AutoFeat: Transitive Feature Discovery over Join Paths

Andra Ionescu, Kiril Vasilev, Florena Buse, Rihan Hai, Asterios Katsifodimos
Delft University of Technology
{a.ionescu-3, r.hai, a.katsifodimos}@tudelft.nl, {k.v.vasilev-1, i.buse}@student.tudelft.nl

***Abstract*—Can we automatically discover machine learning (ML) features in a large data lake in order to increase the accuracy of a given ML model? Existing solutions either focus on simple star schemata, failing to discover features in more complex real-world schemata or consider only PK-FK relationships in clean, curated databases. However, real-world data lakes can contain long join paths of uncurated joinability relationships resulting from automated dataset discovery methods.**

This paper proposes a novel ranking-based feature discovery method called AutoFeat. Given a base table with a target label, AutoFeat explores multi-hop, transitive join paths to find relevant features in order to augment the base table with additional features, ultimately leading to increased accuracy of an ML model. AutoFeat is *general*: it evaluates the predictive power of features without the need to train an ML model, ranking join paths using the concepts of relevance and redundancy. Our experiments on real-world open data show that AutoFeat is *efficient*: it can find features of high predictive power on data lakes with an increased number of dataset joinability relationships 5x-44x faster than baseline approaches. In addition, AutoFeat is *effective*, improving accuracy by 16% on average compared to the baseline approaches, even in noisy, uncurated data lakes.

I. Introduction

Machine learning is widely used in various domains, such as retail, medical diagnosis, and transportation. An ML model's performance (e.g., accuracy) heavily depends on its training data [1]. Although it is a common assumption that the input data of a model is a single table, in practice, the situation can be more complex. Features with high predictive power may reside in different database tables or even in multiple files in an open data repository/lake.

Recent works [2], [3] focus on augmenting a base table with features, using join paths to drive their search. However, they first search for data, then join, and then apply feature selection to prune out noisy or irrelevant features based on ML model performance. Given data lakes where primary-key/foreign-key (PK-FK) constraints are missing, it is necessary to use dataset discovery algorithms as a first step of data augmentation to find relationships between tables. However, this process is known to output spurious connections, where what might be considered a join results in an irrelevant table with unrelated data. This becomes an even bigger problem when two datasets can be joined via multiple join columns. In this context, state-of-the-art augmentation processes fail.

With this paper, we introduce *transitive feature discovery*, which aims to discover and augment relevant features over join paths without using ML models in the process. The main idea behind our approach is to explore the space of joinable tables and to prune out the low-quality join paths based on data quality measures, as well as relevance and redundancy metrics. As shown in Figure 1, our approach (AutoFeat) outperforms the competition both in effectiveness and efficiency. AutoFeat's effectiveness benefits come from its ability to explore join paths beyond the star schemata supported by [2], managing to find join paths that contain features of high predictive power. In addition, by simply ranking join paths using relevance and redundancy metrics, instead of training expensive ML models during search [2], [3], AutoFeat is not only able to explore a larger space of transitive joins paths, but also be 5x-44x more efficient than the competition.

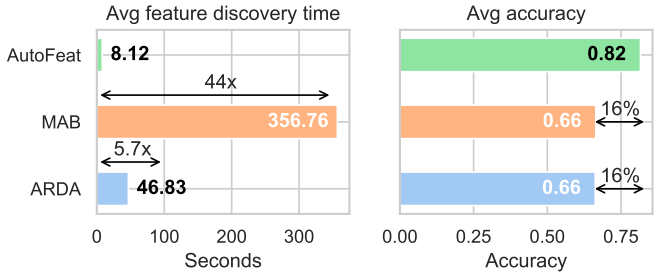


Figure 1: AutoFeat outperforms the state-of-the-art data augmentation frameworks regarding feature discovery/augmentation time, as it is faster than any approach, and the resulting augmented table shows an increase in accuracy when used for ML tasks.

Goal. The goal of feature discovery is to enrich an original base table with new features with high predictive power for a target ML model on a classification task. AutoFeat automates the process of identifying and joining relevant tables from a dataset collection to a base table, thus creating an augmented table. AutoFeat applies heuristic-based feature selection techniques on the augmented table to prune out any noisy or irrelevant features. AutoFeat performs these tasks effectively and efficiently, reducing the need for manual data engineering efforts and improving the performance (i.e., accuracy) of the subsequent ML processes.

Contributions. In short, the proposed method AutoFeat, has the following desirable properties:

- *General*: AutoFeat can explore join paths beyond star schemata in order to augment a given base table. In addition, AutoFeat's join path ranking mechanism does

not depend on training the target ML model.

- *Efficient*: instead of repeatedly training the underlying model to assess the accuracy benefits of a given join path, `AutoFeat`: i) prunes out non-promising join paths using data quality metrics, and ii) proposes a ranking function that chooses the top-k most promising join paths according to information theoretical metrics that encode feature relevance and redundancy. The result is 5x-44x faster feature augmentation.
- *Effective*: while being faster, `AutoFeat`'s feature augmentation strategy achieves on average 16% higher accuracy compared to the state-of-the-art methods on real-world, open data repositories.

Outline. We review the relevant literature in Section II, define our problem and provide an overview of the concepts of relevance and redundancy in Section III. In Section IV, we delve into the concepts and techniques behind the dataset relation graph. Next, we perform an empirical analysis of the popular feature selection strategies in Section V, and in Section VI, we describe the algorithm behind `AutoFeat`. We evaluate our approach, summarise our findings in Section VII, and conclude in Section VIII.

Reproducibility. Our code is open source at https://github.com/delftdata/autofeat. The repository also contains the sources of the datasets used in our evaluation.

II. Related Work

Dataset Discovery. Dataset discovery helps users explore a vast collection of heterogeneous datasets and find related tables to perform a data-driven task [1], [4]. A large corpus of dataset discovery works focus on finding unionable tables [5], joinable tables [6]–[9], while some tackle both relatedness scenarios [10], [11]. We consider feature discovery a more tailored and specific dataset discovery process, where we are only interested in the tables containing features *fit* to augment a base table with more *relevant* information.

Dataset Augmentation. Dataset augmentation has been studied from two angles: when KFK constraints are known and when these are discovered using dataset discovery approaches [3], [12]–[15]. Dataset discovery approaches use joinability graphs to model the datasets and the relations between them, limiting the length of the join paths [2], [6]. These works mostly focus on augmenting directly joinable tables and rely on machine learning models for feature selection [2], [16]. In short, as seen in Table I, ARDA supports single-hop paths (star schemata), while MAB and `AutoFeat` support multi-hop join paths. Both ARDA and MAB require an expensive model execution step to evaluate the quality of a join path, while `AutoFeat` ranks paths according to cheaper metrics (Section V). Finally, for each pair of tables, instead of supporting a single join possibility, `AutoFeat` supports multiple ones (joinability *multigraph*).

Transitive Joins. Transitive joins have been proven to be effective for augmentation in the context of notebooks [11], [17], or as an augmentation strategy for tuples or missing values [14]. `AutoFeat` uses transitive joins to navigate the join space and explore candidates for *feature* augmentation using multi-hop join paths.

Table I: Comparison of state-of-the-art methods.

	Join path Length	Path / Feature Selection	Joinability Graph
ARDA	Single-hop	Model-execution based	Simple Graph
MAB	Multi-hop	Model-execution based	Simple Graph
AutoFeat	Multi-hop	Ranking-based	Multigraph

Feature Selection. Feature selection methods are categorised based on selection strategies into filter, wrapper, and embedded types [18], [19]. Filter methods are independent of the ML model, wrappers assess feature quality based on learner performance, and embedded methods integrate feature selection into the training process. While traditional feature selection assumes training data is stored in a single table, our feature discovery approach addresses challenges arising from data spread across multiple tables.

III. Transitive Feature Discovery

In this section, we provide an overview of the foundational concepts that underpin this paper, such as relevance and redundancy in the context of machine learning; then, we define the problem of transitive feature discovery and provide an overview of our approach.

A. Preliminaries

With transitive feature discovery, we aim to discriminate between relevant and redundant features and reduce useless information [20]. Given a dataset T_i which comprises a collection of features $X_1, X_2, ..., X_n$, and a feature containing the labels Y, and given $S_i = \{X_1, ..., X_{i-i}, X_{i+1}, ..., X_m\}$ a collection of features without X_i, we define the concepts of relevance and redundancy as follows.

Relevance. The relevance of a feature has multiple definitions based on the objective, and it has been shown that a general definition of relevance is not universally applicable [21]. Therefore, in feature selection, we must distinguish between *strong relevance* and *weak relevance* [19], [21], [22].

Strong relevance of a feature X_i means that removing the feature results in the degradation of the optimal feature subset. Contrary to strong relevance, weak relevance of a feature X_i means that the feature is not always necessary (i.e., not relevant), but the performance of a learner on a subset of features S_i is worse than on a subset of features $S_i \cup \{X_i\}$. Finally, the relevant features influence the output, and their role is unique (e.g., no redundancy), while irrelevant features have no influence on the output [20], [22].

Redundancy. Feature redundancy is tightly coupled to feature dependency or feature correlation, meaning that perfectly correlated features are redundant to each other because they do not introduce any new information [19]. The redundant features are those which can take the role of another feature [20]. The most straightforward intuition is that a redundant feature is a duplicate of a relevant feature. It is worth mentioning that feature redundancy is not absolute, as it is conditioned on a

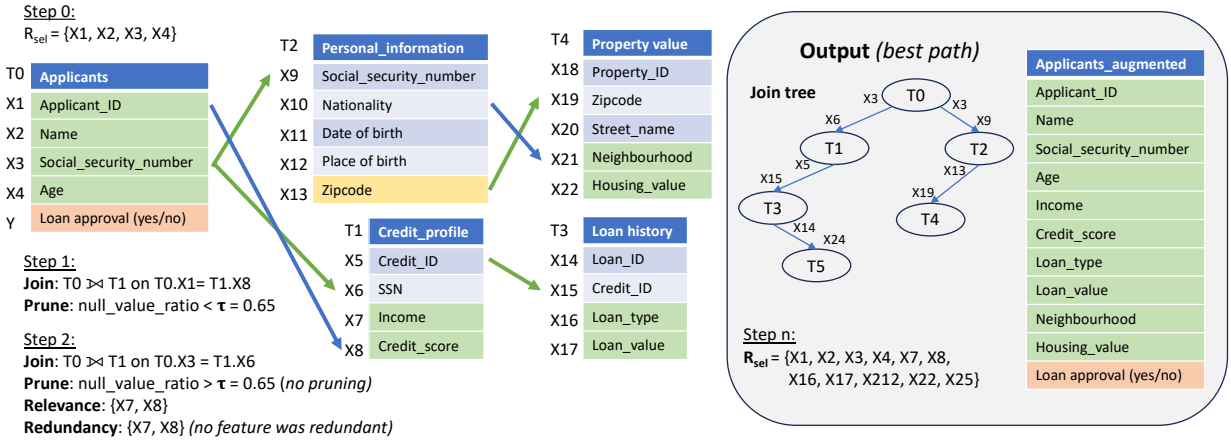


Figure 2: Running example: the base table *Applicants* contains the label *Loan approval*. The green-coloured features represent the features with high predictive power, while the yellow-coloured feature represents the join column used to reach the transitive table *Property value*, which contains relevant features for our classification task. The green arrows represent the paths which contain the relevant features. The best path is abstracted in the right-hand side of the figure as a join tree.

feature subset. Changing the feature subset leads to changing the decision of whether a feature is redundant or not. Finally, feature redundancy aims to identify the redundant features and *remove* them [19].

Summary & Goal. The goal of this work is to maximise relevance and minimise redundancy to find the features which are highly correlated with the label (i.e., relevant) and are not yet represented by any other feature from the selected features subset (i.e., non-redundant).

B. Problem Definition

We introduce *transitive feature discovery*, a process at the intersection of dataset discovery, dataset augmentation and feature selection. We leverage the exploration step and joinability scores from dataset discovery to augment a given table. The (partially) augmented result undergoes a feature selection step, where we select only the features that increase the information value (i.e., relevant and non-redundant). Formally, we define transitive feature discovery as follows:

Definition III.1 (*Transitive feature discovery*)**.** Given a base table T_i with a label column Y and a collection of datasets D with or without Key-Foreign Key (KFK) relations, transitive feature discovery extends the base table with more relevant features X_i with the aim of solving a task M.

Input. We take as an input (i) a table T_0 comprising of n features $\{X_1, X_2, ..., X_n\}$, and a feature Y with the labels, which we further name the *base table*, and (ii) a collection of datasets $\mathbf{T} = \langle T_1, \ldots, T_n \rangle$.

Setting. The base table is located in a data lake, surrounded by a collection of datasets, where we distinguish two scenarios: (i) the relationships between the base table and the other datasets are undiscovered and unknown, or (ii) the base table has known KFK relationships with other datasets.

Output. Our approach outputs a ranked list of top-k join paths. Each join path contains the datasets for augmentation with their respective join keys and a list of selected features, representing the optimal subset of features in the join path, leading to increased performance of an ML classification task.

Example. Take as an example the collection of datasets from Figure 2. Transitive feature discovery aims to enrich the base table *Applicants* with new features that have high predictive power for the target task of predicting bank loan eligibility (*Loan approval*). The candidates for augmentation are the following datasets: *Personal_information*, *Credit_profile*, *Property value*, and *Loan history*. This example contains relationships discovered by dataset discovery algorithms, which can be spurious (e.g., $Applicants.Applicants\_ID \rightarrow Credit\_profile.Credit\_score$). The output shows the result of the best join path, which is depicted as a join tree with its corresponding join keys. It also shows the selected subset of features and the final augmented table.

C. Approach Overview

Our approach consists of an offline component: the joinability graph construction. This phase transforms the base table and the dataset collection into a graph structure, a process we detail in Section IV.

The online component is the entire augmentation pipeline: we traverse the graph, begin the discovery process, and the feature selection phase. During graph traversal, outlined in Section IV-A, we execute the following processes: (i) the identification of joinable tables, (ii) and pruning paths, (iii) the identification of relevant features within a join path, and finally (iv) the decision on which features should be pruned based on redundancy. We further elaborate on our design choices concerning the join type in Section IV-B, and the strategies for table pruning in Section IV-C. Our decision-making process for feature selection follows an empirical approach, presented in detail in Section V.

`AutoFeat` traverses the joinability graph in search of relevant features, then ranks the join paths and produces a

list of the top-k paths along with their corresponding features. The mechanisms and design decisions integral to the ranking process are provided in Section VI. Finally, the top-k join paths are used to train k ML models, and the best join path (i.e., which produces the most accurate result) is returned.

IV. Dataset relation graph

This section delves into the core concepts and techniques behind the offline component. We make the general assumption that we have a collection of various input datasets coming from relational databases or data lakes. In a relational database, the metadata such as the schemata and PK-FK constraints are often defined. In data lakes, we assume the relationships between the datasets can be detected using dataset discovery tools [6]–[11], [13], [23], [24].

We create the joinability graph *Dataset Relation Graph* (DRG). DRG is a weighted graph where the nodes represent the datasets, while the edges represent the relations between these datasets. We use DRG (i) to capture the relationships between datasets, (ii) to traverse the graph following transitive joins, and (iii) to be able to enumerate multi-hop join paths, which we assume they contain valuable and relevant information for the base table. Before we explain how the DRG is constructed, we introduce a few concepts which will be used throughout the rest of the paper.

Definition IV.1 (*Joinability and join column*)**.** Given two datasets T_i and T_j, and their attributes X_i^i and X_j^i, where the superscript denotes the attribute i, while the subscript denotes the originating dataset, T_i is *joinable* with T_j if (1) there is a primary-key/foreign-key relationship between T_i and T_j, i.e., $T_i.X_i^i \rightarrow T_j.X_j^i$ where X_i^i is the foreign key and X_j^i is the corresponding primary key, or (2) X_i^i and X_j^i are related attributes (i.e., their intersection is non-empty). We refer to X_i^i and X_j^i as *join columns*.

Definition IV.2 (*Join graph and path*)**.** Consider a set of datasets $\mathbf{T} = \langle T_1, \ldots, T_n \rangle$. Its join graph $G_{\mathbf{T}} = (V, E)$ is an undirected graph with nodes V and edges E. Each dataset $T_i \in \mathbf{T}$ is represented as a node $v_i \in V$. If two datasets T_i and T_j are joinable, there is an edge $e_{ij} \in E$ between the nodes v_i and v_j. In a join graph, a join path is a finite sequence of edges that connect a sequence of distinct nodes.

Definition IV.3 (*Dataset Relation Graph*)**.** Consider a set of datasets $\mathbf{T} = \langle T_1, \ldots, T_n \rangle$. Its *dataset relation graph* is an undirected multi-graph $G_{\mathbf{T}} = (V, E)$, where each node $v_i \in V$ represents a dataset $T_i \in \mathbf{T}$, and the set of edges between two nodes v_i and v_j is a multiset E_{ij}. The multiset E_{ij} contains all the edges between v_i and v_j, representing multiple join opportunities given different join columns.

The DRG construction follows the next steps. First, we iterate through all the datasets and create the respective nodes in the graph. Second, if the datasets contain information about the integrity constraints, we ingest these constraints as edges with $weight = 1$. Finally, when the relations between tables are unknown, we employ a dataset discovery method: our current prototype uses COMA [25] for schema matching, according to Valentine [24]. The new relationships are modelled as edges with $weight = similarity\_score$, where the $similarity\_score$ is the score returned by COMA. DRG construction is independent of the dataset discovery algorithm; thus any algorithm which outputs a similarity score can be used to model the relationships across datasets.

Join Path Enumeration. We model the DRG as a multi-graph to capture the set of possible join columns between two tables. Given the dataset relation graph $G_{\mathbf{T}}$ and the base table T_0, it is straightforward to enumerate all possible paths starting from the node representing T_0. A path in DRG is a directed join path of minimum $length = 1$.

Definition IV.4 (*Join path search space*)**.** The join path search space J_i is all the acyclic paths in $G_{\mathbf{T}}$ that start from T_i and have $length \geq 1$.

In our approach, we navigate the join path space following transitive joins, thus creating longer join paths. We consider a different join path every edge in the multi-graph and every $n-hop$ traversal, which is a concatenation of paths. In Figure 2, the following is a join path with $length = 1$

$$Applicants.Applicant\_ID \rightarrow Credit\_profile.Credit\_score,$$

while the following is a different join path of $length = 2$

$$Applicants.Applicant\_ID \rightarrow Credit\_profile.Credit\_score \rightarrow Loan\_history.Credit\_ID$$

The DRG plays a crucial role in our methodology, serving as the backbone for all subsequent steps, including graph traversal, join operations, and path pruning, which we will elaborate on next.

A. Graph Traversal

Given a base table T_0, and its corresponding node v_0 in DRG, we set the stage for graph traversal. Among various graph traversal techniques, Depth First Search (DFS) and Breadth First Search (BFS) become particularly pertinent to our context [26], [27]. These methods require only the source node to traverse the graph, an approach that aligns with our setting: the feature discovery process starts with the base table. Exploring a join graph using either BFS or DFS can lead to a greedy approach, as both algorithms aim to find all join paths. However, `AutoFeat` uses BFS to traverse the join graph for feature augmentation for the following reasons.

BFS explores the join graph one level at a time, allowing us to evaluate the data quality after each level of join operations. This early determination of data quality enables us to optimise the data augmentation process and potentially avoid wasting computational resources on irrelevant join paths. While DFS might find a feasible join path quickly, it may not necessarily consider the most relevant datasets early in the process. This can lead to a decrease in data quality as it explores longer, less relevant paths before discovering the more informative ones. Moreover, errors from one join can propagate deeper into the join path, affecting the quality of the results even more. BFS

makes errors easier to manage and contain, as the exploration is performed level by level.

B. Join

Continuing this process, `AutoFeat` only considers *left* joins: we perform a *left* join between the base table T_0 and any other tables T_j under the assumption that $\forall T_j \in \mathbf{T}, Y \notin T_j$, i.e., any other table from the dataset does not contain a feature column with the class labels. For transitive joins, we treat the intermediate join result as a base table and perform a *left* join with the following table along the path, etc.

The *left* join is chosen primarily to maintain the number of tuples and, more critically, the number and distribution of classes in the label Y, which aligns with prior data augmentation approaches [2]. Using a different type of join could either remove or duplicate rows, both of which skew the class distribution and introduce class imbalance [28]. If not handled properly, such imbalance could alter the ML task or degrade performance [29]–[31].

Join Cardinality. To ensure the base table size remains constant (i.e., neither shrinks due to row removal nor expands due to row duplication) even when a *left* join is used, we transform one-to-many and many-to-many joins, thereby preventing data duplication and inconsistencies in labels. We group by the join column and randomly select a row [2]. Given that the DRG is a multi-graph, we apply this strategy for every possible join column due to its direct impact on the subsequent joins.

C. Pruning Paths

To further refine our approach and improve efficiency, it is necessary to consider the complexities involved with the join path search space. Working directly with the raw output of a dataset discovery method results in a significantly expanded search space, demanding the application of robust pruning strategies to manage this increased complexity effectively.

Similarity Score-Based Pruning. There may be instances where multiple possible join columns exist between two nodes as a result of a dataset discovery algorithm. In such cases, we explore each potential join using every respective join column. Preliminary experiments, however, suggest that a significant portion of these join results contain *null* values across their entire right-hand side. This result is far from ideal as an input for any ML algorithm. Thus, we implement our first pruning strategy at the join column level. Using the similarity score, we can prune weaker join columns. Given a base table, a joinable table, and a set of join columns, `AutoFeat` selects the join column with the highest similarity score. When multiple join columns share the same top score, each join from T_i to T_j using the join column X_j^i is an individual join path.

Data Quality-Based Pruning. The resulting augmented table may still suffer from poor quality, even when using the join column with the highest similarity score. A critical dimension of data quality is *completeness* [32], which can be gauged by the amount of *null* values present in a table. Several strategies can enhance table completeness: deletion, which is unsuitable in our context as it involves removing tuples; and imputation, which involves replacing *null* values with mean value, median value, most recurrent value, or a default value [29]. However, these artificially imputed values may skew the data distribution and introduce bias [29]. In light of the drawbacks associated with imputation or deletion, our goal is to augment the base table with datasets that result in a table with the highest possible completeness. As such, with our second pruning strategy based on completeness, we measure the *null* value ratio in the resulting join and prune the joins where this ratio falls below a predefined threshold τ. This threshold is incorporated into our approach as a hyper-parameter, and we demonstrate the effects of tuning it in our experiments (Section VII).

V. Feature Selection Strategies

In this section, we introduce streaming feature selection, explore a variety of relevance and redundancy metrics, and assess their performance. We use this empirical evaluation to drive our design decisions towards the best-performing methods and ensure our effectiveness and efficiency.

A. Streaming Feature Selection

Streaming feature selection assumes a constant number of rows. In contrast, the features arrive in a streaming fashion, one at a time, or in groups, with the goal to determine the subset of relevant features at a given time [18], [33]–[36]. Each new feature batch is derived from a join operation in relational data. Moreover, transitive join paths involve an implicit dependency. Each transitive join relies on the intermediate join and the features representing the join columns. Ensuring the persistence of these join column features is essential, and the feature selection algorithms must not eliminate them. Therefore, we must refrain from pruning intermediate joins, even if they contain irrelevant and redundant features, as they establish the pathway towards multi-hop join paths. `AutoFeat` builds upon the pipeline of streaming feature selection using the high-performance strategies for relevance and redundancy.

B. Empirical Evaluation Setting

Before we delve deeper into the empirical analysis of the methods for relevance and redundancy, we describe the experimental setup.

Datasets. For a comprehensive evaluation, we choose six binary classification datasets varying in domains (e.g., medicine, web data, pattern recognition), the ratio of rows to columns, and types of features (e.g., discrete, continuous, nominal, and ordinal). These datasets are sourced from widely-employed ML repositories: OpenML, Kaggle, and UC Irvine[1].

ML Models. We use LightGBM for our analysis based on its high-dimensional data capacity and resistance to overfitting [37]. This is deployed using AutoGluon [38], an AutoML framework designed for tabular data, which automatically handles data encoding and hyper-parameter tuning.

[1] www.openml.org, www.kaggle.com, https://archive.ics.uci.edu/

Metrics. We measure *accuracy* for effectiveness, and the *feature selection time* and *model training time* for efficiency.

Methodology. Missing values are handled by imputation with the most common value corresponding to the feature. We split the dataset into an 80%-20% train-test set. We use the training set to retain $top-\kappa$ best-performing features from the total set of features [39], where the choice of values for κ varies based on the dimensionality of the dataset. After feature selection, we train the ML model and evaluate it on the test set.

C. Relevance Metrics

In feature selection, relevance is measured using heuristics. One of the most popular heuristics is correlation, which is based on the hypothesis that good features correlate with the label. We analyse two information-theoretic based methods: information gain, symmetrical uncertainty [33], two widely used correlation coefficients: Pearson, Spearman [40], and Relief, primarily used to remove irrelevant features [41].

Information Gain (IG). Information gain helps select features highly correlated with the label. The method assumes that if a feature has a strong correlation with the label, that feature will positively impact the performance of an ML model [33]. Information gain is symmetric (e.g., $I(X;Y) = I(Y;X)$), and it equals zero if two variables X and Y are independent [18].

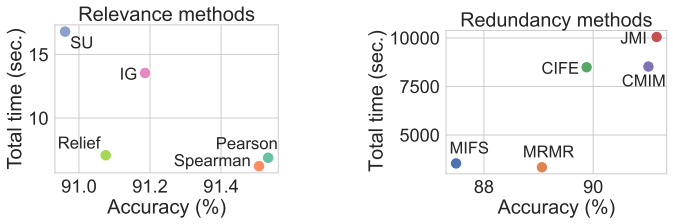
Symmetrical Uncertainty (SU). Symmetrical uncertainty is a correlation metric which measures the linear or non-linear association between two features [42], [43]. SU is based on information gain, and it is able to compensate for the bias towards features with multiple values, one of the shortcomings of IG. SU returns a normalised score in the $[0, 1]$ interval. A score close to zero indicates that the features are independent and, therefore, not relevant for the classification, while a score close to one indicates dependency and, thus, relevancy.

Pearson. Pearson correlation is widely used to assess if two variables are linearly related, independent of any non-linearity that exists in the distribution of the variables. To compute the Pearson correlation between two variables X_i and Y, we must know the covariance and variance of the variables [39].

Spearman. Spearman correlation is a non-parametric measure of rank correlation, which is the statistical dependence between the rankings of two variables. While Pearson's correlation assesses linear relationships, Spearman's correlation assesses monotonic relationships. The distinction between Pearson and Spearman is that Spearman involves transforming the sample values to ranks in the range $[1, N]$. If no repeated data values exist, a perfect correlation of $+1$ or -1 occurs. The higher the value, the higher the correlation [40].

Relief. Relief feature scoring is based on identifying feature value differences between nearest neighbour instance pairs. In other words, the method focuses on separating the data instances from different classes [44].

Choosing a Relevance Metric. Figure 3a shows that Pearson and Spearman are approximately 3x faster than the SU and IG from information theory due to the calculation simplicity



(a) Relevance methods. (b) Redundancy methods.

Figure 3: Comparison of a) relevance methods, and b) redundancy methods based on aggregated accuracy and runtime.

of the first two methods. They also outperform SU and IG in accuracy by around 0.5. While Relief shows comparable efficiency to Pearson and Spearman, its effectiveness is notably lower. Spearman consistently performs best for all datasets, considering both effectiveness and efficiency. Consequently, Spearman is our recommended choice for handling high-dimensional datasets due to its efficiency and effectiveness across varying dataset characteristics.

D. Redundancy Metrics

In feature selection, redundancy is used to assess how much adding a new feature can benefit the performance of a model. Redundant features may have a high degree of overlap in the information they convey: adding a feature X_i, similar to feature X_j, is redundant, as it does not add any new information. We will assess information-theoretical methods based on Shannon's information terms [45].

Definition V.1. (*Shannon's information terms*) Given S a set of currently selected features, Y the feature with the class labels and $X_j \in S$ a feature from the current set S, the unified conditional likelihood maximisation feature selection framework is defined as a linear combination of Shannon's information terms [18]:

$$J(X_k) = I(X_k;Y) - \beta \sum_{X_j \in S} I(X_j;X_k) + \lambda \sum_{X_j \in S} I(X_j;X_k|Y), \quad (1)$$

where $I(X_k;Y)$ represents the information gain, while $I(X_j;X_k|Y)$ represents the conditional information gain.

We will use five methods to measure redundancy, all based on Equation (1).

Mutual Information Feature Selection (MIFS). MIFS helps select the features which are highly correlated with the label and un-correlated with each other [46]. As the name implies, the method uses mutual information to compute the scores and penalises the features correlated with already selected features. The method considers both feature relevancy and redundancy, and it sets the $\lambda = 0$ in Equation (1) [18]. In our analysis, we relied on $\beta = 0.5$.

Minimum Redundancy Maximum Relevance (MRMR). MRMR assumes that with more selected features, the effect of redundancy is gradually reduced. The method sets $\lambda = 0$ and $\beta = \frac{1}{|S|}$ in Equation (1), where $|S|$ is the size of the set of selected features [18].

Conditional Infomax Feature Extraction (CIFE). CIFE is a method that computes the feature-label and feature-feature correlations and the redundancy between the un-selected and already selected features. The method sets $\beta = 1$ and $\lambda = 1$ in Equation (1) [18].

Joint Mutual Information (JMI). JMI aims to increase the complementarity between un-selected and selected features and sets $\lambda = \frac{1}{|S|}$ and $\beta = \frac{1}{|S|}$ in Equation (1), where $|S|$ is the size of the set of selected features [18].

Conditional Mutual Information Maximization (CMIM). CMIM aims to select features with a strong predictive power while reducing the redundancy between the new and selected features. The method is a special case of Equation (1) as follows [18]:

$$J(X_k) = I(X_k; Y) - \max_{X_j \in S}[I(X_j; X_k) - I(X_j; X_k|Y)] \quad (2)$$

Choosing a Redundancy Metric. In Figure 3b, we compare the five information theory methods for feature selection, and we find that MIFS and MRMR are nearly 3x faster than CIFE, JMI, and CMIM. This is due to their computation-saving characteristic of not needing to estimate conditional information gain. However, considering accuracy, CIFE, JMI, and CMIM outperform as they effectively capture inter-column relationships and select less redundant features. Specifically, JMI achieves the highest accuracy but at the cost of the longest runtime, whereas MIFS offers 4x faster runtime but is 6% less accurate. Considering both runtime and accuracy, MRMR emerges as a balanced choice, offering considerably faster runtime while still achieving high model accuracy.

Based on these findings, we devise our algorithm for transitive feature discovery. Spearman rank correlation is employed to measure the relevance, while Maximum Relevance Minimum Redundancy (MRMR) is utilised to quantify redundancy.

VI. AutoFeat's: Ranking-based Feature Discovery

We introduce our ranking algorithm, built upon the foundations of a streaming feature selection pipeline [33], and adapted to suit our specific needs.

Streaming Feature Selection Pipeline. We follow the typical streaming feature selection pipeline, starting with a feature subset's relevance analysis. Subsequently, the resulting subset of relevant features undergoes the redundancy analysis. The final subset contains relevant and non-redundant features, while all others are discarded [18], [33]. As such, we populate the set of selected features with those from the base table T_0. Every join candidate T_i that was not pruned proceeds through the relevance analysis, followed by the redundancy analysis of the subset of relevant features. Algorithm 1 presents our strategy for transitive feature discovery over join paths. The process initiates with a queue containing the base table T_0 and its feature set, which forms the selected feature set R_{sel}.

DRG Traversal. We start the DRG traversal (Lines 6-9) using the strategy described in Section IV-A. Recall that we use dataset discovery algorithms in a data lake context to discover the relationships between datasets, leading to multiple join possibilities between two datasets. We treat each of these join possibilities as a valid join in our approach, allowing our pruning strategy and the feature selection algorithm to determine the significance of the join result (Step 1 and Step 2 in Figure 2). Therefore, in a data lake setting, the $join\_keys$ set results in multiple join possibilities (Line 10). We maintain an additional queue P to preserve all the paths containing relevant information, given a specific join column jk. When the dataset exclusively contains KFK connections, the $join\_keys$ set will always yield a single join column.

Join & Prune. Once the join keys are known, we compute the join (Line 14), as detailed in Section IV-B. If the join is unfeasible due to mismatched join columns (i.e., in a data lake scenario), the join path is pruned according to our first pruning strategy described in Section IV-C. The subsequent

Algorithm 1: Feature Discovery

Input : queue Q containing the base table T_0, R_{sel} selected features from T_0, τ null value ratio threshold, κ maximum number of features to select from a table
Output: dictionary of paths with scores $rank$

```
1  Function autofeat(Q, P=None) is
       // Previous queue initialisation
2      P = Q
3      if Q is empty then
4          return ranking
5      end
6      while Q do
7          T_0 = pop(Q)
8          neighbours = get_neighbours(T_0)
9          foreach node T_n ∈ neighbours do
10             join_keys = get_join_keys(T_0, T_n)
11             C = empty queue
12             while P do
13                 foreach join key jk ∈ join_keys do
14                     R = join(T_0, T_n, jk)
                       // If the join is not possible,
                          prune
15                     D = get_data_quality(R)
                       // If the data quality <
                          threshold τ, prune
16                     R_rel, score_rel =
                        measure_relevance(R, κ)
                       // If R_rel is empty, all
                          features are irrelevant,
                          continue
17                     R_red, score_red =
                        measure_redundancy(R[R_rel], R_sel)
                       // If R_red is empty, all
                          features are redundant,
                          continue
18                     R_sel = R_sel ∪ R_red
19                     score =
                        compute_score(score_rel, score_red)
                       // Save ranking
20                     ranking = (score, R)
                       // Update current queue, C,
                          with the join, R
21                     update(C, R)
22                 end
23                 update(P, C)
24             end
25         end
26     end
27     return autofeat(neighbours, P)
28 end
```

Algorithm 2: Ranking score

Input : $score_{rel}$ the scores from the relevance analysis, $score_{red}$ the scores from the redundancy analysis
Output: $ranking\_score$ one score number

1 **Function** *compute_score*($score_{rel}$, $score_{red}$) **is**
2 $N_{rel} = length(score_{rel})$
3 $N_{red} = length(score_{red})$
4 $sum_{rel} = sum(score_{rel})/N_{rel}$
5 $sum_{red} = sum(score_{red})/N_{red}$
6 $ranking\_score =$ $(N_{red} * sum_{rel} + N_{rel} * sum_{red})/(N_{rel} * N_{red})$
7 **end**

step involves pruning based on data quality (Line 15). More precisely, if the proportion of null values in the join column falls below the predefined hyper-parameter threshold, τ, the join path is pruned.

Feature Selection. All the remaining join paths undergo the feature selection process. We introduce a heuristic approach, named *select κ best*, where we sort the features based on the correlation score, then select the top-κ performers [39]. Consequently, we carry out the relevance analysis (Line 16) using the high-performing method from Section V-C and retain only the top-κ features. If the resulting subset of features is empty, all features are irrelevant. However, we do not prune the join path, as it is an intermediary step towards multi-hop join paths. The final step of the feature selection pipeline is the redundancy analysis (Line 17). Based on the finding from Section V-D, we apply MRMR to the subset of features resulting from the relevance analysis. If the result is an empty set, all the features are redundant. Otherwise, we update the list of selected features R_{sel} with the new subset (Line 18).

Ranking. Upon completion of the feature selection process, we use the scores from the relevance and redundancy analysis to compute a rank for the corresponding join path (Line 19), as explained in Algorithm 2. We compute the sum of the relevance analysis scores, weighted by the cardinality of the selected subset (Line 4). We repeat the process for the redundancy analysis (Line 5). The final ranking score is the sum of sum_{rel} and sum_{red}, weighted by their common divisor. Finally, we store the join path and its corresponding ranking score in an ordered list and update the queues. `AutoFeat` is a recursive algorithm. Once all neighbours are visited, the algorithm starts with the neighbours as base nodes until the entire graph is explored (i.e., all nodes are visited).

From Ranked Paths to Training ML Models. We use the top-k join paths from the list of ranked join paths (Line 20) to systematically train ML models and evaluate the paths. To improve the efficiency of our approach, we use stratified sampling to sample the base table at the beginning of the process. This sampling strategy, however, only impacts the feature selection process and does not limit the scope of data considered during model training. After training, we select the best join path based on the resulting ML model accuracy.

Table II: Overview of datasets used in evaluation.

Dataset	# Rows	# Joinable tables ↓	Total # features	Best accuracy (OpenML.org)
credit	1001	5	21	0.99
eyemove	7609	6	24	0.894
covertype	423682	12	21	0.99
jannis	57581	12	55	0.875
miniboone	73000	15	51	0.9465
steel	1943	15	34	1.0
school	1775	16	731	0.831
bioresponse	3435	40	420	0.885

VII. Evaluation

In this section, we provide quantifiable evidence showing that `AutoFeat` outperforms existing works by being 5x-44x faster on average. Moreover, `AutoFeat` exhibits an average of 16% increase in the predictive power of the augmented features compared to the competition. With our experiments, we aim to show the following:

- *Effectiveness* - `AutoFeat` is able to augment a base table with relevant information, which used to train an ML algorithm, leads to an increase in accuracy.
- *Efficiency* - Relative to SOTA approaches, `AutoFeat` exhibits superior efficiency. This is primarily due to its non-reliance on ML models for augmentation.

A. Experimental Setup

Datasets. We evaluate `AutoFeat` on real-world, open datasets collected from OpenML.org, and summarised in Table II. The *school* dataset originates from the evaluation of ARDA [2]. The datasets are used for binary classification, and we present their best accuracy based on the corresponding tasks found in OpenML.org. For *school* dataset, we report the highest accuracy from the source paper ARDA. We use a collection of eight datasets varying in size and number of joinable tables, which we split into multiple tables using two different settings.

Benchmark Setting. We design a technique to divide a dataset into multiple small tables with known KFK constraints. We extend the setting from ARDA, where they use star schemata with known KFK connections. We call this approach the *benchmark setting*. Here, the DRG contains only KFK relations, which resembles a snowflake schemata. With this setting, we aim to reproduce the results of the baselines [2], [3], and to show that `AutoFeat` can explore longer join paths in search for relevant features.

Data Lake Setting. With the *data lake setting*, we aim to simulate a data lake scenario where the connections between datasets are unknown. Using the same tables from the *benchmark setting*, we remove the edges representing KFK relationships, and we apply dataset discovery algorithms to find the relations. We use COMA [25] as implemented in Valentine [24], with a default schema matching strategy and a similarity threshold of 0.55 to encourage spurious, but not irrelevant, connections. The DRG is now a dense multi-graph. This scenario aims to showcase the predictive power of our

approach in a data lake setting and the fact that we can discriminate between join columns.

Machine Learning Models. To validate that our augmentation approach can add relevant information, we use machine learning tasks. We use four decision tree algorithms (e.g., LightGBM, Extreme Randomised Trees, Random Forest and XGBoost), all available in AutoGluon [38], the AutoML framework for tabular data, which automatically handles data encoding and hyper-parameter tuning. We choose to use decision trees, as they are one of the most popular ML models used in practice; they are explainable and can outperform neural networks [47], [48] in tabular training data.

Metrics. We evaluate the performance of our approach by measuring: (i) *effectiveness* using accuracy, and (ii) *efficiency* by measuring the *feature selection time* - the total time it takes to assess the fitness of features for augmentation.

B. Baselines

`AutoFeat`. `AutoFeat` respresents our feature discovery approach. We use $\tau = 0.65$ as the null value ratio and $\kappa = 15$ as the maximum selected features from a table.

BASE. The BASE approach represents the base table, which is assumed to be performing poorly on any ML model. This represents the table we want to augment with more predictive and relevant features.

ARDA. ARDA is one of the SOTA approaches and is the feature augmentation system which uses a random-injection approach to select the features for augmentation [2]. Since the source code was unavailable, we implemented the feature selection part of the system following the algorithms and details provided in the original paper [2].

MAB. The second SOTA technique we consider is the Multi-Armed Bandit (MAB) method [3]. The study presents two distinct methods for feature augmentation: one leveraging the multi-armed bandit reinforcement learning approach, and another utilising deep Q networks, which is a neural network approach. Our results are consistent with the original study that reported comparable effectiveness (i.e., 0.02 - 0.04 difference in AUC), but major differences in efficiency between the two methods: our experiments showed that the deep Q network method required a prohibitive amount of computational time (i.e., extending to days for a single experiment) which exceeded the limits of our available resources. In contrast, the MAB approach enabled us to conduct the experiments within a reasonable time frame (i.e., hours).

JoinAll. The *JoinAll* baseline is the approach where all the tables which connect with the base table are joined in all possible ways. Given a perfect scenario where: (i) the join cardinality is 1:1, and (ii) the joins are on KFK, *JoinAll* results in a *single* path. However, when the connections are not KFK, regardless of the type of join, the number of possible paths increases tremendously, as the order in which the tables are joined affects the resulting augmented table. We denote D, the maximum depth of the join tree, $N(d)$ the number of nodes at depth $d \in D$, and $k(v)$ the number of un-visited neighbours of node v. The total number of possible *JoinAll* paths P is the product of all choices at all levels:

$$P = \prod_{d=0}^{D} \prod_{v \in N(d)} k(v)!, \tag{3}$$

JoinAll+F. The *JoinAll+F* baseline is the *JoinAll* approach, where we apply filter feature selection on the resulting dataset before training the ML model.

C. Results

The experimental results are split based on the schemata configuration: the *benchmark* setting and the *data lake* setting.

1) Benchmark setting: Recall that the benchmark setting represents the DRG with known KFK relationships without spurious connections. No results are shown for the baselines *JoinAll* and *JoinAll+F* in Figure 4 on the *school* dataset, due to the increased computation time. For *school* dataset, which follows a star schema, and the join cardinality is not 1:1, the number of possible join paths for *JoinAll* and *JoinAll+F* is $P = 15!$ (Equation (3)). As a result, those baselines did not finish within the given time constraint.

Effectiveness. `AutoFeat` shows an average accuracy increase of 16% across all datasets and algorithms. The effectiveness of `AutoFeat` aligns with our expectations due to its ability to navigate through longer join paths, leveraging transitive joins to identify more relevant features. This is in contrast to ARDA, which is limited to star schemata. Given its capability to handle transitive joins, we expected that MAB would be more effective than the results show. Compared to the *JoinAll* and *JoinAll+F* baselines, `AutoFeat` results in similar accuracy, which shows that it finds the right features to improve the performance of a given base table.

Efficiency. The efficiency of `AutoFeat` is noteworthy. On average, `AutoFeat` operates 4.5x faster than ARDA and 12x faster than MAB. Referencing the *credit* dataset from Figure 4, `AutoFeat` is 4x faster than ARDA and 7x faster than MAB, even when the increase in accuracy is marginal. Using heuristics to identify relevant features contributes to the superior efficiency of `AutoFeat`, a stark contrast to ARDA and MAB, which incorporate the ML model into their feature discovery process. Given that the *JoinAll* baseline does not perform feature selection, we can only compare the total runtime, where `AutoFeat` is on average 3x faster than *JoinAll*. In contrast, the feature selection time of the *JoinAll+F* baseline costs less than one second, since it performs feature selection once for a single wide table. However, on average, its total runtime is, 3x slower than `AutoFeat`.

Path Analysis. Figure 4 illustrates the number of datasets each approach uses to find the most relevant features. ARDA is limited to star schemata and joins all the datasets directly connected to the base table. Thus, the number of datasets used to find the most related features is at most the maximum number of directly connected neighbours. Although MAB can handle transitive joins, it does not explore the schemata in depth, a limitation which results from the fact that

Figure 4: Benchmark setting - **Top**: Average runtime in pastel colours. The contrasting colours show the portion of feature selection time out of the total runtime. **Bottom**: Accuracy per dataset, averaged over all tested tree-based ML algorithms. Numbers on bars represent # of joined tables. Horizontal lines represent the best accuracy.

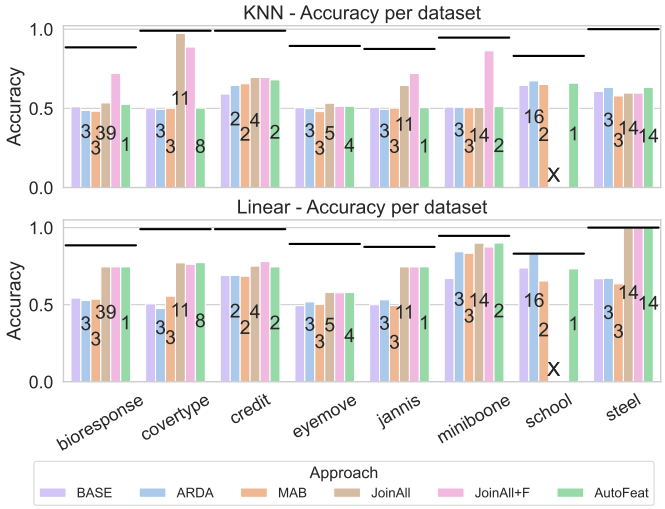


Figure 5: Benchmark setting: accuracy per dataset, for KNN and Linear Regression. Numbers on bars represent # of joined tables. Horizontal lines represent the best accuracy.

MAB requires the same join column name (i.e., PK-FK with the same names). AutoFeat requires fewer datasets while achieving higher accuracy, except for *covertype*, *eyemove*, and *steel*. Here, AutoFeat joins more tables than ARDA and MAB because AutoFeat explores the schemata in depth. We observed that the most relevant features reside via transitive joins; thus, most tables joined by AutoFeat are only required to navigate the join path search space.

Evaluation on Non Tree-Based Models. Figure 5 shows the effectiveness of AutoFeat on non-tree based ML models: K-Nearest Neighbours (KNN) and Linear Regression with L1 regularisation (LR). For LR, AutoFeat achieves top performance together with *JoinAll* and *JoinAll+F* baselines across most datasets, with ARDA surpassing on the *school* dataset. Dataset characteristics influence the effectiveness of KNN: it underperforms on smaller datasets due to insufficient neighbours for reliable decision-making (e.g., *credit*, *eyemove*, *steel*, *school*). Conversely, on larger datasets with fewer features (e.g., *covertype*, *jannis*, *miniboone*), or on high-dimensional datasets (e.g., *bioresponse*), KNN converges effectively with the *JoinAll* and *JoinAll+F* baselines.

2) Data lake setting: Recall that we apply an automated dataset discovery method in a data lake setting to find the relationships between tables. The once snowflake-like schemata transform into a densely connected graph in this context. Figure 6 and Figure 7 do not show results for the baselines *JoinAll* and *JoinAll+F* as the number of possible *JoinAll* paths in this scenario is extremely large (Equation (3)).

Effectiveness. As depicted in Figure 6, AutoFeat consistently demonstrates equal or superior effectiveness to other approaches. On average, AutoFeat outperforms ARDA by 12% and MAB by 2%. In a data lake setting with numerous spurious connections, AutoFeat adeptly prunes tables that introduce noise, indicating the critical role of join ordering for data augmentation.

Efficiency. The AutoFeat approach maintains impressive efficiency, even within a data lake setting. On average, AutoFeat operates 10x faster than MAB and 3x faster than ARDA while delivering comparable or superior accuracy (Figure 6). In this context, we explore various join column pairs to identify the path enriched with the most relevant features, subsequently increasing the time invested in feature discovery. However, using heuristics, instead of relying on machine learning models to predict feature importance, effectively counters the overall time spent, sustaining our approach's efficiency.

Path Analysis. Figure 6 illustrates expected behaviours. With the snowflake schemata disrupted, where dataset discovery algorithms yield many spurious connections, ARDA indiscriminately joins all the datasets neighbouring the base table. MAB suffers from the same limitation as in the *benchmark setting*, where it restricts its joins to tables sharing the same join column name, thereby inhibiting the exploration of transitive connections. AutoFeat can prune out irrelevant tables and explores the join path search space in depth (e.g., *jannis* and *steel* datasets). Once again, we noticed that the high number of joined datasets results from the fact that the most relevant features reside in a multi-hop join path.

Evaluation on Non Tree-Based Models. Figure 7 shows the performance of AutoFeat on non tree based ML models. In the data lake scenario, the baselines may introduce irrelevant features; thus KNN does not achieve optimal results. KNN suffers challenges due to possible noise when irrelevant tables are joined, leading to worse distance measurements. Conversely, LR shows AutoFeat outperforming baselines in the majority of cases (i.e., five out of eight datasets), with ARDA maintaining a consistent lead on the *school* dataset.

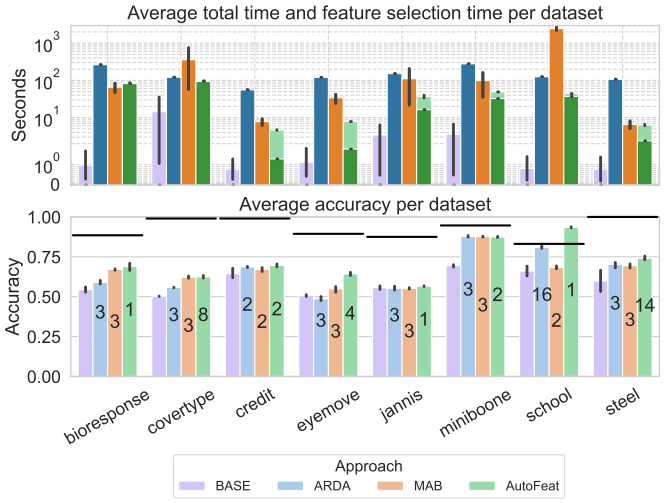


Figure 6: Data lake setting - **Top**: Average runtime in pastel colours. The contrasting colours show the portion of feature selection time out of the total runtime. **Bottom**: Accuracy per dataset, averaged over all tested tree-based ML algorithms. Numbers on bars represent # of joined tables. Horizontal lines represent the best accuracy.

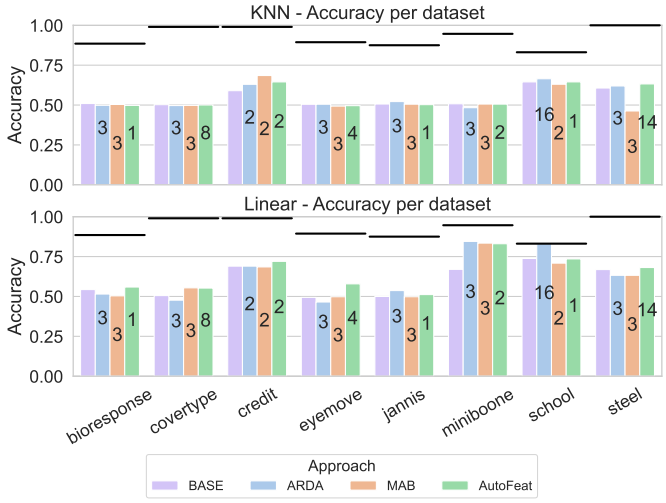


Figure 7: Data lake setting: accuracy per dataset, for KNN and Linear Regression. Numbers on bars represent # of joined tables. Horizontal lines represent the best accuracy.

3) Summary: On a snowflake schema, `AutoFeat` can explore the join path space in depth, searching for relevant features. On a densely connected multi-graph, `AutoFeat` can explore different join column possibilities and follow the path with the highest data quality in search of the most relevant features for augmentation. On both configurations, `AutoFeat` is more efficient for feature discovery than the baselines.

`AutoFeat` is an approach best suited to tree-based ML models that can mitigate the consequences of high dimensionality. Kernel-based ML models and linear models suffer more from the "curse of dimensionality", where data points become sparse and the distance between points loses meaning, which increases the difficulty of finding meaningful patterns in the data, resulting in degraded performance.

The results show that `AutoFeat` is a more efficient and effective method for feature discovery over long join paths on tree-based ML models.

D. Parameter Sensitivity Analysis

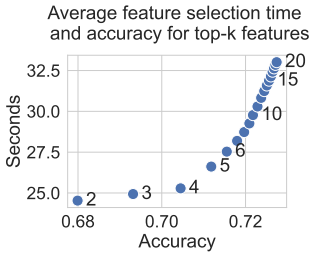
`AutoFeat` relies on two hyper-parameters: the null value ratio (τ) and the maximum number of features to select (κ). This set of experiments aims to demonstrate the sensitivity of our method to variations in τ and κ and their influence on both effectiveness and efficiency.

Sensitivity to κ. Figure 8a illustrates the change in accuracy and feature selection time with different values of κ, denoting the maximum number of features to be selected. As κ varies from 2 to 6, there is a notable 4% increase in accuracy at the cost of an additional 2.5 seconds in feature selection time. However, when κ ranges from 6 to 10, the increase in accuracy is less than 1%, despite the same increment of 2.5 seconds in time. A similar trend is observed when κ varies within the interval [15, 20]. We recommend selecting a value for κ within the range [10, 15], as this provides a high accuracy level with a tolerable trade-off of a few seconds. Any $\kappa > 15$ does not significantly improve accuracy nor efficiency but could potentially risk overfitting the ML model.

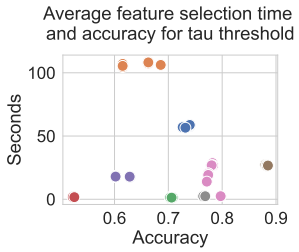
Sensitivity to τ. Our pruning strategy is contingent on data quality, such that a table is pruned if the ratio of null values to the total number of values exceeds a threshold τ. Figure 8b displays the average accuracy and feature selection time across datasets for different values of the hyper-parameter τ within the range of [0.05, 1.0], incremented by steps of 0.05. Most datasets appear to be relatively insensitive to the threshold, barring two exceptions: *covertype* (shown in orange) and *school* (illustrated in pink). In Figure 8c and Figure 8d, we provide a closer look at these two datasets.

When $\tau = 1$, it leaves no room for null values, implying that the tables perfectly match on the join keys, thus leading to peak accuracy. However, the *school* dataset depicted in Figure 8d yields no output when $\tau = 1$, indicating no tables with perfect matches on join keys. Hence, although $\tau = 1$ can result in optimal accuracy in certain instances, it is overly restrictive and fails to generalise across other tables.

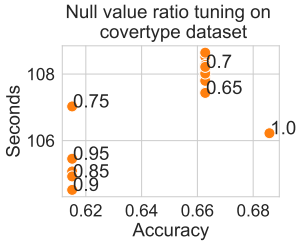
The data indicates that for τ within the range [0.05, 0.6], the results tend to cluster around similar accuracy values or feature selection times. This suggests that not many tables are pruned during this interval, and imputation methods have little effect on accuracy. However, for $\tau > 0.6$, we observe a decrease in accuracy (up to 4% less) and time. Reducing feature selection time implies that more tables are pruned at this stage. The decrease in accuracy signifies potential bias introduced by the imputation strategies. We recommend $\tau = 0.65$ as a balance between accuracy and feature selection time.



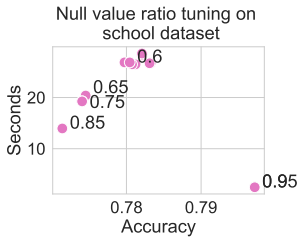
(a) Results averaged over datasets and ML models.



(b) Results averaged over datasets and ML models.



(c) *covertype* - accuracy and feature selection time averaged over ML models.



(d) *school* - accuracy and feature selection time averaged over ML models.

Figure 8: Parameter Sensitivity Analysis.

The hyper-parameter κ directly influences our model's effectiveness, with higher κ leading to improved results. Meanwhile, the hyper-parameter τ greatly impacts the model's efficiency, highlighting the role of our pruning strategy.

E. Ablation Study

AutoFeat uses two core methods: Spearman to measure relevance, and MRMR redundancy. To show that AutoFeat is indeed more efficient and more effective thanks to these methods, we perform an ablation study, where we use different configurations of AutoFeat. One by one, we replace either MRMR with JMI, or Speaman with Pearson, or both. Finally, we turn off the computation of either relevance or redundancy.

The results of the ablation study are presented in Figure 9. The variants of AutoFeat which use JMI are at least two times slower than AutoFeat, consistent with Figure 3. However, both Pearson-JMI and Spearman-JMI are less accurate than AutoFeat on five out of eight datasets and marginally more accurate on *school* and *jannis*.

Compared with AutoFeat, the Spearman-only version results in either degradation in time or accuracy and achieves similar results only on *school* and *steel* datasets.

Pearson-MRMR and MRMR versions exhibit similar behaviour. They are either less accurate (i.e., they fail to find all the relevant features) or slower (i.e., they retain too many features). However, on three of the datasets, they result in similar performance with AutoFeat. One reason for this behaviour is the fairly similar schemata of these datasets, which contain the same amount of joinable tables. However, on *school* dataset, MRMR is the slowest but also the most accurate. Here, the schemata are star-shaped, which results in fewer opportunities for deep pruning. MRMR fails to discard features, exhibiting a behaviour similar to *JoinAll* baseline.

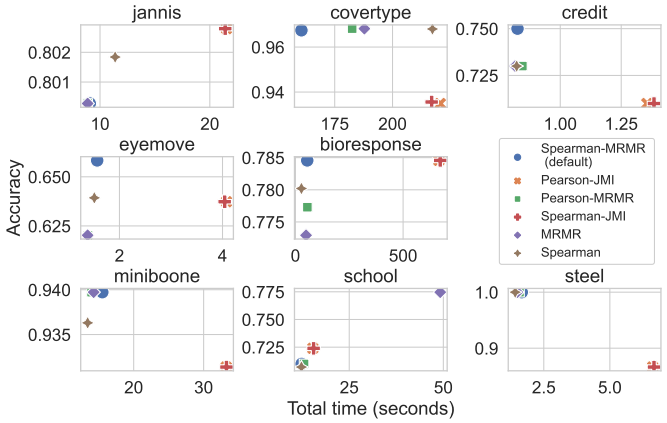


Figure 9: Accuracy and total time on the ablation study with different configurations of AutoFeat.

The proposed version of AutoFeat (i.e., Spearman-MRMR) is the most efficient version with minimal accuracy loss.

VIII. Conclusion

We have introduced AutoFeat, a novel approach for transitive feature discovery. AutoFeat addresses a critical gap in current data science practices by navigating the complex join space, effectively augmenting a base table with relevant features across various datasets. It exploits a relevance-agnostic algorithm for scoring features, not relying on specific ML models, making the approach broadly applicable and versatile. The results demonstrated that AutoFeat achieves high effectiveness using a fraction of the computation time compared to SOTA approaches.

Future Work. Several directions remain open for future exploration. We aim to integrate more complex feature selection strategies that could improve the feature's relevance and overall performance. We plan to explore dynamic hyper-parameter tuning, allowing the algorithm to adapt to different data landscapes and tasks. As datasets grow in complexity and volume, more efficient methods of navigating join paths and reducing the complexity of the problem space will be investigated. We are particularly interested in how graph-based models and representations could be used to achieve this. Other improvement possibilities might arise after evaluating our approach on real data lake scenarios within organisations, where we envision that more aggressive pruning strategies might be required.

Acknowledgments

We want to extend our gratitude to Alexandru Balan for his invaluable support and feedback, and for his outstanding engineering skills, which have played a pivotal role in the success of this project. This work was partially funded by the the European Union's H2020 project OpertusMundi (870228), and by European Union Horizon Programme ExtremeXP (101093164). This publication is part of the project Understanding Implicit Dataset Relationships for Machine Learning (VI.Veni.222.439) of the research programme NWO Veni which is partly financed by NWO.

REFERENCES

[1] G. Fan, J. Wang, Y. Li, and R. J. Miller, "Table discovery in data lakes: State-of-the-art and future directions," in *Companion of the 2023 International Conference on Management of Data*, 2023, pp. 69–75.

[2] N. Chepurko, R. Marcus, E. Zgraggen, R. C. Fernandez, T. Kraska, and D. Karger, "Arda: automatic relational data augmentation for machine learning," *PVLDB*, pp. 1373–1387, 2020.

[3] J. Liu, C. Chai, Y. Luo, Y. Lou, J. Feng, and N. Tang, "Feature augmentation with reinforcement learning."

[4] R. Hai, C. Koutras, C. Quix, and M. Jarke, "Data lakes: A survey of functions and systems," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[5] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller, "Table union search on open data." Association for Computing Machinery, 2018, pp. 813–825.

[6] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker, "Aurum: A data discovery system," in *ICDE*, 2018, pp. 1001–1012.

[7] R. Castro Fernandez, J. Min, D. Nava, and S. Madden, "Lazo: A cardinality-based method for coupled estimation of jaccard similarity and containment," *ICDE*, vol. 2019-April, pp. 1190–1201, 2019.

[8] E. Zhu, D. Deng, F. Nargesian, and R. J. Miller, "Josie: Overlap set similarity search for finding joinable tables in data lakes," in *SIGMOD*, 2019, pp. 847–864.

[9] Y. Dong, K. Takeoka, C. Xiao, and M. Oyamada, "Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach," in *ICDE*. IEEE, 2021, pp. 456–467.

[10] A. Bogatu, A. A. Fernandes, N. W. Paton, and N. Konstantinou, "Dataset discovery in data lakes," in *ICDE*. IEEE, 2020, pp. 709–720.

[11] Y. Zhang and Z. Ives, "Finding related tables in data lakes for interactive data science," in *SIGMOD*, 2020, pp. 1951–1966.

[12] A. Kumar, J. Naughton, J. M. Patel, and X. Zhu, "To join or not to join? thinking twice about joins before feature selection," in *SIGMOD*, 2016, pp. 19–34.

[13] S. Castelo, R. Rampin, A. S. Santos, A. Bessa, F. Chirigati, and J. Freire, "Auctus: a dataset search engine for data discovery and augmentation," 2021.

[14] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri, "Infogather: entity augmentation and attribute discovery by holistic matching with web tables," in *SIGMOD*, 2012, pp. 97–108.

[15] R. Hai, C. Koutras, A. Ionescu, Z. Li, W. Sun, J. Van Schijndel, Y. Kang, and A. Katsifodimos, "Amalur: Data integration meets machine learning," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023, pp. 3729–3739.

[16] S. Galhotra, Y. Gong, and R. C. Fernandez, "Metam: Goal-oriented data discovery," *arXiv preprint arXiv:2304.09068*, 2023.

[17] Y. Zhang and Z. G. Ives, "Juneau: data lake management for jupyter," *Proc. VLDB Endow.*, vol. 12, no. 12, 2019.

[18] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM computing surveys (CSUR)*, vol. 50, no. 6, pp. 1–45, 2017.

[19] S. H. Huang, "Supervised feature selection: A tutorial." *Artif. Intell. Res.*, vol. 4, no. 2, pp. 22–37, 2015.

[20] L. C. Molina, L. Belanche, and À. Nebot, "Feature selection algorithms: A survey and experimental evaluation," in *ICDM*. IEEE, 2002, pp. 306–313.

[21] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[22] L. Yu and H. Liu, "Efficient feature selection via analysis of relevance and redundancy," *The Journal of Machine Learning Research*, vol. 5, pp. 1205–1224, 2004.

[23] S. Bharadwaj, P. Gupta, R. Bhagwan, and S. Guha, "Discovering Related Data at Scale," *PVLDB*, vol. 14, no. 8, pp. 1392–1400, 2021.

[24] C. Koutras, G. Siachamis, A. Ionescu, K. Psarakis, J. Brons, M. Fragkoulis, C. Lofi, A. Bonifati, and A. Katsifodimos, "Valentine: Evaluating matching techniques for dataset discovery," in *ICDE*. IEEE, 2021, pp. 468–479.

[25] H.-H. Do and E. Rahm, "Coma—a system for flexible combination of schema matching approaches," in *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 2002, pp. 610–621.

[26] A. Bundy and L. Wallen, "Breadth-first search," *Catalogue of artificial intelligence tools*, pp. 13–13, 1984.

[27] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[28] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, "Learning from class-imbalanced data: Review of methods and applications," *Expert systems with applications*, vol. 73, pp. 220–239, 2017.

[29] V. Gudivada, A. Apon, and J. Ding, "Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations," *International Journal on Advances in Software*, vol. 10, no. 1, pp. 1–20, 2017.

[30] C. Cortes, L. D. Jackel, and W.-P. Chiang, "Limits on learning machine accuracy imposed by data quality," *Advances in Neural Information Processing Systems*, vol. 7, 1994.

[31] P. Li, X. Rao, J. Blase, Y. Zhang, X. Chu, and C. Zhang, "Cleanml: A study for evaluating the impact of data cleaning on ml classification tasks," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 13–24.

[32] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Biessmann, and A. Grafberger, "Automating large-scale data quality verification," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 1781–1794, 2018.

[33] N. AlNuaimi, M. M. Masud, M. A. Serhani, and N. Zaki, "Streaming feature selection algorithms for big data: A survey," *Applied Computing and Informatics*, vol. 18, no. 1/2, pp. 113–135, 2020.

[34] X. Hu, P. Zhou, P. Li, J. Wang, and X. Wu, "A survey on online feature selection with streaming features," *Frontiers of Computer Science*, vol. 12, pp. 479–493, 2018.

[35] K. Yu, X. Wu, W. Ding, and J. Pei, "Scalable and accurate online feature selection for big data," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 11, no. 2, pp. 1–39, 2016.

[36] H. Li, X. Wu, Z. Li, and W. Ding, "Group feature selection with streaming features," in *2013 IEEE 13th International Conference on Data Mining*. IEEE, 2013, pp. 1109–1114.

[37] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[38] 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.

[39] 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.

[40] J. de Winter, S. Gosling, and J. Potter, "Comparing the pearson and spearman correlation coefficients across distributions and sample sizes: A tutorial using simulations and empirical data," *Psychological Methods*, vol. 21, pp. 273–290, 2016.

[41] R. Duangsoithong and T. Windeatt, "Relevance and redundancy analysis for ensemble classifiers," in *Machine Learning and Data Mining in Pattern Recognition: 6th International Conference, MLDM 2009, Leipzig, Germany, July 23-25, 2009. Proceedings 6*. Springer, 2009, pp. 206–220.

[42] I. H. Witten, E. Frank, M. A. Hall, C. J. Pal, and M. DATA, "Practical machine learning tools and techniques," in *Data Mining*, vol. 2, no. 4, 2005.

[43] L. Yu and H. Liu, "Feature selection for high-dimensional data: A fast correlation-based filter solution," in *ICML*, 2003, pp. 856–863.

[44] R. J. Urbanowicz, M. Meeker, W. La Cava, R. S. Olson, and J. H. Moore, "Relief-based feature selection: Introduction and review," *Journal of biomedical informatics*, vol. 85, pp. 189–203, 2018.

[45] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[46] 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.

[47] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" in *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[48] Z. Qin, L. Yan, H. Zhuang, Y. Tay, R. K. Pasumarthi, X. Wang, M. Bendersky, and M. Najork, "Are neural rankers still outperformed by gradient boosted decision trees?" 2021.