ARDA: Automatic Relational Data Augmentation for Machine Learning

Nadiia Chepurko[1], Ryan Marcus[1], Emanuel Zgraggen[1],
Raul Castro Fernandez[2], Tim Kraska[1], David Karger[1]
[1]MIT CSAIL [2]University of Chicago
{nadiia, ryanmarcus, emzg, kraska, karger}@csail.mit.edu raulcf@uchicago.edu

ABSTRACT

Automatic machine learning (AML) is a family of techniques to automate the process of training predictive models, aiming to both improve performance and make machine learning more accessible. While many recent works have focused on aspects of the machine learning pipeline like model selection, hyperparameter tuning, and feature selection, relatively few works have focused on automatic data augmentation. Automatic data augmentation involves finding new features relevant to the user's predictive task with minimal "human-in-the-loop" involvement.

We present ARDA, an end-to-end system that takes as input a dataset and a data repository, and outputs an augmented data set such that training a predictive model on this augmented dataset results in improved performance. Our system has two distinct components: (1) a framework to search and join data with the input data, based on various attributes of the input, and (2) an efficient feature selection algorithm that prunes out noisy or irrelevant features from the resulting join. We perform an extensive empirical evaluation of different system components and benchmark our feature selection algorithm on real-world datasets.

PVLDB Reference Format:
Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, David Karger. ARDA: Automatic Relational Data Augmentation. *PVLDB*, 13(9): 1373-1387, 2020.
DOI: https://doi.org/10.14778/3397230.3397235

1. INTRODUCTION

Automatic machine learning (AML) aims to significantly simplify the process of building predictive models. In its simplest form, AML tools automatically try to find the best machine learning algorithm and hyper-parameters for a given prediction task [44, 59, 60, 66]. With AML, the user only has to (1) provide a dataset (usually a table) with features and a predictive target (i.e., columns), and (2) specify their model goal (e.g., build a classifier and maximize the F1 score). More advanced AML tools go even further and not only try to find the best model and tune hyper-parameters, but also perform automatic feature engineering. In their current form, AML tools can outperform experts [36] and can help to make ML more accessible to a broader range of users [28, 41].

However, what if the original dataset provided by the user does not contain enough signal (predictive features) to create an accurate model? For instance, consider a user who wants to use the publicly-available NYC taxi dataset [24] to build a forecasting model for taxi ride durations. Suppose the dataset contains trip information over the last five years, including license plate numbers, pickup locations, destinations, and pickup times. A model built only on this data may not be very accurate because there are other major external factors that impact the duration of a taxi ride. For example, weather obviously has a strong impact on the demand of taxis in NYC, and large events like baseball games can significantly alter traffic.

While it is relatively easy to find related datasets using systems like Google Data Search [11] or standardized repository like Amazon's data collections [54], trying to integrate these datasets as part of the feature engineering process remains a challenge. First, integrating related datasets requires finding the right join key between different datasets. Consider, for example, the hypothetical taxi dataset and a related NYC weather dataset. While the taxi data is provided as a list of events (one row per taxi ride), the weather data might be on the granularity of minutes, hours, or days, and might also contain missing information. To join the taxi dataset with the NYC weather dataset, one needs to define an appropriate join key and potentially pre-aggregate and interpolate the weather dataset, as well as deal with null values. Moreover, there might be hundreds of related tables, creating a large amount of work for the user.

There exists a unique opportunity to explore this type of data augmentation automatically as part of an automatic machine learning pipeline. Automatically finding and joining related tables is challenging [26]. Users often have a hard time distinguishing a semantically meaningful join from an meaningless join if they don't know details about the join itself; the relevance of a join depends on the semantics of the table and there is no easy way to quantify semantics. However, in the context of automated machine learning, joining is significantly simpler as a clear evaluation metric exists: does the prediction performance (e.g., accuracy, F1 score) after the join increase? This sidesteps the question of the semantic connection between two tables (perhaps implicitly measures it) while still increasing the metric the user cares most about, the predictive model's performance.

Proceedings of the VLDB Endowment, Vol. 13, No. 9
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3397230.3397235

In this paper, we explore how we can extend existing automatic machine learning tools to also perform automatic data augmentation by joining related tables. A naive solution to the problem would try to find common join keys between tables, then join all possible tables to create one giant "uber"-table, and then use a feature selection method (which are already part of many AML-tools) to select the right features. Obviously, such an approach has several problems. First, in some cases like the weather and taxi data, a direct join-key might not exist, requiring a more fuzzy type of join. Second, joining everything might create a giant table, potentially with more features than rows. This can be problematic even for the best existing feature selection methods (i.e., it creates an underdetermined system). Even with ideal feature selectors, the size of such an "uber"-table may significantly slow down the AML process.

Thus, we developed ARDA, a first Automatic Relational Data Augmentation system. From a base table provided by a user and a foreign key mapping (provided by the user or discovered with an external tool [26]), ARDA automatically and efficiently discovers joins can help model performance, and avoids joins that add little or no value (e.g., noise). At a high level, ARDA works by first constructing a *coreset*, a representative but small set of rows from the original sample table. This coreset is strategically joined with candidate tables, and features are evaluated using a novel technique we call *random injection feature selection (RIFS)*. At a high level, RIFS helps determine if the results of a join are helpful for a predictive model by comparing candidate features against noise: if a candidate feature performs no better than a feature consisting of random noise, that feature is unlikely to be useful. ARDA handles joins between mismatched keys that may not perfectly align (e.g., time) using several interpolation strategies. The final output of ARDA is an augmented dataset, containing all of the user's original dataset as well as additional features that increase the performance of a predictive model.

While there has been prior work on data mining, knowledge discovery, data augmentation and feature selection, to the best of our knowledge no system exists which automatically explores related tables to improve the model performance. Arguably most related to our system are Kumar et. al. [42] and Shah et al. [58]. Given a set of joins, the authors try to eliminate many-to-one (n-to-1) joins which are not necessary (i.e., highly unlikely to improve a predictive model) because the foreign key already contains all the information from the external table. For example, consider again the taxi dataset, but where the start address is modeled as a FK-PK relationship to a street-table. In that case, depending on the model, the join to the street-table might not be necessary as the foreign-key acts as an embedding for the street. In other words, the foreign key (street address) already contains the street name, which is the only extra piece of information that could be gained from joining with the street-table.

However, the techniques in [42, 58] focus only on classification tasks and err on the side of caution, only eliminating joins that are statistically highly unlikely to improve the final model. In other words, these techniques focus on excluding a narrow subset of tables that will not improve a model: just because a table is not ruled out by the techniques of [42, 58] does not mean that this table will improve model performance. In fact, the techniques of [42, 58] are intentionally conservative. ARDA is designed to automatically find feature sets that actually improve the predictive power of the final model. Furthermore, they can only eliminate many-to-one joins and neither deal with one-to-many joins, nor with fuzzy joins (e.g., the weather and taxi data which must be joined on time). Thus, both [42] and [58] are orthogonal to this work, as we are concerned with effectively augmenting a base table to improve model accuracy as opposed to determining which joins are potentially safe to avoid. In our experimental study, we demonstrate that Kumar et al.'s decision rules can be used as a prefiltering technique for ARDA, slightly affecting model performance (sometimes better, sometimes worse) but always improving runtime.

In summary we make the following contributions:

- We introduce ARDA, a system for automatic relational data augmentation which can discover joins that improve the performance of predictive models,
- we propose simple methods for performing one-to-many and "soft key" (e.g., time) joins useful for ML models,
- we introduce RIFS, a specific feature selection technique custom-tailored to relational data augmentation,
- we experimentally demonstrate that effectiveness of our prototype implementation of ARDA, showing that relational data augmentation can be performed automatically in an end-to-end fashion.

2. PROBLEM OVERVIEW

In this section, we describe the setup and problem statement we consider. Our system is given as input a *base table* with labelled data (that is, a specified column contains the target labels) and the goal is to perform inference on this dataset (i.e., be able to predict the target label for a new row of the table). We assume that the base table provided by the user is one table in a (potentially large) data repository (e.g. [2]). We assume the learning model is also specified and is computationally expensive to train. For example, random forest models with a large number of trees or SVM models over large datasets can be time consuming to train. ARDA is otherwise agnostic to the model training process, and can use any type of model training, including simple models like random forest as well as entire AML systems.

Our goal then is to *augment* our base table—to add features such that we can obtain a non-trivial improvement on the prediction task. To this end, we assume that an external *data discovery system* (e.g. [26]) automatically determines a collection of *candidate joins*: columns in the base table that are potentially foreign keys into another table. Since data discovery systems like [26] generally depend on a "human-in-the-loop" to eliminate false positives, ARDA is designed to handle a generated collection that is potentially very large and highly noisy (i.e., that the majority of the joins are semantically meaningless and will not improve a predictive model).

ARDA considers candidate joins based on two types of candidate foreign keys: *hard keys*, which are foreign keys in the traditional sense of a database system and can be joined with their corresponding table using traditional algorithms, and *soft keys*, foreign keys that may not precisely match the values in the corresponding table. For example, a data

discovery system may indicate that a column representing time in the user's base table is a potential foreign key into another table storing weather data. If the user's base table has timestamps at a one-day level of granularity, but the weather table has timestamps at a one-minute level of granularity, joining the two tables together requires special care. Simply using a traditional join algorithm may result in many missing values (when granularity does not precisely align), or semantically meaningless results (e.g., if one arbitrarily determines which weather entry to match with each row of the user's base table). To support joins of such soft keys, ARDA provides the user with several options, including nearest neighbor, linear interpolation, and resampling techniques. We assume that the "hardness" or "softness" of a join key is indicated by the data discovery system [26], and we discuss the details of soft key joins in Section 4.

Example. Traditionally, a user has had to invest significant effort in deciding what data can usefully augment a classification problem. For example, suppose that given a base table TAXI, the learning task is to predict the target column trips that gives the number of trips a given a taxi made or will make for a given day. Traditionally, a user might speculate that the weather might significantly impact demand. Since weather data is available in the user's data repository, she can writing code to join the TAXI and WEATHER datasets on columns date and time. After determining a strategy to join together these soft keys, she then evaluates whether or not the additional weather data improved model accuracy. Afterwards, she can continue to search for other potential augmentations, such as sporting event schedules or traffic data.

The goal of this project is to automate this arduous process with little to no user intervention from the user. In Figure 2 we show an example schema representing foreign keys discovered by a data discovery system on a (potentially heterogeneous) data repository. In reality, this schema would be much bigger, since it would contain a large quantity of irrelevant datasets. Our task is to discover tables that contain information that can improve a predictive model's performance, without requiring a human being to decide which candidate joins are potentially relevant. The central questions that arise here include how to effectively combine information from multiple relations, how to efficiently determine which other relations provide valuable information and how to process a massive number of tables efficiently.

Once the relevant join is performed, we must contend with the massive number of irrelevant features interspersed with a small number of relevant features. We observed that after many joins, the resulting dataset often results in predictive models with worse performance than training a model with no augmentation. This is because machine learning models are can be misled by noise, and large numbers of features provide more opportunity for such confusion. Therefore, our implementation must efficiently select relevant features in the presence of many irrelevant or noisy features. The rest of this work is devoted to describing our system and how it addresses the aforementioned challenges.

3. AUGMENTATION WORKFLOW

We begin with a high-level description of the workflow that our system follows.

Input to ARDA. ARDA requires a reference to a database and a collection of candidate joins from a data discovery system: a description of the columns in the base table that can be used as foreign keys into other tables. Often, data discovery systems (e.g., [2, 26]) provide a ranking of the candidate joins based on expected relevancy (usually determined by simple heuristics). While such an ordering may not directly correspond to an importance of given a table to downstream learning tasks, ARDA can optionally make use of this information to prioritize its search.

Coreset construction. As a first part of ARDA's joining pipeline we construct a *coreset*, a representative sample of rows from the base table. This sampling is done for computational efficiency: if the base table is small enough, no sampling is required. ARDA allows several types of coreset construction, described in Section 3.1. If sample size is specified, ARDA samples rows according to a customizable procedure, the default being uniform sampling. ARDA allows user to specify the desired coreset size, or ARDA can automatically select a coreset size using simple heuristics.

Join plan. During the stage of a *join plan* ARDA decides in which order tables should be considered for augmentation, how many tables to join at a time, and what tables should be grouped together for further feature selection. ARDA has three available options for a join plan described in Section 4. By default, ARDA uses the *budget* strategy.

Join execution. After determining a join plan, ARDA begins testing joins for viable augmentations. Performing each join requires special care for (1) handling soft keys, (2) handling missing values, and (3) handling one-to-many joins which might duplicate certain training examples, biasing the final model. We discuss join execution in Section 4.

Aggregation. This steps pre-aggregates foreign table rows over the given set of join keys to reduce one-to-many or many-to-many join cardinalities to one-to-one or many-to-one. This step includes *time-resampling* technique (discussed in Section 4) for time series columns. Aggregation and time-resampling is done by taking median value for numerical data and random sample for categorical columns.

Feature Selection. When considering whether or not a particular join represents a useful augmentation, ARDA uses a feature selection process. ARDA considers various types of feature selection algorithms that can be run simultaneously, which we discuss in subsequent sections. These methods include convex and non-linear models, such as Sparse Regression and Random Forests. Unless explicitly specified, ARDA uses a new feature selection algorithm method that we introduce in Section 6. This algorithm, random injection feature selection (RIFS), is based on injecting random noise into the dataset. We compare the running time and accuracy for different methods Section 7.

Final estimate. Finally, after ARDA has processed the entire join plan and selected a set of candidate joins to use as augmentations, ARDA trains a machine learning model on the newly collected features. ARDA is agnostic to the ML training process, and in our experimental analysis we tested both random forest models and advanced AML systems [1, 59] AutoML system as an optional estimator.

3.1 Coreset Constructions

In this subsection, we discuss various approaches used by ARDA to sample rows of our input to reduce the time we spend on joining, feature selection, model training, and

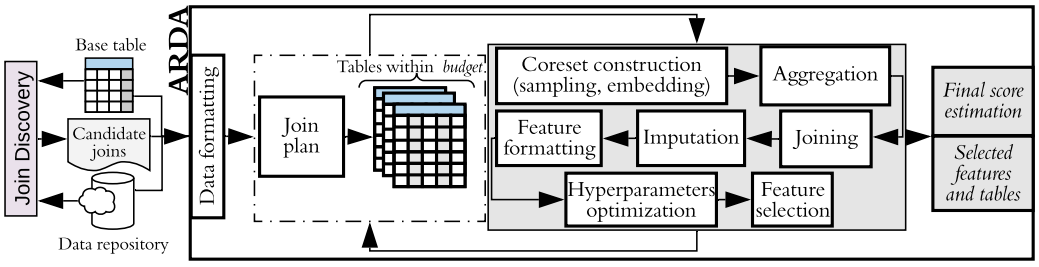

Figure 1: We provide a visual description of the workflow of our system. We start with a database and a base table as input to a join discovery framework that returns a large collection of tables which may contain information relevant to our learning task. After appropriate pre-processing, ARDA joins the candidate tables in batches.

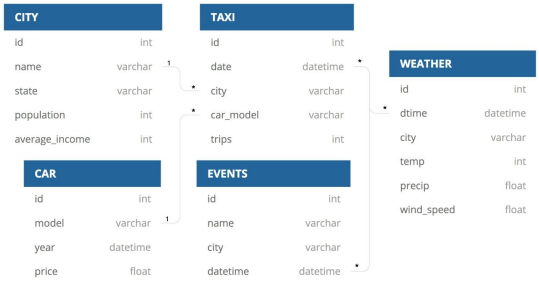

Figure 2: Example Schema: Initially a user has a base table TAXI and she finds a pool of joinable tables to see if some of them can help improve prediction error for taxi demand for a specific date given in target column trips.

score estimation. While we discuss generic approaches to sampling rows, often the input data is structured and the downstream tasks are known to the users. In such a setting, the user can use specialized coreset constructions to sample rows. We refer the reader to an overview of coreset constructions in [55] and the references therein.

Coreset construction can be viewed as a technique to replace large data set with a smaller number of representative points. This corresponds to reducing the number of rows (training points) of our input data, which in turn allows us to run feature selection more quickly, possibly at a cost in accuracy. We consider two main techniques for coreset construction: sampling and sketching. Sampling, as the name suggests, selects a subset of the rows and re-weights them to obtain a coreset. On the other hand, Sketching relies on taking sparse linear combination of rows, which inherently results in modified row values. This limits our use of sketching before we join tables since the sketched data may result in joins that are vastly inconsistent with the original data.

Uniform Sampling. The simplest strategy to construct a coreset is uniformly sampling the rows of our input tables. This process is extremely efficient since it does not require reading the input to create a coreset. However, uniform sampling does not have any provable guarantees and is not sensitive to the data. It is also agnostic to outliers, labels and anomalies in the input. For instance, if our input tables are labelled data for classification tasks, and one label appears way more often than others, a uniform sample might completely miss sampling rows corresponding to certain labels. Thus, the sample we obtain may not be diverse or well-balanced, which can negatively impact learning.

Stratified Sampling. To address the shortcomings of uniform sampling, we consider stratified sampling. Stratification is the process of dividing the input into homogeneous subgroups before sampling, such that the subgroups form a partition of the input. Then simple random sampling or systematic sampling can be applied within each stratum.

The objective is to improve the precision of the sample by reducing sampling error. It can produce a weighted mean that has less variability than the arithmetic mean of a simple random sample of the population. For classification tasks, if we stratify based on labels and use uniform sampling within each stratum, we obtain a diverse, well-balanced sub-sample, and no label is overlooked.

Matrix Sketching. Finally, consider sketching algorithms to sub-sample the rows of our input tables. Sketching has become a useful algorithmic primitive in many big-data tasks and we refer the reader to a recent survey [68], though is only applicable to numerical data. We note that under an appropriate setting of parameters, sketching the rows of the input data approximately preserves the subspace spanned by the columns.

An important primitive in the sketch-and-solve paradigm is a subspace embedding [20, 53], where the goal is to construct a concise describe of data matrix that preserves the norms of vectors restricted to a small subspace. Constructing subspace embeddings has the useful consequence that accurate solutions to the sketched problem are approximate accurately solutions to the original problem.

Definition 1 *(Oblivious subsapce embedding.) Given* $\epsilon, \delta > 0$ *and a matrix* A*, a distribution* $\mathcal{D}(\epsilon, \delta)$ *over* $\ell \times n$ *matrices* Π *is an oblivious subspace embedding for the column space of* A *if with probability at least* $1-\delta$*, for all* $x \in \mathbb{R}^d$*,* $(1-\epsilon)\|Ax\|_2 \leq \|\Pi Ax\|_2 \leq (1+\epsilon)\|Ax\|_2$*.*

We use OSNAP matrix for Π from [53], where each column has only one non-zero entry. In this case, ΠA can be computed in $nnz(A)\log(n)$ time, where nnz denotes the sparsity (number of nonzero entries) of A.

Definition 2 *(OSNAP Matrix.) Let* $\Pi \in \mathbb{R}^{\ell \times n}$ *be a sparse matrix such that for all* $i \in [n]$*, we pick* $j \in [\ell]$ *uniformly at random, such that* $\Pi_{i,j} = \pm 1$ *uniformly at random and repeat* $\log(n)$ *times.*

For obtaining a subspace embedding for A, Π needs to have $\ell = d\log(n)/\epsilon^2$ rows. We note that for tables where

the number of samples (rows) is much larger than the number of features(columns) the above algorithm can be used to get an accurate representation of the feature space in nearly linear time. However, here we note that Π takes linear combinations of rows of A and thus does not preserve numeric values. While the above guarantees hold in the worst-case, for real-world data, we can often use a lot fewer rows in our sketch.

Since sketching methods apply linear combinations of rows, we cannot hope to sketch the base table before the join takes place, without modifying it's contents. Therefore, ARDA sketches tables after the join is performed. Note that ARDA binarizes categorical features into a set of numerical features, which are amenable to sketching. For classification tasks, ARDA sketch rows independently within each label, analogous to stratified sampling.

4. JOINS

After selecting a coreset, ARDA needs to determine which tables will produce valuable augmentations. ARDA provides different strategies for performing joins, as discussed below. The objective of the join is to incorporate information from a foreign table in the optimal way. A key requirement from our join procedure is to preserve all base table rows since we do not want to artificially add or remove training examples.

Joins. There are four common join types between two tables INNER JOIN, FULL JOIN, RIGHT JOIN, and LEFT JOIN. But for our application, the goal is to *add* information to *each* row of the base table. Losing base table rows would lose data, and creating rows that do not correspond to base table rows would not be meaningful. Thus, only certain joins are suitable.

LEFT JOIN selects all records from the left table (base table), and the matched records from the right table (foreign table). The result is NULL from the right side if there is no match. This is the only join type that works for our augmentation task since it both preserves every record of the base table and brings only records from the foreign table that match join key values in base table. LEFT JOIN semantics also include leaving NULL when there is no match and results in missing values in the resulting dataset. There are many standard data imputation techniques in ML that we can use to handle these missing values.

None of the other join types satisfy our requirements. Consider for example the INNER JOIN which selects only records that have matching values in both tables. This type of join will drop training examples from the base table that do not join—losing data. For example, assume relation TAXI from Figure 2 performed an inner join with relation CAR on car model attribute. Further assume there is only a single car model in TAXI that is found in CAR. In this scenario, all the rows with car models that do not exist in CAR would be lost.

Key Matches. ARDA supports single key join, multiple key join (composite keys), mixed key join (composite key consists of soft and hard keys), and multiple-option key join (table has different keys it can be joined on with base table). The latter case implies different joining options with the foreign table. In this scenario ARDA joins on each key separately. An alternative option is to join on the key that gives a larger intersection, but this strategy could fail if such join happens to be useless for a target learning task.

In ARDA, we handle joining on *hard keys*, *soft keys*, and a custom combinations of them. Joining on a *hard key*, like Restaurant ID, implies joining rows of two tables where the values in column-keys are an exact match. When we join on a *soft key* we do not require an exact match between keys. Instead, we join rows with the column corresponding to the closest value. Examples of *soft keys* include time, GPS location, cities, age etc.

We note that in the special case of ARDA receiving keys from the time series data it automatically performs soft join. ARDA has two settings for soft join:

1. *Nearest Neighbour* Join: This joins a base table row value with the nearest value in the foreign table. The distance to rows the foreign table is defined in terms of numerical quantities based on the *soft key*. If tolerance threshold is specified and nearest neighbour does not satisfy the threshold then *null* values are filled instead.
2. *Two-way nearest neighbour* Join: For a given value of (the join column) of a base table row, this method finds the row largest foreign key less than the value and the row with smallest key greater than the value. These two rows from the foreign table are combined into one using linear interpolation on numeric values. For instance, let x be the numerical value for the base table key and $y_{\text{low}}, y_{\text{high}}$ be the values of the foreign keys that matched corresponding to rows r_{low} and r_{high}. Then, $x = \lambda y_{\text{low}} + (1-\lambda) y_{\text{high}}$ for some $\lambda \in [0, 1]$. We then join the row in the base table with $\lambda r_{\text{low}} + (1-\lambda) r_{\text{high}}$. This is used to account for how distant the keys are from the base table key value. If the values are categorical, they are selected uniformly at random.

Time-Resampling. Consider a situation when the base table has time series data specified in month/day/year format, while foreign table has format month/day/year hr:min:sec. In order to perform a join the time data needs to be "aligned" between the two tables.

One option would be to resolve the base table format to the midnight timestamp: month/day/year 00:00:00. However, for a hard join this might result in no matches with a foreign table and for a soft Nearest Neighbour join this would result in joining with only one row from a foreign table that has closest time to a midnight for the same day. This situation would result in information loss and therefore affect the quality of resulting features. ARDA identifies differences in time granularity and aggregates data over the span of time of a less precise key. In our scenario all rows that correspond to the same day would be resampled (aggregated) in foreign table before the join takes place.

Table grouping. ARDA supports grouping tables at three levels of granularity for joining:

1. *Table-join*: One table at a time in the priority order specified by the input. Based on our experiments it is the least desirable type of join since it adds significant time overhead and does not capture co-predicting features partitioned across tables.
2. *Budget-join:* As many tables at a time as we can fit within a predefined *budget*. The budget is a user defined parameter, set to be the number of rows by default. Size of *budget* trades off the number of co-predicting features we might discover versus the amount of noise the

model can tolerate to distinguish between good and bad features.

3. *Full materialization join:* All the tables prior to performing feature selection.

Table grouping is used by ARDA to create a *join plan* that is iteratively executed until all tables are processed or user-specified accuracy/error is achieved. By default, ARDA uses provided scores by Join Discovery system, such as Aurum [26] (or NYU Auctus), or if missing, ARDA computes *intersection-score*. The *Tuple Ratio* from [42] scoring can be specified by the user on demand. Tables are grouped in batches such that one batch does not exceed allowed *budget*. In ARDA *budget* is defines as maximum number of features we process at a time. By default, *budget* equals coreset size. An exception to this rule happens when a single table has more features than rows in a coreset, in this case ARDA ships an entire table to a feature selection pipeline.

Join Cardinality. There are four types of join cardinality: *one-to-one*, *one-to-many*, *many-to-one*, and *many-to-many*. Both *one-to-one* and *many-to-one* preserve the initial distribution of the base table (training examples) by avoiding changes in number of rows. However, for *one-to-many* and *many-to-many* joins, we would need to match at least one record from base table to multiple records from foreign table. This would require repeating the base table records to accommodate all the matches from a foreign table and introduce redundancy. Since we cannot change the base table distribution, such a join is infeasible. To address the issue with *one-to-many* and *many-to-many* joins we pre-aggregate foreign tables on join keys, thereby effectively reducing to the *one-to-one* and *many-to-one* cases.

Imputation. Since in data augmentation workflow we do not assume anything about the input data, we work with simple approaches that reduce the total running time of the system. We implemented a simple imputation technique: use the median value for numeric data and uniform random sampling for categorical.

5. FEATURE SELECTION OVERVIEW

Feature selection algorithms can be broadly categorized into filter models, wrapper models and embedded models. The filter model separates feature selection from classifier learning so that the bias of a learning algorithm does not interact with the bias of a feature selection algorithm. Typically, these algorithms rely on general characteristics of the training data such as distance, consistency, dependency, information, and correlation. Examples of such methods include Pearson correlation coefficient, Chi-squared test, mutual information and numerous other statistical significance tests [19]. We note that filter methods only look at the input features and do not use any information about the labels, and are thus sensitive to noise and corruption in the features.

The wrapper model uses the predictive accuracy of the learning algorithm to determine the quality of selected features. Wrapper-type feature selection methods are tightly coupled with a specific classifier, such as correlation-based feature selection (CFS) [31] and support vector machine recursive feature elimination (SVM-RFE) [35]. The trained classifier is then used select a subset of features. Popular approaches to this include *Forward Selection*, *Backward Elimination* and *Recursive Feature Elimination (RFE)*, which iteratively add or remove features and compute performance of these subsets on learning tasks. Such methods are likely to get stuck in local minimax [35,37,71]. Further, forward selection may ignore weakly correlated features and backward elimination may erroneously remove relevant features due to noise. They often have good performance, but their computational cost is very expensive for large data and training massive non-linear models [25, 46].

Given the shortcomings in the two models above, we focus on the embedded model, which includes information about labels, and incorporates the performance of the model on holdout data. Typically, the first step is to obtain a ranking of the features by optimizing a convex loss function [18, 47]. Popular objective functions include quadratic loss, hinge loss and logistic loss, combined with various regularizers, such as ℓ_1 and elastic net. The resulting solution is used to select a subset of features and evaluate their performance. This information is then used to pick a better subset of features.

One popular embedded feature selection algorithm is Relief, which is considered to be effective and efficient in practice [39,57,63]. However, a crucial drawback of Relief is that in the presence of noise, performance degrades severely [63]. Since Relief relies on finding nearest-neighbors in the original feature space, having noisy features can change the objective function and converge to a solution arbitrarily far from the optimal. While [63] offers an iterative algorithm to fix Relief, this requires running Expectation-Maximization (EM) at each step, which has no convergence guarantees. Further, each step of EM takes time quadratic in the number of data points. Unfortunately, this algorithm quickly becomes computationally infeasible on real-world data sets.

Given that all existing feature selection algorithms that can tolerate noise are either computationally expensive, use prohibitively large space or both, it's not obvious if such methods could be effective in data augmentation scenario when we deal with massive number of spurious features.

6. RANDOM INJECTION BASED FEATURE SELECTION

In this section, we describe our random injection based feature selection algorithm, including the key algorithmic ideas we introduce. Recall, we are given a dataset, where the number of features are significantly larger than the number of samples and most of the features are spurious. Our main task is to find a subset of features that contain signal relevant to our downstream learning task and prune out irrelevant features.

This is a challenging task since bereft of assumptions on the input, any *filter based* feature selection algorithm would not work since it does not take prediction error of a subset of features into account. For instance, consider a set of spurious input features that have high Pearson Correlation Coefficient or Chi-Squared test value. Selecting these features would lead to poor generalization error in our learning task but filter methods do not take this information into account. To this end, we design a comparison-based feature selection algorithm that circumvents the requirement of testing each subset of features. We do this by injecting carefully constructed random features into our dataset. We use the random features as a baseline to compare the input features with.

We train an ensemble of random forests and linear model to compute a joint ranking over the input and injected fea-

tures. Finally, we use a wrapper method on the resulting ranking to determine a subset of input features contain signal. However, our wrapper method only requires training the complex learning model a constant number of times.

6.1 Random Feature Injection

Given that we make no assumptions on the input data, our implementation should capture the following extreme cases: when most of the input features are relevant for the learning task, we should not prune out too many features. On the other hand, when the input features are mostly uncorrelated with the labels and do not contain any signal, we should aggressively prune out these features. We thus describe a random feature injection strategy that interpolates smoothly between these two corner cases.

Algorithm 1 : Feature Selection via Random Injection

Input: An $n \times d$ data matrix A, a threshold τ and fraction of random features to inject η, the number of random experiments performed k.

1. Create $t = \eta d$ random feature vectors $n_1, n_2, \ldots n_t \in \mathbb{R}^n$ using Algorithm 2. Append the random features to the input and let the resulting matrix be $A' = [A \mid N]$.
2. Run a feature selection algorithm (see discussion below) on A' to obtain a ranking vector $r \in [0,1]^{d+t}$. Repeat k times.
3. Aggregate the number of times each feature appears in front of all the random features $n_1, \ldots n_t$ according to the ranking r. Normalize each value by k. Let $r^* \in [0,1]^d$ be the resulting vector.

Output: A subset $\mathcal{S} \subseteq [d]$ of features such that the corresponding value in r^* is at least τ.

We show that in the setting where a majority of the features are good, injecting random features sampled from the standard Normal, Bernoulli, Uniform or Poisson distributions suffices, since our ensemble ranking can easily distinguish between random noise and true features. The precise choice of distribution depends on the input data.

The challenging setting is where the features constituting the signal is a small fraction of the input. Here, we use a more aggressive strategy with the goal of generating random features that look a lot like our input. To this end, we fit a statistical feature model that matches the empirical moments of the input data. Statistical modelling has been extremely successful in the context of modern machine learning and moment matching is an efficient technique to do so. We refer the reader to the surveys and references therein [21, 38, 51, 62].

We compute the empirical mean $\mu = \frac{1}{d}\sum_{i\in[d]} A_{*,i}$ and the empirical covariance $\Sigma = \frac{1}{d}\sum_{i\in[d]}(A_{*,i}-\mu)(A_{*,i}-\mu)^T$ of the input features. Intuitively, we assume that the input data was generated by some complicated probabilistic process that cannot be succinctly describes and match the empirical moments of this probabilistic process. An alternative way to think about our model is to consider μ to be a typical feature vector with Σ capturing correlations between the coordinates. We then fit a simple statistical model to our observations, namely $\mathcal{N}(\mu, \Sigma)$. Finally, we inject features drawn i.i.d. from $\mathcal{N}(\mu, \Sigma)$.

6.2 Ranking Ensembles

We use a combination of two ranking models, Random Forests and regularized Sparse Regression. Random Forests are models that have large capacity and can capture non-linear relationships in the input. They also typically work well on real world data and our experiments indicate that the inherent randomness helps identifying signal on average. However, since Random Forests have large capacity, as we increase the depth of the trees, the model may suffer from over-fitting. Additionally, Random Forests do not have provable guarantees on running time and accuracy. We use an off-the-shelf implementation of Random Forests and ARDA takes care of tuning the hyper-parameters.

Algorithm 2 : Random Feature Injection Subroutine

Input: An $n \times d$ data matrix A.

1. Compute the empirical mean of the feature vectors by averaging the column vectors of A, i.e. let $\mu = \frac{1}{d}\sum_{i\in[d]} A_{*,i}$.
2. Compute the empirical covariance $\Sigma = \frac{1}{d}\sum_{i\in[d]} (A_{*,i}-\mu)(A_{*,i}-\mu)^T$.
3. We model the distribution of features as $\mathcal{N}(\mu, \Sigma)$ and generate ηd i.i.d. samples from this distribution.

Output: A set of ηd random vectors that match the empirical mean and covariance of the input dataset.

We use $\ell_{2,1}$-norm minimization as a convex relation of sparsity. The $\ell_{2,1}$-norm of a matrix sums the absolute values of the ℓ_2-norms of its rows. Intuitively, summing the absolute values can be thought of as a convex relaxation of minimizing sparsity. Such a relaxation appears in Sparse Recovery, Compressed Sensing and Matrix Completion problems [13–15, 50] and references therein. The $\ell_{2,1}$-norm minimization objective has been considered before in the context of feature selection [45, 48, 56, 61, 69]. Formally, let X be our input data matrix and Y is our label matrix. We consider the following regularized loss function :

$$\mathcal{L}(W) = \min_{W\in\mathbb{R}^{c\times d}} ||WX - Y||_{2,1} + \gamma||W^T||_{2,1} \qquad (1)$$

While this loss function is convex (since it is a linear combination of norms), we note that both terms in the loss function are not smooth functions since ℓ_1 norms are not differentiable around 0. Apriori it is unclear how to minimize this objective without using Ellipsoid methods, which are computationally expensive (large polynomial running time). A long line of work studies efficient algorithms to optimize the above objective and use the state-of-the-art efficient gradient based solver from [56] to optimize the loss function in Equation 1. However, this objective does not capture non-linear relationships among the feature vectors. We show that a combination of the two approaches works well on real world datasets.

Additionally, for classification tasks we consider setting where the labels of our dataset may also be corrupted. Here, we use a modified objective from [56], where the labels are included as variables in the loss function. The modified loss function fits a consistent labelling that minimizes the $\ell_{2,1}$-norm from Equation 1. We observe that on certain datasets, the modified loss function discovers better features.

6.3 Aggregate Ranking

We use a straightforward re-weighting strategy to combine the rankings obtained from Random Forests (RF) and Sparse Regression (SR). Given the aforementionned rankings, we compute an aggregate ranking parameterized by $\nu \in [0, 1]$ such that we scale the RF rankings by ν and the SR ranking by $(1-\nu)$. Given an aggregate ranking, one natural way to do feature selection is *exponential search*. We choose the final number of features using a modified exponential search from [6]: we start with 2 features, and repeatedly double the number of features we test until model accuracy decreases. Suppose the model accuracy first decreases when we test 2^k features. Then, we perform a binary search between 2^{k-1} and 2^k. We experimentally observe that even the aggregate rankings are not monotone in prediction error. If instead, we perform a linear search over the ranking, we end up training our model n times, which may be prohibitively expensive (this strategy is also known as forward selection).

Algorithm 3 : Wrapper Algorithm

Input: An $n \times d$ data matrix A, a set of thresholds T
For each $\tau \in$ T in increasing order:

1. For each group run Algorithm 1 with A and τ as input. Let $\mathcal{S} \subseteq [d]$ denote the indices of the features selected by Algorithm 1.
2. Train the learning model on $A_{\mathcal{S}}$, i.e. the subset of features indexed by $\mathcal{S}$ and test accuracy on the holdout set.
3. If the accuracy is monotone, proceed to the new threshold. Else, output the previous subset.

Output: A subset of features indexed by set $\mathcal{S}$.

We therefore inject random features into our dataset and compute an aggregate ranking on the joint input. We repeat this experiment t times, using fresh random features in each iteration. We then compute how often each input feature appears in front of all the injected random features in the ordering obtained by the aggregate ranking. Intuitively, we expect that features that contain signal are ranked ahead of the random features on average. Further, input features that are consistently ranked worse than the injected random features are either very weakly correlated or systematic noise.

7. EXPERIMENTS

In this section, we describe our extensive experimental evaluation. We begin by presenting our main experiment that shows achieved augmentation on every dataset. Next, we evaluate each component of our system, including various coreset constructions, joining algorithms, feature selection algorithms and quality of data augmentation on real-world datasets. Our final experiment shows how well different

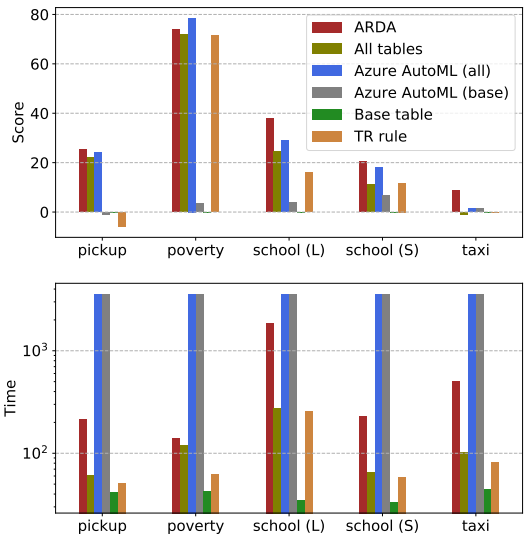

Figure 3: Achieved Augmentation is measured as % improvement over the baseline accuracy using our default fixed estimator and time is measured in seconds, on a log scale. All tables represents a score achieved using our estimator without feature selection. For ARDA we used RIFS as a feature selection method. TR rule is a table-filtering method from [42]. Azure AutoML (all) represents a score achieved on fully materialized join using Azure AutoML over an hour run. Azure AutoML (base) represents a baseline score achieved by Azure AutoML over an hour run.

feature selectors filter out synthetic noise on micro benchmarks. For evaluation of our experiments we used lightly auto-optimized Random Forest model for classification and regression tasks along with SVM with RBF kernel for classification only tasks, such that the best score achieved was reported.

Recall, our feature selection algorithm consists of a learning model used to obtain a ranking and a subset selection such as forward or backward selection. In our plots, we use the following feature selection methods: *Forward Selection*, *Backward Selection (Backward Elimination)*, *Recursive Feature Elimination (RFE)*, *Random Forest*, *Sparse Regression*, *Mutual Information*, *Logistic Regression*, *Lasso*, *Relief*, *Linear SVM*, *F-test*, *Tuple Rule*, RIFS. We also include experiments with two AutoML systems: Microsoft Azure AutoML [28] and Alpine Meadow [59]. Note that AutoML systems optimize the choice of final estimator while ARDA's final estimator is limited to random forest (classification and regression) and SVM (classification only).

Methods such as Random Forest, Sparse Regression, Mutual Information, Logistic Regression, Lasso, Relief, and Linear SVM return ranking that we use to select features using repetitive doubling and binary search algorithm. Forward Selection, Backward Selection, and Recursive Feature elimination (RFE) use Random Forest ranker. We picked Random Forest as the main ranker model for a lot of feature selectors because it was showing consistently good performance. For comparing ranking algorithms such as Random Forest, Sparse Regression, Mutual Information, Logistic Regression, Lasso, Relief, Linear SVM, F-test, we use *exponential search* described in Section 5, as this accurately captures

the monotonicity properties of the ranking and performed the best on our data sets.

For our experiments with RIFS, we inject 20% random features ($\mu = 0.2$) drawn i.i.d from Algorithm 2. We repeat this process t times (in our experiments $t = 10$) and in each iteration we train a Random Forest model as well as a Sparse Regression model to obtain two distinct rankings. For each feature, we compute the fraction of times it appears in front of all random features. We then discard all features less than τ, for $\tau \in$ T. We only increase the threshold as long as the performance of the resulting features on the holdout set increases monotonically.

7.1 Real World Datasets

Real World datasets are such that given a base table you search open sourced datasets for joinable tables using Join Discovery systems such as Aurum or NYU Auctus. All of our regression datasets are composed based on base tables provided by the DARPA D3Mcompetition. We use the NYU Auctus to search over new tables to augment. Our composed real scenario datasets:

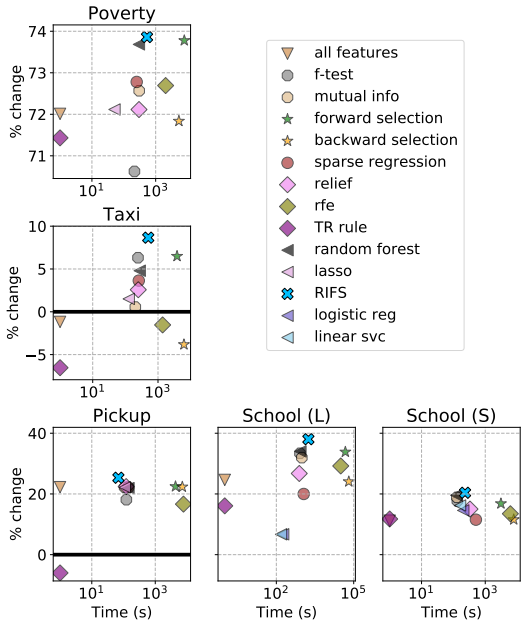

Figure 4: Scores vs. time for real-world datasets, where time along the x-axis is in log scale and represents the time spent on feature selection, and score is the %-improvement over the base table accuracy. Since all features and TR rule do not feature selection, the corresponding time is 0. Our feature selection algorithm, RIFS, consistently performs better augmentation on all datasets.

1. Taxi, Pickup and Poverty: These are regression datasets contains information about vehicle collision in NYC, taxi demand, and socio-economic indicators of poverty respectively. The base table is available through NYC Open Data, which can be retrieved using the Socrata API. In addition to base table we collected 29 joinable tables. Pickup contains hourly passenger pickup numbers from LGA airport by Yellow cabs between Jan 2018 to June 2018. The target prediction is the number of passenger pickups for a given hour. In addition to base table we collected 23 joinable tables. Poverty consists of socio-economic features like poverty rates, population change, unemployment rates, and education levels vary geographically across U.S. States and counties. In addition to base table we collected 39 joinable tables.
2. School (S,L): This is a classification dataset from the DataMart API using exact match as the join operation. The target prediction is the performance of each school on a standardized test based on student attributes. In addition to base table, School (S) collected 16 joinable tables whereas School (L) collected 350 joinable tables.

We note that joinable tables that we obtain using NYU Auctus are treated as "black-box" since they are not annotated and we cannot select features or reason about their relevance based on semantic properties.

7.2 Micro Benchmarks

We construct synthetic data to test the performance of feature selection algorithms. These experiments are done since we do not know the ground truth information about the features for a real world datasets that were constructed in a standard augmentation workflow. Since we do not know ground truth information about the features for a real world datasets we constructed two synthetic datasets to evaluate how well different methods perform in terms of noise filtering.

1. Kraken: A classification dataset consisting of anonymized sensors and usage statistics from the Kraken supercomputer. The target represents machine failure within a 24-hour window. The labels are binary, with 568 samples being 0 and 432 being 1.
2. Digits: A standard multi-label classification data set with roughly 180 samples for each digit between 0 and 9. It ships with Sklearn.

The synthetic features we append are random, uncorrelated noise vectors sampled from standard distributions such as uniform, Gaussian, and Bernoulli with randomly initialized parameters for these distributions. Since we work in the extreme noise regime, the number of noise features we append is 10× more than the number of original features. We note that not all base table features are relevant and thus may be filtered as well.

We compared performance of RIFS with various feature selection methods described in Section 5. Figure 4 presents percentage of improvement over accuracy achieved on the base table for a given algorithm on a given dataset. As we can see RIFS outperforms all of it's competitors and also performs well in terms of running time.

Additionally, as we observe, picking all features for the Taxi dataset can even decrease accuracy below that achieved by the base table. The forward selection algorithm is often a close second behind RIFS in terms of accuracy but is at least an order of magnitude slower. The Random Forest ranker with our noise injection rule also achieves augmentation of all datasets and it marginally faster than RIFS.

Table 1: Results on real world datasets on multiple feature selectors. Error is given as scaled Mean Absolute Error, Time represents feature selection and evaluation time in seconds.

Method	Taxi		Pickup		Poverty		School (S)		School (L)	
	error x10^5	time	error x10^4	time	error x10^6	time	accuracy	time	accuracy	time
baseline (our)	5658	45	2088	42	8116	43	69.05%	33	71.11%	35
all features (our)	572	102	1623	61	2271	121	76.79%	65	88.61%	278
all features (Alpine Meadow)	5874	3600	1761	3600	2684	3600	82.25%	3600	99.73%	3600
baseline (Azure AutoML)	5584	3600	2112	3600	7841	3600	81.42%	3600	91.74%	3600
all features (Azure AutoML)	5572	3600	1586	3600	1742	3600	73.55%	3600	73.93%	3600
TR rule	5676	81	2213	51	2318	43	77.17%	259	82.54%	259
RIFS	5168	553	1559	168	2121	575	83.13%	258	98.15%	1761
backward selection	5874	6277	1621	7181	2286	5399	77.03%	8052	88.22%	64627
forward selection	5291	3881	1619	4412	2128	7974	80.67%	3221	95.16%	47491
RFE	5745	1438	1730	7799	2216	2054	78.37%	6205	91.88%	31611
sparse regression	5452	302	1624	578	2209	298	77.01%	594	85.32%	1151
random forest	5386	152	1630	428	2136	351	82.58%	152	95.29%	856
f-test	5301	243	1710	232	2384	281	81.26%	169	94.85%	824
lasso	5573	187	1624	322	2263	104	n/a			
mutual info	5627	242	1625	286	2226	389	82.10%	179	93.83%	1173
relief	5512	297	1619	184	2263	361	79.42%	381	90.12%	828
linear svc	n/a						80.25%	191	75.87%	201
logistic reg	n/a						79.12%	263	75.87%	243

Table 2: Coreset construction sampling strategies for classification datasets: stratified sampling and sketching techniques (subspace embedding) described in 3.1. This table shows accuracy change of a given technique over uniform sampling.

Method	School (S)		Digits		Kraken	
	Stratified	Sketch	Stratified	Sketch	Stratified	Sketch
f-test	-1.29%	-1.86%	0.82%	-3.06%	0.54%	-4.72%
mutual info	-3.83%	-3.35%	0.28%	-3.69%	-0.66%	-0.56%
random forest	-2.84%	-4.05%	1.05%	-4.09%	7.06%	5.84%
sparse regression	-5.65%	2.85%	-0.15%	0.04%	-2.74%	1.02%
all features	0.58%	-0.97%	0.33%	-0.51%	0.10%	1.19%
RIFS	-2.70%	0.56%	0.13%	1.83%	-1.70%	2.45%
forward selection	-0.13%	1.55%	-1.06%	-0.68%	6.70%	0.62%
linear svc	0.56%	-0.82%	-0.04%	-0.48%	-4.72%	-4.65%
relief	-0.01%	0.21%	0.29%	-0.14%	-0.27%	0.31%

Table 3: Benchmarking sketching performance for various feature selection methods. The entries in the table indicate %-change over uniform sampling.

Method	Taxi	Pickup	Poverty
RIFS	-0.37%	0.77%	0.01%
sparse regression	1.37%	0.03%	3.60%
f-test	-3.42%	4.61%	0.00%
lasso	1.23%	-0.50%	-0.15%
mutual info	1.86%	0.13%	-0.91%
relief	-0.37%	-4.21%	-5.46%
all features	2.16%	-0.08%	-0.53%
random forest	-2.25%	-0.14%	-6.21%
forward selection	-2.06%	-7.56%	-3.07%

7.3 Feature selectors

Coreset Construction. Next, we compare the performance of constructing coresets via Uniform sampling, Stratified sampling and Subspace Embedding. Recall, discussion for these sampling techniques can be found in Subsection 3.1. Results of the experiments are given in Table 3 and 2.

The sketching algorithm (count-sketch) we use provably obtains a subspace embedding for the optimization objective in RIFS and Sparse Regression, given that the number of rows sampled are larger than the number of columns. While provable guarantees don't hold for our datasets (since columns are much larger) we expect sketching algorithms to perform well even with fewer rows.

We observe there is no one approach that always performs best, and the coreset performance is data and model dependent. ARDA uses uniform sampling as default sampling method, but allows users specify and explore other sampling strategies.

Soft Joins on Time-Series. We then experiment with soft join techniques that we described in Section 4. In Figure 5 we compare 4 different techniques for joining on a time series data: simple (hard) join on unmodified keys, Two-way Nearest Neighbour soft join, Nearest Neighbour soft join, and Time-Resampling technique for a simple (hard) join. Every Two-Way Nearest Neighbour and Nearest Neighbour joins also include time-resampling technique.

We can see that in most cases Two-Way Nearest Neighbour soft join beats Nearest Neighbour and both outperform simple hard join. We address rounding for hard join in the Taxi dataset by identifying a target time-granularity and aggregate rows of a foreign table that correspond to same days. We observe that Taxi dataset time-resampling technique performed much better combined with hard join.

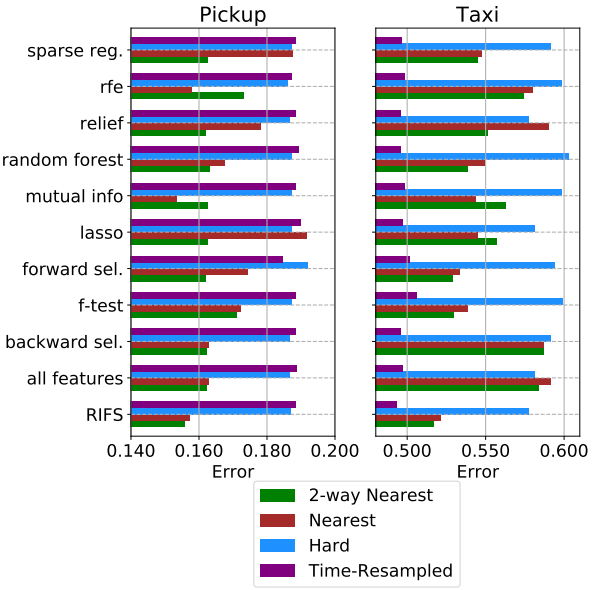

Figure 5: Performance of different techniques for a time series join on multiple feature selectors. Pickup errors were multiplied by 10^2 and Taxi errors were multiplied by 10 for convenience.

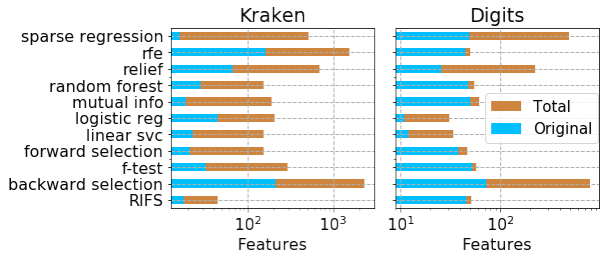

Figure 6: This figure shows the number of features selected by each feature selection method as well as fraction of original features to the total selected features. The difference is planted synthetic noise. X-axis represents the number of features in log scale.

Table 4: Performance of ARDA with RIFS and Tuple Rule as a table filtering step for real world datasets. Hyperparameter τ was optimized for each dataset.

Dataset	Score change	Speed (x faster)	Tables removed	τ
Taxi	-0.04%	3.18	10	24
Pickup	-15.35%	3.50	17	17
Poverty	-1.19%	5.87	36	15
School (S)	-1%	1.14	2	15
School (L)	-5%	1.32	39	17

Tuple Ratio Test. Kumar et al. [42] propose several decision rules to safely eliminate tables before feature selection takes place. Here, we evaluate the most conservative rule proposed, the *Tuple Ratio rule*. The *Tuple Ratio* is defined as $\frac{n_S}{n_R}$, where n_S is the number of training examples in a base table and n_R is a size of a foreign-key domain. Based on an analysis of VC dimensions in binary classification problems, the decision rule suggests that a foreign table cannot help a predictive model if $\frac{n_S}{n_R}$ larger than a threshold.

We note that this decision rule is intended only as a filtering technique (i.e., a small tuple ratio does not imply that a model will improve from the resulting join). We experimented with Tuple Ratio as a stand-alone method for feature augmentation and as a pre-filtering step. During the pre-filtering step we eliminate tables that are above optimized τ and therefore avoid feature selection. Kumar et al. suggest that τ only needs to be tuned per model, not per dataset. However, we found slight improvements from optimizing the threshold per dataset: we report the threshold used for each dataset in Table 4.

Tables 1 show that Tuple Ratio (shown as *TR rule*) does not perform well as a stand-alone selection solution. Table 4 shows experiments that use Tuple Ratio as a filtering tool to eliminate tables before feature selection takes place. We observed that filtering with the TR decision rule caused small to moderate decreases in model accuracy and medium to significant improvements in training time.

The change in model accuracy is most pronounced on the Pickup dataset, which is not surprising given that the TR decision rule was designed for (binary) classification problems and the Pickup dataset represents a regression task. Overall, we conclude that the TR decision rule is a useful filtering technique for ARDA when training time is paramount and some model accuracy can be sacrificed.

Table grouping. Here we evaluate our three table grouping methods described in Section 4: *table-join*, *budget-join*, and *full materialization join*. Table 5 compares the performance of *table-join* and *full materialization join* against *budget-join* for various feature selectors. The fact that *table-join* almost always performs worse than *budget-join* is evidence that these datasets contain *co-predictors*: some features are only useful when combined with other features from different tables. The presence of co-predictors also help explain why full materialization occasionally outperforms *budget-join*. However, with RIFS, full materialization never outperforms *budget-join* by a significant margin, and often degrades performance since full materialization results in many noise features which tend to interfere with the model's ability to learn. Since *budget-join* is not significantly outperformed, we believe that it represents a reasonable compromise between full materialization and *table-join* in terms of finding co-predictors (since *budget-join* still considers multiple tables at once) and creating noise features.

Filtering Synthetic Noise. Using our micro benchmark datasets we compute the number of true and noisy features recovered by different feature selectors. We plot the resulting experiment in Figure 6.

From Figure 6 we can conclude that the amount of noise that is being selected depends both on a feature selection method as well as on a data itself. While not perfect, RIFS shows best selectivity that filters out a lot of noise and redundant features from the original pool, while maintaining the top accuracy (see Table 6).

8. RELATED WORK

There has been extensive prior work on data mining, data augmentation, knowledge discovery, and feature selection. We provide a brief overview of this literature. Perhaps the most pertinent work is Kumar et. al. [42] and Shah et al. [58], which study when joining tables to the base table is unnecessary (i.e., is highly unlikely to improve a predictive

Table 5: We compare change in final accuracy among several feature selectors with *budget-join* being the baseline. *Table-join* joins a single table at a time with feature selection being performed after each join. *Full materialization* joins all tables before feature selection is performed.

Method	Taxi		Pickup		Poverty		School(S)	
	Table	Fullmat	Table	Fullmat	Table	Fullmat	Table	Fullmat
RIFS	-4.10%	0.97%	-25.75%	0.87%	-0.40%	-2.88%	-2.11%	-1.45%
forward selection	-3.92%	1.11%	-6.60%	-7.53%	-8.66%	-0.50%	-1.00%	-0.14%
random forest	-3.55%	2.86%	-22.01%	0.04%	-7.92%	-3.20%	-1.85%	0.12%
sparse regression	-5.35%	-3.27%	-25.30%	-12.67%	-2.51%	0.92%	0.44%	-0.29%

Table 6: Results on micro benchmark datasets on multiple feature selectors. acc. represents accuracy metric.

Method	Kraken		Digits	
	acc.	time	acc.	time
baseline (our)	56.80%	7	39.77%	8
all features (our)	57.16%	29	90.97%	37
all features (Alpine Meadow)	52.72%	3600	81.56%	3600
all features (Azure AutoML)	79.63%	3600	92.14%	3600
baseline (Azure AutoML)	62.77%	3600	45.11%	3600
RIFS	71.44%	466	95.00%	172
backward selection	57.04%	4446	90.92%	1883
forward selection	64.54%	1527	94.24%	1118
RFE	57.18%	1059	94.07%	2174
sparse regression	62.76%	255	91.12%	397
random forest	65.42%	301	94.47%	147
f-test	74.20%	258	93.85%	213
linear svc	63.66%	188	91.08%	522
logistic reg	62.80%	156	91.38%	460
mutual info	57.46%	258	94.27%	227
relief	57.70%	302	91.29%	256

model). The proposed decision rules (which come with theoretical bounds) allow practitioners to rule out joining with specific tables entirely. Both [42] and [58] are orthogonal to this work, as we are concerned with effectively augmenting a base table to improve model accuracy as opposed to determining which joins are potentially safe to avoid. In our experimental study, we demonstrate that Kumar et al.'s decision rules can be used as a prefiltering technique for Arda.

Data discovery. Data discovery systems deal with sharing datasets, searching new datasets, and discovering relationships within heterogeneous data pools [7–9, 12, 17, 29, 30, 32–34, 65, 65]. For example, Aurem [26] helps users automatically identify joins between tables representing similar entities. While these systems help users discover new data and explore relationships between datasets, they do not automatically determine whether or not such new information is useful for a predictive model. Arda uses data discovery systems as an input, combining new datasets and discovered relationships into more powerful predictive models.

Data augmentation. In general, data augmentation involves combining a dataset with additional information (derived or otherwise) in order to improve the accuracy of a predictive model. For example, learned embedding techniques [22, 43] like Word2Vec [52] use a large corpus of unlabeled data to learn an embedding that can then be used to semantically represent new pieces of information. Other systems, like Octopus and InfoGather [70] automatically search the web for information relevant to a user's data (e.g., discover the ZIP codes and populations of a list of cities).

Feature selection. A natural approach to our problem would be to join all compatible tables and rely of the learning algorithm to ignore irrelevant features. The number of irrelevant features may be much greater than the number of relevant features, especially in large data lakes with high data diversity. Unfortunately, almost all ML algorithms are highly sensitive to noise, especially when the noise overwhelms the relevant features [3, 16, 49, 64, 67].

The negative impact of noise on model performance is well-studied [10, 23], resulting in several works on feature selection. For an overview, see [40].

AML. Recently, Automatic Machine Learning (AML) has emerged as independent area of research seeking methods to automate the process of finding the best machine learning pipelines for a given learning task. AML systems often handle ML tasks like feature selection, model selection, and hyperparameter tuning. Examples of AML tools include Autosklearn [27], Alpine Meadow [59], Microsoft Azure AutoML , Google Cloud AutoML , Neural Architecture Search [4, 5], and others. However, these systems generally rely exclusively on the data supplied by the user and expect input in single-table form. While Arda can use any AML system to construct a final model from the features Arda discovers, many techniques presented here could also be integrated into an AML system.

9. CONCLUSION & FUTURE WORK

We built a system that automates the data augmentation workflow and integrates it with the ML model selection process. We demonstrated the effectiveness and versatility of our system through a vast set of experiments that benchmark different available approaches for coreset construction, joining, feature selection etc. We addressed methods for joining on date-time keys, albeit location-based joins remain unexplored. Given the modular nature of our system, it would be interesting to evaluate sophisticated methods for data imputation and aggregation, neural networks as learning models, and statistical significance tests for augmented features. While our work concentrated around achieving augmentation for a single base table, we hope future work can address transitive joins. Another important direction is to build a large corpus of tables with column annotations to allow reasoning about selected features beyond their effect on prediction score and statistical significance.

10. ACKNOWLEDGMENTS

This research is supported by Google, Intel, and Microsoft as part of the MIT Data Systems and AI Lab (DSAIL) at MIT, NSF IIS 1900933, DARPA Award 16-43-D3M-FP040, and the MIT Air Force Artificial Intelligence Innovation Accelerator (AIIA).

11. REFERENCES

[1] Microsoft Azure Services, http://www.microsoft.com/azure/.

[2] NYU Auctus, https://datamart.d3m.vida-nyu.org/.

[3] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 2017.

[4] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[5] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le. Neural optimizer search with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 459–468. JMLR. org, 2017.

[6] J. L. Bentley and A. C.-C. Yao. An almost optimal algorithm for unbounded searching. *Information processing letters*, 5(SLAC-PUB-1679), 1976.

[7] A. Bhardwaj, S. Bhattacherjee, A. Chavan, A. Deshpande, A. J. Elmore, S. Madden, and A. G. Parameswaran. Datahub: Collaborative data science & dataset version management at scale. *arXiv preprint arXiv:1409.0798*, 2014.

[8] A. Bhardwaj, A. Deshpande, A. J. Elmore, D. Karger, S. Madden, A. Parameswaran, H. Subramanyam, E. Wu, and R. Zhang. Collaborative data analytics with datahub. *Proc. VLDB Endow.*, 8(12):1916–1919, Aug. 2015.

[9] S. Bhattacherjee, A. Chavan, S. Huang, A. Deshpande, and A. Parameswaran. Principles of dataset versioning: Exploring the recreation/storage tradeoff. *Proc. VLDB Endow.*, 8(12):1346–1357, Aug. 2015.

[10] L. Breiman. Bias, Variance, and Arcing Classifiers. Technical report, 1996.

[11] D. Brickley, M. Burgess, and N. Noy. Google dataset search: Building a search engine for datasets in an open web ecosystem. In *The World Wide Web Conference*, pages 1365–1375, 2019.

[12] M. J. Cafarella, A. Halevy, and N. Khoussainova. Data integration for the relational web. *Proc. VLDB Endow.*, 2(1):1090–1101, Aug. 2009.

[13] E. J. Candes and Y. Plan. Matrix completion with noise. *Proceedings of the IEEE*, 98(6):925–936, 2010.

[14] E. J. Candès and B. Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717, 2009.

[15] E. J. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on information theory*, 52(2):489–509, 2006.

[16] N. Carlini and D. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14. ACM, 2017.

[17] R. Castro Fernandez, D. Deng, E. Mansour, A. A. Qahtan, W. Tao, Z. Abedjan, A. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, et al. A demo of the data civilizer system. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1639–1642. ACM, 2017.

[18] G. C. Cawley, N. L. Talbot, and M. Girolami. Sparse multinomial logistic regression via bayesian l1 regularisation. In *Advances in neural information processing systems*, pages 209–216, 2007.

[19] G. Chandrashekar and F. Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, Jan. 2014.

[20] K. L. Clarkson and D. P. Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 81–90. ACM, 2013.

[21] A. C. Davison. *Statistical models*, volume 11. Cambridge University Press, 2003.

[22] L. Deng. Table2vec: Neural word and entity embeddings for table population and retrieval. Master's thesis, University of Stavanger, Norway, 2018.

[23] P. Domingos. The role of occam's razor in knowledge discovery. *Data mining and knowledge discovery*, 3(4):409–425, 1999.

[24] B. Donovan and D. Work. New york city taxi trip data (2010-2013), 2016.

[25] P. A. Estévez, M. Tesmer, C. A. Perez, and J. M. Zurada. Normalized mutual information feature selection. *IEEE Transactions on Neural Networks*, 20(2):189–201, 2009.

[26] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1001–1012. IEEE, 2018.

[27] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *Advances in neural information processing systems*, pages 2962–2970, 2015.

[28] N. Fusi, R. Sheth, and M. Elibol. Probabilistic matrix factorization for automated machine learning. In *Advances in Neural Information Processing Systems*, pages 3348–3357, 2018.

[29] W. Gatterbauer and P. Bohunsky. Table extraction using spatial reasoning on the css2 visual box model. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1313. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[30] H. Gonzalez, A. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley, and W. Shen. Google fusion tables: data management, integration and collaboration in the cloud. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 175–180. ACM, 2010.

[31] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002.

[32] A. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. Goods: Organizing google's datasets. In *Proceedings of the 2016 International Conference on Management of Data*, pages 795–806. ACM, 2016.

[33] A. Y. Halevy. Data publishing and sharing using fusion tables. In *CIDR*, 2013.

[34] A. Y. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. Managing google's data lake: an overview of the goods system. *IEEE Data Eng. Bull.*, 39(3):5–14, 2016.

[35] M. A. Hall and L. A. Smith. Feature selection for machine learning: Comparing a correlation-based filter approach to the wrapper. In *Proceedings of the Twelfth International Florida Artificial Intelligence Research Society Conference*, page 235–239. AAAI Press, 1999.

[36] X. He, K. Zhao, and X. Chu. Automl: A survey of the state-of-the-art. *arXiv preprint arXiv:1908.00709*, 2019.

[37] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Machine Learning Proceedings 1994*, pages 121–129. Elsevier, 1994.

[38] S. Khalid, T. Khalil, and S. Nasreen. A survey of feature selection and feature extraction techniques in machine learning. In *2014 Science and Information Conference*, pages 372–378. IEEE, 2014.

[39] K. Kira and L. A. Rendell. A practical approach to feature selection. In *Machine Learning Proceedings 1992*, pages 249–256. Elsevier, 1992.

[40] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial intelligence*, 97(1-2):273–324, 1997.

[41] T. Kraska. Northstar: An Interactive Data Science System. *PVLDB*, 11(12):2150–2164, 2018.

[42] 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*, pages 19–34. ACM, 2016.

[43] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014.

[44] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[45] B. Liu, G. Gui, S. Matsushita, and L. Xu. Dimension-reduced direction-of-arrival estimation based on l2/l1-norm penalty. *IEEE Access*, 6:44433–44444, 2018.

[46] H. Liu, H. Motoda, R. Setiono, and Z. Zhao. Feature selection: An ever evolving frontier in data mining. In *Feature Selection in Data Mining*, pages 4–13, 2010.

[47] H. Liu and L. Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge & Data Engineering*, pages 491–502, 2005.

[48] Y. Ma, C. Li, X. Mei, C. Liu, and J. Ma. Robust sparse hyperspectral unmixing with l2/l1 norm. *IEEE Transactions on Geoscience and Remote Sensing*, 55(3):1227–1239, 2016.

[49] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv:1706.06083*, 2017.

[50] E. C. Marques, N. Maciel, L. Naviner, H. Cai, and J. Yang. A review of sparse recovery algorithms. *IEEE Access*, 7:1300–1322, 2018.

[51] P. McCullagh. What is a statistical model? *Annals of statistics*, pages 1225–1267, 2002.

[52] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[53] J. Nelson and H. L. Nguyên. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 117–126. IEEE, 2013.

[54] G. Paolacci, J. Chandler, and P. G. Ipeirotis. Running experiments on amazon mechanical turk. *Judgment and Decision making*, 5(5):411–419, 2010.

[55] J. M. Phillips. Coresets and sketches. *arXiv preprint arXiv:1601.00617*, 2016.

[56] M. Qian and C. Zhai. Robust unsupervised feature selection. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[57] M. Robnik-Šikonja and I. Kononenko. Theoretical and empirical analysis of relieff and rrelieff. *Machine learning*, 53(1-2):23–69, 2003.

[58] V. Shah, A. Kumar, and X. Zhu. Are key-foreign key joins safe to avoid when learning high-capacity classifiers? *Proc. VLDB Endow.*, 11(3):366–379, Nov. 2017.

[59] Z. Shang, E. Zgraggen, B. Buratti, F. Kossmann, P. Eichmann, Y. Chung, C. Binnig, E. Upfal, and T. Kraska. Democratizing data science through interactive curation of ml pipelines. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 1171–1188, New York, NY, USA, 2019. Association for Computing Machinery.

[60] E. R. Sparks, A. Talwalkar, M. J. Franklin, M. I. Jordan, and T. Kraska. Tupaq: An efficient planner for large-scale predictive analytic queries. *arXiv preprint arXiv:1502.00068*, 2015.

[61] M. Stojnic. L2/l1-optimization in block-sparse compressed sensing and its strong thresholds. *IEEE Journal of Selected Topics in Signal Processing*, 4(2):350–357, 2010.

[62] X. Sun and B. Bischl. Tutorial and survey on probabilistic graphical model and variational inference in deep reinforcement learning. *arXiv preprint arXiv:1908.09381*, 2019.

[63] Y. Sun. Iterative relief for feature weighting: algorithms, theories, and applications. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1035–1051, 2007.

[64] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[65] I. G. Terrizzano, P. M. Schwarz, M. Roth, and J. E. Colino. Data wrangling: The challenging yourney from the wild to the lake. In *CIDR*, 2015.

[66] C. Thornton, F. Hutter, H. H. Hoos, and

K. Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013.

[67] E. Wong and J. Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 2017.

[68] D. P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

[69] Y. Xiao, S.-Y. Wu, and B.-S. He. A proximal alternating direction method for l2/l1 norm least squares problem in multi-task feature learning. *Journal of Industrial and Management Optimization*, 8(4):1057, 2012.

[70] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 97–108. ACM, 2012.

[71] J. Yang and V. Honavar. Feature subset selection using a genetic algorithm. In *Feature extraction, construction and selection*, pages 117–136. Springer, 1998.