Boosting ML via Data Re-weighting

1. INTRODUCTION

- The use of "found data" is very common.
- A major problem in the use of "found data" is that it may not be adequate to the user's intended use.
  - For instance, underrepresented minorities or skew data that can affect the resulting learned model (which can be reveal, e.g., using our pattern based labels).
- To eliminate this problem the user may wish to adjust the data, by introducing new tuples (for underrepresented groups) or remove tuples (for data skew).
- Ideally, in the case of underrepresented groups, the adjustment of the data is done by collecting more data on those groups. However, this task may be expensive, and even impossible, particularly in the case of "found data".
- To this end, we propose a method for data adjustment, based data re-weighting. The goal of the adjustment process is to boost the performance of learned model.
- To achieve this goal the resulting weighted data should eliminate any underrepresented or skew data, while preserving the original data properties (namely, attributes distribution) as possible.
- We propose the local data difference, a new measure for the difference between the original data and the weighted data to capture the changes made to the data.
- The problem is then to re-weight the data in way that satisfies a set of conditions over the patterns count (representing the underrepresented and skew data the user wish to eliminate) while minimizing the local data difference.
- We prove that this problem is NP-hard, and present a heuristic [[[**(Yuval) or approximation?**]]] solution by reducing the problem to an Integer Programming problem.
- We experimentally show the usefulness of our approach in boosting the performance of learned models using the weighted data compared to the original data.

	Gender	Age group	Race	Marital status	Count
t_1	Female	under 20	Hispanic	single	1
t_2	Male	under 20	Hispanic	single	1
t_3	Male	20-39	Hispanic	single	1
	Female	20-39	Hispanic	married	2
	Male	20-39	Hispanic	married	4
	Female	20-39	Hispanic	divorced	2
	Male	20-39	Hispanic	divorced	3
	Female	under 20	African-American	single	2
	Male	under 20	African-American	single	3
	Female	20-39	African-American	married	2
	Male	20-39	African-American	married	4
	Female	20-39	African-American	divorced	3
	Male	20-39	African-American	divorced	5
	Female	under 20	Caucasian	single	2
	Male	under 20	Caucasian	single	3
	Female	20-39	Caucasian	married	2
	Male	20-39	Caucasian	married	4
	Female	20-39	Caucasian	divorced	2
	Male	20-39	Caucasian	divorced	4

Figure 1: Sample data from a simplified version of the COMPAS dataset

2. PRELIMINARIES

We assume the data is represented using a single relational database, and that the relation's attributes values are categorical. Where attribute values are drawn from a continuous domain, we render them categorical by bucketizing them into ranges: very commonly done in practice to present aggregate results.

DEFINITION 2.1 (PATTERNS). *Let D be a database with attributes $\mathcal{A} = \{A_1, \ldots, A_n\}$ and let $Dom(A_i)$ be the active domain of A_i for $i \in [1..n]$. A* pattern *p over D is set of $\{A_{i_1} = a_1, \ldots, A_{i_k} = a_k\}$ where $\{A_{i_1}, \ldots, A_{i_k}\} \subseteq \mathcal{A}$ and $a_j \in Dom(A_{i_j})$ for each A_{i_j} in p. We use $Attr(p)$ to denote the set of attributes in p.*

EXAMPLE 2.2. *Consider the simplified version of the COMPAS dataset given in Figure 1. It includes only four attributes: gender, age group, race and marital status. The count column depicts the number of instances of the tuple in the data. p =$\{$race= Hispanic, marital status = singe$\}$ is a possible pattern and $Attr(p)$ =$\{$race, marital status$\}$.*

DEFINITION 2.3. *We say that a tuple $t \in D$* satisfies *a pattern p and denote it by $t \vDash p$ if $t.A_i = a_i$ for each A_i appearing in p. The count $c_D(p)$ of a pattern p is the number of tuples in D that satisfy p.*

EXAMPLE 2.4. *Consider again the database given in Figure 1. The tuples annotated as t_1, t_2 and t_2 satisfy the pattern p =$\{$race= Hispanic, marital status = singe$\}$ and thus the count of p is $c_D(p) = 3$.*

Insufficient, skewed and unrepresentative data may lead to bias in data driven models, thus, when using "found data", their avoidance are of great importance. To this end, based on the analyst's desired task, she may states constraints over the count of patterns in the data. We capture the desired data count constraints using data conditions as follows.

DEFINITION 2.5 (CONDITION). *Condition C is an expression of the form*

$$c_D(p) \ \texttt{op} \ x$$

where p is a pattern, $\texttt{op} \in \{<, >\}$ and x can be (i) a constant value in $\mathbb{N}$, (ii) an expression of the form $\alpha \cdot c_D(p')$ for any $\alpha \in \mathbb{R}$ and pattern p'.

We use $C.p$ to denote the pattern p in the condition, and say that a database D satisfies a condition C if the condition holds for D.

Intuitively, conditions of type (i) may be used to declare constraints for the avoidance of insufficient groups representation and skew data, while condition of form (ii) aim at asserting a (close) to real world data representation. Recall that a pattern is essentially a set of attribute and value pairs. By this definition definition, a pattern p can also be empty ($p = \{\}$). In this case, all tuples in the data satisfy the pattern p. By setting the pattern p' to be the empty pattern we may restrict the the count of the pattern $C.p$ with respect to the resulting weighted dataset.

EXAMPLE 2.6. *The following are possible conditions over the database D given in Figure 1.*

$$C_1: \ c_D(\{race= \textit{Hispanic}, \textit{marital status} = \textit{singe}\}) > 4$$
$$C_2: \ c_D(\{gender= \textit{Male}\}) < 0.65 \cdot c_D(\{\}) = 0.65 \cdot |D_w|$$

The pattern $C_1.p$ is $\{race= Hispanic, marital\ status = singe\}$ and $C_2.p = \{gender= Male\}$.

Given a set of conditions, "repairing" the data in a way that results in a dataset that satisfies the conditions may be done by adding new tuples and/or removing existing ones. When new data is needed, e.g., to increase the number of underrepresented group in the data, a key question is how to introduce new tuples. One possible solution approach is data acquisition. This way the repaired data is representative and thus this solution is optimal. However, collecting new data may be costly, and even impossible when using "found data". Another possible is to generate synthetic data tuples, based on models learned from the data distributions [[[**(Yuval) cite**]]]. This way the repaired data is an estimated representation of the real world, and may include non realistic tuples [[[**(Yuval) or something in this spirit?**]]]. Our approach is inspired by the wealth body of work on survey data re-weighting [**?**], where weights are assigned to the database tuples. Given a database, we use a weighting function to depict the data modifications as we next define.

DEFINITION 2.7 (DATA WEIGHTING FUNCTION). *Given a database D, a weighting function $w : D \mapsto \mathbb{N}$ is a function that maps each tuple in the database D to a natural number. The weighted database D_w is the database that contains $w(t)$ times the tuple t for each t in D.*

EXAMPLE 2.8. *The function w such that $w(t_1) = w(t_3) = 2$ and $w(t_i) = 1$ for all $i \neq 1, 3$ is a an example of weighting function for the database given in Figure 1. Another possible weighting function is the function w' where $w'(t_2) = w'(t_3) = 2$ and $w(t_i) = 1$ for all $i \neq 2, 3$.*

Intuitively, given a set of conditions, the goal of the data adjustment is to eliminate any condition violation that occur in the original data. We thus define the notion of satisfying weighting function.

DEFINITION 2.9 (SATISFYING WEIGHTING FUNCTION). *Given a database D and a set of conditions $\mathcal{C}$, we say that a weighting function w is* satisfying weighting function *if D_w satisfies C for all $C \in \mathcal{C}$.*

EXAMPLE 2.10. *Consider again the database shown in Figure 1, the conditions C_1 and C_2 given in Example 2.6 and the weighting function from Example 2.8. The number of tuples satisfying the pattern $C_1.p = \{race= Hispanic, marital\ status = singe\}$ is 3 and the number of tuples satisfying the pattern $C_2.p = \{gender= Male\}$ is 32. Since the total number of tuples in the database is 50, $c_D(\{gender= Male\}) = 0.64 \cdot |D|$. Thus, D satisfies the condition C_1 and C_2.*

The weighted database D_w resulting from the weighting function w in Example 2.8 satisfies the conditions, since the number of tuples satisfying the pattern $C_1.p =$ is 5, and the number of tuples satisfying the pattern $C_2.p =$ is $33 > 0.65 \cdot 50$. Thus w is a satisfying weighting function. The weighted database D'_w obtained using w' satisfies the condition C_1, however it does not satisfy C_2 and thus w' is not a satisfying weighting function. (Yifan) I slightly modified this example to make it correct.

3. LOCAL DATA DIFFERENCE

In this section, we present our notion of *local data difference*. Our goal in re-weighting the data is to achieve a database that satisfies the user defined condition. An important goal within this process is to keep the resulting dataset as close as possible to its original form. A key question is then how to quantify the data modification. Intuitively, we wish to maintain the data properties, such as correlations, dependencies and distributions.

In order to satisfy the condition, data re-weighting would typically modify the attribute distributions. As a simple example, consider a dataset containing information of children's age, weight and height. Assume we are given a constraint that require adding information of children within a particular age group (without increasing the number of tuples in other age groups). Satisfying this condition would require increasing the weights of tuple representing children within the group age stated by the constraint (and keeping the rest intact). Obviously, such re-weighting will affect the age attribute value's distribution, but due to the correlation between the attributes, it may also affect the value distributions of the weight and height attributes. Such modifications may be inevitable, however a more refined requirement would be to minimize the difference in the value distribution of weight and height within each age group between the original and weighted data. To this end, we propose the local data difference, a novel measure for data difference.

DEFINITION 3.1 (LOCAL DATA DIFFERENCE). *Given a database D with attributes $\mathcal{A} = \{A_1, \ldots, A_n\}$ we use $P_D(A_i = a_j)$ to*

denote the probability of a tuple t in D to have the value a_j for some $a_j \in Dom(A_i)$.

Given a database D, a pattern p and a weighted database D_w, the local data difference from D to D_w with respect to p in the attribute A is defined as

$$\Delta_{p,A}(D, D_w) = \sum_{x \in Dom(A)} |P_D(A = x \mid p) - P_{D_w}(A = x \mid p)|$$

The local data difference from D to D_w with respect to p is then

$$\Delta_p(D, D_w) = \sum_{A \notin Attr(p)} \Delta_{p,A}(D, D_w)$$

Finally, we define the local data difference from D to D_w with respect to a set of conditions as

$$\Delta_{\mathcal{C}}(D, D_w) = \sum_{C \in \mathcal{C}} \Delta_{C.p}(D, D_w)$$

EXAMPLE 3.2. *[[[**(Yuval) TBD**]]]*

(Yifan) some more analysis on local data difference equation? Delete if unnecessary.

For each condition C, there are $|\mathcal{A}| - Attr(C.p)$ attributes involving in the local data difference calculation. Hence, the total number of terms in such a local data difference equation is:

$$\sum_{C \in \mathcal{C}} \sum_{A \notin Attr(C.p)} |Dom(A)|$$

The complexity of local data difference increases as number of conditions, number of attributes not used to define conditions' patterns or cardinality of attributes increases.

Problem Definition. We are now ready to define the optimal satisfying weighting function.

PROBLEM 3.3. *Given a database D and a set of conditions $\mathcal{C}$, find a satisfying weighting function w with minimal local data difference, namely*

$$\arg\min_{w}(\Delta_{\mathcal{C}}(D, D_w))$$

4. REDUCTION TO MIXED INTEGER NONLINEAR PROGRAMMING

In this section, we present our approach for finding an optimal satisfying weighting function.

One naive approach to generate a weight function is to infer bounds of weight variables from user-defined conditions, and then perform exhaustive search over all weight variables. This approach has an exponential complexity. A weight function can be infeasible to generate in a reasonable time if some weight variables are unbounded or some weight variables' domain is too large.[[[**(Yuval) this is essentially what we do, no?**]]]

The core idea of our approach is to model the data reweighting problem as a constrained optimization problem, specifically as Mixed-Integer Nonlinear Programming (MINLP). The idea is to assign a variable w_i for each tuple $t_i \in D$. Every user-defined condition is modeled as an constraint over the variables w_i's, and the objective function is defined to be the local data difference. We then use a solver to find an optimal assignment to the variables which corresponds to an optimal weighting function.

4.1 Translating Conditions into constraints

We start by presenting the encoding of user defined condition over pattern count into constraints. Algorithm 1 depicts this transformation process.

Notations. Given a condition $C = c_D(p)$ `op` x we use C.`op` to denote `op` (i.e., $<$ or $>$), and $C.x$ is use to denote the right-hand side of the condition. Recall that x can be either a constant or a fraction of other pattern count. In the latter case $C.x = \alpha \cdot c_D(p')$ for some pattern p' over the database and $\alpha \in \mathbb{R}$.

The input to the algorithm is a database D and a condition C. In line 1 the algorithm retrieves the indexes of the tuples from D that satisfy the pattern $C.p$ using the procedure `getSatisfyingIdx`. Next, (line 2) the first part of the constraint is constructed, as the sum of the variables w_i that correspond to the tuples in D that satisfy $C.p$. The second part of the constraint is generated in lines 3–7. If $C.x$ is a constant , the second part of the constraint is simply $C.x$ (lines 3–4). Otherwise (line 5), the algorithm apply the procedure `getSatisfyingIdx` to retrieve the indexes of the tuples from D that satisfy the pattern p' where p' is the pattern appearing in $C.x$ (line 6). In this case, the right-hand side of the constraint is the sum of variables that correspond to the weights of tuples that satisfy p' (line 7). Finally, the algorithm return the expression obtained by joining the expression generated in the first part, C.`op` and the second part of the constraint (line 8).

Algorithm 1: Transform a Condition

input : A database D and a condition C
output: A constraint expression representing the input condition

1 $\mathcal{I} \leftarrow$ `getSatisfyingIdx`$(D, C.p)$
2 $left\_sum \leftarrow \sum_{i \in \mathcal{I}} w_i$
3 **if** *$C.x$ is a constant value* **then**
4 $\quad right\_sum \leftarrow C_j.x$
5 **else**
$\quad$ Let $C_j.x = \alpha \cdot c_D(p')$
6 $\quad \mathcal{I}' \leftarrow$ `getSatisfyingIdx`(D, p')
7 $\quad right\_sum \leftarrow \alpha \cdot \sum_{i \in \mathcal{I}'} w_i$
8 **return** $left\_sum\ C_j.op\ right\_sum$

EXAMPLE 4.1. *[[[**(Yuval) tbd**]]]*

4.2 Modeling as an Optimization Problem

The key part of modeling an optimization problem is to explicitly and correctly represent its objective function and constraints. The formulation of local data difference is nonlinear by nature, and we want resulting weights are all integers, so we model the data-reweighting problem as an Mixed-Integer Nonlinear Programming (MINLP) problem. For efficiency, we use off-the-shelf open-source mixed-integer nonlinear programming solver: Couenne [1] to solve the formulated constrained optimization problem. MINLP is NP-hard in general [2]. Fortunately, MINLP solvers such as Couenne utilizes a combination of linear programming solvers and nonlinear programming solvers, and a variety of algorithms such as branch-and-bound to improve efficiency of solving MINLP problems.

Following our definition in Sec. 2 and Sec. 3, given a database D and a set of conditions C, the objective function of our MINLP model is:

$$\Delta_{\mathcal{C}}(D, D_w) = \sum_{C \in \mathcal{C}} \Delta_{C.p}(D, D_w)$$

Except the constraint: all weight variables should be integers, other model constraints come from user-defined conditions. Without loss of generality, we define three types of conditions in Sec. 2 in the form:

$$c_D(p) \ ? \ x$$

Next, we show how to transform them into constraints suitable for modeling purposes. Algorithm 1 illustrates this transformation process assuming the inequality sign ? is $<$ for simplicity.

(Yifan) Do we need an example to illustrate this process? I feel like there are not many paper showing concrete examples in Algorithm section. [[[**(Yuval) would be good if we can an example using the running example**]]]

Overall, Algorithm 2 shows an overview of our approach for generating a weight function. Our approach firstly performs initialization (lines 1-4). Then, it process conditions to add constraints and objective function to an optimization solver (lines 5-11). Finally, it assigns solver-calculated weights to weight variables of data tuples (lines 12-17). One key factor of solver performance is the number of variables it has to deal with. We use a set S to store these variables covered by one or more conditions. These variables need solver to assign weights to them, and the remaining weight variables remains 1.

Algorithm 2: MINLP based algorithm

input : A database D with n tuples and a set of conditions $\mathcal{C}$
output: A weighting function with minimal local data difference

```
1  Initialize MINLP solver s
2  Initialize w_1, w_2, ..., w_n = 1
3  Initialize obj = 0
4  Initialize S an empty set to store weight variables
5  for C_i ∈ C do
6      Put weights of all tuples satisfying C_i.p into S
7      obj += Δ_{C_i.p}(D, D_w)
8      Transform condition C_i into constraint i
9      Add constraint i to s
10 Add obj as the objective function to s
11 s solves the optimization problem
12 for w_i do
13     if w_i ∈ S then
14         w_i ← arg min_{w∉S}(Δ_C(D, D_w))
15     else
16         w_i = 1
17 return {w_1, w_2, ..., w_n}
```

5. EXPERIMENTS

In this section, we evaluate usefulness of local data difference and our approach in terms of:

Objectives:

- Usefulness of the approach: increasing accuracy of different models.
  - Different datasets
  - Varied "boosting" parameter
- Objective function justification:
  - Compare the distributions of attributes in the weighted dataset to the original dataset using KL-divergence.
  - Compare to other possible definitions?
- Algorithm performance:
  - As a function of data size (number of tuples)
  - As a function of number of attributes
  - As a function of number of constraints
  - If we end up using any optimizations: effect of optimizations

5.1 Dataset Description

We use three real-world datasets for evaluations:

- *COMPAS*[1]: ProPublica is an independent, non-profit organization that produces investigative journalism. This dataset contains records for defendants from Broward County from 2013 to 2014. We bucketized attributes as follows: `sex` (0: male and 1: female), `age` (0: under 20, 1: between 20 and 39, 2: between 40 and 59, and 3: above 60), `race` (0: African-American, 1: Caucasian, 2: Hispanic, and 3: other races), and `martial status` (0: single, 1: married, 2: separated, 3: widowed, 4: significant other, 5: divorced, and 6: unknown). We used one additional discrete attribute: `decile_score` as one extra training feature, and `is_recid` as the binary prediction label. Totally, five training features are used to predict whether a criminal has re-offended.
- *BlueNile*[2]: Blue Nile is the world's leading online diamond retailer. We used 15,000 diamond records. We categorized continuous-valued attributes `carat, depth, table, LengthWidthRatio`. In addition, we chose 6 categorical attributes, namely `shape, cut, color, clarity, polish`, and `symmetry`. All of those attributes are used as training features to predict `price` bin of diamonds.
- *Adult data*[3]: This dataset was extracted from the census bureau database. We bucketized attributes `age, hours-per-week, capital-gain`, and `capital-loss`. Discrete attributes, namely `workclass, education, marital-status, occupation, relationship, race, sex`, and `native-country` are used as training features as well. The prediction task is to determine whether a person makes over $50k$ a year.

[1] `https://www.propublica.org/datastore/dataset/compas-recidivism-risk-score-data-and-analysis`
[2] `https://www.bluenile.com/diamond-search`
[3] `https://archive.ics.uci.edu/ml/datasets/adult`

5.2 Hardware and Platform

The experiments were conducted on a Linux machine with a 1.7 GHz Intel Xeon processor and 128 GB memory. We use the open-source, constrained-optimization solver Couenne [1]. We use Pyomo [3], a Python-based, open-source optimization modeling language with a diverse set of optimization capabilities, as a wrapper to interact with the Couenne solver.

5.3 Minority Group Identification

(Yifan) I'm planning to list the minority groups I found in this subsection

5.4 Evaluation

(Yifan) list reweighting conditions + result number as table and plot results

6. REFERENCES

[1] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tighteningtechniques for non-convex minlp. *Optimization Methods & Software*, 24(4-5):597–634, 2009.

[2] S. Burer and A. N. Letchford. Non-convex mixed-integer nonlinear programming: A survey. *Surveys in Operations Research and Management Science*, 17(2):97–106, 2012.

[3] W. E. Hart, J.-P. Watson, and D. L. Woodruff. Pyomo: modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219–260, 2011.