Data Exploration with SQL using Machine Learning Techniques

Julien Cumin
Orange Labs & Inria
Meylan, France
julien1.cumin@orange.com

Jean-Marc Petit
Univ Lyon, INSA Lyon, CNRS
LIRIS, France
jean-marc.petit@liris.cnrs.fr

Vasile-Marian Scuturici
Univ Lyon, INSA Lyon, CNRS
LIRIS, France
marian.scuturici@liris.cnrs.fr

Sabina Surdu
Babes-Bolyai Univ
Cluj-Napoca, Romania
sabina@cs.ubbcluj.ro

ABSTRACT

Nowadays data scientists have access to gigantic data, many of them being accessible through SQL. Despite the inherent simplicity of SQL, writing *relevant and efficient SQL queries* is known to be difficult, especially for databases having a large number of attributes or meaningless attribute names. In this paper, we propose a "rewriting" technique to help data scientists formulate SQL queries, to rapidly and intuitively explore their big data, while keeping user input at a minimum, with no manual tuple specification or labeling. For a user specified query, we define a *negation query*, which produces tuples that are not wanted in the initial query's answer. Since there is an exponential number of such negation queries, we describe a pseudo-polynomial heuristic to pick the negation closest in size to the initial query, and construct a balanced learning set whose positive examples correspond to the results desired by analysts, and negative examples to those they do not want. The initial query is reformulated using machine learning techniques and a new query, more efficient and diverse, is obtained. We have implemented a prototype and conducted experiments on real-life datasets and synthetic query workloads to assess the scalability and precision of our proposition. A preliminary qualitative experiment conducted with astrophysicists is also described.

CCS Concepts

•Information systems → Data mining; Query reformulation; *Users and interactive retrieval;* **•Human-centered computing → Interactive systems and tools; •Computing methodologies →** *Machine learning;*

Keywords

Big Data; query rewriting; pattern mining; query recommendation; knowledge discovery in databases

1. INTRODUCTION

There's a pressing need on the discovery rate of interesting knowledge to keep pace with the booming data stores, as is the case in scientific fields like astronomy or earth observation, with their continuous advent of data-producing instruments and techniques. For instance, the Large Synoptic Survey Telescope [2] (LSST) currently under construction in Chile is projected to produce about 15 TB of imaging data each night, and over 10 petabytes of relational data in its 10 years lifetime. Clearly, our ability to stockpile data has long left behind our data analysis and exploitation capacities. Trying to make sense of data of the mentioned sizes with today's systems and techniques is an extremely difficult, if not an impossible, task, as even simple queries can return results so large they are practically incomprehensible [3]. Data storage and throughput related issues may be quite challenging, but problems linked with data exploration and system usage are even more so.

Scientific data commonly appear as SQL queryable relations, with hundreds of attributes holding numerical values from physical measurements or observations[1]. The attribute-based selection criteria are not always easily expressible. In order to formulate the right SQL query, analysts need to correctly define the appropriate attribute thresholds and make sure no attribute condition is missing from the query, which can be a real hurdle. Moreover, it is often the case that scientists do not know exactly what they are searching for, until after they find it, e.g., astrophysicists searching for interesting patterns in the data. Consider an astrophysicist exploring a huge database holding data about stars. She wants to select stars likely to harbor planets. But the data's attributes are mostly related to light magnitude and amplitude, and the presence of planets is only confirmed for a small number of stars. The astrophysicist can easily pose an initial query retrieving those stars but has no idea about another SQL query whose results would be similar, i.e., pro-

[1]See for example the LSST website: http://lsst-web.ncsa.illinois.edu/schema/index.php

viding a fair number of true positives (stars with planets) and a limited number of false positives (stars which surely lack planets), but also introducing new stars, which are *likely* to have planets.

Given the range of fields which produce or benefit from increasingly sized data, more and more difficult to query by a large spectrum of users, we believe it is crucial to enable data exploration, while keeping the expertise requirements as low as possible.

Example 1 Consider the `CA (CompromisedAccounts)` relation given in figure 1, inspired from the Ashley Madison data breach[2] [32]. Ashley Madison is a website targeting people that seek out an extramarital affair. `AccId` is a user identifier and `BossAccId` is the boss of the user, only if she is also registered online. `JobRating` is a job performance rating for some of the accounts and normalized to a scale of 1 to 5. `MoneySpent` and `DailyOnlineTime` refer to the money spent on the site, and the average daily time spent online by the owner's account, respectively. `Status` tells whether the user is a governmental employee or not. The meaning of the other attributes is clear from context.

Assume a zealous reporter is searching for *governmental users that spend more time online than their bosses*. The reporter has built up a somewhat good background in SQL, and poses the following query (initial query):

```
SELECT AccId, OwnerName, Sex
FROM CompromisedAccounts CA1
WHERE Status = 'gov' AND
 DailyOnlineTime > ANY
 (SELECT DailyOnlineTime
      FROM CompromisedAccounts CA2
      WHERE CA1.BossAccId = CA2.AccId)
```

This query produces the tuples corresponding to *Casanova* and *PrinceCharming*. But the reporter is interested in as many government officials as possible. Clearly, the problem of relaxing some conditions of the query to get more results has no solution for her. Hovewer, as we will see in the paper, the above query can be reformulated as follows:

```
Reformulated query:

  SELECT AccId, OwnerName, Sex
  FROM CompromisedAccounts
  WHERE (MoneySpent >= 90000 AND JobRating >= 4.5) OR
    (MoneySpent < 90000 AND DailyOnlineTime >= 9)
```

We notice that the reformulated query:

- is very much different from the initial one: it includes the new attributes *MoneySpent* and *JobRating*;
- it does not contain imbrications, therefore being more efficient, only going once through the relation; when the data volumes are very large, this is obviously of major importance;
- it maintains *Casanova* and *PrinceCharming* in the result, but
- is not equivalent to the first query, introducing some diversity in the results: tuples *RhetButtler*, *MrDarcy* and *BigBadWolf* are only found in the new query's result.

However, the reporter does not know in advance the conditions in the reformulation's *WHERE* clause, which in fact describe a *pattern* hidden in the data. According to this pattern, accounts spending more than 90k and whose owners do very well at work are likely to belong to cheating governmental employees who spend more time on the website than their bosses do. Same can be assumed about accounts spending less than 90k, but more than one third of a day online. A set of attributes is described based on another set of attributes. We could uncover this pattern by feeding a data mining tool with all these data, then go back to the database and pose the found query. But this poor reporter already deserves all the credit and our sympathy for digging into databases, a field unfamiliar to her. Do we really want to put her through the wringer of delving into the fascinating, yet complex world of data mining and machine learning? Moreover, if on a 10 rows example one could come up with a pattern close to the real one, this is impossible to achieve in a realistic setting.

Problem statement.

The problem we are interested in can be formulated as follows:

> Given a database and a user-specified SQL query that selects data based on some initial condition, we are interested in a reformulation of the initial query that captures the *pattern* revealed from the initial query's result data.

We do not search for an equivalent query, but for one whose result overlaps the initial query's result to a certain extent, and that also introduces new tuples in its result. The latter represents the exploratory potential of the new query. We set out to build an interaction that allows analysts to explore their data in this fashion, solely through SQL queries.

For a user-specified SQL query Q on a database d, we can easily define *positive examples*, or simply *examples*, based on Q's answer on d. They correspond to tuples that are wanted by the analyst. One of the challenges is to define the *negative examples* or *counter-examples*, i.e., tuples the analyst does not want to see in the result. This, in turn, raises the question of defining a query $\overline{Q}$ that could be considered as a *negation* of Q. If the generation of this query $\overline{Q}$ is possible, we have a set of *example* tuples, the results of Q's evaluation, and a set of *counter-example* tuples, obtained from $\overline{Q}$'s evaluation. We can then use a data mining technique in a supervised pattern learning approach on these two tuple sets.

For a user-specified query Q on a database d, there is an exponential number of possible negation queries $\overline{Q}$ that can be defined by negating different parts of Q's selection condition. We aim to find the one whose answer size is closest to Q's answer size, i.e., the set of positive examples and the set of negative examples are as close as possible in size. We call it the *balanced negation query*. The "more" balanced the learning set is, the higher its entropy, the better

[2]This is an example we came up with, in no way related to real data from the breach.

AccId	Owner Name	Age	Sex	Money Spent	Daily Online Time	JobRating	Status	BossAccId
100	Casanova	50	M	100k	5h	4.5	gov	350
200	DonJuanDeMarco	20	M	20k	1h	2.1	null	null
350	PrinceCharming	28	M	90k	4h	4.8	gov	230
40	Playboy	40	M	10k	35min	2	nongov	700
700	Romeo	50	M	30k	30min	3	nongov	null
90	RhetButtler	40	M	95k	4h	4.9	null	null
80	Shrek	40	M	25k	1h	1	nongov	700
70	MrDarcy	35	M	97k	3h	4.6	null	null
230	JackSparrow	61	M	30k	2h	3	gov	null
59	BigBadWolf	31	M	70k	9h	3	null	200

Figure 1: The `CompromisedAccounts (CA)` relation

for the decision tree algorithm working on it. Pruning the exponential space of negation queries boils down to solving an heuristic based on the Knapsack problem, known to be pseudo-polynomial.

The decision-tree algorithm automatically proposes a new query to the data analyst. This system-generated query can be interesting in the exploratory quest, since its results are close to, yet different from the initial query's results, including new tuples that were not directly accessible before. We propose some quality measures to evaluate the reformulated query quality, especially with respect to its diversity in terms of returned tuples.

We emphasize that:

- the user's input is kept to a minimum, i.e., only a SQL query is expected from her; she does not need to manually label any tuples, or input actual tuple values, an approach frequently used by alternative systems;
- even for users with a data mining background, the job is now much easier, since they only use one system, without any disrupting switches between various tools.

Paper contribution.

To sum up, we propose a simple but powerful approach to deal with the above problem statement. Our main contributions can be summarized as follows:

- For a class of SQL queries, we introduce the notions of *positive examples* - from the answer set of a query - and *negative examples* - from the answer set of a *negation query*.
- Since the set of negation queries is exponential, we propose a pseudo-polynomial Knapsack-based heuristic to identify a negation query whose size is close to the size of the initial query answer.
- The initial query and the selected negation query allow building a learning set on which machine learning techniques, derived from decision trees, are applied. A new SQL query, called a transmuted query, is proposed from the decision tree. Some criteria are proposed to evaluate the quality of the transmuted query.
- We have implemented a prototype and conducted experiments on real-life datasets and synthetic query workloads to assess the scalability and precision of our proposition. A preliminary qualitative experiment conducted with astrophysicists is also described.

Paper Organization.

The rest of this paper is organized as follows. Section 2 discusses queries and their *negation queries*. A heuristic that prunes the exponential space of negation queries is fully described. In section 3 we present our approach to data exploration, based on automatic query rewriting. We discuss *transmuted queries* and metrics that assess the quality of the rewriting. Section 4 describes preliminary results on an astrophysics database and the experimental setting for a newly implemented prototype. Section 5 positions our approach with respect to related work. Finally, section 6 concludes our paper.

2. SQL QUERIES AND THEIR NEGATION

The challenge we take on is to define the set of tuples that *do not verify* a query Q, which reduces to defining a *negation query* $\overline{Q}$ of Q. We then explore the exponential space of possible negation queries, and finally present a pseudo-polynomial Knapsack-based heuristic to find the negation query whose answer size is closest to the answer size of Q.

2.1 Preliminaries

We briefly introduce the notations used throughout this paper (see for example [4]). Let d be a database defined on a schema $\mathbf{R}$ and Q be a query on $\mathbf{R}$. We denote by $ans(Q, d)$ the result of the evaluation of Q on d. The domains of attributes are assumed to give either categorical values or numerical values. We assume the database allows null values. As relational languages, we consider both relational algebra for formal notations and SQL for the running example.

2.2 Considered relational queries

For the sake of clarity, we consider a simple class of relational queries, basically conjunctive queries extended with some binary operators and a construct to check for NULL values. Identifying a larger class of relational queries is left for future work.

Queries in the considered class have the following form:

$$Q = \pi_{A_1,\ldots,A_n}(\sigma_F(R_1 \bowtie \ldots \bowtie R_p))$$

where

- π is the projection, σ the selection and $\bowtie$ the natural join as usual
- F is a conjunction of m atomic formulas of the form $\gamma_1 \wedge ... \wedge \gamma_m$ with $m \geq 1$

- Each atomic formula (or predicate) γ_i in F has the form $A\ bop\ B$, $A\ bop\ a$, $A\ IS\ NULL$, where A, B are from R, a is a real value or a categorical one and $bop \in \{=, <, >, \leq, \geq\}$
- An atomic formula γ can be negated; its negation is denoted by $\neg(\gamma)$.

We denote by $attr(F)$ the set of attributes that appear in a selection formula F. The cardinal of a set E is denoted by $|E|$.

Example 2 To comply with this class of queries, the initial query from example 1 is rewritten as:

```
SELECT CA1.AccId, CA1.OwnerName, CA1.Sex
FROM CompromisedAccounts CA1, CompromisedAccounts CA2
WHERE CA1.Status = 'gov' AND
  CA1.DailyOnlineTime > CA2.DailyOnlineTime AND
  CA1.BossAccId = CA2.AccId
```

For a query Q of the above form, we consider the "reservoir of diversity" (diversity tank) to consist of those tuples for which:

- there exists at least one predicate γ_i whose evaluation is NULL; (1)
- all the predicates in Q that do not evaluate to NULL, evaluate to TRUE (2).

Some tuples in this diversity tank might turn out to be interesting for the user, even if they do not appear in the initial query's result. Condition (1) states that, for a tuple to have an exploratory potential, some data the user is interested in about the tuple needs to be unknown. Condition (2) asks that none of Q's predicates evaluate to false. Otherwise, the tuple does not satisfy Q and shouldn't be explored at all, since the user is interested in tuples meeting Q's condition.

Example 3 For our running example, tuples corresponding to employees (CA1.OwnerName) *DonJuanDeMarco*, *RhetButtler*, *MrDarcy*, *JackSparrow* and *BigBadWolf* form the diversity tank (and indeed, the reformulated query's new tuples, *RhetButtler*, *MrDarcy* and *BigBadWolf*, came from this set).

Positive examples are denoted by $E_+(Q)$ and come from the query's evaluation result. When the result's size is reasonable, we can consider all its tuples as examples. Otherwise, we can use stratified random sampling for instance to extract a subset of tuples as positive examples. We keep all the possible attributes, so later on in the learning phase we have as many as possible options to learn on. Therefore, we eliminate the projection on $A_1, ..., A_n$ and obtain: $E_+(Q) \subseteq \sigma_F(R_1 \bowtie \ldots \bowtie R_p)$.

Example 4 The positive tuples of query Q are those corresponding to employees *Casanova* and *PrinceCharming*.

2.3 The negation of queries

Let Q be a query. Its answer set is obtained by simply evaluating Q on the database. We pose the problem of defining $\overline{Q}$, a *negation query* of Q, i.e., a query whose evaluation produces tuples we do not want to see when evaluating Q. Intuitively, $\overline{Q}$ should not touch the same tuples of Q, i.e., the intersection of their respective answer sets should be empty.

Since we aim to uncover dependencies between attributes in the query rewriting process, we keep all the possible attributes that might allow us to distinguish between tuples in the learning step. As for positive examples, we eliminate the projection from the negation queries' definitions hereafter.

One way of computing a negation query for Q is to consider its entire tuple space $R_1 \bowtie \ldots \bowtie R_p$ and eliminate those tuples that belong to Q's answer. This is in fact the complement of Q. We consider it to be Q's *complete negation* and denote it by $\overline{Q_c}$:

$$\overline{Q_c} = (R_1 \bowtie \ldots \bowtie R_p) \setminus (\sigma_{\gamma_1 \wedge \ldots \wedge \gamma_m}(R_1 \bowtie \ldots \bowtie R_p)) \quad (1)$$

There is no guarantee on the answer size of $\overline{Q_c}$; it may be quite different than Q's size. We thus set out to explore the space of alternative negation queries for Q. We consider negation queries whose selection conditions are generated from Q's selection formula. In this case, we consider that a negation query $\overline{Q}$ needs to negate at least one of Q's predicates.

We exclude foreign key join predicates from the set of predicates that can be negated. They help narrow down the set of tuples that can be used as examples and counter-examples.

To simplify notations, we describe Q's selection formula F as $F_{\overline{k}} \wedge F_k$, where F_k is the conjunction of all the foreign key predicates in F, if any, and $F_{\overline{k}}$ the conjunction of all the other predicates, i.e., the *negatable* predicates.

Property 1 Let Q be a query and $n = |F_{\overline{k}}|$, the number of negatable predicates. The number of negation queries $\overline{Q}$ with respect to Q is exponential in n.

Proof. For each negatable predicate γ in Q, there are three possibilities: (1) keep it in $\overline{Q}$ as it is, (2) take its negation $\neg(\gamma)$ or (3) do not consider it at all. Then there are 3^n possible negation queries, but 2^n out of them are invalid, since they do not contain any negated predicate from $F_{\overline{k}}$. We get $3^n - 2^n$, which is in $O(2^n)$. □

We denote the set of the valid negation queries by $\{\overline{Q}\}$. All the negated γ_i come from $F_{\overline{k}}$. We denote by $attr(F_{\overline{k}})$ all the attributes from $F_{\overline{k}}$ that appear in predicates that are negated in $\overline{Q}$.

Similarly to positive examples, the negative examples are denoted by $E_-(Q)$ and verify the following:
$E_-(Q) \subseteq ans(\overline{Q}, d)$.

Example 5 Let us denote the three predicates in Q CA1.Status = 'gov' by γ_1, CA1.DailyOnlineTime > CA2.DailyOnlineTime by γ_2 and CA1.BossAccId = CA2.AccId by γ_3. In our approach there are five possible negations of Q, with selection formulas: $\neg(\gamma_1) \wedge \gamma_3$, $\neg(\gamma_2) \wedge \gamma_3$, $\neg(\gamma_1) \wedge \gamma_2 \wedge \gamma_3$, $\gamma_1 \wedge \neg(\gamma_2) \wedge \gamma_3$ and $\neg(\gamma_1) \wedge \neg(\gamma_2) \wedge \gamma_3$. If we choose the third one, the negation query $\overline{Q}$ is:

```
SELECT *
FROM CompromisedAccounts CA1, CompromisedAccounts CA2
WHERE NOT (CA1.Status = 'gov') AND
  CA1.DailyOnlineTime > CA2.DailyOnlineTime AND
  CA1.BossAccId = CA2.AccId
```

Accounts *Playboy* and *Shrek* are thus considered to be negative examples. Both Q and $\overline{Q}$ answer sizes are equal to two.

2.4 Pruning the space of negation queries

In this subsection we solve the problem of finding the closest negation query, in terms of its answer size, for a given query, by proposing an heuristic based on the subset-sum problem (related to the knapsack problem). The subset-sum problem is known to be pseudo-polynomial (weakly NP-complete) [7]. The additional constraints we add do not affect the complexity of the algorithm.

Notation.

To further simplify notations, when there's no confusion between a query Q and its answer, we denote the latter by Q instead of $ans(Q, d)$. Z denotes the entire tuple space $R_1 \bowtie \ldots \bowtie R_p$. Similarly, for a predicate $\gamma_i \in Q$, we refer to the query $\sigma_{\gamma_i}(Z)$ simply by γ_i. Since we only touch the selection formula, we ignore Q's projection attributes. We denote the negation of a predicate γ by $\overline{\gamma}$.

Assumption.

DataBase Management Systems (DBMS) maintain many statistics for cost-based optimization of query processing. Moreover, to estimate the size of query results and make a decision for choosing a physical plan, data are often assumed to be uniformly distributed. In the sequel, we borrow the same assumptions, i.e., data is uniformly distributed in Z and for a given query Q, an estimate of the size of its answer $|Q|$ is supposed to be known. We denote the probability that a tuple in Z verifies γ_i by $P(\gamma_i)$. The cardinality of γ_i is $|\gamma_i| \simeq P(\gamma_i) * |Z|$. The probability that a tuple satisfies both γ_i and γ_j is $P(\gamma_i \wedge \gamma_j) = P(\gamma_i) * P(\gamma_j)$. The cardinality of the set of rows satisfying both predicates is estimated by $|\gamma_i \wedge \gamma_j| \simeq P(\gamma_i) * P(\gamma_j) * |Z|$. The probability of a negated predicate $\overline{\gamma_i}$ is $P(\overline{\gamma_i}) = 1 - P(\gamma_i)$, and $P(Q \cup \overline{Q_c}) = 1$.

We can now give a more formal description of our problem of finding the most balanced learning set corresponding to a query Q.

Balanced negation query.

Let us consider a query Q with n negatable predicates in $F_{\overline{k}}$ and l foreign key join predicates in F_k. The problem statement is the following:

> Given such a query Q, find a negation query $\overline{Q}$ of Q such that:
>
> (1) its answer size $|\overline{Q}|$ is closest to $|Q|$, i.e., $abs(|Q| - |\overline{Q}|)$ is minimized,
>
> (2) $\overline{Q}$ negates at least one predicate from $F_{\overline{k}}$,
>
> (3) $\overline{Q}$ can contain any number of the rest of the predicates from $F_{\overline{k}}$, negated or not, and
>
> (4) $\overline{Q}$ contains all the predicates from F_k.

The possible solutions to this problem have the form $\overline{Q} = \bigwedge_{i=1}^{n+l} a_i$, where a_i is a predicate. Let us consider the predicates for $\overline{Q}$ as a (n+l)-tuple denoted by s. The components of s are:

(1) the predicates from F_k,

(2) any predicate from $F_{\overline{k}}$, or its negation, or the "identity" element $Q \cup \overline{Q_c}$, i.e., the predicate is not considered at all in $\overline{Q}$.

s can be represented as: $s = (a_1, \ldots, a_{n+l}) = (\gamma_{k_1}, \ldots, \gamma_{k_l}, e_1, \ldots, e_n)$ with $\gamma_{k_i} \in F_k$ and $e_j \in \left\{\gamma_{\overline{k}_j}, \overline{\gamma_{\overline{k}_j}}, Q \cup \overline{Q_c}\right\}$, for all $\gamma_{\overline{k}_j} \in F_{\overline{k}}$.

Therefore every negation query $\overline{Q}$ can be represented as such a tuple s and its cardinality is estimated by $|\overline{Q}| = |s| = \left(\prod_{i=1}^{n+l} P(a_i)\right) * |Z| = \left(\prod_{i=1}^{n+l} \frac{|a_i|}{|Z|}\right) * |Z|$.

Our problem now is in fact a particular case of the *subset product problem* [19, 26], known to be NP-hard [24]:

> Given a set $E = \{e_1, \ldots, e_n\}$ of integer values and a number m, does there exist a subset F of E, such that $\prod_{e_i \in F} e_i = m$?

For a given query Q, we need to choose all the predicates in F_k and one of three possible versions for the predicates in $F_{\overline{k}}$, such that the product of their probabilities multiplied by the cardinal of the tuple space Z is as close as possible to the size of Q, the target number.

Applying logarithms on this product, we pass from the subset-product problem to a kind of subset-sum problem on real numbers (the pseudo-polynomial algorithm is working on integers).

$$\sum_{e_i \in F} log(e_i) = log(m)$$

We propose a heuristic by introducing a tolerable approximation in the precedent relation:

$$\sum_{e_i \in F} \lfloor log(e_i) * sf \rfloor = \lfloor log(m) * sf \rfloor$$

where the *scale factor* $sf \geq 1$ is used to reduce the approximation due to rounding logarithms.

Our problem can now be expressed as a particular case of the subset-sum problem, also known to be NP-complete:

> Given a total weight $|Q|$ and a set of $n + l$ a_i objects, l of them with fixed weights $|\gamma_{k_i}|$, $i = 1, \ldots, l$, and n of them with three possible weights $|e_j| \in \left\{|\gamma_{\overline{k}_j}|, |\overline{\gamma_{\overline{k}_j}}|, 1\right\}$, $j = 1, \ldots, n$, choose all the γ_{k_i} objects and a version for **each** of the e_j objects, such that: (1) the sum of the combined weights of the chosen objects is as close as possible to $|Q|$ and (2) at least one of the e_j objects is negated.

If s is the solution tuple, its combined weight is its cardinality as defined above, and condition (1) translates to minimizing $abs(|Q| - |s|)$.

The heuristic described in algorithm 1 solves this problem. All the objects in the fixed-weights vector f_k are in the output; the target weight w is updated accordingly (lines 2-3). Function *weight* calculates the weight of s, i.e., the number of rows estimated to satisfy all the predicates currently in s. From here on out, we only work with the $f_{\overline{k}}$ vector of negatable objects / predicates. For each such predicate (weight: $lWs[i].pW$), BALANCEDNEGATION assumes its negation (weight: $lWs[i].nW$) is part of the solution. It

finds an optimal solution for the rest of the predicates and a correspondingly updated weight (lines 5 - 15). This solution could contain only positive objects, but the current predicate is always added with its negated form, so restriction (2) in the problem statement is met. Out of the n possible solutions, the one that gets closer to the target weight is chosen. We set the scale factor sf parameter value to 1000 (see experiment 2 in section 4.1).

Algorithm 1: Knapsack-based heuristic to find the balanced negation of a query

```
1  procedure BalancedNegation (|Z|, |Q|, f_k, f_k̄, sf) ;
   Input  : the size of the tuple space |Z|; the target
            weight |Q|; l-vector of fixed weights f_k;
            n-vector of negatable weights f_k̄; scale factor
            sf
   Output: l + n-vector of chosen objects s; s's weight s_w
2  s := f_k; s_w := weight(s) ;
3  w := |Q| / s_w ;
4  mW := 0 ;
5  for i ← 1 to n do
6  |  lWs := f_k̄ ;
7  |  rW := lWs[i].pW ;
8  |  lWs.Remove(i) ;
9  |  tW := w*|Z| / (|Z| − rW) ;
10 |  tW := − ⌊ln(tW/|Z|) * sf⌋ ;
11 |  for j ← 1 to n − 1 do
12 |  |  lWs[j].pW := − ⌊ln((lWs[j].pW)/|Z|) * sf⌋ ;
13 |  |  lWs[j].nW := − ⌊ln((|Z| − lWs[j].pW)/|Z|) * sf⌋ ;
14 |  end
15 |  SubsetSum(lWs, tW, out oObj, out oW) ;
16 |  oW := ⌊e^((−oW)/sf) * |Z|⌋ ;
17 |  mWL := ⌊(|Z| − rW)/|Z| * oW⌋ ;
18 |  if mWL > mW then
19 |  |  mW := mWL ;
20 |  |  CompleteSol(i, s, s_w, oObj, mWL) ;
21 |  end
22 end
```

In lines 10-14 **BalancedNegation** transforms the input in order to apply a classic Knapsack algorithm on it. For each predicate, the algorithm computes its probability and then logarithms it. Since probabilities are subunitary, all the logarithms are negative and quite small, so they are multiplied by the scale factor sf, and the opposite of their integer part is retained. The same treatment is applied to the target weight. SubsetSum in line 15 computes an optimal solution in vector *oObj* with the corresponding sum of the solution subset in $oW \leq tW$. The only difference from the classic algorithm is that, if object $lWs[i]$'s positive version $lWs[i].pW$ is chosen, then its negation can not be part of the solution and vice-versa.

The output weight oW is transformed back in line 16. Lines 17-21 choose the best out of the n possible solutions adding the temporarily removed object at position i with its negated version (*CompleteSol*).

3. AUTOMATIC QUERY REWRITING

We now describe the machine learning stage that uncovers relevant patterns in the data. Starting from a user's initial query Q, the system automatically generates its negation $\overline{Q}$. It then assembles a learning set from $E_+(Q)$ and $E_-(Q)$, which is fed into an implementation of the C4.5 decision tree learning algorithm. The output decision tree's patterns are forthrightly translated into a relational selection condition, yielding a new SQL query, the transmuted query of Q. Different criteria evaluating the quality of our machine learning-based rewriting approach are defined.

3.1 Learning set construction

Once we have the positive and the negative example sets, we can build a learning set. As stated before, we do not aim at producing a reformulated query that provides the exact, precise answer of the initial query. Instead, we want to help the analyst formulate a query that better answers his expectations. These guesswork stages do not require the consideration of all the data if its size is very large. A detailed study of the guarantees we can provide for learning is outside the scope of this paper.

Definition 1 Given a query Q and its negation $\overline{Q}$, a *learning set* is defined on the schema of $(R_1 \bowtie \ldots \bowtie R_p) \setminus attr(\overline{F_{\bar{k}}}) \cup Class$. Its tuples come from $E_+(Q)$ and $E_-(Q)$, with the addition of the $+$, and the $-$ value, respectively, for the new `Class` attribute.

We exclude attributes in $attr(\overline{F_{\bar{k}}})$ from the learning set's schema to avoid learning (part of) the selection condition expressed in the initial query.

Example 6 Going further with the running example, we obtain the learning set described in Figure 2. The `Status` attribute, i.e., the only attribute in $attr(\overline{F_{\bar{k}}})$, has been suppressed and the `Class` attribute has been added with the corresponding $+$ and $-$ values.

Decision tree-based machine learning methods construct a tree that determines a class variable as a function of input variable values. A path from its root to a leaf forms a conjunction of conditions on the input variables. The class of the data that fall through this path is given by the label of its leaf. The supervised model construction, i.e., tree structure and conditions placed on internal nodes, based on a learning set, depends on the used algorithms. We chose the C4.5 algorithm [29]. It allows us to predict the values of the *Class* attribute depending on a set of attributes from the learning set.

3.2 Building the selection condition from a decision tree

Starting from a decision tree, it's relatively straightforward to build a relational selection condition by traversing the tree in depth. A branch in a tree is a direct path from its root to a leaf. A branch in the decision tree is a conjunction of boolean conditions on a tuple's attributes values. The class of the tuple is the label on the leaf of the branch. The set of branches leading to the class of positive tuples "+" can thus be seen as a disjunction of conjunctive clauses obtained from the branches. This disjunction can hence be used like a new relational selection condition.

CA	AccId	Owner Name	Age	Sex	Money Spent	Daily Online Time	JobRating	BossAccId	Class
	100	Casanova	50	M	100k	5h	4.5	350	+
	350	PrinceCharming	28	M	90k	4h	4.8	230	+
	40	Playboy	40	M	10k	35min	2	700	-
	80	Shrek	40	M	25k	1h	1	700	-

Figure 2: Learning set built from $E_+(Q)$ and $E_-(Q)$

Definition 2 Let *lSet* be a learning set obtained from a query Q. Let *decisionTree* be the decision tree learned from *lSet* to predict the values of the *Class* attribute. Let b be a positive branch of *decisionTree*, i.e., from the root to a leaf labeled +. We use the following notation for the disjunction of conjunctions leading to positively labeled leaves:

$$F_{new} = \bigvee_{b \in decisionTree} \bigwedge_{e \in b} e$$

where e has the form $A_i\, bop\, v$, A_i is an attribute in *lSet*, *bop* is a usual binary operator, and v is a numerical or categorical value.

Definition 3 Let Q be a query. The rewritten query obtained from Q, denoted by ${}^tQ$, is defined as follows:

$${}^tQ = \pi_{A_1,\ldots,A_n}(\sigma_{F_{new}}(R_1 \bowtie \ldots \bowtie R_p)).$$

We call it *the transmuted query* thereafter.

The transmuted query has a completely new selection condition that can include attributes not identified as useful in the initial query. Moreover, ${}^tQ$ is obviously simpler and quicker if the initial query is nested with, for example, the `EXISTS` or *bop* `ANY` operators. The rewriting is mechanically simplified by a single selection (a single data scan only).

Example 7 In the running example, a decision tree algorithm finds the condition `(MoneySpent >= 90000 AND JobRating >= 4.5) OR (MoneySpent < 90000 AND DailyOnlineTime >= 9)` . The corresponding rewritten query ${}^tQ$ is:

```
SELECT AccId, OwnerName, Sex
FROM CompromisedAccounts
WHERE (MoneySpent >= 90000 AND JobRating >= 4.5) OR
  (MoneySpent < 90000 AND DailyOnlineTime >= 9)
```

3.3 Quality criteria

It is difficult to make guarantees about the precise relationships between the initial query Q and its rewriting ${}^tQ$, since the latter depends on the patterns discovered in the learning phase. We can however describe the sets of tuples involved in the rewriting process and give their optimal properties:

- $\boldsymbol{Z}$: the entire tuple space $R_1 \bowtie \ldots \bowtie R_p$
- tuples fulfilling F_k and tuples fulfilling $F_{\overline{k}}$
- $\boldsymbol{ans(Q, d)}$: the set of tuples from the initial query Q on d, i.e., tuples that meet both F_k and $F_{\overline{k}}$
- $\boldsymbol{ans(\overline{Q}, d)}$: the set of tuples from the negation $\overline{Q}$ of Q on d, i.e., tuples that meet F_k, but do not meet $F_{\overline{k}}$
- $\boldsymbol{E_+(Q)}$: the set of positive tuples
- $\boldsymbol{E_-(Q)}$: the set of negative tuples
- $\boldsymbol{ans({}^tQ, d)}$: the set of tuples from the new query ${}^tQ$ on d

We now explicitly define a number of criteria and metrics that assess the quality of ${}^tQ$, the query obtained from Q.

3.3.1 Representativeness of the initial data

Our objective is to obtain a query ${}^tQ$ whose evaluation is representative of Q's results. Since the supervised learning process uses examples and counter-examples, we can expect to obtain patterns that help meet this criterion. We concretely measure it with the following formulas:

$$\frac{|{}^tQ \cap Q|}{|Q|} \underset{\text{optimal}}{=} 1 \tag{2}$$

$$\frac{|{}^tQ \cap \pi_{A_1,\ldots,A_n}(\overline{Q})|}{|\pi_{A_1,\ldots,A_n}(\overline{Q})|} \underset{\text{optimal}}{=} 0 \tag{3}$$

Equation 2 justifies the direct representativeness of the data obtained from ${}^tQ$ with respect to the data obtained from Q: optimally, we should find in ${}^tQ$ all the tuples of Q. In a similar manner, equation 3 evaluates the proportion of tuples from $\overline{Q}$ found in ${}^tQ$, which should be as small as possible.

Example 8 Criteria 2 and 3 are optimal for the transmuted query in example 7. Indeed, this query retrieves both positive tuples *Casanova* and *PrinceCharming*, and does not produce any of the negative tuples *Playboy* or *Shrek*.

3.3.2 Diversity with respect to the initial data

The objective of the rewriting process is to produce a query that is similar to an initial query specified by the user (measurable by the previous criterion), but that also answers the user's exploratory expectation, and thus, presents new tuples to the user. Therefore, it is important that this set of new tuples is not only not empty (equation 4), but also of a suitable size: not too small with respect to the data initially obtained by the user (equation 5), nor comparable to the size of the entire set of tuples (equation 6). If the last condition is not met, the user will most likely have difficulties interpreting the result.

$${}^tQ \cap (\pi_{A_1,\ldots,A_n}(Z) - (Q \cup \pi_{A_1,\ldots,A_n}(\overline{Q}))) \neq \emptyset \tag{4}$$

$$|{}^tQ \cap (\pi_{A_1,\ldots,A_n}(Z) - (Q \cup \pi_{A_1,\ldots,A_n}(\overline{Q})))| \ll |Q| \tag{5}$$

$$|{}^tQ \cap (\pi_{A_1,\ldots,A_n}(Z) - (Q \cup \pi_{A_1,\ldots,A_n}(\overline{Q})))| \ll |\pi_X(Z)| \tag{6}$$

Example 9 The rewritten query ${}^tQ$ from example 7 produces three new tuples *RhetButtler*, *MrDarcy* and *BigBadWolf*, so criterion 4 is fulfilled. These three tuples are numerically comparable with respect to the 2 initial tuples (5), and are less numerous than the ten possible tuples (6).

We give the compact representation of the query rewriting approach in algorithm 2.

Algorithm 2: query rewriting

1 procedure QueryRewriting (Q, d) ;
Input : a query Q, database d
Output: the transmuted query ${}^tQ$
2 *let* $Q = \pi_{A_1,\ldots,A_n}(\sigma_{F_k \wedge F_{\overline{k}}}(R_1 \bowtie \ldots \bowtie R_p))$;
3 $SplitInTrainingAndTestSets(d, out\ trSet, out\ teSet)$;
4 $E_+(Q) := EvaluateQuery(Q, trSet)$;
5 $\overline{Q} := BalancedNegation(|trSet|, |Q|, F_k, F_{\overline{k}}, 1000)$;
6 $E_-(Q) := EvaluateQuery(\overline{Q}, trSet)$;
7 $lSet := BuildLearningSet(E_+(Q), E_-(Q), attr(\overline{F_k}))$;
8 $decisionTree := FindC45(lSet)$;
9 $F_{new} := \bigvee_{b \in decisionTree_+} \bigwedge_{e \in b} e$;
10 return ${}^tQ = \pi_{A_1,\ldots,A_n}(\sigma_{F_{new}}(R_1 \bowtie \ldots \bowtie R_p))$;

4. IMPLEMENTATION AND EXPERIMENTAL RESULTS

We have implemented the proposition made in this paper in `C#` and `SQL Server`, with the C45Learning classifier in the `Accord.NET` library [1], which implements the C4.5 algorithm [29]. We have conducted experiments first to evaluate the accuracy of the generated negation query with respect to an optimal negation query and second, to study the efficiency of the generation of negation queries. Based on previous results [13], we also have briefly described a validation of SQL data exploration with Astrophysicists on a real-life example.

4.1 Scalability and precision

Our proposition makes use of several existing pieces of work (e.g. decision tree, query evaluation) which are not detailed here. We focus on the most difficult part of our proposition, i.e., the heuristic to identify a negation query whose result size is similar to the size of the answer set of the initial query. Quality criteria detailed in section 3.3 require a cohort of users to assess the results and will be addressed in future work.

For a given query, our Knapsack-based heuristic is evaluated with respect to its *accuracy*, how good is the result with respect to the best possible negation query and its *efficiency*.

Both accuracy and efficiency have been studied with respect to different query workloads, a varying number of predicates in the initial SQL query and the values of the `sf` parameter.

Experimental setup.

In our experiments we used two datasets:

- `Iris`: a small well-known dataset describing the properties of some species of iris flowers. The dataset has 150 tuples only, four numerical attributes and one categorical attribute. It was chosen to easily compute (and understand) all the possible negation queries for a given query.
- `Exodata`: a scientific dataset containing 97717 tuples and 62 attributes (see next section for more details).

We generated a query workload as follows: for a fixed number of predicates in a query (from 1 to 200), a predicate of the form A *bop* *value* is generated by randomly choosing an attribute A, the operator *bop* from a list of possibilities ($\{=\}$ for categorical attributes, $\{<, <=, >, >=\}$ for numerical attributes), and the corresponding value $value \in Dom(A)$ for attribute A.

We assume to have statistics available for each attribute in the database, and hence the size of the database does not interfere with the performance of the algorithm.

To assess the distance between the negation proposed by our heuristic and the "best negation" for a query, we proceeded as follows: for a given query Q, we computed all its negations $\overline{Q}$; it has been indeed possible since the number of predicates remained small on our workloads.

We denote by $\overline{Q}_T$ the negation query closest in size to Q and by $\overline{Q}_K$ our approximated negation of Q.

The **distance** between $\overline{Q}_K$ and $\overline{Q}_T$ is defined by

$$abs(|\overline{Q}_K| - |\overline{Q}_T|)/|Z|$$

where $|Z|$ is the size of all possible tuples. The closer the distance is to zero, the better the heuristic (and conversely, the closer to one, the worse the heuristic).

Experiment 1.

To evaluate the impact of the *number of predicates* on the **accuracy** and *computation time* of the approximated negation, we fixed the scale factor `sf`= 1000 and we generated a query workload of 10 random queries with 1 to 9 predicates.

A *query type* is defined by its number of predicates. For each query type, we processed 10 queries and we displayed several values: the minimum distance, the first quartile, the third quartile and the maximum distance via a box plot. The average values for the distance are also given.

Figure 3 (top-left) shows the distance between the proposed heuristic and the closest negation (accuracy). For three predicates, we have a very bad result for one generated query (distance around 0.84) but, on average, the errors are around 0.2, which remains acceptable. The results show that the more predicates a query has, the better the heuristic is, very close to the best solution. With more than 6 predicates, the heuristic turns out to be very precise. Accuracy for both datasets is always excellent whenever the number of predicates exceeds six.

Figure 3 (top-right) shows good performances for the proposed heuristic on both datasets, always less than 0.2s.

Experiment 2.

The second test evaluates the impact of the **scale factor** sf on the **accuracy** of the approximated negation. The scale factor `sf` varies between 1 and 10000. As in the previous experiment, we used a query workload with 10 random queries for each type of test defined by the number of predicates (between 5 and 20) and different values of sf.

In our tests on *Exodata*, we observe that the accuracy is affected by different values of `sf`. For the same number of predicates, as the value of `sf` is increasing, the accuracy

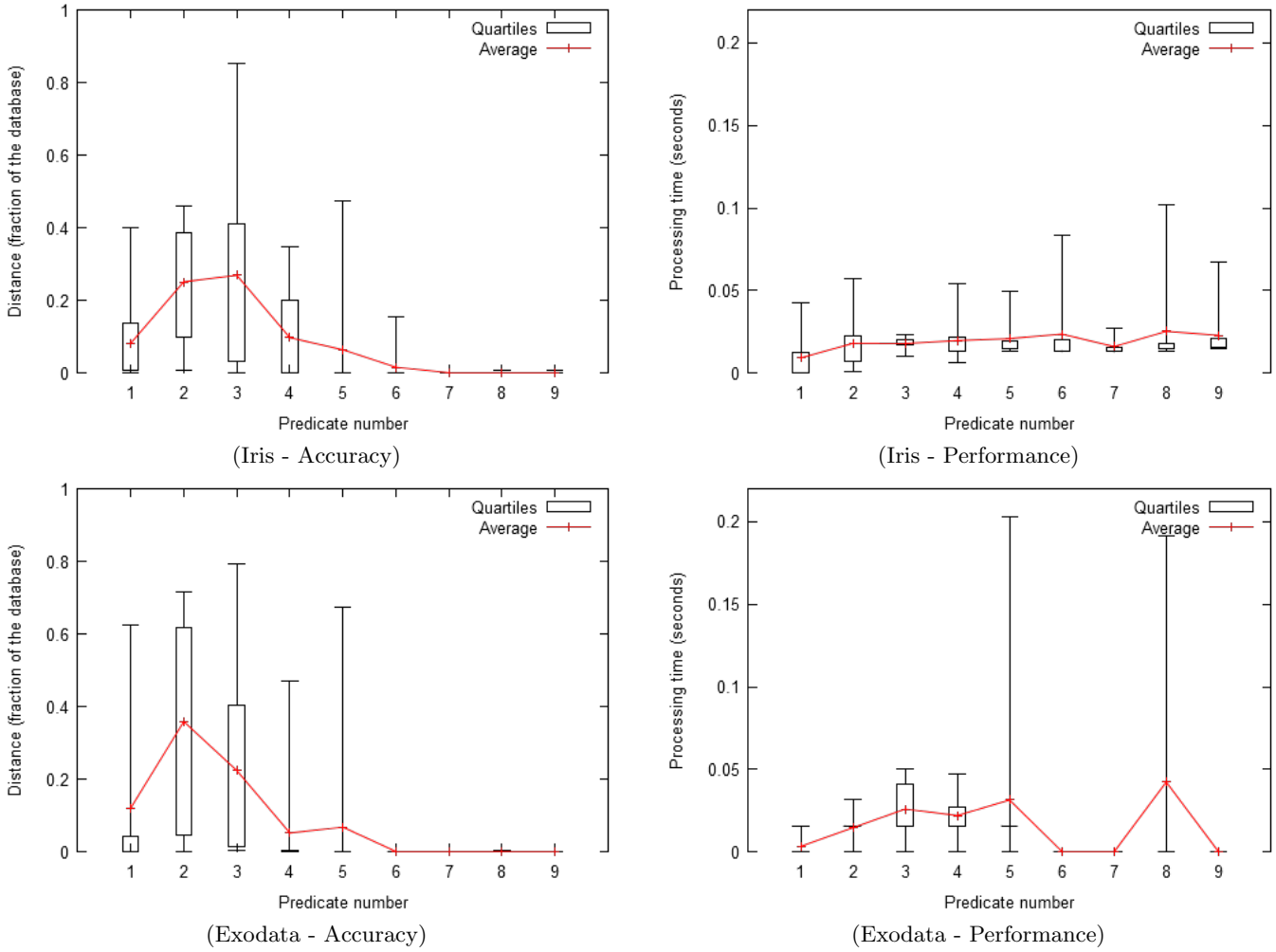


Figure 3: Impact of the number of predicates on the accuracy and computation time of the approximated negation w.r.t. *Iris* dataset (top) and *Exodata* dataset (bottom).

is also improving. Whenever the value of `sf` exceeds 1000, the heuristic behaves very well (distance gets closer to 0, see figure 4-left).

Experiment 3.

As expected, the scale factor `sf` has an influence on the processing time. As seen in experiment 2, a greater value for `sf` allows a better approximation for the negation in our heuristic, but the search space increases. For a large number of predicates we execute the same tests only on the `Exodata` schema in order to estimate the overhead introduced by our heuristic in computing the negation for a given query. We observe that the processing time is increasing with the number of predicates in the original query and the value of `sf` (figure 4-right). However the processing time remains around 1 second for a query with 200 predicates and $sf = 10000$.

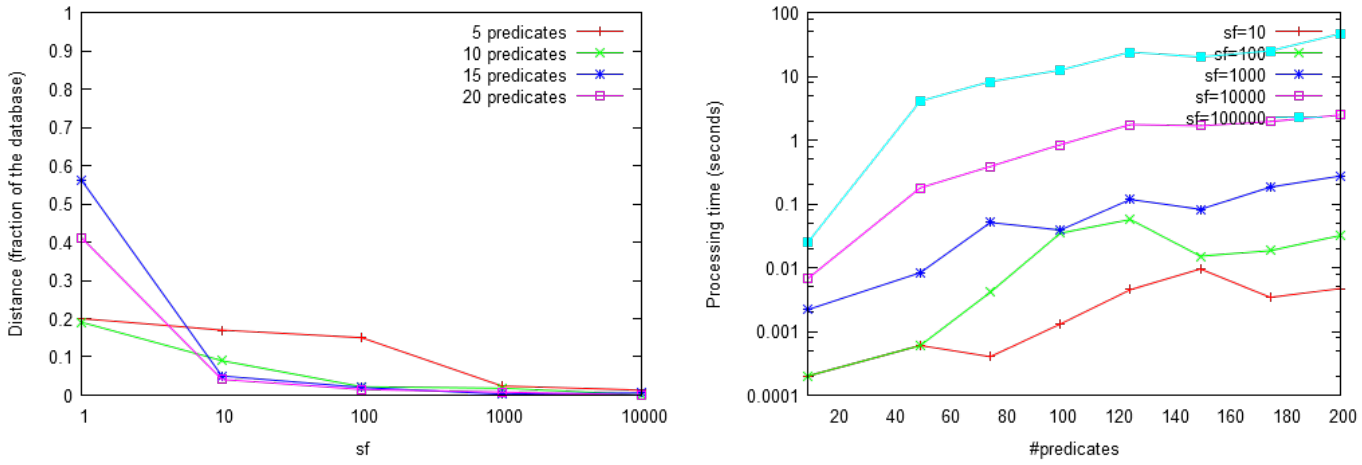
4.2 Validation with astrophysicists

We describe the validation conducted on an astrophysics database derived from the European project `CoRoT`[3] (COnvection, ROtation and planetary Transits), which studies star seismology and searches for extra-solar planets. `CoRoT` has observed for years the stars in our galaxy in order to study them and to discover planets beyond our solar system. A sample of the `EXODATA`[4] database was extracted into one table (`EXOPL`) with 97717 tuples and 62 attributes. A tuple represented a star and attributes included the position of the star, its magnitude at different wavelengths, the degree of its activity, etc. A special attribute `Object` described the presence of planets around the star - value `p`, the absence of planets - value `E`, or the lack of knowledge concerning possible revolving planets - `NULL`.

Most of the stars had not been classified, having the `NULL` value for the `Object` attribute. The objective was to obtain a set of stars that potentially harbor planets. We wanted to identify some conditions that allow to infer the presence of planets for stars that have not yet been studied, starting from the stars for which the presence or absence of planets has been confirmed. With astrophysicists, the initial query was thus very simple to identify:

[3]http://smsc.cnes.fr/COROT/

[4]http://cesam.lam.fr/exodat



(Queries on Exodata) (Queries on Exodata schema)

Figure 4: Impact of the scale factor sf on the accuracy of the approximated negation w.r.t. *Exodata* dataset (left) and the computation time overhead needed to find this negation on *Exodata* schema (right).

```
SELECT DEC, FLAG, MAG_V, MAG_B, MAG_U
FROM EXOPL
WHERE OBJECT = 'p'
```

The negation query was straightforward to obtain (without the machinery introduced in this paper), simply by changing the condition to `OBJECT = 'E'` (see [13] for a detailed discussion). There were 50 positive examples (`OBJECT = 'p'`) and 175 counter-examples (`OBJECT = 'E'`) among the 97717 tuples in the database. Discussions with astrophysicists emphasized different magnitude and amplitude attributes as pertinent attributes to learn on. They hold data concerning observed light under a variety of wavelength filters. Starting from this expert but easily exploitable information, we tried out, in a couple of minutes, several sets of attributes on which to learn. The expert selected attributes `MAG_B, AMP11, AMP12, AMP12, AMP13` and `AMP14`. The learning phase was then launched and generated a decision tree from which the following new condition was extracted: `MAG_B > 13.425 AND AMP11 <= 0.001717`, easily leading to the following transmuted query:

```
SELECT *
FROM EXOPL
WHERE MAG_B > 13.425 AND AMP11 <= 0.001717
```

It is worth noting that such an SQL query had very little chance of germinating in the initial stage of data exploration. This new query identified 22% of the initial positive examples, 0% of the negative examples and 1337 new tuples. These new tuples, representing stars around which the presence of revolving planets has not been studied, could thus be priority study targets due to their proximity, in the data exploration space, to a subset of stars around which planet presence has been confirmed.

The new SQL query turned out to be itself interesting: it showed the detectability limits of current instruments: for magnitudes greater than 13.425, i.e., for dimmer stars, it's mandatory to have star variability amplitudes less than 0.001717, i.e., the light emitted by the star must have a small variability.

Astrophysics scientists have found our approach satisfactory and easy to use, all of them having basic knowledge in SQL. The proposed transmuted query was a contribution per se. The use of machine learning techniques behind the scenes for proposing new SQL queries was completely transparent for them and very much appreciated. Clearly, other validations with domain experts should be conducted to assess the interest of our approach for data exploration in SQL, but these first results were very encouraging.

5. RELATED WORK

Data exploration is a very active research field in databases, data mining and machine learning. This section is organized according to several themes, namely query exploration, recommendation-based exploration, query by output and why-not queries.

Query exploration [25] shows that the time spent to assemble an SQL query is significantly higher than the query's execution time, even for SQL experts and decent-sized data. This is even more true in discovery-oriented applications, where the user does not know very well neither the database (schema / content), nor exactly what she is looking for (the right / exact queries to pose). [25] proposes a set of principles for a guided interaction paradigm, using the data to guide the query construction process. The user successively refines the query considering the results obtained at each iteration in order to reach a satisfactory solution.

[20] theorizes a new class of *exploration-driven applications*, characterized by *exploration sessions* with several interlinked queries, where the result of a query determines the formulation of the next query. [10] also talks about *interactive data exploration* applications characterized by *human-in-the-loop* analysis and exploration. It advocates the need for systems that provide session-oriented usage patterns, with sequences of related queries, where each query is the starting gate for the next one.

Query morphing [21] proposes a technique in which the user is presented with extra data via small changes of the initial query. By contrast, our solution exploits the examples and counter-examples sets obtained from the user's initial query, to obtain a new query via rules learned from these sets. We believe it is most of the time quite difficult to

define *small* modifications for a given SQL query.

Some database exploration approaches are based on manual labeling of examples and counter-examples [34, 8, 30, 14, 23], on which machine learning approaches are eventually used to generate queries. Manual labeling has two main drawbacks: it is quickly tiring for the user, reducing system interactivity, simplicity and attractiveness; and it is far from trivial when the user does not know the data very well (frequently the case in an exploration task) or if the labeling criteria are complex (again, frequently the case, otherwise the user could probably obtain the data with a simple query).

Recommendation-based exploration [17] describes a template-based framework that recommends SQL queries as the user is typing in keywords. A top-k algorithm suggests queries derived from the queryable templates identified as relevant to the keywords provided by the user, based on a probabilistic model. The QueRIE framework [16] uses Web recommendation mechanisms to assist the user in formulating new queries. A query expressed by a user is compared with similar queries in the system log and a set of recommendations are proposed based on the behaviour of other users in a similar context. [6, 5] help the user formulate quantified, exploratory SQL queries. SnipSuggest [22] provides users with context-aware SQL query suggestions based on a log of historical queries. [15] assists the user in the exploratory task by suggesting additional *YMAL* items, not part of, but highly correlated with the results of an original query. It exploits offline computed statistics to identify subtuples appearing frequently in the original result, then builds exploratory queries, guiding the user to different directions in the database, not included in the original query. While we also make use of common statistics maintained by the optimizer, we incur no overhead by computing any other statistics. The entire process of aiding the user formulating her queries unfolds online.

Query by output solutions find a query that produces the data specified by the user, no more, no less. [31] finds an alternative path in the schema whose corresponding query produces the same data; an initial query may or may not be specified. The reformulated query in our approach does overlap the initial one to some extent, but is not equivalent to it. Similarly, [33] finds a join query, given its output, using arbitrary graphs. By contrast, [30] searches for a minimal valid project-join query starting from a few example tuples specified by the user. The generated query is expected to produce extra tuples, just like in our approach. The approach is however based on the user manually introducing example tuples. Likewise, [28] discovers queries whose answers include user-specified examples, for sample-driven schema mapping. [27] follows a similar approach to [30], but generates and ranks multiple queries that can partially contain the input tuples supplied by the user.

Why-not queries A database exploration approach tries to explain why a query on a database does not return desired tuples in the response, initially proposed in [11]. The framework proposed in [11] identifies the components from the guery evaluation plan responsible for filtering desired data items. This approach has been studied for different types of queries, as for reverse top-k queries [18]. As an alternative idea, the data provenance may be useful in our context by explaining why some tuples are included in the proposed exploration [9].

6. CONCLUSION AND DISCUSSION

In this paper we presented an approach that shows to be very promising in tackling one of the Big Data challenges: formulate SQL queries that correspond to what the analyst searches for and that efficiently execute on data of huge size. Starting from a query issued by a non-expert user, we explore the space of corresponding negation queries and we choose the one whose result size is as close as possible to the initial query's result size using a Knapsack-based heuristic. We then obtain a balanced set of examples and counter-examples that can feed a decision tree learning process. The learned model allows to directly rewrite the initial query using the obtained rules. This new transmuted query returns results that are similar to the initial query's ones, while also producing new tuples, which again are similar to the ones returned by the initial query. Such a form of diversity would have been practically impossible to achieve without the help of machine learning to formulate the query.

The user can also assess the global quality for the rewriting of her query, using diverse criteria that compare for instance the percentage of the new data obtained, or the number of tuples from the initial query that are retrieved. She can thus quickly evaluate the direction of her exploration, without having to first interpret the data she obtains for each query.

We have implemented and conducted several experiments to evaluate the main technical contribution of the paper, i.e., the Knapsack-based heuristic. Results are quite promising both in term of accuracy and performance. Preliminary validation results obtained on an astrophysics dataset with a simpler version of our current prototype [13], that automatically obtains a negation query from an initial so-called *discriminatory query*, are promising and open the way for an extensive experimental study.

The transparent integration of learning techniques with SQL is a significant change in scientists' way of working. The user just explores the data by posing questions in the golden SQL query language, both easy to use and intuitive. The user could therefore focus on the science part instead of the computational one. She does not have to switch between various systems and to re-load data several times, rending the exploration process "seamless".

A large number of possibilities has been opened up by this approach. We can easily imagine its extension to other types of SQL queries or to evaluate the scalability on very large datasets. We can also extend this work to pattern mining based on declarative languages like RQL [**?**], or, more generally, to all pattern types. For a given hypothesis (or pattern or query), the notion of examples and counter-examples seems to be relatively universal. This type of approach could thus play an interesting role in Big Data exploration in the years to come.

Acknowledgments: This work has been partially supported by `Petasky` and `QualiSky` projects under the CNRS `Mastodon` program. We thank the anonymous reviewers for their valuables remarks and Fabien Rouge and Christian Surace for their feedbacks on preliminary version of this work.

7. REFERENCES

[1] Accord .NET Framework. http://accord-framework.net/.

[2] Large Synoptic Survey Telescope. http://www.lsst.org//. [Online; accessed 11-Dec-2016].

[3] European Cooperation in the field of Scientific and Technical Research. Memorandum of understanding. http://w3.cost.eu/fileadmin/domain_files/TDP/Action_TD1403/mou/TD1403-e.pdf, 2014. [Online; accessed 11-Dec-2016].

[4] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[5] A. Abouzied, D. Angluin, C. H. Papadimitriou, J. M. Hellerstein, and A. Silberschatz. Learning and verifying quantified boolean queries by example. In *PODS*, pages 49–60, 2013.

[6] A. Abouzied, J. M. Hellerstein, and A. Silberschatz. Playful Query Specification with DataPlay. *PVLDB*, 5(12):1938–1941, Aug. 2012.

[7] R. Bellman. Notes on the theory of dynamic programming iv-maximization over discrete sets. *Naval Research Logistics Quarterly*, 3(1-2):67–70, 1956.

[8] A. Bonifati, R. Ciucanu, and S. Staworko. Interactive Inference of Join Queries. In *EDBT*, pages 451–462, 2014.

[9] P. Buneman, S. Khanna, and T. Wang-Chiew. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330. Springer Berlin Heidelberg, 2001.

[10] U. Çetintemel, M. Cherniack, J. DeBrabant, Y. Diao, K. Dimitriadou, A. Kalinin, O. Papaemmanouil, and S. B. Zdonik. Query Steering for Interactive Data Exploration. In *CIDR*, 2013.

[11] A. Chapman and H. Jagadish. Why not? In *SIGMOD*, pages 523–534. ACM, 2009.

[12] B. Chardin, E. Coquery, M. Pailloux, and J. Petit. RQL: A SQL-Like Query Language for Discovering Meaningful Rules. In *ICDM*, pages 1203–1206, 2014.

[13] J. Cumin, J. Petit, F. Rouge, V. Scuturici, C. Surace, and S. Surdu. Requêtes discriminantes pour l'exploration des données. In *Extraction et Gestion des Connaissances*, pages 195–206, 2016.

[14] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Explore-by-example: an automatic query steering framework for interactive data exploration. In *SIGMOD*, 2014.

[15] M. Drosou and E. Pitoura. YmalDB: Exploring Relational Databases via Result-driven Recommendations. *The VLDB Journal*, 22(6):849–874, Dec. 2013.

[16] M. Eirinaki, S. Abraham, N. Polyzotis, and N. Shaikh. Querie: Collaborative database exploration. *IEEE TKDE*, 26(7):1778–1790, 2014.

[17] J. Fan, G. Li, and L. Zhou. Interactive SQL Query Suggestion: Making Databases User-friendly. In *ICDE*, ICDE '11, pages 351–362, 2011.

[18] Y. Gao, Q. Liu, G. Chen, B. Zheng, and L. Zhou. Answering why-not questions on reverse top-k queries. *PVLDB*, 8(7):738–749, 2015.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[20] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of Data Exploration Techniques. In *SIGMOD*, pages 277–281, 2015.

[21] M. L. Kersten, S. Idreos, S. Manegold, and E. Liarou. The Researcher's Guide to the Data Deluge: Querying a Scientific Database in Just a Few Seconds. *PVLDB*, 4(12):1474–1477, 2011.

[22] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: Context-aware autocompletion for SQL. *PVLDB*, 4(1):22–33, 2010.

[23] H. Li, C. Chan, and D. Maier. Query From Examples: An Iterative, Data-Driven Approach to Query Construction. *PVLDB*, 8(13):2158–2169, 2015.

[24] A. Marchetti-Spaccamela and G. Romano. On different approximation criteria for subset product problems. *Information processing letters*, 21(4):213–218, 1985.

[25] A. Nandi and H. V. Jagadish. Guided Interaction: Rethinking the Query-Result Paradigm. *PVLDB*, 4(12):1466–1469, 2011.

[26] C. T. Ng, M. S. Barketau, T. C. E. Cheng, and M. Y. Kovalyov. "Product Partition" and related problems of scheduling and systems reliability: Computational complexity and approximation. *European Journal of Operational Research*, 207(2):601–604, 2010.

[27] F. Psallidas, B. Ding, K. Chakrabarti, and S. Chaudhuri. S4: Top-k Spreadsheet-Style Search for Query Discovery. In *SIGMOD*, pages 2001–2016, 2015.

[28] L. Qian, M. J. Cafarella, and H. V. Jagadish. Sample-driven schema mapping. In *SIGMOD*, pages 73–84, 2012.

[29] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[30] Y. Shen, K. Chakrabarti, S. Chaudhuri, B. Ding, and L. Novik. Discovering queries based on example tuples. In *SIGMOD*, pages 493–504, 2014.

[31] Q. T. Tran, C. Chan, and S. Parthasarathy. Query by output. In *SIGMOD*, pages 535–548, 2009.

[32] D. Victor. The Ashley Madison Data Dump, Explained. New York Times, 2015-08-20, 2015.

[33] M. Zhang, H. Elmeleegy, C. M. Procopiuc, and D. Srivastava. Reverse Engineering Complex Join Queries. In *SIGMOD*, SIGMOD '13, pages 809–820, New York, NY, USA, 2013. ACM.

[34] M. M. Zloof. Query-by-example: A Data Base Language. *IBM Syst. J.*, 16(4):324–343, Dec. 1977.