be estimated based on their relative distance to the ones with the actual importance assigned.

FEATPILOT's exploitation stage address *Challenge 3* discussed in Section I. It leverages both the feature importance estimator and integration quality model to conduct an efficient and effective search over all candidate features and their associated join paths. It iteratively selects the most important feature and uses the integration quality model to select the join path with the best integration quality until a predefined number of features is reached. FEATPILOT then follows the selected join paths to integrate the selected features with the base table and generates the final augmented table for the given ML task. Finally, AutoML solutions can train, select and deploy the best performing ML model on the augmented table [20].

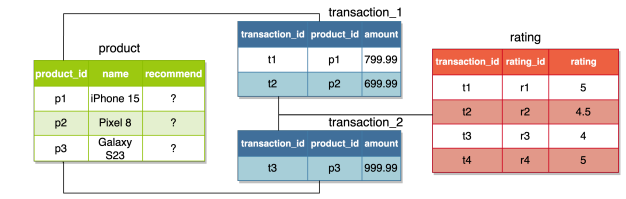
III. UTILITY GAIN MODELING

The problem of *maximizing utility gain* for a given ML task is intractable. Hence we propose to decompose this problem into two sub-problems, namely maximizing feature importance – how useful is a feature to the ML model, and maximizing integration quality – how many data instances in the base table can be augmented by a feature.

Feature Importance. To define the *feature importance*, we first introduce the concept of a *virtual table*. Let T_{aug} denote a resulting augmented table from a base table T_{base} and a candidate table T_c. The virtual table T_{virt} refers to the ideal case where there exists a join path P such that every instance in T_{base} can be augmented by a feature instance from T_c via this P (i.e., iPhone 15 with a rating of 5, Pixel 8 with a rating of 4.5, and Galaxy S23 with a rating of 4 shown in Fig. 3b). Note that a virtual table is specific to a base table and an individual candidate feature.

Note that, for ease of illustration, we assume a 1:1 join relationship in this example. In practice, FEATPILOT supports 1:1, 1:m, and m:m join relationships between the base table T_{base} and a candidate table T_j. To handle these scenarios, FEATPILOT employs commonly used methods based on feature data types, including aggregation (e.g., count, mean, min/max, etc. for numerical features), pooling on feature embeddings (for textual features), one-hot encoding (for categorical features).

[6] https://auto.gluon.ai/



(a) Join graph

(b) T_{virt} (c) T_{aug_1} (d) T_{aug_2}

Fig. 3: A join graph of augmenting the `product` table with the feature `rating` via different integration paths.

Open data repositories often contain noisy data, where incorrect values in candidate tables can negatively impact feature importance and, ultimately, utility gain. In other words, erroneous values may lead to a lower utility gain, reducing the likelihood of an attribute being selected for augmenting the base table. However, this challenge is not specific to FEATPILOT; rather, it is a general challenge to feature augmentation when dealing with noisy data [21], [22].

Definition 7 (Feature Importance): Feature importance is defined as the largest utility gain achievable by augmenting the base table with a candidate feature $f_{i,j} \in T_i$, i.e.,

$$FI(f_{i,j}) = US(T_{virt(i,j)}) - US(T_{base}),$$

where $f_{i,j}$ is the j-th feature from the table T_i, $T_{virt(i,j)}$ is the virtual table augmented with feature $f_{i,j}$.

Integration Quality. On the other hand, *integration quality* measures the proportion of instances in the base table that can be augmented by the candidate table via a join path. Figs. 3c and 3d show two augmented tables using the same feature `rating`. The join path P_1 = `product` ⋈ `transaction_1` ⋈ `rating` brings 2 out of 4 instances from `rating` to `product`. However, only 1 instance from `rating` can be augmented to the base table by following P_2 = `product` ⋈ `transaction_2` ⋈ `rating`. Hence P_1 is a better integration path than P_2.

When data instances in the base table cannot be augmented by a candidate feature, we use the special token `null` to fill the missing feature value. AutoML solutions (e.g., AutoGluon) can automatically handle `null` values during model training [20]. Therefore, an augmented table can be split into two parts, namely *non-NA* and *NA*. *non-NA* refers to the set of instances in T_{aug} that are augmented by feature instances (e.g., two products $p1$ and $p2$ in the red box belong to *non-NA* in Fig. 3c). And *NA* denote the data instances in T_{aug} filled with `null` values (e.g., $p3$ in Fig. 3c).

Definition 8 (Integration Quality): Integration quality of an augmented table T_{aug} is defined as a ratio between the number of feature instances that contribute to the augmentation and the total number of instances in the base table:

$$IQ(T_{aug}) = \frac{|\textit{non-NA}|}{|T_{aug}|} = \frac{\#\ \textit{augmented instances}}{\#\ \textit{instances}} \text{ in } T_{aug},$$

where $T_{aug} = Augment(T_{base}, f_{i,j}, P_k)$ indicates that T_{aug} is produced by augmenting the feature $f_{i,j}$ to the base table, following the path P_k. $IQ(T_{aug})$ ranges from 0 to 1.

Decomposition of *UG*. We argue that the *utility gain* (UG) can be expressed as $UG(T_{aug}) = FI(f_{i,j}) \times IQ(T_{aug})$. To prove this hypothesis, we will discuss utility score estimation as well as its error bound of an augmented table with missing feature values below.

Lemma 1: $US(T_{aug})$ is the weighted average of $US(\textit{non-NA})$ and $US(\textit{NA})$, namely, $US(T_{aug}) = p \times US(\textit{non-NA}) + (1-p) \times US(\textit{NA})$, where p denotes the integration quality measurement for simplicity purposes.

Proof. Without loss of generality, we choose the classification accuracy as the utility score. By definition, the classification accuracy for *non-NA* and *NA* in T_{aug} is:

$$US(\textit{non-NA}) = \frac{CP_{\textit{non-NA}}}{|\textit{non-NA}|} \text{ and } US(\textit{NA}) = \frac{CP_{\textit{NA}}}{|\textit{NA}|},$$

where CP indicates the correct predictions including true positives and negatives, $|\textit{non-NA}|$ and $|\textit{NA}|$ are the cardinality of *non-NA* and *NA* in T_{aug}. Consequently, the accuracy of the augmented table T_{aug} can be expressed as follows:

$$\begin{aligned} US(T_{aug}) &= \frac{CP_{T_{aug}}}{|T_{aug}|} = \frac{CP_{\textit{non-NA}} + CP_{\textit{NA}}}{|T_{aug}|} \\ &= \frac{CP_{\textit{non-NA}}}{|\textit{non-NA}|} \times \frac{|\textit{non-NA}|}{|T_{aug}|} + \frac{CP_{\textit{NA}}}{|\textit{NA}|} \times \frac{|\textit{NA}|}{|T_{aug}|} \\ &= p \times US(\textit{non-NA}) + (1-p) \times US(\textit{NA}) \end{aligned} \quad (2)$$

Lemma 2: If a random variable x, s.t. $-1 < x < 1, E(x) = 0, Var(x) \leq \epsilon < 1$ ($\epsilon \in \mathbb{R}$), then $0 \leq E(|x|) \leq 2\sqrt{\epsilon}$ and $0 \leq Var(|x|) \leq \epsilon$.

Proof. To begin with, we have

$$E(x^2) = Var(x) - E(x)^2 \leq \epsilon \quad (3)$$

Let a be a constant $0 < a < 1$, and we have $E(|x|) = E(\sqrt{x^2})$. For $a < |x| < 1$, we have $\sqrt{x^2} < \frac{1}{a}x^2$, and for $0 \leq |x| \leq a$, we have $\sqrt{x^2} \leq a$. Consequently, $\sqrt{x^2} \leq a + \frac{1}{a}x^2$. Thus for $E(|x|)$, we have

$$E(|x|) \leq E(a + \frac{1}{a}x^2) = a + \frac{1}{a}E(x^2) \quad (4)$$

Based on Equations 3 and 4, we have $E(|x|) \leq a + \frac{1}{a}\epsilon$, and let $a = \sqrt{\epsilon}$, we have $E(|x|) \leq 2\sqrt{\epsilon}$. For $Var(|x|)$, we have

$$Var(|x|) = E(|x|^2) - E(|x|)^2 \leq E(|x|^2) = E(x^2) \leq \epsilon \quad (5)$$

Given that $0 \leq E(|x|)$ and $0 \leq Var(|x|)$, we prove that $0 \leq E(|x|) \leq 2\sqrt{\epsilon}$ and $0 \leq Var(|x|) \leq \epsilon$.

Lemma 3: The utility score of *non-NA* part of T_{aug} can be estimated by the one of T_{virt} with an error. The expectation and variance of the absolute error are bounded by the reciprocal of the number of rows in T_{virt}. Namely,

$$error = (US(\textit{non-NA}) - US(T_{virt})) \times \frac{|\textit{non-NA}|}{|T_{virt}|},$$

with $0 \leq E(|error|) \leq \frac{2}{\sqrt{|T_{virt}|}}$ and $0 \leq VAR(|error|) \leq \frac{1}{|T_{virt}|}$. Thus $E(|error|) \approx 0$ and $Var(|error|) \approx 0$, when $|T_{virt}| \to +\infty$.

Proof. Without loss of generality, we again choose the classification accuracy as the utility score. Let us denote N as the total number of instances (rows) in T_{virt}, M as the number of correct predictions (i.e., true positives and true negative) when the model is trained on T_{virt}, n as the number of instances (rows) in *non-NA*, and m as the number of correct predictions when the model is trained on *non-NA*.

We express the accuracy attained through training on T_{virt} and *non-NA* as $US(T_{virt}) = \frac{M}{N}$ and $US(\textit{non-NA}) = \frac{m}{n}$, respectively. The approximation error is expressed as:

$$\begin{aligned} error &= (US(\textit{non-NA}) - US(T_{virt})) \times \frac{|\textit{non-NA}|}{|T_{virt}|} \\ &= \frac{n}{N}(\frac{m}{n} - \frac{M}{N}) \end{aligned} \tag{6}$$

Let us assume that n instances are randomly selected from N instances, implying that m follows a hyper-geometric distribution, i.e., $Pr(m = x) = \frac{\binom{M}{x}\binom{N-M}{n-x}}{\binom{N}{n}}$. The expectation of m is $E(m) = \frac{nM}{N}$ and the variance of m is $Var(m) = \frac{nM(N-n)(N-M)}{N^2(N-1)}$. Then we have

$$\begin{aligned} E(\frac{n}{N}(\frac{m}{n} - \frac{M}{N})) &= 0, \\ Var(\frac{n}{N}(\frac{m}{n} - \frac{M}{N})) &= \frac{nM(N-n)(N-M)}{N^4(N-1)}. \end{aligned} \tag{7}$$

According Equations 6 and 7, we can infer that $E(error) = 0$ and $Var(error) = \frac{nM(N-n)(N-M)}{N^4(N-1)}$. Furthermore, as $N > M$, we have $\frac{M(N-n)(N-M)}{N^2(N-1)} \leq 1$, and hence $Var(error) \leq \frac{n}{N^2} \leq \frac{1}{N} < 1$. By setting $\epsilon = \frac{1}{N}$, we have $E(error) = 0$ and $Var(error) \leq \epsilon < 1$ $(-1 < error < 1)$. Based on Lemma 2, we have $E(|error|) \leq 2\sqrt{\epsilon} = \frac{2}{\sqrt{N}}$ and $Var(|error|) \leq \epsilon = \frac{1}{N}$. Subsequently, we have $0 \leq E(|error|) \leq \frac{2}{\sqrt{N}}$ and $0 \leq Var(|error|) \leq \frac{1}{N}$. Hence we prove $E(|error|) \approx 0$ and $Var(|error|) \approx 0$ when $N \to +\infty$.

Theorem 1: The utility gain of an augmented table $T_{aug} = Augment(T_{base}, f_{i,j}, P_k)$ can be approximated by a multiplication of the feature importance of $f_{i,j}$ and the integration quality of $f_{i,j}$ following the join path P_k, namely $UG(T_{aug}) = FI(f_{i,j}) \times IQ(T_{aug})$.

Proof. Theorem 1 can be proved by using Definition 6, Lemma 1, and Lemma 3.

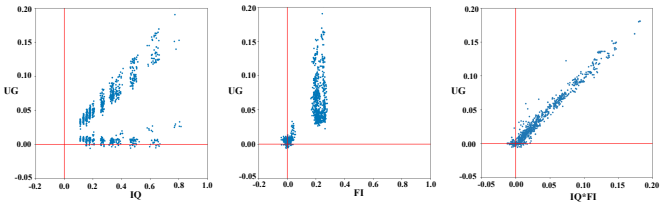
$$\begin{aligned} UG(T_{aug}) &= US(T_{Aug}) - US(T_{base}) \\ &= p \times US(\textit{non-NA}) + (1-p) \times US(NA) - US(T_{base}) \\ &\approx p \times US(T_{virt}) + (1-p) \times US(T_{base}) - US(T_{base}) \\ &= p \times US(T_{virt}) - p \times US(T_{base}) \\ &= p \times (US(T_{virt}) - US(T_{base})) \\ &= FI(f_{i,j}) \times IQ(T_{aug}) \end{aligned} \tag{8}$$

Theorem 2: The problem of utility gain maximization

$$\max UG(T_{aug}) = \max \sum_{T_i \in \mathcal{T}',\ P_k \in \mathcal{P}'} UG(T_{aug_i})$$

can be approximated by

$$\max \sum_{T_i \in \mathcal{T}',\ P_k \in \mathcal{P}'} IQ(T_{aug_i}) \times FI(f_{i,j}),$$



(a) IQ and UG (b) FI and UG (c) $IQ{\cdot}FI$ and UG

Fig. 4: Utility gain, feature importance, and integration quality.

where $f_{i,j} \in T_i$ denotes a candidate feature $f_{i,j}$ in T_i and T_{aug_i} represents the augmented table $Augment(T_{base}, f_{i,j}, P_k)$.

Empirical Validation. We conduct an empirical study to validate the derived relationship in Theorem 1. We choose `DonorsChoose` dataset[7] from Kaggle and randomly sample 1,000 candidate features and their corresponding join paths. We use these sampled features to augment the base table and compute the metrics UG, IQ, and FI, subsequently plotting the data points in Fig. 4. We observe that (1) lower IQ and FI values lead to lower UG values, and (2) $IQ \cdot FI$ has a linear relationship with UG, which are consistent with Theorem 1.

IV. FeatPilot System Design

The utility gain has been shown to have a linear relationship with integration quality (IQ) and feature importance (FI) in Theorem 1. However, both IQ and FI are computational expensive to obtain as they require resource intensive joins and model training, particularly when dealing with a large number of features and their associated join paths. As a result, estimating IQ and FI to avoid costly joins and model training becomes imperative. In this section, we delves into our proposed solution for IQ and FI estimation as well as data selection strategy for estimation model training.

A. Integration Quality Estimation

According to Definition 8, the IQ of a join path, $P = T_{base} \bowtie_{0,1} T_1 \bowtie_{1,2} T_2 \cdots \bowtie_{n-1,n} T_n$, is determined by the number of instances in T_{base} that can be augmented by the instances in the feature column $c_{n,j}$ from T_n. This essentially boils down to the *sequence of join operations* in a path. Inspired by cardinality estimation in database systems [23], [24], we introduce a Long Short-Term Memory (LSTM)-based model designed specifically to estimate the integration quality of a given join path. The idea is to leverage the pairwise table features and statistics to estimate the IQ without materializing the join. We use a variety of input features extracted from join operations, including transitivity (*Tran*), variance (*Var*), entropy (*Ent*), KL-divergence (*KL*), Pearson correlation coefficient (*PC*), and mutual information (*MI*). *Tran* represents the IQ for 1-hop joins, making it a reasonable choice for estimating multi-hop join paths. We adopt *Var*, *PC*, and *MI* from AutoFeature [4], where these features are utilized to assess the potential benefit of a candidate feature. *Ent* and

[7] `https://tinyurl.com/mv7y96fh`

KL capture key aspects of the data distribution in addition to *Var*, offering a more comprehensive statistical representation. The detailed implementations can be found in [25].

We use the above feature characteristics of join operations to train a Long Short-Term Memory (LSTM) model for IQ estimation. The LSTM architecture is chosen due to its lightweight yet powerful capability in capturing the inherent order and dependencies within a sequence of joins in a join path. These features, namely *Tran*, *Var*, *Ent*, *KL*, *PC* and *MI*, are concatenated together to represent each join operation $T_i \bowtie_{i,j} T_j$. Note that we use the feature column $f_{i,j}$ instead of the join column in T_4 to compute the above features as it determines the ultimate IQ. The training objective of the LSTM network uses Mean Squared Error as the loss function:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i(\theta))^2,$$

where N is the number of training examples, y_i is the true IQ for the i-th join path sample, and θ denotes the parameters of the LSTM model, and $\hat{y}_i(\theta)$ is the predicted value by the model parameterized by θ. The training objective is to minimize the MSE loss by adjusting the weights of the LSTM. The method of selecting data points for training the LSTM network will be described in Section IV-C.

B. Feature Importance Estimation

Feature importance estimation has been studied [26]–[28] in recent years. However, these methods are often computationally expensive and model specific. Hence we introduce a simple and effective clustering-based method to estimate the feature importance. The intuition is that features with similar representations in the embedding space tend to have a similar impact (i.e., feature importance) on the ML task [5], [26]. The clustering algorithm computes the representative features' FI in each cluster and estimates FI of other features based on their proximity to the representative ones, which leads to substantial computational savings.

Feature Representation. The representation of a feature column $c_{i,j}$ consists of two parts: (1) column metadata, and (2) column data instances. We use a pre-trained language model (PLM) such as BERT [29] to generate the embeddings of column metadata such as column names, column semantic types, column description, if available. For data instances in a feature column, we sample the column, serialize the sample data, and use a PLM to generate the embeddings as well. Finally, we concatenate the embeddings of both column metadata and data instances as the feature representation.

Feature Clustering. We design the clustering method based on DBSCAN [30]. To be specific, we first randomly select an unexplored feature and check the number of features that lie within its ϵ-neighborhood, where ϵ denotes the threshold of cosine similarity between two feature representations. Then we check if the ϵ-neighborhood of this selected feature contains at least m features. If so we form a cluster by adding all unexplored features within the selected feature's ϵ-neighborhood.

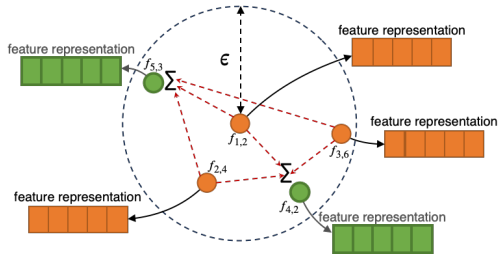


Fig. 5: Clustering-based FI estimation. The feature importance of $f_{1,2}$ is estimated based on the actual FI scores of its neighbors (circles in green).

We iterate this process until all features have been assigned to a cluster. Unlike DBSCAN, we do not consider any feature as noise. Namely, the remaining unexplored features will be considered as clusters of their own. The advantage of this lightweight method is that it bounds the similarity between the centroid feature and other candidate features within a cluster, i.e., $sim(Repr(centroid), Repr(candidate_i)) \leq \epsilon$.

Compute FI. According to Definition 7, we need to follow one or more join paths to integrate a feature to the base table in order to compute its FI. Naively, one could utilize all join paths between the base table and the feature table to obtain its maximum FI, which is very costly. Hence we introduce an efficient data selection strategy in Section IV-C that chooses features and their associated join paths for FI computation.

Estimate FI for Unseen Features. For a unseen feature, we estimate its FI by a weighted average over the FI of its surrounding features in the same feature cluster. Precisely,

$$FI(c) = \sum_{n=1}^{N} FI(c_n) \cdot \frac{sim(c, c_n)}{\sum_{i=1}^{N} sim(c, c_i)},$$

where c denotes the unseen feature, N denotes the set of features with known FI in the same cluster, and $cos(\cdot)$ represents the cosine similarity between the unseen feature and its neighboring feature. For example, Fig. 5 illustrates $f_{1,2}$'s ϵ-neighborhood. The FI scores of the feature represented in orange circles $f_{1,2}$, $f_{2,4}$ and $f_{3,6}$ are known. The FI scores of the unseen features $f_{4,2}$ and $f_{5,3}$ (green circles) are estimated by referring to the three explored features.

C. Data Selection for Estimation

Now we introduce an effective data selection strategy for IQ model training and FI estimation. Our goal is to choose high quality data points to serve both IQ and FI estimations, in the form of *feature-path* $(f_{i,j}, P_k)$ pairs. Ideally, the selected data points should have two desired properties. First, the selected integration paths should have a good coverage over all joins among any candidate tables and the base table. The covered joins can be stitched together to accurately estimate unexplored integration paths. Second, the selected features should be evenly distributed across all feature clusters. The number of features chosen from one cluster should be proportioned to the size of the cluster.

Algorithm 1: Greedy Data Point Selection

Input: *candidateSet*: all feature-path pairs, *budget*: the max number of paths to select
Output: *selected*: the set of selected feature-paths

$unselected \leftarrow candidateSet$
$selected \leftarrow \emptyset$
$path\_coverage \leftarrow init\_pc(unselected)$
$path\_group\_coverage \leftarrow init\_pgc(unselected)$
$cluster\_coverage \leftarrow init\_fc(unselected)$
$weights \leftarrow init\_weight(unselected)$
while $budget > 0$ *and* $unselected \neq \emptyset$ **do**
  $(f, p) \leftarrow min\_weight(unselected, w)$
  $selected \leftarrow selected \cup \{(f, p)\}$
  $unselected \leftarrow unselected - \{(f, p)\}$
  for $e \in p$ **do**
    $path\_coverage[e] \leftarrow True$
  end
  $path\_group\_coverage[len(p)]$ += 1
  $cluster\_coverage[f.getCluster()]$ += 1
  $weights \leftarrow init\_weight(unselected)$
  for $(f, p) \in unselected$ **do**
    $weights[(f, p)] \leftarrow w(f, p, path\_coverage, path\_group\_coverage, cluster\_coverage)$
  end
  $budget \leftarrow budget - 1$
end
return *selected*

To achieve two properties above, we consider the following factors regarding a feature-path pair and propose a weight function to rank all feature-path pairs.

(1) Single join path coverage (PC) represents how much a join path has been explored, namely the ratio between the number of join edges in a join path that have been explored and the total number of join edges in the path.

(2) l-hop join path group coverage (PGC) represents the number of join paths of a length l that have been fully explored, measured by the ratio between the number of fully explored l-hop join paths and the total number of l-hop paths.

(3) Feature cluster coverage (FC) represents the number of features that have been examined in a feature cluster. This is measured by the ratio between the number of tested features and the total number of features in a cluster.

Note that all the above factors range between [0, 1] and a lower value of each factor indicates that a path or a feature should be explored. Hence the weight function is defined as

$$w(f_{i,j}, P_k) = PC \times PGC \times FC. \tag{9}$$

Intuitively, Eq. 9 ensures that the weight is significantly reduced when any of the contributing factors (PC, PGC, FC) approaches zero. This property aligns with our objective of selecting data points that comprehensively represent the structure coverage and feature diversity.

We assign a score to each feature-path pair using this weight function, rank them in an ascending order to select the one with the smallest weight in each iteration. After each iteration, we update the weights, re-rank the feature-path pairs and continue the selection until a given budget is exhausted. Algorithm 1 details the greedy data point selection method.

D. Feature Search and Integration

To identify an augmentation plan that maximizes the utility gain of a ML task, we design an efficiently and effectively feature search and integration algorithm, exploiting the integration quality and feature importance estimations.

1) Join Graph Model: We first introduce FEATPILOT join graph model to capture the search space of integration plans.

*Definition 9 (*FEATPILOT *Join Graph):* A FEATPILOT join graph $\mathcal{G} = (V, E)$ is a unweighted and undirected graph with a set of vertices and a set of edges. A vertex $v \in V$ represents a table, consisting of a set of features $F = \{f_1, ..., f_n\}$, and an edge $e \in E$ represents a join between two tables.

Given a join graph $\mathcal{G}$, a source vertex v_b (i.e., the base table), and a given budget B, the goal is to identify no more than B features and their respective integration paths in $\mathcal{G}$ to maximize the utility score of the ML task.

2) Search and Refinement: We introduce two pruning principles below to progressively reduce the search space, preventing the search explosion in the early stage without sacrificing the quality of feature augmentation.

Pruning Principle 1 (IQ Monotonicity). Based on Definition 8, the integration quality monotonically decreases as a path extends. In other words, if the IQ of a path P is too low, it is not beneficial to explore any path that has P as its prefix. This fundamental principle is critical for path exploration with early termination.

Pruning Principle 2 (FI Lower Bound). If the feature importance of a feature f_i is too low, then f_i and all integration paths to f_i can be safely pruned. The rationale behind this principle is based on Theorem 1. Specifically, if f_i's FI is too low, its UG would still be negligible even there were a perfect integration path (i.e., IQ = 1). Note that the lower bound of FI can be progressively increased as we explore the search space and identify more useful features.

Search with Progressive Pruning. With these two pruning principles and inspiration from Beam Search [31], we introduce a BFS-based search algorithm (Algorithm 2) that iteratively selects feature candidates and their integration paths.

It starts with the base table and iteratively extends to k-hop features. We maintain a min-heap that keeps top-H candidate feature-path pairs $selected$, sorted by the UG score in a descending order (Line 1). The heap size H is larger than the feature budget B so as to maintain a superset of candidates for further refinement (Line 2). We then iterate through each k-hop path, extending the path to include an additional vertex v directly connected to the last vertex u in the path (Lines 3-5). Subsequently, we compute the IQ for the extended path (p') and discard it if its IQ is below T_{IQ} (Lines 6-11). For the qualified p', we add it to the set of $(k+1)$-hop paths, and evaluate the UG for each feature on vertex v. The feature-path pair with the lowest UG score in $selected$ is used as the lower bound (Lines 12-14). We update $selected$ with the qualified feature-path pair as well as the feature budget (Lines 15-16). Finally, the algorithm returns the set of selected feature-paths.

Algorithm 2: Search and Refinement

Input: $G = (E, V)$: join graph, T_{IQ}: IQ threshold, H: heap size, B: feature budget
Output: *selected*: the set of selected feature-paths

$P_{cur} \leftarrow \{(v_b)\}$, $selected \leftarrow minHeap(H)$
while $|P_{cur}| > 0$ *and* $B > 0$ **do**
  $P_{next} \leftarrow \emptyset$
  for $p \in P_{cur}$ **do**
    /* get the last vertex of path p */
    $u \leftarrow p.getLastV()$
    /* get one-hop neighbors of v */
    for $v \in u.getNeighbor()$ **do**
      /* avoid cyclic joins */
      if $v \in p$ **then**
        continue
      end
      $p' \leftarrow p \oplus v$, $iq \leftarrow estimateIQ(p')$
      /* Pruning Principle 1 */
      if $iq \geq T_{IQ}$ **then**
        $P_{next} \leftarrow P_{next} \cup p'$
        for $f \in v.getFeatures()$ **do**
          /* Pruning Principle 2 */
          if $estimateFI(f) \times iq \geq selected.minUG()$ **then**
            $selected.insert((f, p'))$
            $B \leftarrow B - 1$
          end
        end
      end
    end
  end
  $P_{cur} \leftarrow P_{next}$
end
return *selected*

Among the selected feature-path candidates, similar features in the same cluster are expected to have comparable effect on the downstream ML task [5]. In light of this observation, we design a weight decay method to mitigate the local optimum incurred by choosing similar or redundant features during the search process. Specifically, the weight decay method relies on a scaling factor using feature similarity to regulate the FI on related features, subsequently optimizing the overall efficacy of selected features in the downstream tasks. The weight decay function on a feature f is

$$FI(f) = \frac{1}{|N|} \sum_{i=1}^{N} FI(f_i) \cdot (1 - sim(f, f_i)),$$

where N is the set of chosen features in the same cluster. We use cosine similarity based on the feature representation introduced in Section IV-B, which captures contextual relevance, as the representation is derived from both column metadata and instances. By incorporating both metadata and instance-level information, this representation ensures that features identified as similar are likely to exhibit a comparable effect (i.e., *utility gain*) on the ML task.

General feature engineering and model optimization techniques, such as regularization, redundancy detection, or validation-based feature pruning, can further enhance model performance by identifying and mitigating overfitting risks introduced by noisy features. AutoML solutions (e.g., AutoGluon) incorporate some of these techniques, while others can be integrated into FEATPILOT for enhanced robustness.

V. EXPERIMENTAL EVALUATION

A. Experimental Setup

Datasets. We evaluate FEATPILOT over six public datasets (Table I), including `School`, `DonorsChoose`, `Diabetes`, `Fraud Detection`, `Poverty`, and `Air`, by performing classification and regression tasks. All six datasets are composed of base tables provided by Kaggle and D3M[8]. Given a base table, we identify candidate tables for augmentation on open source datasets using NYU Auctus [19].

- **School** is a dataset commonly used in baselines [3]–[5] for a classification task. The target prediction is school performance on a standardized test based on student attributes, course attributes and other historical surveys.
- **DonorsChoose**[9] is a Kaggle dataset for a classification task. The target prediction is whether or not a DonorsChoose proposal was accepted. We identified 122 candidate features for augmentation.
- **Diabetes**[10] is a Kaggle dataset for a binary classification task that predictions diabetes. We identified 72 candidate potential features for augmentation.
- **Fraud Detection**[11] is a Kaggle dataset consisting of over 800 candidate features. The task is to predict whether an online transaction is fraudulent.
- **Poverty** consists of socioeconomic features like poverty rates, population change, unemployment rates, and education levels across U.S. States and counties. The regression task is to predict poverty levels at different locations.
- **Air** dataset contains tables/features such as temperature, PM2.5, and sulfur dioxide. The regression task is to predict the air quality of a city on a given date.

Baselines. We compared FEATPILOT with a variety of baselines, including heuristic-based solutions such as Deep Feature Synthesis (DFS) [2], ARDA [3], AutoFeat [32] as well as ML-based solutions such as AutoFeature [4] and METAM [5]. We also compare against a naive baseline (Exhaustive) that uses all candidate tables to augment the base table.

- Exhaustive method is a naive baseline that augments the base table with all candidate tables and subsequently selects features to train the model.
- Deep Feature Synthesis (DFS) [2] is a pioneer work that automatically generates features for relational datasets. It utilizes join paths in the data to a base table, and then applies mathematical functions along these paths to create the final features. We set the maximum depth of the features to 5, which covers all potential candidate tables.
- ARDA [3] is a feature augmentation system, which employs a variety of heuristics to select top-k tables to join and uses a

[8] `https://datadrivendiscovery.org/about-d3m/`
[9] `https://tinyurl.com/mv7y96fh`
[10] `https://tinyurl.com/9tzsrs9r`
[11] `https://tinyurl.com/4urssxkm`

Datasets	#Tables	#Columns	#1-hop Features	#Distant Features
School	121	1,295	440	625
DonorsChoose	73	221	48	74
Diabetes	71	204	22	50
Fraud Detection	81	254	30	97
Poverty	98	408	93	117
Air	75	603	172	308

TABLE I: Statistics of datasets.

random injection-based feature augmentation to search candidate feature subsets. We adopt the setting from ARDA by providing star schemata with known PK-FK relationships.

- AutoFeature [4] is a reinforcement learning based framework to augment the features following an exploration-exploitation strategy over the search space of candidate tables (features). We use the Multi-Arm Bandit solution with the hyperparameter of 100 action selection iterations, l=1 and γ=0.6 for optimal performance, as suggested in [4].
- AutoFeat [32] is a ranking-based feature discovery method that identifies relevant features by exploring multi-hop, transitive join paths within complex data lakes. We use τ = 0.65 as the `null` value ratio and κ = 15 as the maximum selected features from a table.
- METAM [5] is a goal-oriented framework that queries the downstream task with a candidate dataset, using a feedback loop that automatically steers the discovery and augmentation process. We use ϵ = 0.05, which controls the trade-off between the number of clusters and their quality. We also use θ = 0.90 as the required utility score and τ is set to the number of identified clusters.

Evaluation Metrics. Since we evaluate FEATPILOT using both classification and regression tasks, we adopt the commonly used metrics, Accuracy and F1 score for measuring the effectiveness. In particular, we report Accuracy for the classification tasks on `DonorChoose`, `Diabetes` and `School` because of the balanced label distribution over the classes. We report F1 score for the classification task on `Fraud Detection` since the label distribution is skew towards the negative classes and typically a fraudulent (positive) case is a more valuable prediction. For the regression tasks, we follow [3], [4] and report the Mean Absolute Error (MAE) or Mean Square Error (MSE) between the estimated and ground truth values. In ablation studies, we also measure the accuracy of each FEATPILOT's component by reporting the MAE or MSE. We use AutoGluon to automatically train multiple models (e.g., XGBoost model, kNN model, and neural network models), and ensemble the models to create the final predictor.

B. End-to-end Evaluation

We first conduct an end-to-end evaluation of FEATPILOT against the state-of-the-art solutions for feature augmentation. In these set of experiments, we study the effectiveness of FEATPILOT on finding the most useful features from candidate tables within a given budget. The budget B is defined as the number of features allowed to augment the base table. We vary the budget from 1, 5 to 10 and report the utility measurements of different methods on the datasets. As shown in Table II, when $B > 1$, FEATPILOT consistently demonstrates superior performance on finding and integrating features that contribute the most to the task on the base table. The reason is that FEATPILOT not only accurately identifies useful features that are different hops away but also exploits these features with high-quality integration paths.

When the budget is extremely limited (i.e., B = 1), FEATPILOT is still among the top 2 performing methods. On both `DonorChoose` and `Air` datasets, the most useful feature is from an 1-hop candidate table, which also presents high joinability to the base table. And this information is given to ARDA and hence it is able to outperform FEATPILOT. METAM on the `Fraud Detection` dataset delivers the best performance by finding a 1-hop join path to the most useful feature. FEATPILOT is slightly worse than METAM due to its IQ and FI estimation errors amplified in this extreme case.

Overall, the naive Exhaustive method exhibits moderate performance but incurs substantial computational costs and excessive intermediate memory usage due to evaluating all possible augmentations. This inefficiency arises because not all feature-path pairs contribute positively to ML tasks. Instead, indiscriminate augmentation may introduce noise, redundancy, and high dimensionality, complicating subsequent feature selection and degrading model performance. ARDA falls behind as it fails to leverage distant features. FEATPILOT outperforms other multi-hop approaches by addressing key limitations: DFS performs feature augmentation without considering downstream ML performance, often leading to suboptimal results. METAM and AutoFeature prioritize feature importance while neglecting integration quality, which can result in misidentifying the best candidate features. Furthermore, FEATPILOT outperforms AutoFeat by decomposing utility gain into integration quality and feature importance, reducing the problem's dimensionality and enabling more efficient and reliable optimization.

C. Ablation Study

To better understand how our key designs of FEATPILOT benefit the overall performance feature augmentation, we conduct extensive ablation studies to evaluate the performance of FEATPILOT's individual components and their variants.

1) Effectiveness of FEATPILOT*'s IQ Model.:* First, we evaluate the accuracy of FEATPILOT's LSTM-based IQ estimator. The ground truth is established by materializing the join paths and computing the corresponding IQ scores according to Definition 8. We measure and report the MSE between the estimated values and the ground truth values. We compare against three baselines: (1) *Production*, multiplication of *Transitivity* scores between the tables on a join path, (2) *Random [0,1]*, randomly predicting an IQ score between 0 and 1, and (3) an LSTM-based IQ estimator trained on randomly selected feature-path pairs and vary the number of training pairs.

Overall, FEATPILOT's LSTM-based IQ estimator achieves the best (lowest) MSE compared to all baselines across six different datasets, as shown in Figure 6. The random estimator performs the worst (highest) MSE. Notably, our method consistently outperforms both Production and Random methods,

Datasets	Metrics	Feature Budgets	Methods						
			Exhaustive	DFS	ARDA	AutoFeature	AutoFeat	METAM	FEATPILOT
School	Accuracy	1	0.704(4)	0.704(4)	0.697(7)	0.708(3)	0.704(4)	0.790(2)	**0.823**(1)
		5	0.730(4)	0.700(7)	0.808(2)	0.704(6)	0.710(5)	0.801(3)	**0.891**(1)
		10	0.704(6)	0.692(7)	0.794(3)	0.723(4)	0.718(5)	0.816(2)	**0.880**(1)
DonorsChoose	Accuracy	1	0.682(4)	0.656(5)	**0.856**(1)	0.708(3)	0.656(5)	0.656(5)	0.822(2)
		5	0.834(4)	0.820(5)	0.890(2)	0.852(3)	0.681(6)	0.659(7)	**0.954**(1)
		10	0.837(5)	0.854(4)	0.901(2)	0.896(3)	0.818(7)	0.820(6)	**0.961**(1)
Diabetes	Accuracy	1	0.521(6)	0.521(6)	0.525(4)	0.585(3)	0.525(4)	0.616(2)	**0.678**(1)
		5	0.740(2)	0.631(5)	0.584(7)	0.649(3)	0.605(6)	0.647(4)	**0.742**(1)
		10	**0.746**(1)	0.647(5)	0.616(7)	0.651(4)	0.618(6)	0.655(3)	0.744(2)
Fraud Detection	F1	1	0.325(4)	0.068(7)	0.416(3)	0.145(6)	0.296(5)	**0.437**(1)	0.435(2)
		5	0.440(3)	0.070(7)	0.422(4)	0.152(6)	0.422(4)	0.446(2)	**0.493**(1)
		10	0.517(2)	0.084(7)	0.450(4)	0.162(6)	0.441(5)	0.464(3)	**0.540**(1)
Poverty	MAE	1	8781.90 (2)	13620.14 (7)	12389.54 (3)	13532.57 (6)	12944.16(4)	13077.66 (5)	**8222.34** (1)
		5	7373.94 (2)	13410.07 (6)	12389.54 (3)	13532.57 (7)	12558.93(4)	12956.29 (5)	**7322.44** (1)
		10	7309.52 (2)	13077.66 (6)	12164.23 (4)	13411.85 (7)	11213.23(3)	12786.82 (5)	**7182.38** (1)
Air	MSE	1	1.201 (7)	1.184 (5)	**0.969** (1)	1.259 (6)	1.061(4)	1.101 (2)	1.101 (2)
		5	0.983 (5)	0.985 (6)	0.793 (2)	1.219 (7)	0.915(4)	0.900 (3)	**0.762** (1)
		10	0.873(5)	0.943 (6)	0.761 (2)	1.202 (7)	0.820(4)	0.762 (3)	**0.715** (1)

TABLE II: Baseline comparison against the state-of-the-art methods. The top ranked results are bold. The numbers in the parentheses indicate the result ranking on a particular dataset.

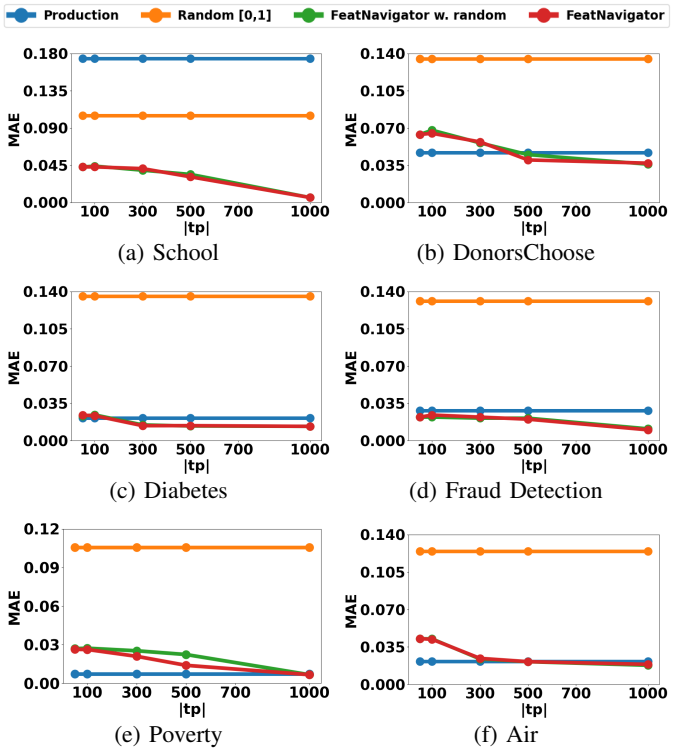


Fig. 6: Effectiveness of FEATPILOT's IQ model.

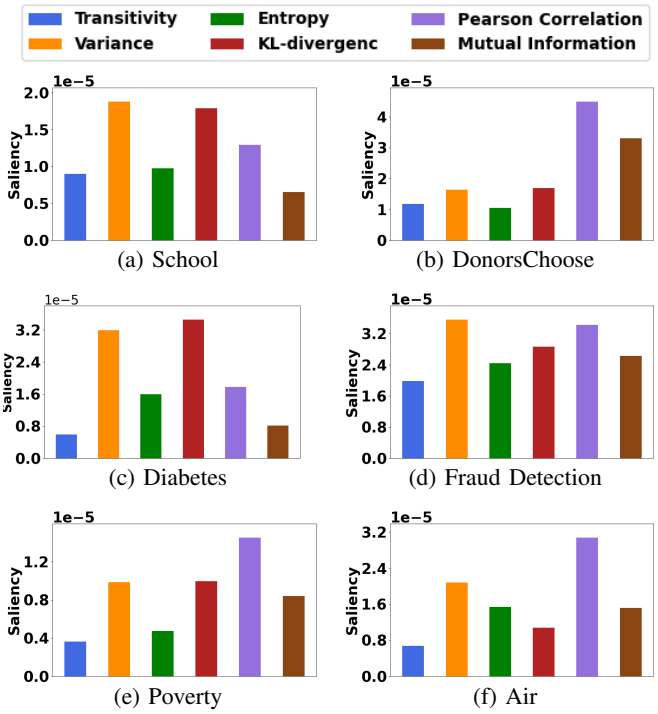


Fig. 7: Saliency analysis of features used in IQ model.

which empirically verifies the usefulness of the proposed features for our IQ estimator. With the increasing number (from 50 to 1000) of the training join paths $|tp|$, our IQ estimator produces more accurate estimations. Note that 1000 training feature-path pairs only account for approximately 0.1% of all possible pairs in the smallest dataset (i.e., DonorsChoose). We also observe that random data selection strategy demonstrates competitive performance. It is expected as all test sets are randomly drawn from six datasets, respectively.

Saliency Analysis of Input Features to IQ We further employ the Saliency method [33] to measure the importance of each input feature to the IQ model, by assessing their respective gradient concerning the predicted IQ value. As shown in Figure 7, the IQ model benefits the most from *Pearson Correlation* and *Variance*, and the other features also contribute to the IQ model to varying degrees with different datasets. This confirms that the chosen features are essential to a high-quality IQ estimator in FEATPILOT.

2) Effectiveness of FEATPILOT*'s FI Estimator.:* Second, we evaluate the accuracy of our FI estimator. The ground truth is established by materializing the test join paths and computing the corresponding FI scores using AutoGluon. We measure and report the MAE between the estimated values and the ground truth values. We also vary the size of the training set from 50 to 1000. We compare against three alternative

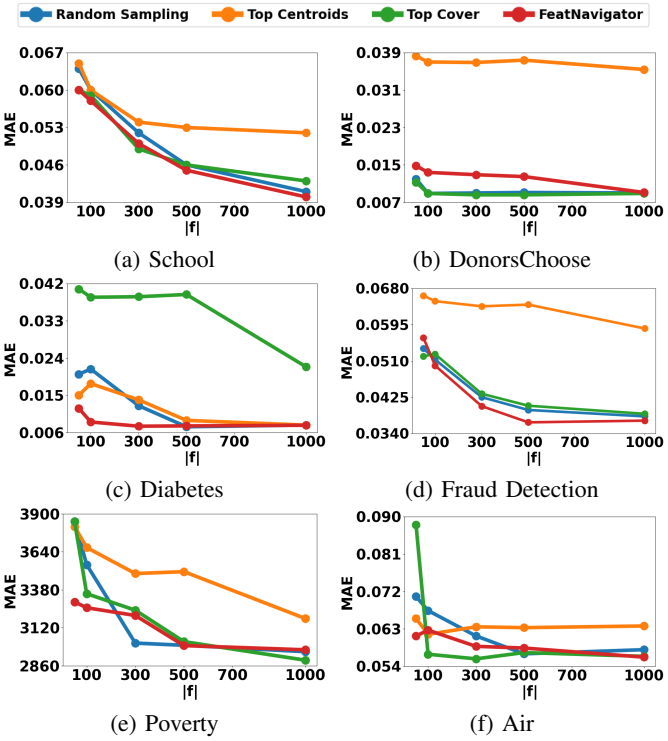


Fig. 8: Effectiveness of FEATPILOT 's FI estimation.

designs: (1) a random sampling method that randomly choose $|f|$ destination features. We ensure that at least one sample is drawn from each cluster; (2) a top centroid-based method that allocate budgets uniformly across clusters. Within each cluster, we select the top central features; and (3) a top cover method that equally distributes budgets across clusters. The objective within each cluster is to identify points that minimize the *coverage radius*.

Overall, as shown in Figure 8, we observe that FEATPILOT's estimation quality (i.e., lower MAE) improves with an increasing number of data points collected from feature clusters. FEATPILOT outperforms the other baselines on `School`, `Diabetes`, `Fraud Detection` and `Air` datasets, and its performance on `DonorsChoose` and `Poverty` is also very close to the best performing *top cover* method. When the given budget is small (i.e., $|f| \leq 100$), the random sampling method is slightly more effective on `DonorsChoose` and `School` datasets. This is consistent with our observation in the above integration quality evaluation. Hence, one could take a hybrid approach that randomly samples the first 50 or 100 feature-path pairs and then adapts to the proposed data selection strategy when establishing the FI estimator and the IQ model.

D. FEATPILOT's Search Algorithm Performance.

We evaluate the effectiveness and efficiency of FEATPILOT's search algorithm. We report the final Accuracy, F1, MAE or MSE as well as the compute time spent on search. To make sense of the measurements, we compare FEATPILOT's search algorithm against two baselines: (1) an exhaustive method that uses BFS to exhaustively search and estimate all possible combinations of destination features and integration paths; and (2) a greedy method that chooses features with highest estimated FI for candidate features. For each candidate feature, it further selects the path with the fewest number of joins (i.e., shortest integration path). If more than one integration paths are the shortest, the greedy method picks the one with the highest estimated IQ. We vary the feature budget from 1 to 10, to be consistent with Table II.

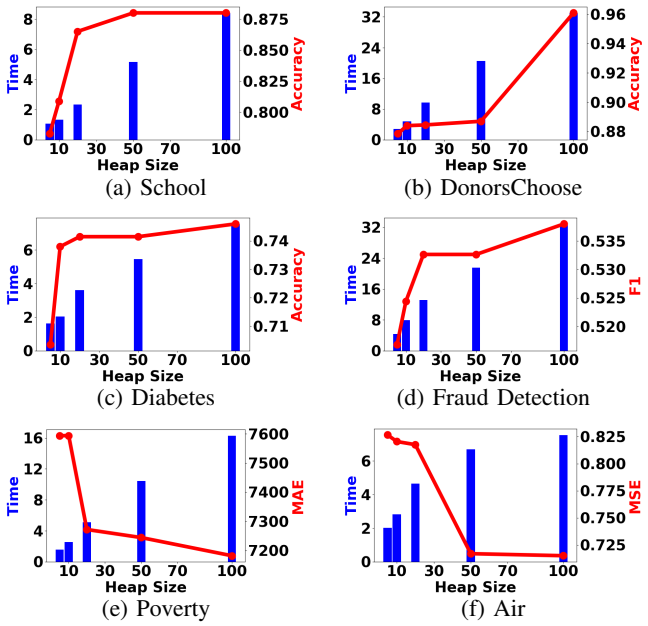


Fig. 9: Varying heap size for FEATPILOT's search algorithm. Blue bars represent search time, while red lines indicate ML performance metrics.

As shown in Table III, FEATPILOT's search algorithm significantly outperforms the greedy algorithm and achieves similar performance to the exhaustive algorithm in Accuracy or F1. Regarding the efficiency, FEATPILOT's search algorithm also achieves 2-5 times speedup compared to the exhaustive algorithm. Moreover, FEATPILOT's compute time, on average, increases 2% when increasing the feature budget from 1 to 10. This shows that FEATPILOT's search algorithm is scalable to a large number of features allowed for augmentation.

We further evaluate the performance of FEATPILOT's search algorithm by varying heap sizes H from 5 to 100 and report the final model performance and the search compute time. As shown in Figure 9, the model performance shows solid improvements with an increasing heap size, while the latency only increases sub-linearly. This showcases that FEATPILOT's search and refinement algorithms are effective and efficient.

End-to-End Latency. In Table IV, we report the end-to-end latency of FEATPILOT and other baselines over six datasets. The feature budget is set to 10 to be consistent with Table II. Specifically, FEATPILOT is 2.1% and 18.4% faster than METAM on `Fraud Detection` and `School` datasets, and 6.2% faster than ARDA on `DonorsChoose` dataset. This shows that FEATPILOT not only delivers the best model performance but also has lower latency compared to the best performing baselines.

Methods	Exhuasive						Greedy						FEATPILOT					
Datasets	$\|B\|$=1		$\|B\|$=5		$\|B\|$=10		$\|B\|$=1		$\|B\|$=5		$\|B\|$=10		$\|B\|$=1		$\|B\|$=5		$\|B\|$=10	
	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
School	0.823	45.33	0.880	45.53	0.885	45.69	0.720	3.02	0.801	3.08	0.805	3.16	0.823	8.046	0.891	8.194	0.880	8.392
DonorsChoose	0.822	71.99	0.954	72.21	0.961	72.52	0.855	1.77	0.925	1.88	0.955	2.05	0.822	32.51	0.954	32.70	0.961	32.97
Diabetes	0.678	7.75	0.742	7.98	0.744	8.16	0.678	2.12	0.742	2.16	0.744	2.25	0.678	7.39	0.742	7.57	0.744	7.85
Fraud Detection	0.403	61.54	0.513	61.78	0.538	61.80	0.403	3.55	0.492	3.62	0.478	3.76	0.403	32.58	0.513	32.66	0.538	32.81
Poverty	8222.34	36.66	7739.44	36.75	7473.87	36.85	8222.34	1.34	7885.65	1.39	7716.36	1.43	8222.34	16.15	7322.44	16.20	7182.38	16.26
Air	1.044	16.79	0.7619	16.92	0.7158	17.15	1.090	2.06	0.766	2.20	0.732	2.33	1.101	7.19	0.762	7.32	0.715	7.53

TABLE III: Effectiveness and efficiency of FEATPILOT's search algorithm (Time in seconds).

Datasets	School		DonorsChoose		Diabetes		Fraud Detection		Poverty		Air	
Methods	Actual Score	Time	Actual Score	Time	Actual Score	Time	Actual Score	Time	Actual Score	Time	Actual Score	Time
DFS	0.692	53	0.854	140	0.647	100	0.084	95	13077.66	184	0.9425	121
ARDA	0.794	1,517	0.901	1,707	0.616	1,446	0.450	921	12164.23	3,155	0.7614	3,003
AutoFeature	0.723	3,488	0.896	3,403	0.651	2,927	0.162	3,942	13077.66	9,268	1.2016	4,747
AutoFeat	0.718	303	0.818	1,950	0.618	1,743	0.441	982	11213.23	1,714	0.820	3,366
METAM	0.816	1,723	0.820	1,120	0.655	2,141	0.464	1,912	12786.82	1,651	0.7618	2,249
FEATPILOT	0.880	1,406	0.961	1,602	0.744	1,144	0.540	1,872	7182.38	1,059	0.7154	1,488

TABLE IV: End-to-end latency (Time in seconds).

VI. RELATED WORK

Feature Augmentation. Automatic feature augmentation has been intensively studied and shown effective for various ML tasks including classification, regression, clustering, etc. Kumar et al. [10] avoid primary key-foreign key join without sacrificing the performance of the model. Deep Feature Synthesis (DFS) [2] is one of the early work in generating features for relational datasets. DFS follows joinable relationships in the data to a base table, and then sequentially applies mathematical functions along that path to create the final features. However, DFS fails to address any of three challenges in this paper, as it relies on a brute-force approach to explore all potential joinable tables. ARDA [3] is a feature augmentation framework that leverages existing data discovery tools [18] to score the candidate tables and uses a heuristic algorithm to select features. Hence, the performance of ARDA heavily relies on the scores given by Aurum, which are not always accurate as they are not model-aware. ARDA supports only single-hop paths (star schemata), thereby bypassing the challenges we address with FEATPILOT. However, this limitation prevents it from leveraging useful tables that are multi-hops away.

Recently, AutoFeature [4] leverages reinforcement learning (RL) for feature augmentation. It uses either a multi-armed bandits or a branch Deep Q Networks to choose between exploring features that could lead to performance improvement and exploiting features that are rarely selected. METAM [5] is a goal-oriented framework that also utilizes the multi-arm bandit method for feature discovery and augmentation to achieve the desired performance of a given task. It uses data properties and utility functions from downstream tasks to drive the discovery and augmentation process. AutoFeat [32] also explores multi-hop join paths to find relevant features in order to augment the base table with additional features. It uses relevancy and redundancy scores derived from the join paths to rank feature candidates. While AutoFeature, METAM, and AutoFeat incorporate multi-hop join paths to identify relevant features, AutoFeature and METAM suffer from expensive join operations and search complexities, resulting in higher end-to-end latency. AutoFeat mitigates the cost of joins by ranking paths based on cheaper metrics (i.e., relevancy and redundancy scores), but it does so at the expense of search quality, as it does not effectively address search complexities.

Coreset selection [34], [35] aims to select a high-quality coreset without materializing the augmented table and uses its weighted gradients to approximate the full gradient of the entire train dataset. Such method avoids unnecessary joins and keeps the performance of the model, rather than augmenting useful features to improve the performance of the model.

Feature Selection. Feature selection [36], [37] aims to identify a subset of the most relevant features from a large dataset for a specific ML task, thereby reducing feature dimensionality. In contrast, feature augmentation involves increasing the number of features to enhance the model's predictive capability. The ranking-based methods evaluate the importance of each feature using statistical metrics like correlation coefficient or mutual information with the labels [38] and then select the top relevant features. However, ranking-based methods may yield a sub-optimal subset with highly correlated variables, where a smaller subset would suffice. Sequential algorithms [39] either add or remove features from an initial empty (or full) set until the utility gain is maximized, using a criterion to incrementally enhance the utility gain with fewer features. Heuristic algorithms [40] explore different subsets, either by searching within a space or generating solutions to optimize the utility gain. Due to the NP-hard nature of evaluating 2^N subsets, these methods find sub-optimal subsets heuristically. Cherepanova et al. [41] propose an input-gradient-based analogue of Lasso for neural networks that outperforms classical feature selection methods on challenging problems such as selecting from corrupted or second-order features in the tabular deep learning setting.

VII. CONCLUSION

In this paper, we propose FEATPILOT, an automatic feature augmentation framework by decomposing it into feature importance and integration quality. We design a lightweight clustering-based method for feature importance estimation and a learning-based model for integration quality. FEATPILOT's feature path search algorithm efficiently identifies high-quality features and their optimized integration paths. Our experiments on six public datasets show the effectiveness of FEATPILOT compared to state-of-the-art methods, improving ML model performance by up to 10.27%.

REFERENCES

[1] T. R. Hoens and N. V. Chawla, "Learning in non-stationary environments with class imbalance," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 168–176.

[2] J. M. Kanter and K. Veeramachaneni, "Deep feature synthesis: Towards automating data science endeavors," in *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA*, 2015, pp. 1–10.

[3] N. Chepurko, R. Marcus, E. Zgraggen, R. C. Fernandez, T. Kraska, and D. Karger, "Arda: Automatic relational data augmentation for machine learning," *Proc. VLDB Endow.*, vol. 13, no. 9, p. 1373–1387, 2020.

[4] J. Liu, C. Chai, Y. Luo, Y. Lou, J. Feng, and N. Tang, "Feature augmentation with reinforcement learning," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 3360–3372.

[5] S. Galhotra, Y. Gong, and R. C. Fernandez, "Metam: Goal-oriented data discovery," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, 2023, pp. 2780–2793.

[6] X. L. Dong and T. Rekatsinas, "Data integration and machine learning: A natural synergy," in *Proceedings of the 2018 International Conference on Management of Data*. Association for Computing Machinery, 2018, p. 1645–1650.

[7] Z. Chen and M. J. Cafarella, "Integrating spreadsheet data via accurate and low-effort extraction," in *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*. ACM, 2014, pp. 1126–1135.

[8] R. Chen, C. Dewi, S. Huang, and R. E. Caraka, "Selecting critical features for data classification based on machine learning methods," *J. Big Data*, vol. 7, no. 1, p. 52, 2020.

[9] X. Ying, "An overview of overfitting and its solutions," in *Journal of physics: Conference series*, vol. 1168. IOP Publishing, 2019, p. 022022.

[10] A. Kumar, J. Naughton, J. M. Patel, and X. Zhu, "To join or not to join? thinking twice about joins before feature selection," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, p. 19–34.

[11] A. Khatiwada, R. Shraga, W. Gatterbauer, and R. J. Miller, "Integrating data lake tables," *Proc. VLDB Endow.*, vol. 16, no. 4, pp. 932–945, 2022.

[12] A. Khatiwada, R. Shraga, and R. J. Miller, "DIALITE: discover, align and integrate open data tables," in *Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023*, 2023, pp. 187–190.

[13] J. Vermorel and M. Mohri, "Multi-armed bandit algorithms and empirical evaluation," in *Machine Learning: ECML 2005, 16th European Conference on Machine Learning, Porto, Portugal, October 3-7, 2005, Proceedings*, 2005, pp. 437–448.

[14] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, 2016, p. 1995–2003.

[15] C. Chai, J. Liu, N. Tang, G. Li, and Y. Luo, "Selective data acquisition in the wild for model charging," *Proc. VLDB Endow.*, vol. 15, no. 7, p. 1466–1478, 2022.

[16] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," in *Machine Learning, Proceedings of the Eleventh International Conference*, 1994, pp. 121–129.

[17] R. Battiti, "Using mutual information for selecting features in supervised neural net learning," *IEEE Transactions on Neural Networks*, vol. 5, no. 4, pp. 537–550, 1994.

[18] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker, "Aurum: A data discovery system," in *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, 2018, pp. 1001–1012.

[19] S. Castelo, R. Rampin, A. Santos, A. Bessa, F. Chirigati, and J. Freire, "Auctus: A dataset search engine for data discovery and augmentation," *Proc. VLDB Endow.*, vol. 14, no. 12, p. 2791–2794, 2021.

[20] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola, "Autogluon-tabular: Robust and accurate automl for structured data," *arXiv preprint arXiv:2003.06505*, 2020.

[21] S. H. Lim, N. B. Erichson, F. Utrera, W. Xu, and M. W. Mahoney, "Noisy feature mixup," *CoRR*, vol. abs/2110.02180, 2021. [Online]. Available: https://arxiv.org/abs/2110.02180

[22] S.-H. Hwang, M. Kim, and S. E. Whang, "Rc-mixup: A data augmentation strategy against noisy data for regression tasks," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 1155–1165.

[23] J. Sun and G. Li, "An end-to-end learning-based cost estimator," *Proc. VLDB Endow.*, vol. 13, no. 3, p. 307–319, 2019.

[24] S. Kwon, W. Jung, and K. Shim, "Cardinality estimation of approximate substring queries using deep learning," *Proc. VLDB Endow.*, vol. 15, no. 11, p. 3145–3157, 2022.

[25] A. Authors, "Source code," 2024. [Online]. Available: https://anonymous.4open.science/r/FeaturePilot-C308

[26] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[27] S. Hooker, D. Erhan, P.-J. Kindermans, and B. Kim, "Evaluating feature importance estimates," 2018.

[28] B. Škrlj, S. Džeroski, N. Lavrač, and M. Petkovič, "Feature importance estimation with self-attention networks," *arXiv preprint arXiv:2002.04464*, 2020.

[29] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Jun. 2019, pp. 4171–4186.

[30] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, 1996, pp. 226–231.

[31] P. S. Ow and T. E. Morton, "Filtered beam search in scheduling," *The International Journal Of Production Research*, vol. 26, no. 1, pp. 35–62, 1988.

[32] A. Ionescu, K. Vasilev, F. Buse, R. Hai, and A. Katsifodimos, "Autofeat: Transitive feature discovery over join paths," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 1861–1873.

[33] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: visualising image classification models and saliency maps," in *Proceedings of the International Conference on Learning Representations (ICLR)*. ICLR, 2014.

[34] J. Wang, C. Chai, N. Tang, J. Liu, and G. Li, "Coresets over multiple tables for feature-rich and data-efficient machine learning," *Proc. VLDB Endow.*, vol. 16, no. 1, p. 64–76, 2022.

[35] C. Chai, J. Liu, N. Tang, J. Fan, D. Miao, J. Wang, Y. Luo, and G. Li, "Goodcore: Data-effective and data-efficient machine learning through coreset selection over incomplete data," *Proc. ACM Manag. Data*, vol. 1, no. 2, pp. 157:1–157:27, 2023.

[36] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.

[37] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Computing Survey*, vol. 50, no. 6, dec 2017.

[38] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[39] P. Pudil, J. Novovičová, and J. Kittler, "Floating search methods in feature selection," *Pattern Recognition Letters*, vol. 15, no. 11, pp. 1119–1125, 1994.

[40] D. E. Goldberg, *Genetic algorithms*. pearson education India, 2013.

[41] V. Cherepanova, R. Levin, G. Somepalli, J. Geiping, C. B. Bruss, A. G. Wilson, T. Goldstein, and M. Goldblum, "A performance-driven benchmark for feature selection in tabular deep learning," in *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.