Scorpion: Explaining Away Outliers in Aggregate Queries

Yujia Fan

11/07/2019

What is Scorpion?

Scorpion is a system that takes a set of user-specified outlier points in an aggregate query result as input and finds predicates that explain the outliers in terms of properties of the input tuples that are used to compute the selected outlier results.

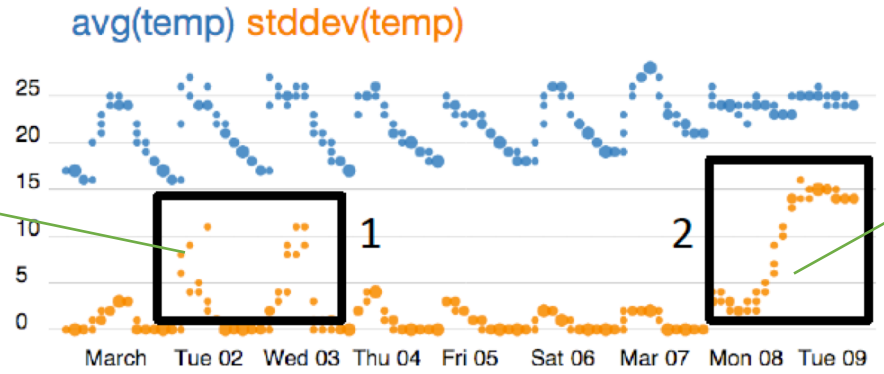
More understandably, *Scorpion* is a python library that helps answer these “why” questions. The user simply selects examples of outlier and normal results and Scorpion will look for subsets of the input table that potentially explains the outliers and finds a combination of attribute values (predicate) that describes that subset.

To find such predicates, *Scorpion* develops a score for how much influence a predicate on the outliers, and design efficient algorithms to find predicates with lots of influence.

Why Scorpion?

For example, Figure 1 shows a visualization of data from the Intel Sensor Data Set. Each point represents an aggregate(either mean or standard deviation) of data over an hour from 61 sensors.

The standard deviation fluctuates heavily. Why?



The temperature stops oscillating. Why?

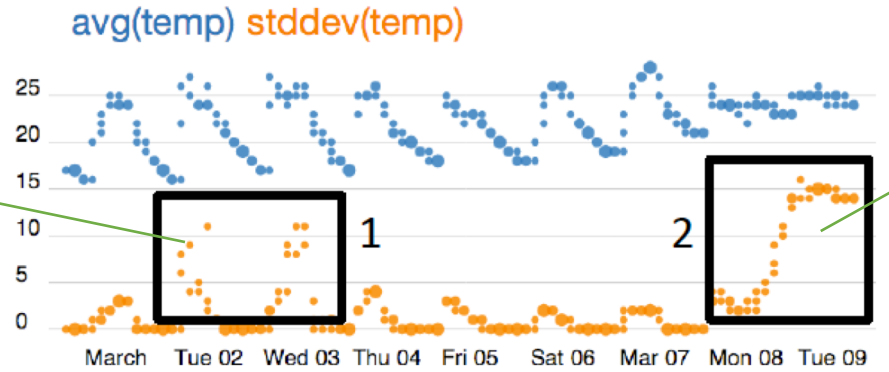
Figure 1: Mean and standard deviation of temperature readings from Intel sensor dataset.

Why Scorpion?

In this case, it turns out that Region 1 is due to sensors near windows that heat up under the sun around noon, and the Region 2 is by another sensor running out of energy (indicated by low voltage) that starts producing erroneous readings.

These facts are not obvious from the visualization and require manual inspection of the attributes of the readings that contribute to the outliers to determine what is going on.

Because the sensor is near windows



Because the sensor is running out of energy.

Figure 1: Mean and standard deviation of temperature readings from Intel sensor dataset.

Problem Setup

Scorpion seeks to find a predicate over an input dataset that most influences a user selected set of query output. So we have to define influence, and also additional information needed in specifying the problem.

Notation	Description
D	The input relational table with attributes $attr_1, \cdots, attr_k$
A_{gb}, A_{agg}	Set of attributes referenced in GROUPBY and aggregation clause
$p_i \prec_D p_j$	Result set of p_i is a subset of p_j when applied to D
α	The set of aggregate result tuples, α_i's
g_{α_i}	Tuples in D used to compute α_i e.g., have same GROUPBY key
O, H	Subset of α in outlier and hold-out set, respectively
v_{α_i}	Normalized error vector for result α_i

User-specified

Consider a single relation D, with attributes $A = attr_1, attr_2, .., attr_k$. Let Q be a group-by SQL query grouped by attributes $A_{gb} \in A$, with a single aggregate function, $agg()$, that computes a result using aggregate attributes $A_{agg} \in A$. $A_{gb} \cap A_{agg} = \emptyset$. Finally, we have $A_{rest} = A - A_{agg} - A_{gb}$ be the attributes that are used to construct the explanations.

Problem Setup

A predicate, p, is a conjunction of range clauses over the continuous attributes and set containment clauses over the discrete attributes, where each attribute is present in at most one clause.

$p_i \prec_D p_j$	Result set of p_i is a subset of p_j when applied to D

Let $p(D) \subseteq D$ be the set of tuples in D that satisfy p. A predicate p_i is contained in p_j with respect to a dataset D if the tuples in D that satisfy p_i are a subset of those satisfying p_j : $p_i \prec_D p_j$ and $p_i(D) \subset p_j(D)$. Let P_A be the space of all possible predicates over the attributes in A.

Problem Setup

The user can also specify an error vector that describes how an outlier result looks wrong (e.g., temperature is too high). v_{O_i} is a normalized error vector for the result, O_i that points in the direction of the error.

Result id	Time	AVG(temp)	Label	v
α_1	11AM	34.6	Hold-out	-
α_2	12PM	56.6	Outlier	$< -1 >$
α_3	1PM	50	Outlier	$< -1 >$

Table 2: Query results (left) and user annotations (right)

For example, if the user thinks that α_1 of Q1 is too low, she can define the vector $v_{\alpha_1} =< -1 >$, whereas she can specify that α_2 is too high using $v_{\alpha_2} =< 1 >$.

Predicate Influence

Suppose the model is an aggregate function, agg, that takes a set of tuples, g_o, as input, and outputs a result, o. The influence of a predicate, p, on o depends on the difference between the original result $o.res$ and the updated output after deleting $p(g_o)$ from g_o.

$$\Delta_{agg}(o,p) = agg(g_o) - agg(g_o - p(g_o))$$

The influence is defined as the ratio between the change in the output and the number of tuples that satisfy the predicate:

$$inf_{agg}(o,p) = \frac{\Delta o}{\Delta g_o} = \frac{\Delta_{agg}(o,p)}{|p(g_o)|}$$

Predicate Influence

In the Intel sensor dataset, suppose the individual influences of each tuple in $g_{\alpha_2} = \{T4, T5, T6\}$. Base on that,

$$inf_{agg}(\alpha_2, \{T4\}) = \frac{\Delta\alpha_2}{\Delta g_o} = \frac{\Delta_{agg}(o,p)}{|p(g_o)|} = \frac{56.6 - 67.5}{1} = -10.8$$

Result id	Time	AVG(temp)	Label	v
α_1	11AM	34.6	Hold-out	-
α_2	12PM	56.6	Outlier	$< -1 >$
α_3	1PM	50	Outlier	$< -1 >$

Tuple id	Time	SensorID	Voltage	Humidity	Temp.
T1	11AM	1	2.64	0.4	34
T2	11AM	2	2.65	0.5	35
T3	11AM	3	2.63	0.4	35
T4	12PM	1	2.7	0.3	35
T5	12PM	2	2.7	0.5	35
T6	12PM	3	2.3	0.4	100
T7	1PM	1	2.7	0.3	35
T8	1PM	2	2.7	0.5	35
T9	1PM	3	2.3	0.5	80

α_2 → $\frac{100 + 35}{2} = 67.5$

Predicate Influence

Error Vector:

The previous formulation does not take into account the error vectors. For example, if the user thinks that the average temperature was too low, then removing T6 would, contrary to the user's desire, further decrease the mean temperature.

This intuition suggests that only the components of the basic definition that align with the error vector's direction should be considered.

$$inf_{agg}(o, p, v_o) = inf_{agg}(o, p) * v_o$$

Hold-out Result:

The user may select a hold-out result, h, that the returned predicate should not influence. Intuitively, p should be penalized if it influences the hold-out results in any way.

$$inf_{agg}(o, p, v_o) = \lambda inf_{agg}(o, p, v_o) - \underbrace{(1 - \lambda)|inf_{agg}(h, p)|}_{\text{penalty}}$$

where λ is a parameter that represents the importance of not changing the value of the hold-out set.

Predicate Influence

Multiple Results:

The user will often select multiple outlier results, O, and hold-out results, H. We extend the notion by averaging the influence over the outlier results and penalizing the maximum influence over the hold-out set:

$$inf_{agg}(O, H, p, V) = \lambda \frac{1}{|O|} \sum_{o \in O} inf_{agg}(o, p, v_o) - (1 - \lambda) \max_{h \in H} |inf_{agg}(h, p)|$$

We chose to use the maximum in order to provide a hard cap on the amount that a predicate can influence any hold-out result.

Influential Predicates Problem

The Influential Predicates (IP) Problem is defined as follows: Given a select-project-group-by query Q, and user inputs O, H, and V , find the predicate, p^*, from the set of all possible predicates, $P_{A_{rest}}$, that has the maximum influence:

$$p^* = \arg \max_{p \in P_{A_{rest}}} inf(p)$$

Core Problem

Influential Predicates Problem

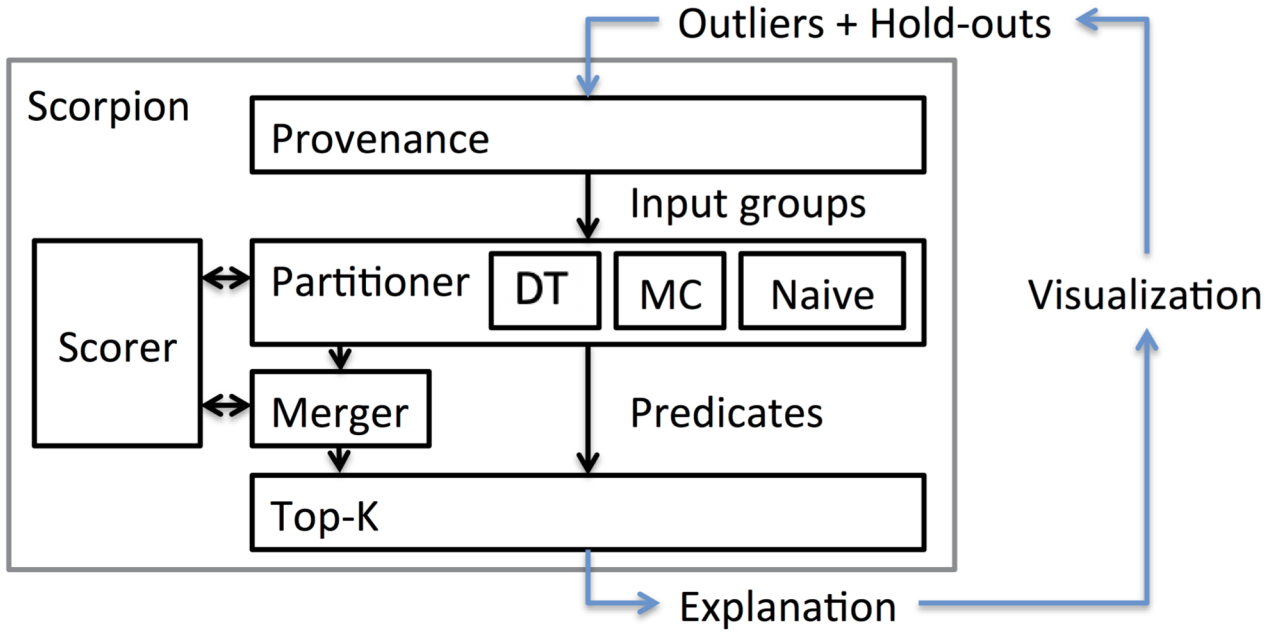
- Scorpion needs to consider how combinations of input tuples affect the outlier results, which depends on properties of the aggregate function. Possible predicates maybe exponential in the number of and cardinalities of attributes.

- Scorpion returns predicates, rather than individual tuples, to provide the user with understandable explanations of anomalies in the data.

- The influence of a predicate relies on statistics of the tuples in addition to their individual influences, and the specific statistic depends on the particular aggregate function. For example, AVG depends on both the values and density of tuples.

- In the presence of a hold-out set, simple greedy algorithms may not work because an influential set of tuples in the outlier set may also influence the hold-out results.

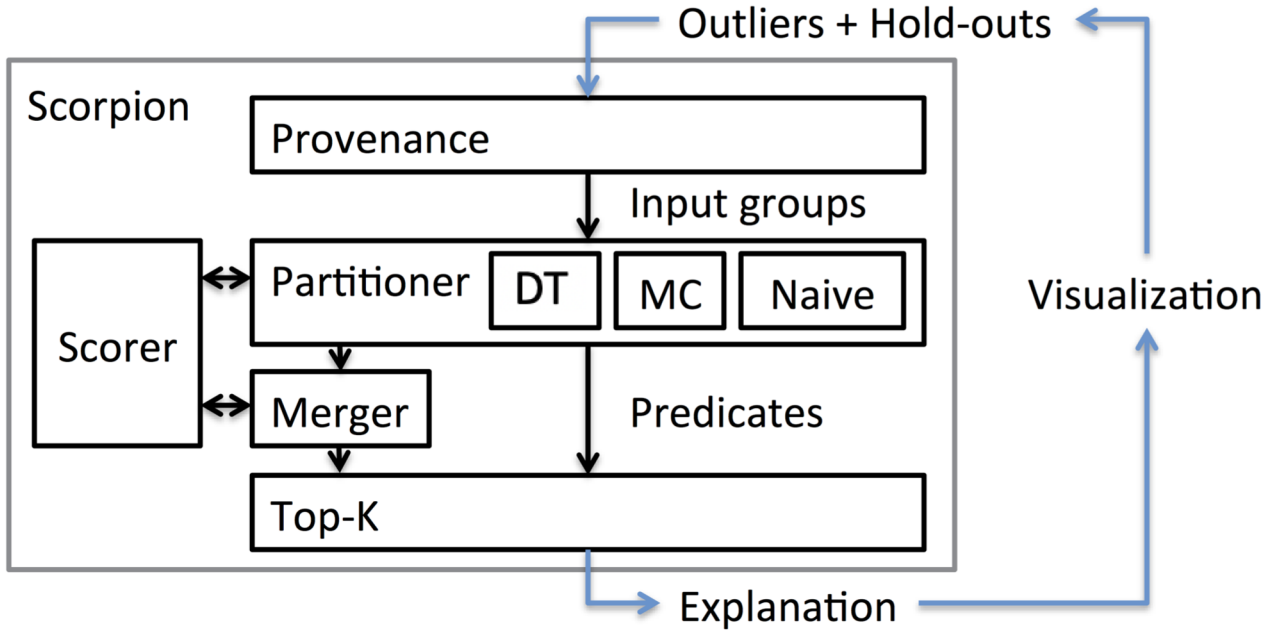
Scorpion Architecture



Provenance:

Scorpion first uses the *Provenance* component to compute the provenance of the labeled results and returns their corresponding input groups. In this work, the queries are group-by queries over a single table, so computing the input groups is straightforward. The input groups, along with the original inputs, are passed to the *Partitioner*.

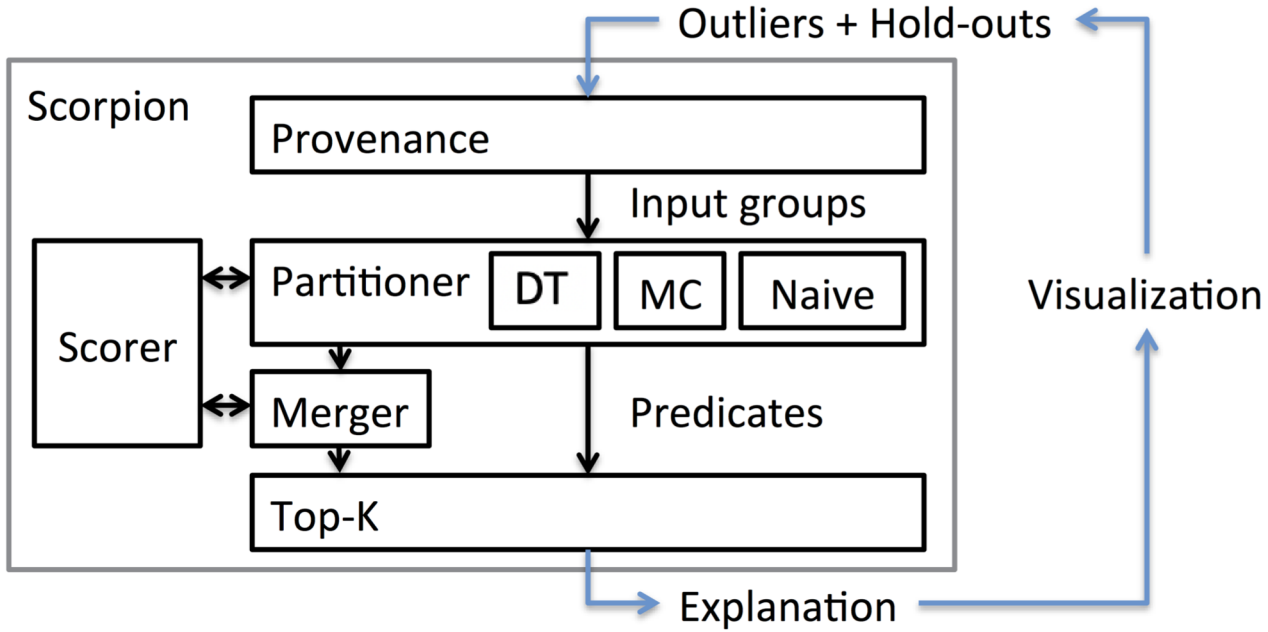
Scorpion Architecture



Partitioner:
Partitioner chooses the appropriate partitioning algorithm based on the properties of the aggregate. The algorithm generates a ranked list of predicates, where each predicate is tagged with a score representing its estimated influence.

Merger:
Merger greedily merges similar predicates as long as it increases the influence.

Scorpion Architecture



Scorer:
The Partitioner and Merger send candidate predicates to the Scorer, which computes the influence as defined in the previous section.

In the whole process, the need to read the entire dataset to evaluate the influence is overly expensive if the dataset is large, or the aggregate function needs multiple passes over the data. Some properties of aggregate can reduce these costs.

Aggregate Properties

Useful in optimizations

Incrementally Removable*:*

In general, a computation is *incrementally removable* if the updated result of removing a subset, s, from the inputs, D, can be computed by only reading s. For example, SUM is incrementally removable because $SUM(D - s) = SUM(D) - SUM(s)$.

Independent*:*

The *independence* property allows Scorpion to assume that the input tuples influence the aggregate result independently.

Useful in MC algorithm

1. $t_a < t_b \rightarrow inf_{\mathcal{F}}(T \cup \{t_a\}) < inf_{\mathcal{F}}(T \cup \{t_b\})$ and
2. $t_a > t^* \rightarrow inf_{\mathcal{F}}(T \cup \{t_a\}) \geq inf_{\mathcal{F}}(T) | t^* = \arg\min_{t \in T} inf_{\mathcal{F}}(t)$

The algorithm first sorts the tuples by influence and then incrementally adds the most influential tuple to the candidate set until the influence of the set does not increase further.

Anti-monotonic Influence*:*

The anti-monotonic property is used to prune the search space of predicates. In general, a property is anti-monotone if, whenever a set of tuples s violates the property, so does any subset of s.

Naïve Partitioner

- For an arbitrary aggregate function without nice properties, it is difficult to improve beyond an exhaustive algorithm that enumerates and evaluates all possible predicates. The Naïve algorithm first defines all distinct single-attribute clauses, then enumerates all conjunctions of up to one clause from each attribute.

- This algorithm is inefficient because the number of single-attribute clauses increases exponentially (quadratically) for a discrete (continuous) attribute as its cardinality increases. Additionally, the space of possible conjunctions is exponential with the number of attributes.

- Naïve computes the influence of each predicate by sending it to the *Scorer*, then the *Merger*, and returns the most influential predicate.

Decision Tree(DT) Partitioner

DT is a top-down partitioning algorithm for independent aggregates. It is based on the intuition that the Δ function of independent operators cannot decrease when tuples with similar influence are combined together.

DT Partitioner is based on regression tree algorithms.

Recursive Partitioning:

DT recursively splits the attribute space of an input group to create a set of predicates. Because outlier groups are different than hold-out groups, DT partition these groups separately, resulting in a set of outlier predicates and hold-out predicates. These are combined into a set of predicates that differentiates ones that only influence outlier results from those that also influence hold-out results.

Decision Tree(DT) Partitioner

The key insight is that it is more important that partitions containing influential tuples be accurate than non-influential partitions, thus the error metric threshold can be relaxed for partitions that don't contain any influential tuples.

Its value is based on the maximum influence in a partition, inf_{max}, and the upper, inf_u, and lower, inf_l, bounds of the influence values in the dataset. The threshold can be computed via any function that decreases from a maximum to a minimum threshold value as inf_{max} approaches inf_u.

Threshold is flexible with different partitions

$$threshold = \omega * (inf_u - inf_l)$$

$$\omega = min(\tau_{min} + s * (inf_u - inf_{max}), \tau_{max})$$

$$s = \frac{\tau_{min} - \tau_{max}}{(1-p) * inf_u - p * inf_l}$$

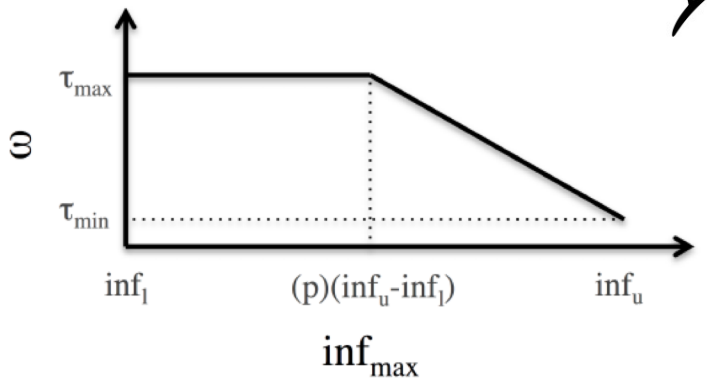


Figure 4: Threshold function curve as inf_{max} varies

Decision Tree(DT) Partitioner

Combining Partitionings

The recursive partitioning step generates an outlier partitioning, $partitions_O$, and hold-out partitioning, $partitions_H$, for their respective input groups and then combine them together into a single partitioning, $partitions_C$.

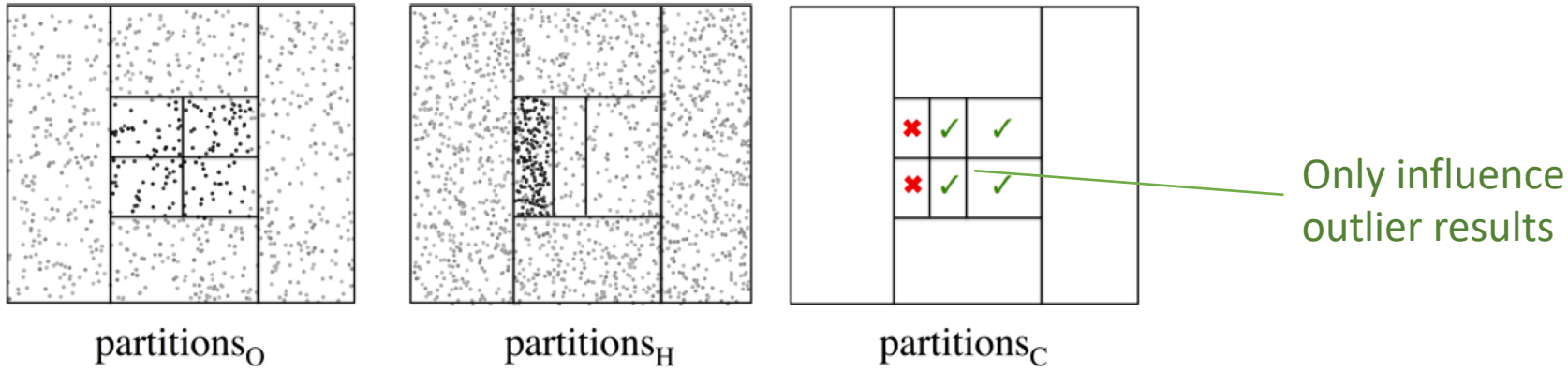


Figure 5: Combined partitions of two simple outlier and hold-out partitionings

For example, $partitions_H$ in Figure 5 contains a partition that overlaps with two of the influential partitions in $partitions_O$.

Bottom-Up(MC) Partitioner

The MC algorithm is a bottom-up approach for independent, anti-monotonic aggregates, such as *COUNT* and *SUM*. It can be much more efficient than DT for these aggregates.

MC Partitioner is similar to algorithms used for subspace clustering.

Subspace clustering problem:

Each iteration computes the intersection of all units kept from the previous iteration whose dimensionality differ by exactly one attribute. Thus, the dimensionality of the units increase by one after each iteration. Non-dense units are pruned, and the remaining units are kept for the next iteration. The algorithm continues until no dense units are left. Finally, adjacent units with the same dimensionality are merged.

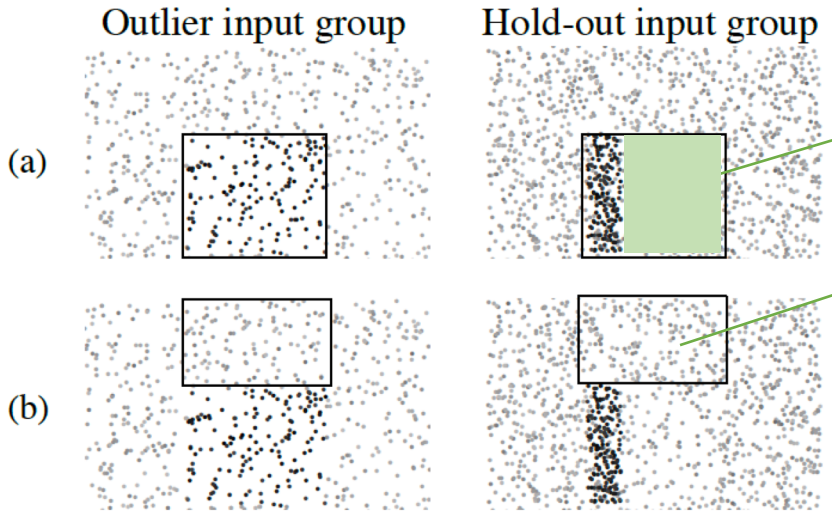
Bottom-Up(MC) Partitioner

Modification 1 :

Merge adjacent units after each iteration to find the most influential predicate. If the merged predicate is not more influential than the optimal predicate so far, then the algorithm terminates.

Modification 2 :

Modify the pruning procedure to account for two ways in which the influence metric is not anti-monotonic.



There must exists a smaller p that only influences the outlier results

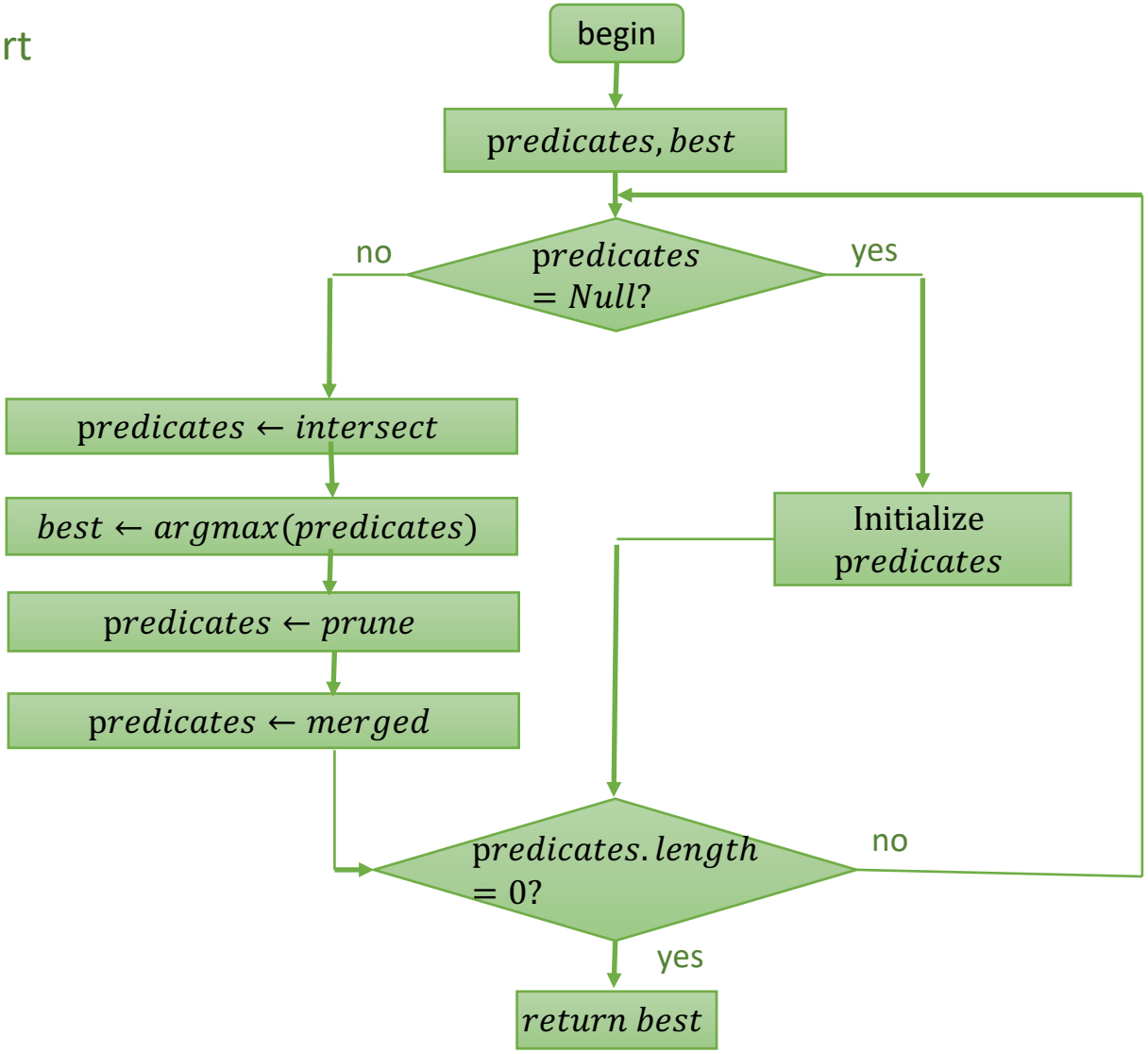
Pruned safely

function PRUNE($predicates$, O, V, $best$)
$ret = \{p \in predicates | \inf(O, \emptyset, p, V) < \inf(best)\}$
$ret = \{p \in ret | \arg\max_{t^* \in p(O)} \inf(t^*) < \inf(best)\}$
return ret

Figure 6: A predicate is not influential if it (a) influences a hold-out result or (b) doesn't influence outlier result.

Bottom-Up(MC) Partitioner

Algorithm Flow Chart



Synthetic Dataset:

SELECT SUM(A_v) FROM synthetic GROUP BY A_d

Drawn from one of three gaussian distributions, depending on if the tuple is normal or outlier

10 distinct A_d values (10 groups), containing 2,000 tuples randomly distributed in n dimensions

$$A_v \begin{cases} \text{Normal tuples} \quad \text{->} \quad N(\,10, 10\,) \\ \text{High-valued outliers} \quad \text{->} \quad N(\,\mu, 10\,) \\ \text{Medium valued outliers} \quad \text{->} \quad N(\,\frac{\mu + 10}{2}, 10\,) \end{cases}$$

μ is a parameter to vary the difficulty of distinguishing normal and outlier tuples. The problem will be harder the closer μ is to 10.
In the experiment, half of the groups (the hold-out groups) exclusively sample from the normal distribution, while the rest (the outlier groups) sample from all three distributions.

Visualization of Synthetic Dataset:

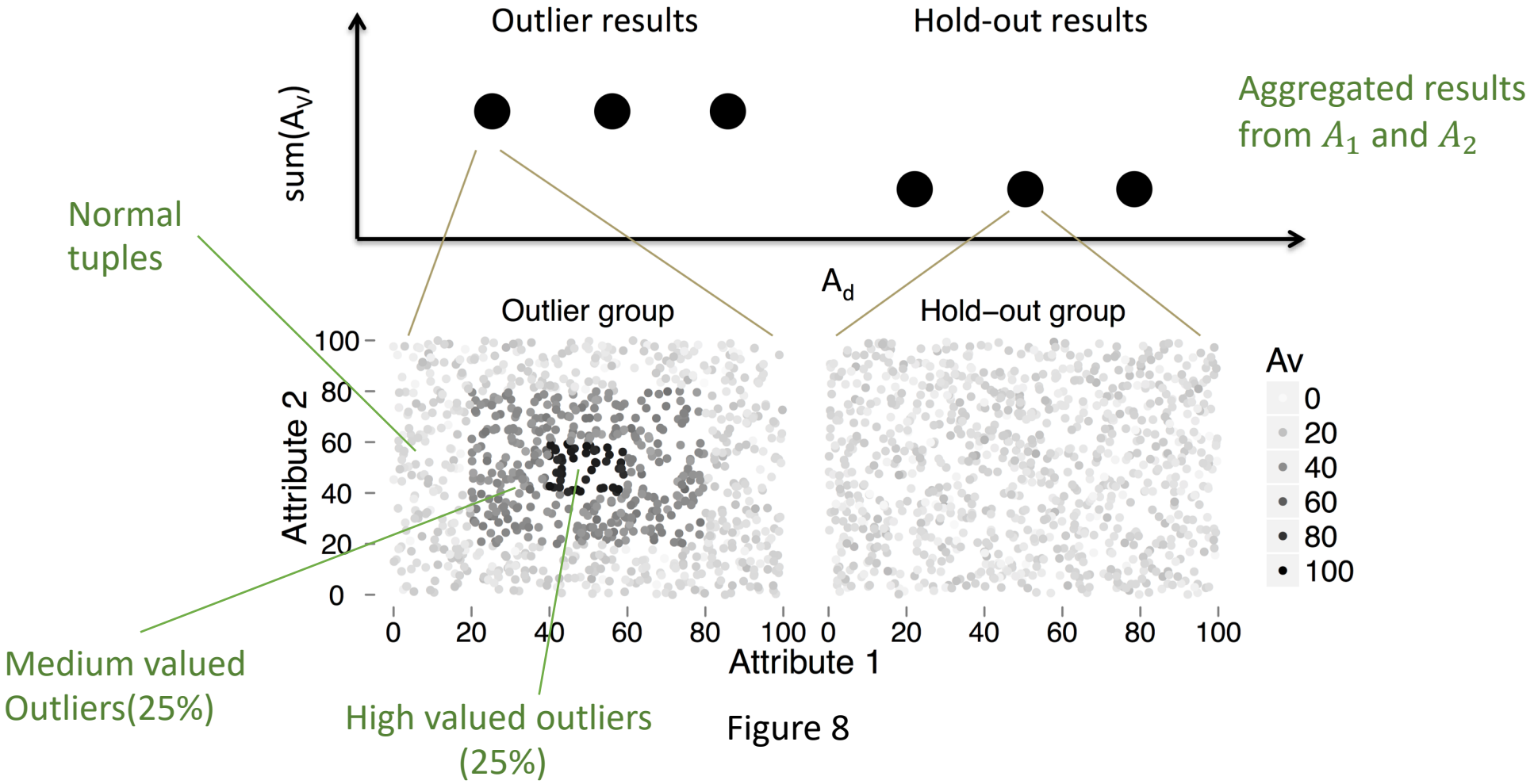


Figure 8

Visualize a synthetic 2D dataset with $\mu = 90$, the outer cube as $A_1 \in [20, 80]$, $A_2 \in [20, 80]$ and the inner cube as $A_1 \in [40, 60]$, $A_2 \in [40, 60]$. A_1 and A_2 are used to generate the explanatory predicates.

Other Datasets:

Intel Sensor Dataset:

Sensorid	Humidity	Light	Voltage

General queries for this experiment are both related to the impact of sensor failures on the standard deviation of the temperature:

SELECT truncate('hour', time) as hour, STDDEV(temp) FROM readings
WHERE *STARTDATE* ≤ time ≤ *ENDDATE* GROUP BY hour → Independent aggregate

For example, the query occurs when a sensor starts to lose battery power, indicated by low voltage readings, which causes above 100 °C temperature readings. The user selects 138 outliers and 21 hold-out results, and indicates that the outliers are too high.

Other Datasets:

Campaign Expenses Dataset:

Recipient	Dollar amount	State	Zip-code	Organization type

The expenses dataset contains all campaign expenses between January 2011 and July 2012 from the 2012 US Presidential Election.

```
SELECT sum(disb_amt)
FROM expenses WHERE candidate = 'Obama'
GROUP BY date
```

→ Independent, anti-monotonic aggregate

The SQL query sums the total expenses per day in the Obama campaign. The typical spending is around $5,000 per day. So we flag 7 outlier days where the expenditures are over $10 M, and 27 hold-out results from typical days.

Experimental Methodology:

The experiments compare Scorpion using the three partitioning algorithms along metrics of Precision, recall, F-score and running. It computes precision and recall of a predicate, p, by comparing the set of tuples in $p(g_O)$ to a ground truth set.

The F-score if defined as the harmonic mean of the precision and recall:

$$F = 2 \times \frac{precision \times recall}{precision + recall}$$

expresses the ability to find all relevant instances in a dataset

Naïve Partitioner:

When the Naïve algorithm has executed for a user specified period of time, it will automatically terminate and return the most influential predicate generated so far. In this paper's experiments, it ran the exhaustive algorithm for up to 40 minutes, and also logged the best predicate at every 10 second interval.

An Additional Knob

This additional knob is to address what to do if the user specifies that an outlier result is “too high”. We modify the basic definition of influence as follows:

$$inf_{agg}(o, p, c) = \frac{\Delta o}{(\Delta g_o)^c}$$

How aggressively Scorpion should reduce the result

The exponent, $c \geq 0$, is a user-controlled parameter that trades off the importance of keeping the size of s small and maximizing the change in the aggregate result. For example, when $c = 0$, Scorpion will reduce the aggregate result without regard to the number of tuples that are used, producing predicates that select many tuples.

Synthetic Dataset Experiment:

Recall that:
$$\Delta\, inf_{agg}(o,p,c) = \frac{\Delta_o}{(\Delta g_o)^c}$$

1. How the c parameter impacts the quality of the optimal predicate.

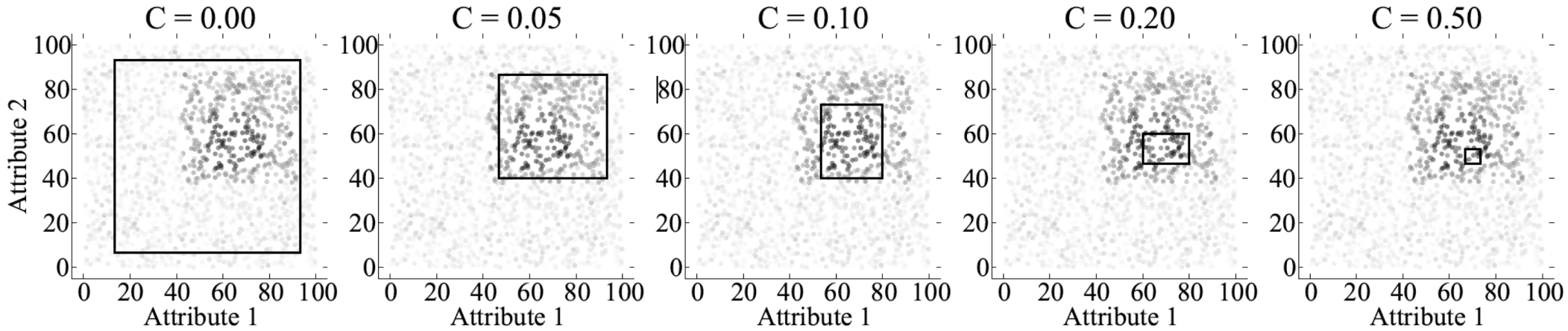
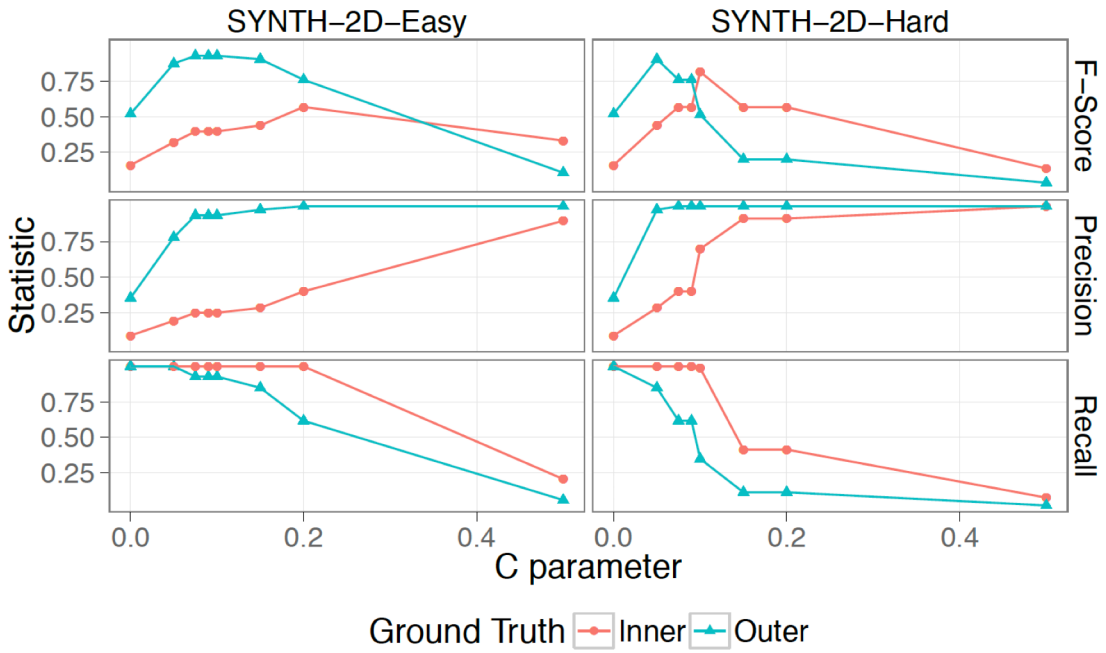


Figure 9 plots the optimal predicate that *Naïve* finds for different c values on the Synthetic-2D-Hard dataset. When $c = 0$, the predicate encloses all of the outer cube, at the expense of including many normal points. When $c = 0.05$, the predicate contains most of the outer cube, but avoids regions that also contain normal points. Increasing c further reduces the predicate and exclusively selects portions of the inner cube.

Synthetic Dataset Experiment:

Accuracy statistics with different c parameters



Inner/outer cubes as the ground truth

Figure 10 plots the accuracy statistics as c increases. As expected, the F-score of the outer curve peaks at a lower c value than the inner curve. This is because the precision of the outer curve quickly approaches 1.0, and further increasing c simply reduces the recall. In contrast, the recall of the inner curve is maximized at lower values of c and reduces at a slower pace.

Synthetic Dataset Experiment:

Accuracy statistics as execution time increases

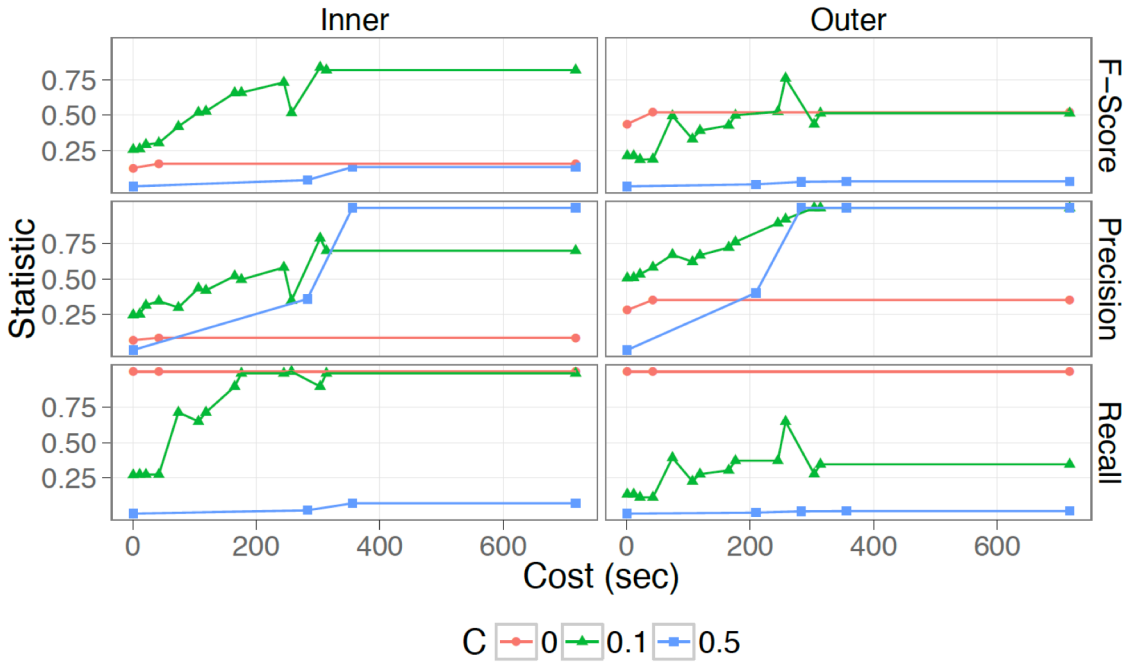
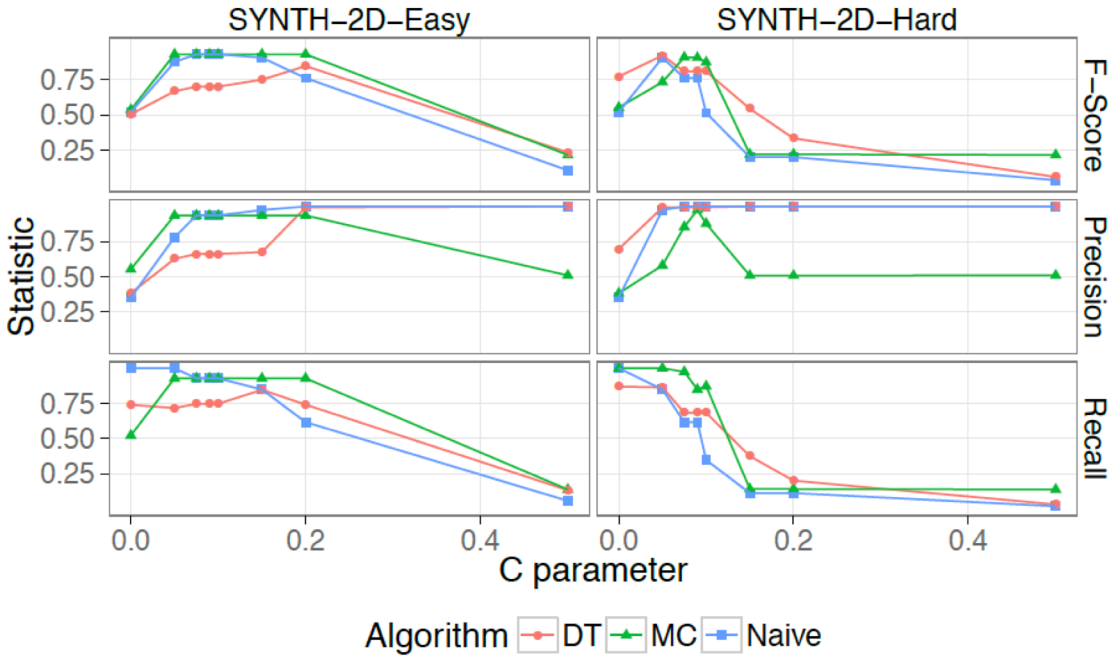


Figure 11 plots the amount of time it takes for Naïve to converge when executing on Synth-2D-Hard. Naïve tends to converge faster when c is close to zero, because the optimal predicate involves fewer attributes.

Synthetic Dataset Experiment:

2. Comparison with DT, MC and Naïve algorithms by varying the c parameter

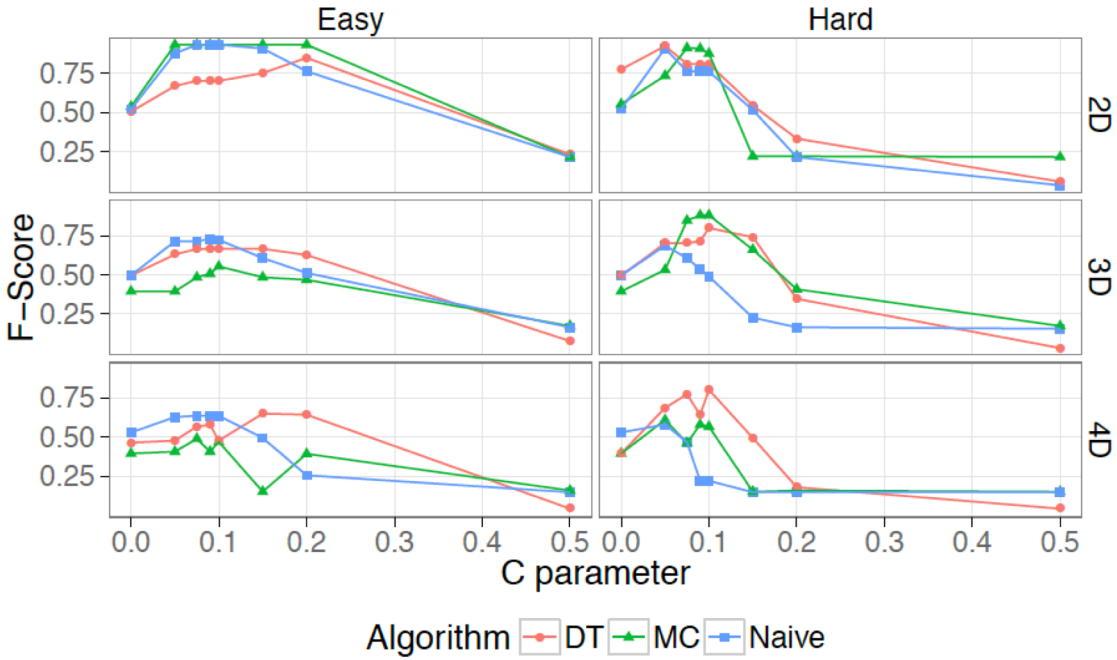


Not too much difference

Figure 12 varies c parameter and computes the accuracy statistics using the outer cube as the ground truth. Both DT and MC generate results comparable with those from the Naïve algorithm.

Synthetic Dataset Experiment:

2. Comparison with DT, MC and Naïve algorithms by varying the dimensionality



Not too much difference

Figure 13 compares the F-scores of the algorithms as the dimensionality varies from 2 to 4. Each row and column of plots corresponds to the dimensionality and difficulty of the dataset, respectively. As the dimensionality increases, DT and MC remain competitive with Naïve.

Synthetic Dataset Experiment:

Improve a lot than Naïve !!

2. Comparison with DT, MC and Naïve algorithms of runtime and number of tuples

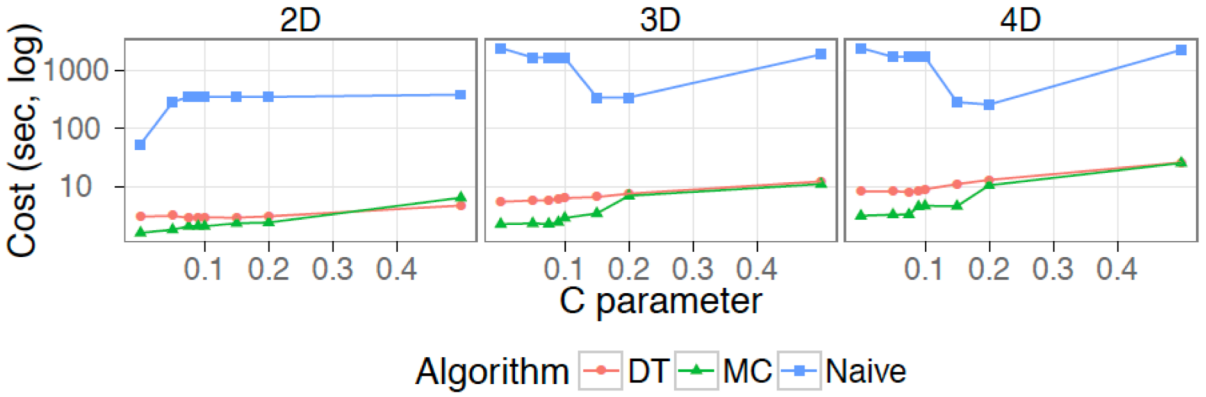


Figure 14 compares the algorithm runtimes while varying the dimensionality of the Easy synthetic datasets. We can see that DT and MC are up to two orders of magnitude faster than Naïve.

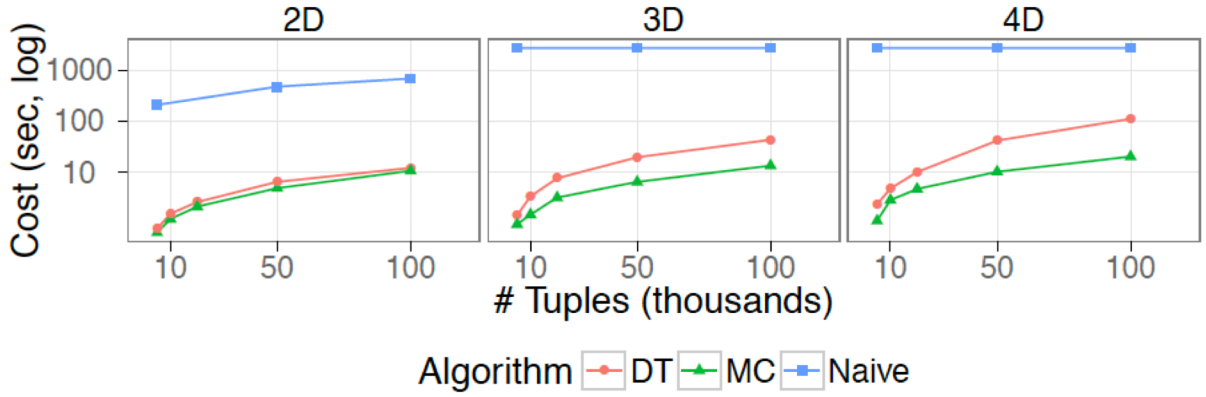


Figure 15 uses the Easy datasets and varies the number of tuples per group from 5k tuples to 10k tuples for a fixed $c = 0.1$. And DT spends significant time splitting non-influential partitions.

Synthetic Dataset Experiment:

3. A Caching Based Optimization for the DT algorithm and the Merger

The previous experiments showed that the result predicates are sensitive to c, thus the user or system may want to try different values of c.

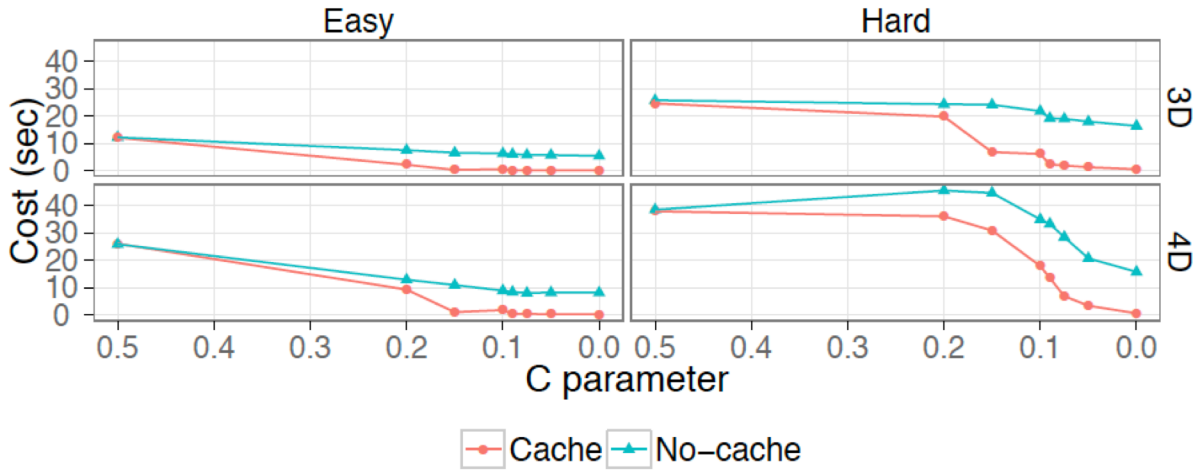
→ Caching is needed for duplicate computation of c

DT can cache and re-use its results because the partitioning algorithm is agnostic to the c value. Thus, the DT partitioner only needs to execute once for Scorpion queries that only change c.

Almost the same for the Merger. Scorpion can initialize the merging process to the results of any prior execution with a higher c value. For example, if the user first ran a Scorpion query with $c = 1$, then those results can be re-used when c reduces to 0.5.

Synthetic Dataset Experiment:

3. A Caching Based Optimization for the DT algorithm and the Merger



Improve a lot than no caching

Figure 16: Cost with and without caching enabled

Figure 16 executes Scorpion using the DT partitioner on the synthetic datasets with decreasing values of c (from 0.5 to 0), and cache the results so that each execution can benefit from the previous one. Caching DT and Merger results for low c values reduces execution cost by up to 25×.

Experiment Results with Real-World Datasets:

Vary c from 0 to 1

INTEL

Scorpion returned $Light \in [283, 354]$ & $sensorid = 18$ when $c = 1$. Sensor 18' voltage is abnormally low, which causes it to generate high temperature readings(90 ℃-122 ℃). The readings are particularly high(122 ℃) when the light levels are between 283 and 354. Scorpion only returns $sensorid = 18$ at lower c values.

EXPENSES

Scorpion executed the MC algorithm. It generated the predicate $recipient\ st =' DC'$ & $recipient_{nm} =' GMMB\ INC'$ & $file_{num} = 800316$ & $disb_{desc} =' MEIDA\ BUY'$. This predicates best describes Obama's highest expenses on advertising and pollical consulting and matches all over $1M expenditures for an average of $2.6M.

Thank you for listening!