A Human-in-the-loop Attribute Design Framework for Classification

Md Abdus Salam
The University of Texas at Arlington
mdsalam@uta.edu

Mary E. Koone
The University of Texas at Arlington
mary.koone@mavs.uta.edu

Saravanan Thirumuruganathan
QCRI, HBKU
sthirumuruganathan@hbku.edu.qa

Gautam Das
The University of Texas at Arlington
gdas@uta.edu

Senjuti Basu Roy
New Jersey Institute of Technology
senjutib@njit.edu

ABSTRACT

In this paper, we present *a semi-automated, "human-in-the-loop" framework for attribute design* that assists human analysts to transform raw attributes into effective derived attributes for classification problems. Our proposed framework is optimization guided and fully agnostic to the underlying classification model. We present an algebra with various operators (arithmetic, relational, and logical) to transform raw attributes into derived attributes and solve two technical problems: (a) the **top-k buckets design** problem aims at presenting human analysts with k buckets, each bucket containing promising choices of raw attributes that she can focus on only without having to look at all raw attributes; and (b) the **top-l snippets generation** problem, which iteratively aids human analysts with top-l derived attributes involving an attribute. For the former problem, we present an effective exact bottom-up algorithm that is empowered by pruning capability, as well as random walk based heuristic algorithms that are intuitive and work well in practice. For the latter, we present a greedy heuristic algorithm that is scalable and effective. Rigorous evaluations are conducted involving 6 different real world datasets to showcase that our framework generates effective derived attributes compared to *fully manual or fully automated methods*.

CCS CONCEPTS

• **Information systems** → *Crowdsourcing*;

KEYWORDS

human computation; crowdsourcing; attribute design; feature engineering

ACM Reference Format:
Md Abdus Salam, Mary E. Koone, Saravanan Thirumuruganathan, Gautam Das, and Senjuti Basu Roy. 2019. A Human-in-the-loop Attribute Design Framework for Classification. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3308558.3313547

1 INTRODUCTION

Attribute design (also known as feature engineering) is one of the most challenging aspects of a data science pipeline, and is considered to be an "arduous process for data scientists"[1] [3, 14], where the raw attributes often need to be transformed into derived attributes that can be more effective for building predictive models such as classifiers. For example, in a healthcare setting involving large, high dimensional and heterogeneous electronic health records (EHR) datasets, an attribute such as the average length of prior hospitalization can be very useful to build effective predictive or classification models on future hospitalization (i.e., readmission) [4]); however, such attributes are not readily available in the raw dataset. For example, Table 1 shows that the average length of hospitalization needs to be computed per hospitalization window considering admission and discharge date, and taking the average of these windows.

The state-of-the-art attribute design techniques fall into one of these two extremes: (a) *fully manual*, painstakingly slow and heavily reliant on domain expertise, often requiring data scientists to go through a repetitive trial-and-error exercise until the set of designed attributes are satisfactorily effective or (b) *fully automated* techniques (some notable systems are, Data Science Machine [18][2], ExploreKit [20][3], One Button Machine [23], or Featuretools[4]). Such methods are not model agnostic, require substantial time to identify derived attributes that are opaque to the human analyst.

In this paper, we investigate a semi-automated "human-in-the-loop" framework that is agnostic to any classification model. Our work is inspired by a handful of recent works [8, 36, 50] which propose a conceptual framework and empirically argue that attribute design by involving human analysts can effectively substitute for either of the two extremes (fully manual or fully automated). However, these do not study how to optimize human involvement in the process or how to guide it.

Proposed Framework: Our proposed framework is developed around three fundamental aspects.

Model agnostic measure: We adopt a principled and quantifiable measure of the effectiveness of a derived attribute that is agnostic to the specifics of any underlying classification model. We consider *Mutual Information* (MI in short) [38] to determine

[1]https://www.itproportal.com/features/dont-let-feature-engineering-stagnate-your-ml-projects/
[2]https://people.csail.mit.edu/kalyan/dsm/
[3]https://github.com/giladkatz/ExploreKit
[4]https://docs.featuretools.com/#minute-quick-start

the predictive ability of an attribute. Intuitively, MI is a symmetric measure that captures "correlation" between a pair of attributes and quantifies how much information is contained in one attribute about the other. We are interested in producing derived attributes that have high MI with the target variable (or the class label). MI is known to have several desirable properties: (a) MI is proved to have certain qualitative guarantees when chosen for attribute selection for multiple popular classification models, such as Naive Bayes [38], or linear regression [12]; (b) prior works have also shown that MI optimizes important properties in attribute selection, such as; relevance, complementarity, and redundancy [38] and (c) finally, as we shall show later in the paper, MI satisfies upward closure [6] which is useful for designing effective algorithms.

Attributes algebra: The second aspect of our framework is an algebra that dictates how raw attributes (numerical or categorical) are to be combined to produce the derived attributes. We study arithmetic, logical, and relational operators. Our initial example of average length of prior hospitalization is an example of arithmetic operator, whereas an example of the logical operation is the following Boolean attribute: *elderly* AND *diabetic* OR *covered under medicaid*. A patient is more vulnerable to future hospitalization if she gets an "yes" on this attribute. A logical operator on the other hand is *obesity* that is set to True, when $BMI > 30$.

Guiding human analysts: The third aspect of our framework is the investigation of how to optimize the involvement of humans in the attribute design process and ensure their efforts are successful. This is motivated by a handful of recent works that argue that humans must not be left unguided in the attribute design process [8, 36, 50], Simply presenting the entire set of raw attributes or hundreds and thousands of raw records to the human analysts might overwhelm them. We propose to present each analyst with small summarized portions of the raw data and raw attributes that is likely to be most helpful for the human to analyze and create useful attributes from. We formulate two technical problems that need to be addressed:

(1) Top-k buckets design: We present each analyst with only k (a small number) *buckets*, each with at most x attributes, that are most predictive. For example, when $x = 3$, a useful bucket that is likely to have predictive qualities may contain {elderly, diabetic, covered-by-medicaid}. We show that generating these buckets is NP-hard. We provide an exact algorithm, `ExBKT`, and an even more efficient heuristic algorithm called `RandomizedBKT` that performs random walks on the attribute lattice to design the top-k buckets. However, just producing buckets that are highly predictive (high MI) may not be enough. We propose `RandomizedCovBKT` that finds top-k buckets that are not only effective wrt MI, but also they together cover many "good" raw attributes of the dataset.

(2) Top-l snippets generation: Even after the buckets of raw attributes have been generated, the human analyst may have to sift through large volumes of data (i.e., the tuples with those raw attribute values) to design good derived attributes, and this task may still be overwhelming. Thus, we propose an interactive procedure by which an analyst may approach this task. Given a bucket, the analyst starts designing a derived attribute (i.e., an algebraic expression involving the raw bucket attributes) interactively (i.e, term by term). At each iteration, our framework recommends l *snippets*, i.e., it suggests the l best ways to extend the partially created derived attributes using the algebra. More formally, at a given iteration the snippet generation process suggests how to augment the partially composed derived attribute by length j, i.e., combining j new raw attributes with the derived attribute developed thus far. For instance, if *elderly* has already been selected by the analyst, and if $j = 1$, an example snippet will suggest a visual distribution that shows how *elderly* AND *diabetic* correlates with the prediction target variable *hospital readmission*.

Evaluations: Rigorous evaluations are conducted considering 6 real world datasets by comparing our solutions with two fully automated methods (ExploreKit and Featuretools), and a fully manual domain expert guided attribute design process, as well as intuitive baselines. Our experimental results demonstrate that we scale up to $7x - 20x$ faster compared to fully automated (ExploreKit) and the fully manual process, while ensuring similar quality (average improvement 14%). By leveraging the domain expert, our framework avoids falling into the pitfall where derived features could actually be detrimental to the performance (unlike FeatureTools). It also has other appealing properties, such as being easily parallelizable, and exhibiting "anytime behavior" whereby it gives meaningful results at any point of execution. Our scalability results demonstrate that both bucket design and snippet generation procedures are efficient and can work interactively with the human in the loop.

In summary, the paper makes the following contributions:

- Proposed Framework: We initiate the study of a model agnostic semi-automated attribute design framework (Section 2) that judiciously involves human analysts in the loop.
- Technical contributions: We formalize two technical problems around the framework and present several theoretical and algorithmic results (Sections 3 and 4).
- Experimental Results: We conduct extensive experiments to demonstrate both effectiveness and scalability of our proposed solutions (Section 5).

2 PRELIMINARIES AND FORMALISM

In this section, we describe our data model, present our framework, and formalize the technical problems.

Example 2.1. We present a toy running example in Table 1 that provides longitudinal data of a heart failure datamart in a hospital. The objective is to build a classifier that predicts whether a patient getting discharged from the hospital will be readmitted within 6 months of discharge, considering predictors from base attributes (first 10 columns) and derived attributes (designed using the base attributes). The data is augmented by adding the last column, representing the class label per patient per admission instance.

2.1 Data Model

Base attributes, records, and target variable: A given dataset is comprised of a set $\mathcal{A}$ of n attributes and m records, as well as an additional target attribute (column) Z. These attributes are referred to as base or raw attributes. In this work, we consider all major types of attributes including numeric, Boolean, and categorical. Our proposed approach can work for a wide variety of predictive modeling tasks including classification and regression (we refer to them classification in general). Depending on the nature of the

patient-id	admission date	dis. date	gender	BMI	income	medicaid	senior	diabetic	primary-diagnosis	readmission
p1	10-1-2013	10-23-2013	M	32.5	low	Y	Y	Y	Congestive heart failure	N
p1	5-2-2014	5-10-2014	M	32.2	low	Y	Y	Y	Heart Attack	Y
p2	5-12-2014	5-14-2014	F	26.7	medium	N	N	N	Arrhythmia	N
p3	10-1-2015	11-1-2015	M	29.9	low	N	N	Y	Cardiomyopathy	N
p3	6-10-2016	6-21-2016	M	29.8	low	N	N	Y	Cardiomyopathy	N
p4	12-12-2017	12-14-2017	F	23.4	medium	N	N	N	Arrhythmia	N
p4	9-2-2018	9-29-2018	F	27.8	medium	N	N	N	Heart Attack	Y

Table 1: Example 2.1 toy heart failure datamart

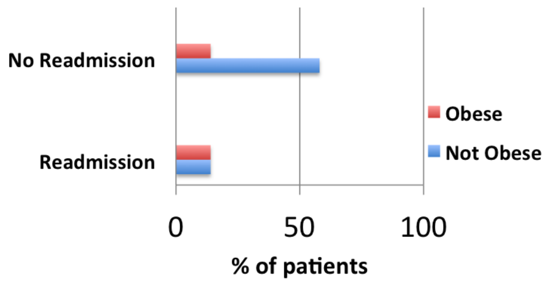

Figure 1: A snippet on Obesity

problem, Z is a continuous variable (regression problem), or has discrete values (classification problem). Using Example 2.1, $n = 10$ and $m = 7$, Z corresponds to readmission within 6 months of discharge (discrete).

Algebra, $\mathcal{L}$: A set of operators (arithmetic, relational and logical) which are applied to one or more base attributes to combine them. When the base attributes are numeric, we consider arithmetic operators to combine them: addition (+), subtraction (−), multiplication (x), division (/). For Boolean attributes, we consider logical operators, AND, OR. We also support all relational operators that compare two base attributes or a base attribute with a constant (e.g. $A_i > A_j$ or $A > 100$). Our framework is flexible enough to support arbitrary operators. We also support aggregate operators over a set of tuples.

Derived Attribute, d: An attribute that combines two or more base attributes using the algebra $\mathcal{L}$. Using Example 2.1, we can create a derived attribute *length of stay* by subtracting discharge date from the admission date. Another derived attribute *obesity* can be obtained through the relational operator > as $BMI > 30$.

Independent variable, dependent variable: A base attribute or a derived attribute is an independent variable (V) in our problem as that is used to predict the target variable Z, which is the dependent variable. Our overall intention is to craft a set of derived attributes as independent variables that are highly "predictive" to the target (dependent) variable.

Mutual Information (MI): We are interested in calculating "predictiveness" of an independent variable V to the target variable Z. For that, we use Mutual Information (MI) that captures information theoretic "correlation" (indeed there exists a relationship between MI and correlation [25] between two random variables that quantifies the amount of information obtained about one through the other). When V and Z are discrete [5], $MI(Z, V)$ is defined as follows:

$$MI(Z,V) = \sum_{z \in Z} \sum_{v \in V} p(z,v) \log \frac{p(z,v)}{p(z)p(v)} \quad (1)$$

where $p(z, v)$ is the joint probability function of Z and V, and $p(z)$ and $p(v)$ are the marginal probability distribution functions of Z and V respectively. Of course, V could be a single base attribute, a small set of base attributes, or a derived attribute.

2.2 Proposed Framework

Our proposed framework consists of two technical steps. We first design a set of k buckets, each with at most x base attributes. Given a bucket and the algebra $\mathcal{L}$, the analyst then starts composing a small number of derived attributes from the bucket. This next step is referred to as *snippet generation*, and works iteratively with the analyst until she decides to stop. In each step in snippet generation, the analyst extends the currently composed derived attributes by a small amount. Our algorithmic contributions are focused on these two steps. In Section 3, we analyze the Top-k buckets design problem and describe our solutions. In Section 4, we present our solutions for the Top-l snippets generation problem.

Definition 2.2. **Score of a bucket,** $sc(b)$ **:** For a bucket b with $A_1, A_2, \ldots A_x$ base attributes, the score of b, i.e., $sc(b)$ is the MI between Z and the Cartesian product of the base attributes that are part of b.

$$sc(b) = MI(Z, [A_1 \times A_2 \times A_3 .... \times A_x]) \quad (2)$$

Using Example 2.1, $sc(gender, senior) = MI(readmission, [gender \times senior])$

Definition 2.3. **Coverage of a set of buckets:** coverage of a set of buckets $Cov(b_1, b_2, ..b_k)$ is the size of the union of the base attributes that are present in these buckets.

$$Cov(b_1, b_2, ..b_k) = |b_1 \cup b_2 \cup ....b_k| \quad (3)$$

Using Example 2.1, if $k = 2, x = 2$, $b_1 = [gender, senior]$, $b_2 = [medicaid, gender]$, then $Cov(b_1, b_2) = 3$.

Definition 2.4. **Snippet,** s**:** A snippet s is a visual representation of a joint distribution between Z and a derived attribute d^s in the snippet.

Figure 1 shows one such snippet between obesity and hospital readmission.

[5]We consider the numeric variables are appropriately discretized, when needed

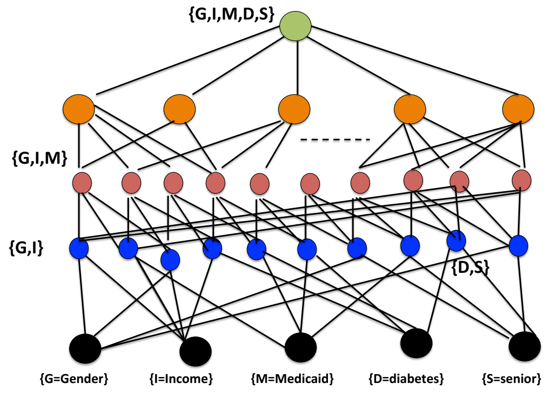

Figure 2: Attribute Lattice

Definition 2.5. **Score of Snippet,** $Sc(s)$**:** Score of a snippet is the MI between Z and derived attribute d^s in the snippet s, i.e., $MI(Z, d^s)$.

Using Figure 1, $Sc(obesity) = MI(readmission, obesity)$.

2.3 Problem Definitions

Problem 1: Top-k-Buckets Design: Given a set of attributes A, number of required buckets k and maximum number of attributes in each bucket x, our objective here is to design Top-k buckets based on MI, i.e., finding the k-buckets (each with at most x attributes) with the highest MI and present those buckets to the the human analyst to investigate further for creating derived attributes.

We also investigate the top-k coverage aware bucket generation problem, where the objective is to create "high quality" buckets *that together cover* all base attributes that have high MI with the target variable.

Problem 2: Coverage Aware Top-k buckets: The objective is to create Top-k buckets such that the score of each bucket, $sc(b)$ in Top-k is above a certain threshold δ and $Cov(b_1, b_2, ..b_k)$ is maximized.

Problem 3: Top-l Interactive Snippets Generation: Each step of the interactive snippet generation takes as inputs a bucket b, an integer j, the currently composed set of derived attributes $\mathcal{D}'$, the algebra $\mathcal{L}$; and produces l snippets with highest scores, where each derived attribute d_i'' in snippet s_i is created by extending d_i', that is, by adding j additional attributes that are part of b, involving $\mathcal{L}$.

3 TOP-k BUCKETS DESIGN

Top-k Buckets Design takes x (the maximum size of each bucket), k (the number of buckets), the dataset (base attributes A, target variable Z, and the records), and produces Top-k buckets, each of size at most x that are highest in MI with Z.

THEOREM 3.1. *The Top-k Buckets Design Problem is NP-hard, for an arbitrary x and k.*

PROOF. (Sketch): As shown in [46], for an arbitrary maximum bucket size x and number of buckets k, the problem of identifying the number of distinct buckets of maximum size x (that have maximal MI in our case) is #P-Hard. Therefore, the enumeration problem of finding top-k buckets of maximum size x is NP-hard. □

Intuitively, the bucket design problem bears similarity with the itemset mining problems based on association rules (support) or other correlation measures [1], such as Chi-Square [6]. WRT our problem, a base attribute could be considered an item, and therefore mining a bucket with at most x-base attributes is akin to mining an itemset with at most x items that satisfy a certain property. Thereby, the Top-k buckets generation problem seemingly appears similar to Top-k itemset mining problems.

Most popular and effective algorithms in this problem space make use of the *upward or downward closure property of the itemsets* based on the underlying measures (for example, support is downward closed, while chi-square is upward closed [6]). These properties enable efficient algorithm design for the itemset mining problems.

Popular algorithms such as Apriori [1] make use of this property extensively in mining frequent itemsets. They execute in a bottom-up manner by discarding any itemset from consideration whose subsets are not frequent based on the support threshold [1].

However, designing an Apriori [1] type of algorithm is impractical for our problem for several reasons: firstly, because *MI is not downward closed;* secondly, because our problem does not have a *support threshold* like Apriori, and finally because our problem has *more constraints* (x and k) as inputs.

LEMMA 3.2. *Mutual Information is upward closed.*

PROOF. Without loss of generality, let us assume Z is the target variable and A_1, A_2, A_3 are three base features. We have to prove $MI(Z, [A_1, A_2]) \leq MI(Z, [A_1, A_2, A_3])$.

Based on MI definition [26]:

$$MI(Z, [A_1, A_2]) = H(Z) - H(Z|[A_1, A_2]) \tag{4}$$

where $H(Z|[A_1, A_2])$ is the conditional entropy of Z given A_1, A_2.

Using Equation 4, we therefore prove $H(Z|[A_1, A_2]) \geq H(Z|[A_1, A_2, A_3])$.

$H(Z|[A_1, A_2, A_3])$

$$\begin{aligned}
&= \Sigma_{a_1,a_2,a_3} p(a_1, a_2, a_3) \times \\
&\Sigma_z p(z|a_1, a_2, a_3) \log p(z|a_1, a_2, a_3) \\
&\leq \Sigma_{a_1,a_2,a_3} p(a_1, a_2) \times p(a_3) \Sigma_z p(z|a_1, a_2) \times \\
&p(z|a_3) \log p(z|a_1, a_2) \log p(z|a_3) \\
&\leq H(Z|[A_1, A_2]) \times \Sigma_{a_3} p(a_3) \Sigma_z p(z|a_3) logp(z|a_3) \\
&\leq H(Z|[A_1, A_2])
\end{aligned} \tag{5}$$

Therefore, $H(Z|[A_1, A_2]) \geq H(Z|[A_1, A_2, A_3])$, proving $MI(Z, [A_1, A_2]) \leq MI(Z, [A_1, A_2, A_3])$. □

An Apriori-Like Algorithm is Impractical. Since MI is upward closed, a level-by-level algorithm such as Apriori must be done in a top-down fashion for our problem, as opposed to the typical bottom-up fashion. Our problem has an additional constraint on the maximum bucket size that requires the algorithm to climb to level x first before it can start generating possible buckets. Moreover, our bucket generation problem does not come with any provided threshold of MI. Instead, it has the number of buckets constraint k. Therefore, it has to start at level x (which will have $\binom{n}{x}$ number of buckets of size x) with the highest MI value possible as the threshold and determine all the size x buckets that satisfy the threshold. If the number of generated buckets is less than k, it then has to reduce the MI threshold value systematically until a total of exactly k buckets have been generated. Naturally, to faithfully reproduce Apriori for our problem, one has to make several runs of the algorithm, until exactly k buckets have been produced. Clearly, such a process is computationally impractical for large n, x, and k.

Next, we describe our solutions for this problem - first an exact algorithm (ExBKT) that produces the top-k buckets of at most size x with the highest MI, and then random walk based algorithms that are highly efficient and work well in practice.

3.1 Exact Algorithm

Algorithm ExBKT extensively exploits a few observations that we make about MI. In particular, while computing the MI of a set of attributes with the target variable Z, we observe that one can produce effective *lower and upper bounds* on MI. These bounds shall allow us to design a bottom up algorithm that goes level by level and effectively prunes a set of candidate attributes (buckets) by using upper bounds. The observation is rather simple - between two subsets of candidate buckets, if one has higher lower bound of MI than the other bucket's upper bound of MI, then the former bucket should get promoted as the winner between these two buckets. Algorithm ExBKT makes use of this observation to prune buckets that are never going to be part of the top-k results. Before we describe this algorithm in detail, we describe how to effectively compute lower and upper bound (LB and UB respectively) of MI of a set of attributes (candidate buckets).

THEOREM 3.3. *Upper Bound: Given a set of t base attributes $A_1, A_2, \ldots A_t$ and the target variable Z, $MI(Z, [A_1, A_2, \ldots A_t]) \leq H(Z) + H(A_1, A_2, \ldots A_t)$*

PROOF.

$$\begin{aligned} MI(Z, [A_1, A_2, \ldots A_t]) &= H(Z) - H(Z|[A_1, A_2, \ldots A_t]) \\ &= H(Z) - H(Z, [A_1, A_2, \ldots A_t]) + \\ H(A_1, A_2, \ldots A_t) &\text{ since } [H(Y|X) = H(X, Y) - H(X)] \end{aligned} \quad (6)$$

Therefore,

$$MI(Z, [A_1, A_2, \ldots A_t]) \leq H(Z) + H(A_1, A_2, \ldots A_t) \quad (7)$$

Applying the chain rule of entropy, the right hand side of the inequality could be expressed further as $H(Z) + H(A_1) + H(A_2|A_1) + H(A3|A_1A_2) + \ldots H(A_t|A_1, A_2, \ldots A_{t-1})$ □

THEOREM 3.4. *Lower Bound: Given a set of t base attributes $A_1, A_2, \ldots A_t$ and the target variable Z, $MI(Z, [A_1, A_2, \ldots A_t]) \geq MI(Z, [A_1, A_2, \ldots A_{t-1}])$*

PROOF. This comes directly from lemma 3.2. □

Algorithm Description: Algorithm ExBKT runs in a bottom-up fashion. It starts at the bottom of attribute lattice with singleton base attributes as buckets, and gradually walks up the lattice, level by level. For illustration purposes, consider only 5 attributes from Example 2.1, abbreviated $\{G, I, M, D, S\}$. The bottom layer refers to the 5 singleton base attributes and their computed MI wrt the target variable Z. Once the score of each of size 1 bucket is computed, then in the next level, the algorithm combines two base attributes and computes 10 buckets of size 2. For each bucket b_i, it maintains two scores: the lower bound score of b_i, $sc^{lb}(b_i)$ derived from Theorem 3.4 and the upper bound score of b_i, $sc^{ub}(b_i)$ derived from Theorem 3.3. Between two buckets b_i and b_j, if the upper bound of b_i (i.e., $sc^{ub}(b_i)$) is smaller than the lower bound of score of b_j (i.e., $sc^{lb}(b_j)$), then b_i and all of its supersets get dropped from further consideration. Using Figure 2, if $sc^{lb}\{G, I\}$ is larger than the upper bound score $sc^{ub}\{M, D\}$, then the latter bucket gets dropped from further consideration. Additionally, all buckets that contain $\{M, D\}$ as some base attributes do not need to be considered. If $x = 3$, then the algorithm continues to climb up the attribute lattice to the next level, finding buckets with at most size 3 base attributes, but applies pruning between the buckets using the lower and upper bound as before. Once it finishes the traversal, it produces the top-k buckets with the k-highest MI with the target variable. The pseudo-code is described in Algorithm 1.

Algorithm 1 Algorithm ExBKT

```
inputs: $\mathcal{A}$, $m$ records, $x$, $k$, target variable $Z$.
output: Top-$k$ buckets, each of size at most $x$.
compute all buckets of size 1 and their score.
$i = 2$
while   $i \leq x$ do
    Compute $sc^{ub}(b)$ and $sc^{lb}(b)$ of each bucket $b$;
    IF $sc^{ub}(b_w) \leq sc^{lb}(b_y)$
        Drop $b_w$ and any superset of $b_w$
    $i = i + 1$
end while
```

LEMMA 3.5. *Algorithm ExBKT produces the exact top-k buckets.*

PROOF. (sketch:) The intuitive argument is that ExBKT only drops a bucket and its super-set if its upper bound score is not larger than the lower bound score of other buckets, which will produce the exact solution, because of Lemma 3.2. □

Running Time: In the worst case ExBKT may take $O(n^x)$, hence it is exponential. However, in practice, the pruning can be very effective and the algorithm converges much sooner.

3.2 Random Walk Based Algorithms

While ExBKT works well in practice, it has an exponential running time in the worst case. Furthermore, ExBKT is not easily extensible when we have to optimize any other criteria in addition to MI. Specifically, for our proposed problem *Coverage Aware Top-k buckets*, ExBKT can not be adapted to generate the best k-buckets with the highest coverage. Therefore, we propose faster heuristic alternatives that provide good solutions most of the time.

Our proposed Algorithm RandomizedBKT is motivated by random walk on the attribute lattice [6] and has a unified solution for both *Top-k buckets Design* and *Coverage Aware Top-k buckets Design* problems. The core idea is to compose a bucket of size x, by performing a random walk on the attribute lattice (refer to Figure 2). An attribute is added to a bucket by using weighted sampling without replacement. The weight of an attribute is based on the optimization criteria. A bucket is formed by performing a random walk on the attribute lattice, until its maximum size x has been reached. This random walk is repeated, until k-unique buckets have been derived and any additional criteria (e.g., threshold δ for the latter problem) has been satisfied. With this high level description above, now we describe how to compute the weight of an attribute in different scenarios. For the *Top-k buckets Design* problem, the weight of an item is directly proportional to its MI, i.e., $weight(A) = MI(Z, A)$.

The *Coverage Aware Top-k buckets Design* problem requires that in addition to MI threshold δ, the coverage of the Top-k buckets also must be maximized. The weight of an attribute is a ratio; it is proportional to its MI, but inversely proportional to the number of times it is present in other buckets that have been computed thus far. Indeed, this latter criteria is designed to ensure high coverage.This condition can be extended to include feature cost as well, so that costly features are assigned lower preference. As before, the algorithm terminates when we obtain top-k unique buckets. Using Figure 2, if $k = 5$ and $x = 2$, when the random walk produces the 5th bucket, if M has appeared in all other 4 buckets, the weight of M is dampened by a factor of 4 even if it is high in MI with readmission. The pseudo-code is described in Algorithm 2.

Algorithm 2 Algorithm RandomizedBKT

```
inputs: set of base attributes $\mathcal{A}$, $m$ records, target variable $Z$, $x$, $k$, $\delta$ (for the coverage aware problem)
output: Top-$k$ buckets of size at most $x$
$\mathcal{B} = \{\}$
while $|\mathcal{B}| < k$ do
    while $size(b_i) < x$ do
        $weight(A) = MI(Z, A)$    ▷ for the top-$k$ variant
        $weight(A) \propto \frac{MI(Z,A)}{\#A \text{ has been used before}}$    ▷ for the Coverage aware variant
        Sample $A$ based on $weight(A)$ and add it to $b_i$
    end while
    IF $sc(b) < \delta$    ▷ only for the Coverage aware variant
        Drop $b$
    $\mathcal{B} = \mathcal{B} \cup b$
end while
```

Running Time: Each run of RandomizedBKT takes at most $O(n)$ time. The total running time of the algorithm is dominated by the number of different random walks that needs to be performed before it returns k buckets.

4 TOP-l INTERACTIVE SNIPPETS GENERATION

Recall that we propose snippet generation as an interactive process that continues until the human is done crafting the attributes. Each step of this process takes as inputs a bucket b, an integer j, the currently composed set of derived attributes $\mathcal{D}'$, and the algebra $\mathcal{L}$ to produce l snippets with highest scores (MI with Z), where each derived attribute d_i'' in snippet s_i is created by extending d_i' by adding j additional attributes that are part of b, involving $\mathcal{L}$. Each of these l snippets are recommended to the human as visual distributions to aid her design derived attributes.

Therefore, our computational challenge is to produce l snippets effectively in each step involving each $d_i' \in \mathcal{D}'$. A natural choice is to investigate a greedy algorithm that builds the final derived attribute bottom-up by selecting the best base attributes from the bucket and combining them with the best operators to recommend a snippet. In order for this greedy algorithm to provide any provable guarantee, this greedy selection process needs to satisfy certain properties. For example, given four attributes A_i, A_j, A_r, A_s, if $MI(Z, A_i) > MI(Z, A_r)$, and $MI(Z, A_j) > MI(Z, A_s)$, then it seems intuitive that if we combine the two attributes with higher MI (A_i, A_j) with an operator, the resulting derived attribute should have higher MI than the combination of the other two attributes (A_r, A_s) using the same operator. Unfortunately, as we prove below with counter examples, such a property fails to hold *even for very simple operators such as arithmetic addition.* This makes the snippet generation more challenging than the previous step of bucket generation.

A_i	A_j	A_r	A_s	Z
1	-1	0	0	0
2	-2	1	0	1
3	-3	0	1	0
4	-4	1	1	1

Table 2: Example where $MI(Z, A_i + A_j) < MI(Z, A_r + A_s)$

LEMMA 4.1. *Given four attributes A_i, A_j, A_r, A_s, if $MI(Z, A_i) > MI(Z, A_r)$, and $MI(Z, A_j) > MI(Z, A_s)$, $MI(Z, A_i + A_j)$ may or may not be greater than $MI(Z, A_r + A_s)$*

PROOF. (Sketch): We prove this by counter examples. We present two examples - one shows $MI(Z, A_i + A_j) < MI(Z, A_r + A_s)$ and the other shows the inequality other way. Table 2 shows the first scenario. To create the second scenario, if we replace A_j with $5, 6, 7, 8$ in the 4 different rows respectively, we have $MI(Z, A_i + A_j) > MI(Z, A_r + A_s)$. □

Proposed Algorithm: Our proposed algorithm GSnippet is a greedy algorithm - it still generates a snippet bottom up in each step. Given a bucket b, it first sorts the base attributes in the bucket in descending MI. Then given j (the number of additional attributes to extend a partially created derived attribute d_i'), it finds the top-j attributes in b that are not in d_i'. These are the candidate set of attributes for the snippets that involve d_i'. After that, it attempts to combine these j attributes considering all the operators in $\mathcal{L}$ with d_i'. It ranks each of these created combinations and produces the l-combinations as snippets that have the top-l MI score with Z. Algorithm 3 presents the pseudo-code.

The reason that GSnippet exhaustively considers all the operators in $\mathcal{L}$ (and cannot make any greedy or more efficient look-ups) is because of Lemma 4.1. In order to avoid exhaustive search, other alternative metrics to MI need to be explored that can guarantee upper/lower bound for features combined under algebraic operators. Nevertheless, as we shall show in our experiment, GSnippet runs at interactive speeds and produces effective recommendations for the human analyst.

Using Example 2.1, if $b = \{admission.date, dis.date\}$, $l = 1$, $j = 2$, $\mathcal{L} = \{+, -, \times, /\}$, and current $d = \{\}$ (i.e., empty), then GSnippet will rank $admission.date + dis.date$, $admission.date -$

dis.date, *admission.date* × *dis.date*, *admission.date*/*dis.date*, and take the one which has the highest MI with readmission.

Algorithm 3 Algorithm `GSnippet`

```
inputs: bucket $b$, a derived attribute $d'_i$, target variable $Z$, $j$, $l$, $\mathcal{L}$
output: Top-$l$ snippets involving $d'_i$
Sort attributes in $b$ in descending order of MI
Select set $S$ with top-$j$ attributes from $b$ that are not in $d'_i$
$C = S \bigcup d'_i$
Combine $c_i \oplus c_j \oplus .... \oplus d'_i$, $\oplus \in \mathcal{L}$, $c_i \in C$
Rank each combination wrt $MI$
Return top-$l$
```

Running Time: Each run takes $O(|\mathcal{L}|^j logl)$ time, the majority of which is spent on brute-forcing on the operator set $\mathcal{L}$.

5 EXPERIMENTAL EVALUATION

We conducted comprehensive experimental analysis to compare our proposed approach with fully automated methods, as well as fully manual (domain expert guided) solution. We investigated both quality and running time in this process.

5.1 Experimental Setup

Hardware and Platform. All our experiments were conducted on a quad-core 2.2 GHz machine with 16 GB of RAM and 1 TB of hard disk. We used Python to implement our algorithms and used scikit-learn for building the classification models.

Datasets. We evaluated our algorithms against a wide variety of datasets that are considered to be popular choice for attribute design problems. Due to lack of space, we report our results on 5 datasets from UCI repository and one from Kaggle. They cover a diverse array of domains (agriculture, medicine, and e-commerce) and contain attributes that are amenable to constructing derived features. Table 3 has more details.

DataSet	# Records	# raw attributes
pollen	3848	5
delta_elevators	9517	6
mammography	11183	6
space	3107	6
diabetes	768	8
home	506	14

Table 3: Reported datasets characteristics

Compared Methods. *(1) Fully Automated Methods.* We implemented two fully automated state-of-the-art approaches for comparison - Featuretools[6] and ExploreKit[7]. They both have open source repositories that allow us to evaluate them fairly.

(2) Our proposed algorithms. These are the solutions that are presented in Sections 3 and 4.

[6]https://docs.featuretools.com/#minute-quick-start
[7]https://github.com/giladkatz/ExploreKit

(3) Buckets Design Baseline algorithms. We compared our proposed buckets design algorithm against an intuitive baseline algorithm (referred to as `Greedy`) that groups attributes on mutual information. It started with an empty bucket and x attributes were added through importance sampling greedily, where the importance is proportional to the marginal increase in MI. The process was repeated k times.

(4) Fully manual method. In this scenario, a domain expert was involved in crafting the derived attributes. We present a case study towards that in Section 5.5.

Evaluation of our proposed framework. Our proposed framework has two steps: the first one, top-k buckets design, is fully automated and does not require any human involvement. We have described three algorithms for that. The second step, top-l snippets generation, proposes the list of top l snippets for each of the derived attribute chosen by the human analyst. The human analyst can either accept the choice, select another snippet or even construct a new one. This process is repeated interactively. The framework continues to retain the top-l choices, if no human guidance is given and is fast enough to be done in near real-time. Finally, the designed derived and the base attributes are passed through existing popular classification models with an average of 10 runs.

Parameter Settings. *x, k, j, l, random walk:* We varied the size of a bucket x between 2 to the maximum number of base attribute with the default value being 5. The snippet extension parameter j is set to 1. Finally, both k and l were set to 5 by default. We ran the random walk for 1000 iterations and picked top-k from it.

Classification models. While our process is classifier agnostic, for the purpose of comparison, we considered two popular classifiers, Support Vector Machines (SVM) and Random Forests (RF). For the latter, we use 100 trees using a depth of 2. Training and testing are performed with a 70% − 30% split of the data, akin to ExploreKit.

Algebra. By default, we used only arithmetic operators. We then varied the grammar to include logical, relational and other aggregate operators. Overall, our approach can support all the operators described in both ExploreKit and Featuretools.

Performance Measures. *Qualitative measures.* For measuring the classifier performance, we reported the Area Under the Curve (AUC). This has been used in prior work such as ExploreKit as it provides a holistic view of the classifier and attribute design. Higher the AUC, better the classifier performance. We reported the percentage improvement in AUC with base(raw) attributes and base+derived attributes. For example, if the classifier had an AUC of 0.7 with base attributes but 0.8 with base+derived attributes, the improvement is $\frac{0.8-0.7}{0.7} = 14\%$.

Scalability measure. We used time in seconds to evaluate the efficiency of our algorithms.

5.2 Summary of Results

Our experimental analysis answered the following key questions:

- **How did our proposed framework compare against fully automated or fully manual solutions?** We found that we scale significantly better ($10x - 20x$ faster than ExploreKit, $7x$ better than fully manual) than both of these extremes, with comparable AUC improvements (on an average 14%). Due to human involvement, it also avoided the worst case

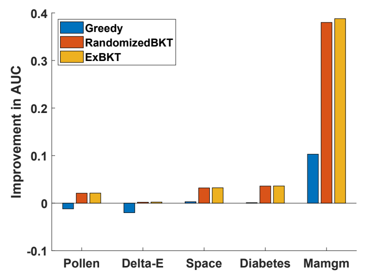

Figure 3: Comparing Baselines : AUC

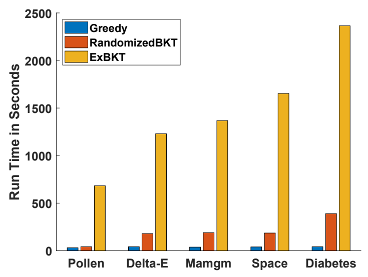

Figure 4: Comparing Baselines : Running Time

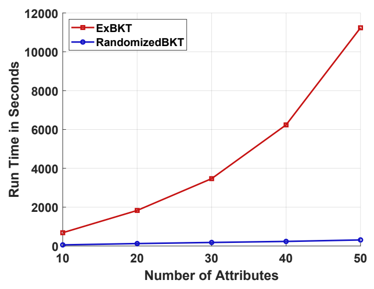

Figure 5: Varying Number of Attributes

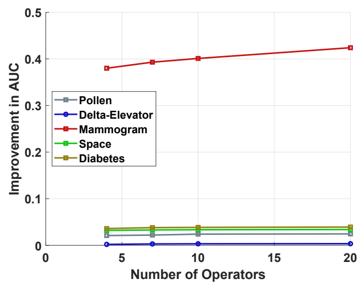

Figure 6: Varying Algebra

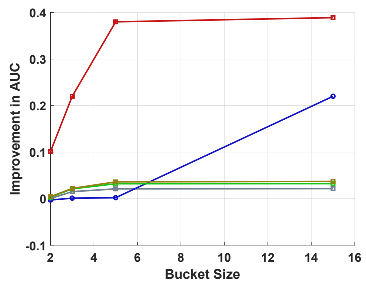

Figure 7: Varying Bucket Size

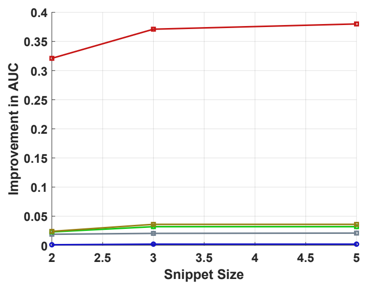

Figure 8: Varying Snippet Size

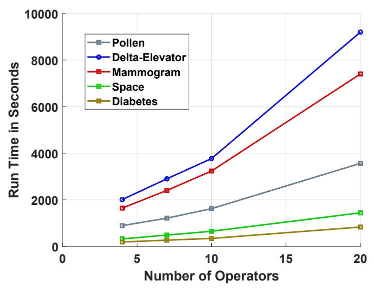

Figure 9: Varying Algebra

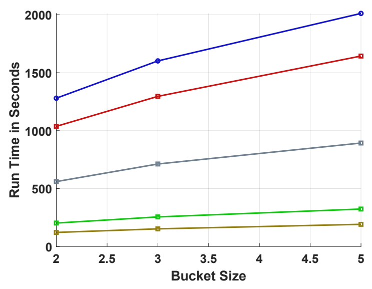

Figure 10: Varying Bucket Size

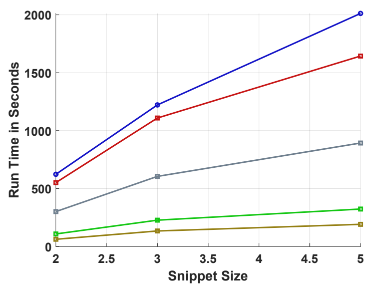

Figure 11: Varying Snippet Size

Data	EK-SVM	Time (s)	FT-SVM	Time (s)	MI-SVM	Time (s)
pollen	1.93%	1063	2.07%	93	2.10%	42
delta_elevators	0.23%	3980	-3.20%	465	0.20%	180
mammography	31.61%	2413	41.93%	204	38.00%	191
space	3.83%	986	1.77%	135	3.20%	186
diabetes	4.63%	343	-13.64%	468	3.60%	390

Table 4: AUC improvement and runtime comparison with fully automated methods using SVM

behavior of the automated methods (FeatureTools showed decrease in performance at times). Sections 5.3, 5.5 present these results.

- **How did our proposed algorithms compare?** We observed that we attained higher qualitative performance, while being scalable compared to other baselines. The exact algorithm `ExBKT` produced optimal buckets, while `RandomizedBKT` was much faster and produced results comparable to `ExBKT`. `Greedy` was inferior in quality. We focused on the coverage variant of `RandomizedBKT` so as to provide a diverse set of buckets

Data	EK-RF	Time (s)	FT-RF	Time (s)	MI-RF (s)	Time
pollen	10.56%	10233	-3.71%	40	-1.00%	42
delta_elevators	0.75%	37611	-1.13%	106	0.70%	180
mammography	-0.01%	16502	5.34%	113	4.00%	191
space	4.09%	10676	4.92%	106	4.20%	186
diabetes	4.46%	3474	3.11%	462	3.30%	390

Table 5: AUC improvement and runtime comparison with fully automated methods using RandomForest

to the human analyst. Algorithm GSnippet was interactive to the human analyst and works in real time. Section 5.4 presents these results.

- **What kind of attributes did our proposed framework generate?** We observed that the attributes produced by our framework were intuitive and meaningful to human analyst. Section 5.5 presents some of these results.

5.3 Comparison with fully automated methods

We compared the performance of our proposed approach against automated systems in both classifier performance and efficiency. Tables 4 and 5 show the results for SVM and RF classifiers respectively. On an average, the AUC improvement of the classifiers were more than 14% through attribute design. Our proposed framework was almost 10x-20x faster than ExploreKit, comparable with FeatureTools (but FeatureTools caused sudden decline in AUC at times with the derived attributes).

5.4 Analysis of presented algorithms

In this section, we present quality and running time study of our algorithms with the baselines, described in Section 5.1. We note that the implemented baselines were inferior in AUC improvements, hence we only present running time of our solutions.

Comparison with Baselines. We began by comparing our algorithms for bucket construction : an exact algorithm ExBKT, a random walk based algorithm handling coverage RandomizedBKT and Greedy. Figures 3 and 4 show the results. As expected, ExBKT took substantial amount of time but gave the best results. RandomizedBKT was much faster and provided almost identical results for AUC improvement. Greedy was very efficient but qualitatively inferior.

Varying Number of Operators in the Algebra. We began by supporting arithmetic operators ($+, -, \times, /, \%$) and then systematically added more logical, relational and aggregate operators. As expected, increasing the number of operators causes a slow down in our approach. Recall that our first stage of bucket generation was agnostic to both classifier and grammar. The resulting buckets often contained only a handful of attributes and hence our algorithm was quite fast even for a large number of operators. This can be seen in Figure 9 where the improvement is (sub)-linear in the number of operators. Figure 6 shows the corresponding impact on AUC. This shows that the additional operators often only provides negligible improvement in AUC. We conjuncture that for most attributes, only a small set of operators are most relevant.

Varying Bucket Size x. Next, we varied the maximum number of attributes x in each bucket. The impact of k (the number of buckets to return) was minimal. Our algorithms have a natural anytime property where it can be stopped at any time and pick the best k buckets. x affected the length of the random walk and had a major impact on runtime and AUC. Figures 7 and 10 show the result. As expected, increasing bucket size improved the AUC but it stagnated quickly. This confirmed with our hypothesis that most derived attributes often consist of few base attributes. The bucket size must not be too small - otherwise, we might miss meaningful group of attributes. It must also not be too large - otherwise, we might expend runtime for no meaningful improvement of AUC. We found a value of 5 to be good both from runtime perspective and the required cognitive impact on the human analyst. The runtime increased almost linearly with larger bucket size.

Varying Snippet Size j. In our final set of experiments, we varied the snippet size. Here we also noted that the impact of l (the number of snippets to return) was minimal. Figures 8 and 11 show the results. As expected, increasing snippet size had minimal impact on AUC. The improvement more or less topped out at snippet of size 3. This once again confirmed our hypothesis that derived attributes were often constructed from a handful of base attributes. The increase in time for larger snippet size was mostly linear.

Varying Number of Attributes. In order to highlight the scalability of our algorithms, we varied the number of base attributes by duplicating some of the base attributes randomly. Figure 5 shows the impact on running time of our exact and random walk based algorithm. As expected, the running time increased dramatically for ExBKT while the increase was marginal for RandomizedBKT. Our approach was scalable to large increase in the number of columns. The impact of increasing number of rows were rather minimal.

5.5 Comparison with fully manual method

We present a case study comparing a fully manual approach (a typical trial-and-error based attribute design exercise a data scientists goes through) and our approach. We used the popular Boston dataset from Kaggle [8] that seeks to predict house prices from from features like its area, number of bedrooms, etc.

We involved a data scientist to construct the derived attributes and she came up with the following 6 additional derived attributes after extensive analysis. lstat / tax, ptratio / tax, dis * rad / tax, black * crim, rm * (zn + indus) and (nox * indus) / tax. The interpretation of these attributes are presented in Table 6. The data scientist took about 10 hours to manually craft these 6 derived attributes, out of which 3 hours were needed to understand and explore the data. The remaining 7 hours were needed for a trial and error process, where the expert tried many possible derived attributes, analyzed the correlation, tweaked the attributes, and repeated the process. While another data scientist with similar expertise was guided by

[8]https://www.kaggle.com/c/boston-housing/data

Attribute	Interpretation
lstat	lower status of the population (percent).
tax	full-value property-tax rate per \$10,000.
ptratio	pupil-teacher ratio by town.
dis	weighted mean of distances to five Boston employment centres.
rad	index of accessibility to radial highways.
black	$1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town.
crim	per capita crime rate by town.
rm	average number of rooms per dwelling.
zn	proportion of non-retail business acres per town.
nox	nitrogen oxides concentration (parts per 10 million).

Table 6: Boston home dataset attributes

our framework, she took *only* 1 *hour* to craft the same 6 derived attributes. For both cases, we observed an AUC improvement of 14%. This case study anecdotally showcase that our proposed framework is capable to drastically reduce the latency (7 hour vs 1 hr, i.e, 7x improvement) by aiding the domain expert.

6 RELATED WORK

Human-in-the-loop Query Answering : A growing number of systems make use of lay workers (known as crowdsourced workers) using commercial platforms (e.g., AMT and CrowdFlower) or for academic use. Examples of applications include not only sentence translation, photo tagging and sentiment analysis, but also query answering (CrowdDB [11], Qurk [28], Deco [32], sCOOP, FusionCOMP, MoDaS, CyLog/Crowd4U), entity resolution (such as CrowdER [40]), planning queries [19], perform matching [41], or counting [27]. A series of works [39–43] have proposed variations of crowdsourced join to address entity resolution task for query answering. Authors in [7, 9, 13, 21, 33, 47] have proposed various techniques of performing crowdsourced version of top-k item selection. Crowdsourcing based filtering has been extensively introduced in [30, 31]. Crowdsourced *find* focuses on selecting one or more qualified items [34, 44]. Unlike these works, we involve humans for attributes design, which occurs at a later stage of the data science pipeline — therefore these prior works do not extend to our problem.

Human-in-the-loop Supervised Modeling: Machine learning literature has involved humans to obtain labels [5, 15, 16, 37] primarily for the classification problem. Active learning involving humans (expert as well as lay workers) has also been discussed in recent works such as [10, 29, 45, 49]. These works utilize humans for accurately predicting class labels, thus the humans function as an added module to help the algorithm steer toward the correct assignment of class labels. The main distinction between these works and ours is that, these works leverage human mainly for data labeling and not for attribute design.

Attribute Design: How to develop automated methods for designing/engineering attributes (commonly referred to as as feature engineering) has been discussed in recent works [2, 18, 20, 22, 23, 48], primarily in machine learning literature. Attribute engineering tools, such as, ExploreKit, Featuretools, Data Science Machine, One Button Machine [17, 18, 20, 23, 24, 35] rely on fully automated approaches. Typically they take longer to run and do not offer adequate explainability or intuition, as the discovered attributes remain opaque to the human analyst interested in broader ad-hoc data exploration. They also ignore the availability of humans to guide their exploration. Finally, the effectiveness of most of these tools depend on the underlying classification model, whereas, we present a model agnostic approach. Nevertheless, we use some of these tools in our experimental analysis for comparison purposes. A very few recent works [8, 36, 50] propose a conceptual framework and describe how to involve humans for designing attributes to improve the accuracy of predictive machine learning models, such as classifiers. While we borrow inspiration from these recent works [8, 36, 50], we present the framework with mathematical rigor and investigate optimization opportunities.

7 CONCLUSION

We present an optimization guided semi-automated model-agnostic "human-in-the-loop" framework for designing derived attributes by leveraging humans. The main contribution of our work is to provide an optimization guided framework for data scientists that will help them to craft meaningful derived attributes much quicker than a fully manual method. On the other hand, running time of the fully automated techniques vary greatly; for example, FeatureTools is significantly faster compared to ExploreKit, but they all have the limitations of producing derived attributes that are opaque. Contrarily, the derived attributes produced by our framework are interpretable.

We present two computational problems inside our proposed framework - *top-k buckets design* and *top-l snippets generation* problems. We present effective solutions for solving both problems. We compare our proposed approach with two fully automated state-of-the-art tools, as well as fully manual domain expert designed derived attributes. Our rigorous quality evaluations using 6 real world datasets demonstrate that we are as effective as *fully automated methods* and we scale up significantly better compared to a fully manual solution involving two domain experts. Our scalability results demonstrate that both bucket design and snippet generation are interactive and ensure real time response with the analysts. As an ongoing problem, we are investigating how to revisit these problems when the datasets contain significant missing values. We are also investigating how to involve multiple domain experts in buckets design and snippets generation problems.

ACKNOWLEDGEMENTS

The work of Senjuti Basu Roy is supported by the National Science Foundation under Grant No.: 1814595 and Office of Naval Research under Grant No.: N000141812838. The work of Gautam Das is supported in part by grant W911NF-15-1-0020 from the Army Research Office, grant 1745925 from the National Science Foundation, and a grant from AT&T.

REFERENCES

[1] Rakesh Agarwal, Ramakrishnan Srikant, et al. 1994. Fast algorithms for mining association rules. In *Proc. of the 20th VLDB Conference*. 487–499.

[2] Michael R Anderson, Dolan Antenucci, Victor Bittorf, Matthew Burgess, Michael J Cafarella, Arun Kumar, Feng Niu, Yongjoo Park, Christopher Ré, and Ce Zhang. 2013. Brainwash: A Data System for Feature Engineering.. In *CIDR*.

[3] Michael R Anderson and Michael Cafarella. 2016. Input selection for fast feature engineering. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*. IEEE, 577–588.

[4] Senjuti Basu Roy, Ankur Teredesai, Kiyana Zolfaghar, Rui Liu, David Hazel, Stacey Newman, and Albert Marinez. 2015. Dynamic hierarchical classification for patient risk-of-readmission. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1691–1700.

[5] Jonathan Bragg, Daniel S Weld, et al. 2013. Crowdsourcing multi-label classification for taxonomy creation. In *AAAI Conference on Human Computation and Crowdsourcing*.

[6] Sergey Brin, Rajeev Motwani, and Craig Silverstein. 1997. Beyond market baskets: Generalizing association rules to correlations. In *Acm Sigmod Record*, Vol. 26. ACM, 265–276.

[7] Xi Chen, Paul N Bennett, Kevyn Collins-Thompson, and Eric Horvitz. 2013. Pairwise ranking aggregation in a crowdsourced setting. In *ACM International Conference on Web Search and Data Mining*. ACM, 193–202.

[8] Justin Cheng and Michael S Bernstein. 2015. Flock: Hybrid crowd-machine learning classifiers. In *ACM Conference on Computer Supported Cooperative Work & Social Computing*. ACM, 600–611.

[9] Brian Eriksson. 2013. Learning to top-k search using pairwise comparisons. In *Artificial Intelligence and Statistics*. 265–273.

[10] Meng Fang, Jie Yin, and Dacheng Tao. 2014. Active Learning for Crowdsourcing Using Knowledge Transfer.. In *AAAI*. 1809–1815.

[11] Amber Feng, Michael J. Franklin, Donald Kossmann, Tim Kraska, Samuel Madden, Sukriti Ramesh, Andrew Wang, and Reynold Xin. 2011. CrowdDB: Query Processing with the Crowd. *PVLDB* 4, 12 (2011), 1387–1390.

[12] Benoît Frénay, Gauthier Doquire, and Michel Verleysen. 2013. Is mutual information adequate for feature selection in regression? *Neural Networks* 48 (2013), 1–7.

[13] Stephen Guo, Aditya Parameswaran, and Hector Garcia-Molina. 2012. So who won?: dynamic max discovery with the crowd. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 385–396.

[14] Jeff Heaton. 2016. An empirical analysis of feature engineering for predictive modeling. In *SoutheastCon, 2016*. IEEE, 1–6.

[15] Chien-Ju Ho, Shahin Jabbari, and Jennifer W Vaughan. 2013. Adaptive task assignment for crowdsourced classification. In *International Conference on Machine Learning*. 534–542.

[16] Muhammad Imran, Carlos Castillo, Ji Lucas, Patrick Meier, and Sarah Vieweg. 2014. AIDR: Artificial intelligence for disaster response. In *International Conference on World Wide Web*. ACM, 159–162.

[17] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *ACM International Conference on Multimedia*. ACM, 675–678.

[18] James Max Kanter and Kalyan Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *IEEE International Conference on Data Science and Advanced Analytics*. IEEE, 1–10.

[19] Haim Kaplan, Ilia Lotosh, Tova Milo, and Slava Novgorodov. 2013. Answering Planning Queries with the Crowd. In *PVDLB*.

[20] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. 2016. Explorekit: Automatic feature generation and selection. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 979–984.

[21] Asif R Khan and Hector Garcia-Molina. 2014. *Hybrid strategies for finding the max with the crowd: technical report*. Technical Report. Stanford InfoLab.

[22] Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Parthasrathy. 2016. Cognito: Automated feature engineering for supervised learning. In *IEEE International Conference on Data Mining Workshops*. IEEE, 1304–1307.

[23] Hoang Thanh Lam, Johann-Michael Thiebaut, Mathieu Sinn, Bei Chen, Tiep Mai, and Oznur Alkan. 2017. One button machine for automating feature engineering in relational databases. *arXiv preprint arXiv:1706.00327* (2017).

[24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436.

[25] Wentian Li. 1990. Mutual information functions versus correlation functions. *Journal of statistical physics* 60, 5-6 (1990), 823–837.

[26] Mokshay Madiman. 2008. On the entropy of sums. In *Information Theory Workshop, 2008. ITW'08. IEEE*. IEEE, 303–307.

[27] Adam Marcus, David Karger, Samuel Madden, Robert Miller, and Sewoong Oh. 2012. Counting with the crowd. *Proceedings of the VLDB Endowment* 6, 2, 109–120.

[28] Adam Marcus, Eugene Wu, David Karger, Samuel Madden, and Robert Miller. 2011. Human-powered sorts and joins. *Proceedings of the VLDB Endowment* 5, 1 (2011), 13–24.

[29] Barzan Mozafari, Purna Sarkar, Michael Franklin, Michael Jordan, and Samuel Madden. 2014. Scaling up crowd-sourcing to very large datasets: a case for active learning. *Proceedings of the VLDB Endowment* 8, 2 (2014), 125–136.

[30] Aditya Parameswaran, Stephen Boyd, Hector Garcia-Molina, Ashish Gupta, Neoklis Polyzotis, and Jennifer Widom. 2014. Optimal crowd-powered rating and filtering algorithms. *Proceedings of the VLDB Endowment* 7, 9 (2014), 685–696.

[31] Aditya G Parameswaran, Hector Garcia-Molina, Hyunjung Park, Neoklis Polyzotis, Aditya Ramesh, and Jennifer Widom. 2012. Crowdscreen: Algorithms for filtering data with humans. In *ACM SIGMOD International Conference on Management of Data*. ACM, 361–372.

[32] Hyunjung Park and Jennifer Widom. 2013. Query optimization over crowdsourced data. *Proceedings of the VLDB Endowment* 6, 10, 781–792.

[33] Thomas Pfeiffer, Xi Alice Gao, Yiling Chen, Andrew Mao, and David G Rand. 2012. Adaptive Polling for Information Aggregation.. In *AAAI*.

[34] Anish Das Sarma, Aditya Parameswaran, Hector Garcia-Molina, and Alon Halevy. 2014. Crowd-powered find algorithms. In *IEEE International Conference on Data Engineering*. IEEE, 964–975.

[35] Frank Seide, Gang Li, Xie Chen, and Dong Yu. 2011. Feature engineering in context-dependent deep neural networks for conversational speech transcription. In *IEEE Workshop on Automatic Speech Recognition and Understanding*. IEEE, 24–29.

[36] Micah J Smith, Roy Wedge, and Kalyan Veeramachaneni. 2017. FeatureHub: Towards collaborative data science. In *IEEE International Conference on Data Science and Advanced Analytics*. IEEE, 590–600.

[37] Chong Sun, Narasimhan Rampalli, Frank Yang, and AnHai Doan. 2014. Chimera: Large-scale classification using machine learning, rules, and crowdsourcing. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1529–1540.

[38] Jorge R Vergara and Pablo A Estévez. 2014. A review of feature selection methods based on mutual information. *Neural computing and applications* 24, 1 (2014), 175–186.

[39] Norases Vesdapunt, Kedar Bellare, and Nilesh Dalvi. 2014. Crowdsourcing algorithms for entity resolution. *Proceedings of the VLDB Endowment* 7, 12 (2014), 1071–1082.

[40] Jiannan Wang, Tim Kraska, Michael J Franklin, and Jianhua Feng. 2012. Crowder: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment* 5, 11 (2012), 1483–1494.

[41] Jiannan Wang, Guoliang Li, Tim Kraska, Michael J Franklin, and Jianhua Feng. 2013. Leveraging transitive relations for crowdsourced joins. In *ACM SIGMOD International Conference on Management of Data*. ACM, 229–240.

[42] Sibo Wang, Xiaokui Xiao, and Chun-Hee Lee. 2015. Crowd-based deduplication: An adaptive approach. In *ACM SIGMOD International Conference on Management of Data*. ACM, 1263–1277.

[43] Steven Euijong Whang, Peter Lofgren, and Hector Garcia-Molina. 2013. Question selection for crowd entity resolution. *Proceedings of the VLDB Endowment* 6, 6 (2013), 349–360.

[44] Tingxin Yan, Vikas Kumar, and Deepak Ganesan. 2010. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *International Conference on Mobile Systems, Applications and Services*. ACM, 77–90.

[45] Yan Yan, Romer Rosales, Glenn Fung, and Jennifer G Dy. 2011. Active learning from crowds.. In *International Conference on Machine Learning*, Vol. 11. 1161–1168.

[46] Guizhen Yang. 2004. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 344–353.

[47] Peng Ye and David Doermann. 2013. Combining preference and absolute judgements in a crowd-sourced setting. In *International Conference on Machine Learning*. 1–7.

[48] Ce Zhang, Arun Kumar, and Christopher Ré. 2016. Materialization optimizations for feature selection workloads. *ACM Transactions on Database Systems* 41, 1 (2016), 2.

[49] Jinhong Zhong, Ke Tang, and Zhi-Hua Zhou. 2015. Active Learning from Crowds with Unsure Option.. In *IJCAI*. 1061–1068.

[50] James Y Zou, Kamalika Chaudhuri, and Adam Tauman Kalai. 2015. Crowdsourcing feature discovery via adaptively chosen comparisons. *arXiv preprint arXiv:1504.00064* (2015).