Query Log Compression for Workload Analytics

Ting Xie
University at Buffalo, SUNY
tingxie@buffalo.edu

Varun Chandola
University at Buffalo, SUNY
chandola@buffalo.edu

Oliver Kennedy
University at Buffalo, SUNY
okennedy@buffalo.edu

ABSTRACT

Analyzing database access logs is a key part of performance tuning, intrusion detection, benchmark development, and many other database administration tasks. Unfortunately, it is common for production databases to deal with millions or more queries each day, so these logs must be summarized before they can be used. Designing an appropriate summary encoding requires trading off between conciseness and information content. For example: simple workload sampling may miss rare, but high impact queries. In this paper, we present LogR, a lossy log compression scheme suitable for use in many automated log analytics tools, as well as for human inspection. We formalize and analyze the space/fidelity trade-off in the context of a broader family of "pattern" and "pattern mixture" log encodings to which LogR belongs. We show through a series of experiments that LogR compressed encodings can be created efficiently, come with provable information-theoretic bounds on their accuracy, and outperform state-of-art log summarization strategies.

PVLDB Reference Format:
Ting Xie, Varun Chandola, Oliver Kennedy. Query Log Compression for Workload Analytics. *PVLDB*, 12(3): 183-196, 2018.
DOI: https://doi.org/10.14778/3291264.3291265

1. INTRODUCTION

Automated analysis of database access logs is critical for solving a wide range of problems, from database performance tuning [10], to compliance validation [14], and query recommendation [12]. For example, the Peloton self-tuning database [39] searches for optimal configurations by repeatedly simulating database performance based on statistical properties of historical queries. Unfortunately, query logs for production databases can grow to be large — A recent study of queries at a major US bank for a period of 19 hours found nearly 17 million SQL queries and over 60 million stored procedure executions [30] — and computing these properties from the log itself is slow.

Proceedings of the VLDB Endowment, Vol. 12, No. 3
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3291264.3291265

Tracking only a sample of these queries is not sufficient, as rare queries can disproportionately affect database performance, for example, if they benefit from an otherwise unnecessary index. Rather, we need a compressed *summary* of the log on which we can compute aggregate statistical properties. The problems of compression and summarization have been studied extensively (e.g., [47, 48, 21, 33, 24, 8, 42, 29]). However, these schemes either require the use of heavyweight inference to desired statistical measures, or produce unnecessarily large encodings.

In this paper, we adapt ideas from pattern mining and summarization [35, 16] to propose a middle-ground: LogR, a summarization scheme that facilitates efficient (both in terms of storage and time) approximation of workload statistics. By adjusting a tunable parameter in LogR, users can choose to obtain a high-fidelity, albeit large summary, or obtain a more compact summary with lower fidelity. Constructing the summary that best balances compactness and fidelity is challenging, as the search space of candidate summaries is combinatorially large [35, 16]. LogR offers a new approach to summary construction that avoids searching this space, making inexpensive, accurate computation of aggregate workload statistics possible. As a secondary benefit, the resulting summaries are also human-interpretable.

LogR does not admit closed-form solutions to classical fidelity measures like information loss, so we propose an alternative called *Reproduction Error*. We show through a combination of analytical and experimental evidence that Reproduction Error is highly correlated with several classical measures of encoding fidelity.

LogR-compressed data relies on a codebook of structural elements like `SELECT` items, `FROM` tables, or conjunctive `WHERE` clauses [3]. This codebook provides a bi-directional mapping from SQL queries to a bit-vector encoding, reducing the compression problem to one of compactly encoding a collection of feature-vectors. We further simplify the problem by observing that a common theme in use cases like automated performance tuning or query recommendation is the need for predominantly aggregate workload statistics. As these are order-independent, we are able to focus exclusively on compactly representing *bags* of feature-vectors.

LogR works by identifying groups of co-occurring structural elements that we call patterns. We define a family of *pattern encodings* of access logs, which map patterns to their frequencies in the log. For pattern encodings, we consider two idealized measures of fidelity: (1) Ambiguity, which measures how much room the encoding leaves for interpretation; and (2) Deviation, which measures how reliably the encoding approximates the original log. Neither Ambiguity nor Deviation can be computed efficiently for pattern encodings. Hence we propose a measure called *Reproduction Error* that is efficiently computable and that closely tracks both Ambiguity and Deviation.

In general, the size of the encoding is inversely related with Reproduction Error: The more detailed the encoding, the more faithfully it represents the original log. Thus, log compression may be defined as a search over the space of pattern-based encodings to identify the one that best trades off between these two properties. Unfortunately, searching for such an ideal encoding from the space can be computationally expensive [35, 16]. To overcome this limitation, we reduce the search space by first clustering entries in the log and then encoding each cluster separately, an approach that we call *pattern mixture encoding*. Finally we identify a simple approach for encoding individual clusters that we call *naive mixture encodings*, and show experimentally that it produces results competitive with more general techniques for log compression and summarization.

Concretely, in this paper we make the following contributions: (1) We define two families of compression for query logs: pattern and pattern mixture, (2) We define a computationally efficient measure, Reproduction Error, and demonstrate that it is a close approximation of Ambiguity and Deviation (two commonly used measures), (3) We propose a clustering-based approach to efficiently search for naive mixture encodings, and show how these encodings can be further optimized, and, (4) We experimentally validate LogR and show that it produces more precise encodings, faster than several state-of-the-art pattern encoding algorithms.

Roadmap. The paper is organized as follows: Section 2 formally defines the log compression problem and the summary representation; Section 3 then defines information loss of the summaries; Section 4 explains the difficulty in computing classical loss measures and provides a practical alternative; Section 5 motivates data partitioning and generalizes the practical loss measure to partitioned data; Section 6 then introduces the proposed LogR compression scheme; Section 7 empirically validates the practical loss measure and evaluates the effectiveness of LogR by comparing it with two state-of-the-art summarization methods; Section 8 further evaluates LogR under applications of comparison methods; Section 9 discusses related work. Section 10 concludes the paper and Section 11 discusses future work.

2. PROBLEM DEFINITION

In this section, we introduce and formally define the log compression problem. We begin by exploring several applications that need to repeatedly analyze query logs.

Index Selection. Selecting an appropriate set of indexes requires trading off between update costs, access costs, and limitations on available storage space. Existing strategies for selecting a near-optimal set of indexes typically repeatedly simulate database performance under different combinations of indexes, which in turn requires repeatedly estimating the frequency of specific predicates in the workload.

Materialized View Selection. The results of joins or highly selective selection predicates are good candidates for materialization when they appear frequently in the workload. Like index selection, view selection is a non-convex optimization problem, typically requiring repeated simulation, which in turn requires repeated frequency estimation over the workload.

Online Database Monitoring. In production settings, it is common to monitor databases for atypical usage patterns that could indicate a serious bug or security threat. When query logs are monitored, it is often done retrospectively, some hours after-the-fact [30]. To support real-time monitoring it is necessary to quickly compute the frequency of a particular class of queries in the system's typical workload.

In each case, the application's interactions with the log amount to counting queries that have specific features: selection predicates, joins, or similar.

2.1 Preliminaries and Notation

Let L be a log, or a finite collection of queries $\mathbf{q} \in L$. We write $f \in \mathbf{q}$ to indicate that $\mathbf{q}$ has some *feature* f, such as a specific predicate or table in its `FROM` clause. We assume (1) that the universe of features in both a log and a query is enumerable and finite, (2) that the features are selected to suit specific applications and (3) optionally that a query is isomorphic to its feature set (motivated in Section 2.3.2). We outline one approach to extracting features that satisfies all three assumptions below. We abuse syntax and write $\mathbf{q}$ to denote both the query itself, as well as the set of its features.

Let $\mathbf{b}$ denote some set of features $f \in \mathbf{b}$. We write these sets using vector notation: $\mathbf{b} = (x_1, \ldots, x_n)$ where n is the number of distinct features in the entire log and x_i indicates the presence (absence) of ith feature with a 1 (resp., 0). For any two patterns $\mathbf{b}$, $\mathbf{b}'$, we say that $\mathbf{b}'$ *is contained* or *appears* in $\mathbf{b}$ if $\mathbf{b}' \subseteq \mathbf{b}$, or equivalently if $\forall i$, $x'_i \leq x_i$.

2.2 Coding Queries

For this paper, we specifically adopt the feature extraction conventions of a query summarization scheme by Aligon et al. [3]. In this scheme, each feature is one of the following three query elements: (1) a table or sub-query in the `FROM` clause, (2) a column in the `SELECT` clause, and (3) a conjunctive atom of the `WHERE` clause.

Example 1. *Consider the following example query.*

```
SELECT _id, sms_type, _time FROM Messages
WHERE status=? AND transport_type=?
```

The query has 6 *features*: ⟨ `_id`, `SELECT` ⟩, ⟨ `sms_type`, `SELECT` ⟩, ⟨ `_time`, `SELECT` ⟩, ⟨ `Messages`, `FROM` ⟩, ⟨ `status=?`, `WHERE` ⟩, *and* ⟨ `transport_type=?`, `WHERE` ⟩

Although this scheme is simple and limited to conjunctive queries, it fulfills all three assumptions we make on feature extraction schemes. The features of a query (and consequently a log) are enumerable and finite, and the feature set of the query is isomorphic to the original query. Furthermore, even if a query is not itself conjunctive, it may be rewritable into a conjunctive equivalent.

Although we do not explore more advanced feature encoding schemes in detail here, we direct the interested reader to work on query summarization [34, 7, 30]. For example, a scheme by Makiyama et. al. [34] also captures aggregation-related features like group-by columns, while an approach by Kul et. al. [30] encodes partial tree-structures in the query.

```
SELECT  sms_type, external_ids, _time, _id
  FROM  messages
 WHERE  (sms_type=?) ∧ (status=?)
```

(a) *Correlation-ignorant*: Features are highlighted independently

```
SELECT sms_type FROM messages WHERE sms_type=?
SELECT sms_type FROM messages WHERE status=?
```

(b) *Correlation-aware*: Pattern groups are highlighted together.

Figure 1: **Example Encoding Visualizations**

2.3 Log Compression

As a lossy form of compression, LogR only approximates the information content of a query log. We next develop a simplified form of LogR that we call pattern-based encoding, and develop a framework for reasoning about the fidelity of a LogR-compressed log. As a basis for this framework, we first formulate the information content of a query log to allow us to adapt classical measures of information content.

2.3.1 Information Content of Logs

We define the information content of the log as a distribution $p(Q \mid L)$ of queries Q drawn uniformly from the log.

Example 2. *Consider the following query log, which consists of four conjunctive queries.*

```
1. SELECT _id FROM Messages WHERE status = ?
2. SELECT _time FROM Messages
          WHERE status = ? AND sms_type = ?
3. SELECT _id FROM Messages WHERE status = ?
4. SELECT sms_type, _time FROM Messages
          WHERE sms_type = ?
```

Drawing uniformly from the log, each entry will appear with probability $\frac{1}{4} = 0.25$. The query q_1 ($= q_3$) occurs twice, so the probability of drawing it is double that of the others (i.e., $p(\mathbf{q}_1 \mid L) = p(\mathbf{q}_3 \mid L) = \frac{2}{4} = 0.5$)

Treating a query as a vector of its component features, we can define a query $\mathbf{q} = (x_1, \ldots, x_n)$ to be an observation of the multivariate distribution over variables $Q = (X_1, \ldots, X_n)$ corresponding to features. The event $X_i = 1$ occurs if feature i appears in a uniformly drawn query.

Example 3. *Continuing, the universe of features for this query log is (1) $\langle$ `_id`, `SELECT` $\rangle$, (2) $\langle$ `_time`, `SELECT` $\rangle$, (3) $\langle$ `sms_type`, `SELECT` $\rangle$, (4) $\langle$ `status = ?`, `WHERE` $\rangle$, (5) $\langle$ `sms_type = ?`, `WHERE` $\rangle$, and (6) $\langle$ `Messages`, `FROM` $\rangle$. Accordingly, the queries can be encoded as feature vectors, with fields recording each feature's presence: $\mathbf{q}_1 = \langle 1,0,0,1,0,1 \rangle$, $\mathbf{q}_2 = \langle 0,1,0,1,1,1 \rangle$, $\mathbf{q}_3 = \langle 1,0,0,1,0,1 \rangle$, $\mathbf{q}_4 = \langle 0,1,1,0,1,1 \rangle$*

Patterns. Our target applications require us to count the number of times features (co-)occur in a query. For example, materialized view selection requires counting tables used together in queries. Motivated by this observation, we begin by defining a broad class of *pattern-based encodings* that directly encode co-occurrence probabilities. A *pattern* is an arbitrary set of features $\mathbf{b} = (x_1, \ldots, x_n)$ that may co-occur together. Each pattern captures a piece of information from the distribution $p(Q \mid L)$. In particular, we are interested in the probability of uniformly drawing a query $\mathbf{q}$ from the log that *contains* the pattern $\mathbf{b}$ (i.e., $\mathbf{q} \supseteq \mathbf{b}$):

$$p(Q \supseteq \mathbf{b} \mid L) = \textstyle\sum_{\mathbf{q} \in L \wedge \mathbf{q} \supseteq \mathbf{b}} p(\mathbf{q} \mid L)$$

When it is clear from context, we abuse notation and write $p(\cdot)$ instead of $p(\cdot \mid L)$. Recall that $p(Q)$ can be represented as a joint distribution of variables $(X_1, \ldots, X_n)$ and probability $p(Q \supseteq \mathbf{b})$ is equivalent to $p(X_1 \geq x_1, \ldots, X_n \geq x_n)$.

Pattern-Based Encodings. Denote by $\mathcal{E}_{max} : \{0,1\}^n \to [0,1]$, the mapping from each pattern ($\mathbf{b}$) to its frequency in the log: $\mathcal{E}_{max} = \{ \ (\mathbf{b} \to p(\mathbf{b}))\ \mid \mathbf{b} \in \{0,1\}^n \ \}$

A *pattern-based encoding* $\mathcal{E}$ is any such partial mapping $\mathcal{E} \subseteq \mathcal{E}_{max}$. We denote the frequency of pattern $\mathbf{b}$ in encoding $\mathcal{E}$ by $\mathcal{E}[\mathbf{b}]$ ($= p(Q \supseteq \mathbf{b})$). When it is clear from context, we abuse syntax and also use $\mathcal{E}$ to denote the set of patterns it maps (i.e., *domain*($\mathcal{E}$)). Hence, $|\mathcal{E}|$ is the number of mapped patterns, which we call the encoding's *Verbosity*. A *pattern-based encoder* is any algorithm $\texttt{encode}(L, \epsilon) \mapsto \mathcal{E}$ whose input is a log L and whose output is a set of patterns $\mathcal{E}$, with Verbosity thresholded at some integer ϵ. Many pattern mining algorithms [35, 16] can be used for this purpose.

2.3.2 Communicating Information Content

A side-benefit of pattern-based encodings is that, under the assumption of isomorphism in Section 2.1, patterns can be translated to their query representations and used for human inspection of the log. Figure 1 shows two examples. The approach illustrated in Figure 1a uses shading to show each feature's frequency in the log, and communicates frequently occurring predicates or columns. This approach might, for example, help a human to manually select indexes. A second approach illustrated in Figure 1b conveys correlations, showing the frequency of entire patterns. The accompanying technical report [45] explores visualizations of pattern-based summaries in greater depth.

3. INFORMATION LOSS

Our goal is to encode the distribution $p(Q)$ as a set of patterns: obtaining a less verbose encoding (i.e., with fewer patterns), while also ensuring that the encoding captures $p(Q)$ with minimal information loss. In this section, we define information loss for pattern-based encodings.

3.1 Lossless Summaries

To establish a baseline for measuring information loss, we begin with the extreme cases. At one extreme, an empty encoding ($|\mathcal{E}| = 0$) conveys no information. At the other extreme, we have the encoding $\mathcal{E}_{max}$ which is the full mapping from all patterns. Having this encoding is a sufficient condition to exactly reconstruct the original distribution $p(Q)$.

Proposition 1. *For any query $\mathbf{q} = (x_1, \ldots, x_n) \in 0, 1^n$, the probability of drawing exactly $\mathbf{q}$ at random from the log (i.e., $p(X_1 = x_1, \ldots, X_n = x_n)$) is computable, given $\mathcal{E}_{max}$.*

3.2 Lossy Summaries

Although $\mathcal{E}_{max}$ is lossless, its Verbosity is exponential in the number of features (n). Hence, we will focus on lossy encodings that can be less verbose. A lossy encoding $\mathcal{E} \subset \mathcal{E}_{max}$ may not precisely identify the distribution $p(Q)$, but can still be used to approximate it. We characterize the information content of a lossy encoding $\mathcal{E}$ by defining a *space* (denoted by $\Omega_{\mathcal{E}}$) of distributions $\rho \in \Omega_{\mathcal{E}}$ allowed by an encoding $\mathcal{E}$. This space is defined by constraints as follows: First, we have the general properties of probability distributions:

$$\forall \mathbf{q} \in \{0,1\}^n : \rho(\mathbf{q}) \geq 0 \qquad\qquad \textstyle\sum_{\mathbf{q}} \rho(\mathbf{q}) = 1$$

Each pattern $\mathbf{b}$ in the encoding $\mathcal{E}$ constrains relevant probabilities in distribution ρ to sum to the target frequency:

$$\forall \mathbf{b} \in \mathit{domain}(\mathcal{E}) : \quad \mathcal{E}[\mathbf{b}] = \sum_{\mathbf{q} \supseteq \mathbf{b}} \rho(\mathbf{q})$$

Note that the dual constraints $1 - \mathcal{E}[\mathbf{b}] = \sum_{\mathbf{q} \not\supseteq \mathbf{b}} \rho(\mathbf{q})$ are redundant under constraint $\sum_{\mathbf{q}} \rho(\mathbf{q}) = 1$.

The resulting space $\Omega_{\mathcal{E}}$ is the set of all query logs, or equivalently the set of all possible distributions of queries, that obey these constraints. From the outside observer's perspective, the distribution $\rho \in \Omega_{\mathcal{E}}$ that the encoding conveys is ambiguous: We model this ambiguity using a random variable $\mathcal{P}_{\mathcal{E}}$ with support $\Omega_{\mathcal{E}}$. The true distribution $p(Q)$ derived from the query log must appear in $\Omega_{\mathcal{E}}$, denoted as $p(Q) \equiv \rho^* \in \Omega_{\mathcal{E}}$ (i.e., $p(\mathcal{P}_{\mathcal{E}} = \rho^*) > 0$). Of the remaining distributions ρ admitted by $\Omega_{\mathcal{E}}$, it is possible that some are more likely than others. For example, a query containing a column (e.g., `status`) is only valid if it also references a table that contains the column (e.g., `Messages`). This prior knowledge may be modeled as a prior on the distribution of $\mathcal{P}_{\mathcal{E}}$ or equivalently by an additional constraint. However, for the purposes of this paper, we take the uninformed prior by assuming that $\mathcal{P}_{\mathcal{E}}$ is uniformly distributed over $\Omega_{\mathcal{E}}$:

$$p(\mathcal{P}_{\mathcal{E}} = \rho) = \begin{cases} \frac{1}{|\Omega_{\mathcal{E}}|} & \text{if } \rho \in \Omega_{\mathcal{E}} \\ 0 & \text{otherwise} \end{cases}$$

Naive Encodings. One specific family of lossy encodings that treat each feature as being independent (e.g., as in Figure 1a) is of particular interest to us. We call this family *naive encodings*, and return to it throughout the rest of the paper. A naive encoding $\ddot{\mathcal{E}}$ is composed of all patterns that have exactly one feature with non-zero frequency.

$$\mathit{domain}(\ddot{\mathcal{E}}) = \{\ (0, \ldots, 0, x_i, 0, \ldots, 0) \mid i \in [1, n],\ x_i = 1\ \}$$

3.3 Idealized Information Loss Measures

Based on the space of distributions constrained by the encoding, the information loss of an encoding can be considered from two related, but subtly distinct perspectives: (1) *Ambiguity* measures how much room the encoding leaves for interpretation and (2) *Deviation* measures how reliably the encoding approximates the target distribution $p(Q)$.

Ambiguity. We define the Ambiguity $\mathrm{I}(\mathcal{E})$ of an encoding as the entropy of the random variable $\mathcal{P}_{\mathcal{E}}$. The higher the entropy, the less precisely $\mathcal{E}$ identifies a specific distribution.

$$\mathrm{I}(\mathcal{E}) = \sum_{\rho} p(\mathcal{P}_{\mathcal{E}} = \rho) \log \left(p(\mathcal{P}_{\mathcal{E}} = \rho) \right)$$

Deviation. The deviation from any permitted distribution ρ to the true distribution ρ^* can be measured by the Kullback-Leibler (K-L) divergence $\mathcal{D}_{KL}(\rho^* || \rho)$. We define the Deviation $\mathrm{d}(\mathcal{E})$ of a encoding as the expectation of the K-L divergence over all permitted $\rho \in \Omega_{\mathcal{E}}$:

$$\mathrm{d}(\mathcal{E}) = \mathbb{E}_{\mathcal{P}_{\mathcal{E}}} \left[\mathcal{D}_{KL}(\rho^* || \mathcal{P}_{\mathcal{E}}) \right] = \sum_{\rho \in \Omega_{\mathcal{E}}} p(\mathcal{P}_{\mathcal{E}} = \rho) \cdot \mathcal{D}_{KL}(\rho^* || \rho)$$

Limitations. There are two limitations to these idealized measures in practice. First, K-L divergence is not defined on any permitted distribution ρ where the true distribution ρ^* is not *absolutely continuous* (denoted $\rho^* \ll \rho$). Second, neither Deviation nor Ambiguity has a closed-form formula.

4. PRACTICAL LOSS MEASURE

Computing either Ambiguity or Deviation requires enumerating the entire space of permitted distributions. One approach to approximating either measure is repeatedly sampling from, rather than enumerating the space. However, accurate approximations require a large number of samples, rendering this approach similarly inefficient. In this section, we propose a faster approach to assessing the fidelity of a pattern encoding. Specifically, we select a single representative distribution $\overline{\rho}_{\mathcal{E}}$ from the space $\Omega_{\mathcal{E}}$, and use $\overline{\rho}_{\mathcal{E}}$ to approximate both Ambiguity and Deviation.

4.1 Reproduction Error

Maximum Entropy Distribution. The representative distribution is chosen by applying the maximum entropy principle [23] commonly used in pattern-based summarization [35, 16]. That is, we select the distribution $\overline{\rho}_{\mathcal{E}}$ with maximum entropy:

$$\overline{\rho}_{\mathcal{E}} = \arg\max_{\rho \in \Omega_{\mathcal{E}}} \mathcal{H}(\rho) \qquad \text{where } \mathcal{H}(\rho) = \sum_{\mathbf{q} \in \{0,1\}^n} -\rho(\mathbf{q}) \log \rho(\mathbf{q})$$

The maximum entropy distribution best represents the current state of knowledge. That is, a distribution with lower entropy assumes additional constraints derived from patterns that we do not know, while one with higher entropy violates the constraints from patterns we do know.

Maximizing an objective function belonging to the exponential family (entropy in our case) under a mixture of linear equalities/inequality constraints is a convex optimization problem [9] which guarantees a *unique* solution and can be efficiently solved using the cvx toolkit [18, 38], and/or by *iterative scaling* [35, 16]. For naive encodings specifically, we can assume independence between each feature X_i. Under this assumption, $\overline{\rho}_{\mathcal{E}}$ has a closed-form representation:

$$\overline{\rho}_{\mathcal{E}}(\mathbf{q}) = \prod_i p(X_i = x_i) \qquad \text{where } \mathbf{q} = (x_1, \ldots, x_n)$$

We define *Reproduction Error* $e(\mathcal{E})$ as the entropy difference between the representative and true distributions:

$$e(\mathcal{E}) = \mathcal{H}(\overline{\rho}_{\mathcal{E}}) - \mathcal{H}(\rho^*) \qquad \text{where } \overline{\rho}_{\mathcal{E}} = \arg\min_{\rho \in \Omega_{\mathcal{E}}} -\mathcal{H}(\rho)$$

4.2 Practical vs Idealized Information Loss

In this section we prove that Reproduction Error closely parallels Ambiguity. We define a partial order lattice over encodings and show that for any pair of encodings on which the partial order is defined, a like relationship is implied for both Reproduction Error and Ambiguity. We supplement the proofs given in this section with an empirical analysis relating Reproduction Error to Deviation in Section 7.1.

Containment. We define a partial order over encodings $\leq_\Omega$ based on *containment* of their induced spaces $\Omega_{\mathcal{E}}$:

$$\mathcal{E}_1 \leq_\Omega \mathcal{E}_2 \quad \equiv \quad \Omega_{\mathcal{E}_1} \subseteq \Omega_{\mathcal{E}_2}$$

That is, one encoding (i.e., $\mathcal{E}_1$) precedes another (i.e., $\mathcal{E}_2$) when all distributions admitted by the former encoding are also admitted by the latter.

Containment Captures Reproduction Error. We first prove that the total order given by Reproduction Error is a superset of the partial order $\leq_\Omega$.

Lemma 1. *For any pair of encodings $\mathcal{E}_1, \mathcal{E}_2$ that induce spaces $\Omega_{\mathcal{E}_1}, \Omega_{\mathcal{E}_2}$ and maximum entropy distributions $\overline{\rho}_{\mathcal{E}_1}, \overline{\rho}_{\mathcal{E}_2}$ it holds that $\mathcal{E}_1 \leq_\Omega \mathcal{E}_2 \rightarrow e(\mathcal{E}_1) \leq e(\mathcal{E}_2)$.*

Proof. First we have $\Omega_{\mathcal{E}_2} \supseteq \Omega_{\mathcal{E}_1} \rightarrow \overline{\rho}_{\mathcal{E}_1} \in \Omega_{\mathcal{E}_2}$. Since $\overline{\rho}_{\mathcal{E}_2}$ has the maximum entropy among all distributions $\rho \in \Omega_{\mathcal{E}_2}$, we have $\mathcal{H}(\overline{\rho}_{\mathcal{E}_1}) \leq \mathcal{H}(\overline{\rho}_{\mathcal{E}_2}) \equiv e(\mathcal{E}_1) \leq e(\mathcal{E}_2)$. □

Containment Captures Ambiguity. Next, we show that the partial order based on containment implies a like relationship between Ambiguities of pairs of encodings.

Lemma 2. *Given encodings $\mathcal{E}_1, \mathcal{E}_2$ with uninformed prior on $\mathcal{P}_{\mathcal{E}_1}, \mathcal{P}_{\mathcal{E}_2}$, it holds that $\mathcal{E}_1 \leq_\Omega \mathcal{E}_2 \rightarrow I(\mathcal{E}_1) \leq I(\mathcal{E}_2)$.*

Proof. Given an uninformed prior: $\mathrm{I}(\mathcal{E}) = \log |\Omega_\mathcal{E}|$, we have $\mathcal{E}_1 \leq_\Omega \mathcal{E}_2 \rightarrow |\Omega_{\mathcal{E}_1}| \leq |\Omega_{\mathcal{E}_2}| \rightarrow \mathrm{I}(\mathcal{E}_1) \leq \mathrm{I}(\mathcal{E}_2)$ □

5. PATTERN MIXTURE ENCODINGS

Thus far we have defined the problem of log compression, treating the query log as a multivariate distribution $p(Q)$ where patterns capture positive frequencies of feature (co-)occurrence. However in cases like logs of *mixed* workloads, there are also many cases of anti-correlation between features. For example, consider a log that includes queries drawn from a mixture of two workloads with disjoint feature sets. Pattern-based summaries can not convey such anti-correlations easily. As a result, patterns including features from both workloads never actually co-occur in the log, but a pattern-based summary of the log will suggest otherwise. Such false positives are especially problematic for use-cases of LogR involving outlier detection (e.g., [32]). Even in other settings, capturing correlations reduces data dimensionality and improves both runtime and effectiveness of state-of-the-art pattern mining algorithms (See Section 8.1).

In this section, we propose a generalization of pattern encodings where the log is modeled not as a single probability distribution, but rather as a mixture of several simpler distributions. The resulting encoding is likewise a mixture: Patterns for each component of the mixture are stored independently. Hence, we refer to it as a *pattern mixture encoding*, and it forms the basis of LogR compression. We first focus on a simplified form of this problem, where we only mix *naive* encodings (we explore more general mixtures in Section 6.4). We refer to the resulting scheme as *naive mixture encodings*, and give examples of the encoding in Section 5.1. Then we generalize Reproduction Error and Verbosity to pattern mixture encodings in Section 5.2. Finally, with generalized encoding evaluation measures, we evaluate several clustering methods for creating naive mixture encodings.

5.1 Example: Naive Mixture Encodings

Consider a toy query log with only 3 conjunctive queries.

1. `SELECT id FROM Messages WHERE status = ?`
2. `SELECT id FROM Messages`
3. `SELECT sms_type FROM Messages`

The codebook of this log includes 4 features: ⟨ `id`, `SELECT` ⟩, ⟨ `sms_type`, `SELECT` ⟩, ⟨ `Messages`, `FROM` ⟩, ⟨ `status = ?`, `WHERE` ⟩. Re-encoding the three queries as vectors, we get:

1. $\langle\, 1, 0, 1, 1 \,\rangle$ 2. $\langle\, 1, 0, 1, 0 \,\rangle$ 3. $\langle\, 0, 1, 1, 0 \,\rangle$

A naive encoding of this log can be expressed as:

$$\left\langle\ \frac{2}{3},\ \frac{1}{3},\ 1,\ \frac{1}{3}\ \right\rangle$$

This encoding captures that all queries in the log pertain to the `Messages` table, but obscures the relationship between the remaining features. For example, this encoding obscures the anti-correlation between `id` and `sms_type`. Similarly, the encoding hides the correlation between `status = ?` and `id`. Such relationships are critical for evaluating the effectiveness of views or indexes.

Example 4. *The maximum entropy distribution for any naive encoding assumes that features are independent. Assuming independence, the probability of query 1 uniformly drawn from the log is estimated as:*

$$p(\mathit{id}) \cdot p(\neg \mathit{sms\_type}) \cdot p(\mathit{Messages}) \cdot p(\mathit{status{=}?}) = \frac{4}{27} \approx 0.148$$

This is a significant difference from the true probability of this query (i.e., $\frac{1}{3}$). Conversely queries not in the log, such as the following, have non-zero probability in the encoding.

`SELECT sms_type FROM Messages WHERE status = ?`

$$p(\neg \mathit{id}) \cdot p(\mathit{sms\_type}) \cdot p(\mathit{Messages}) \cdot p(\mathit{status{=}?}) = \frac{1}{27} \approx 0.037$$

To achieve a more faithful representation of the original log, we could partition it into two components, with the corresponding encoding parameters:

$$\begin{array}{cc|c} \multicolumn{2}{c}{\textbf{Partition 1 } (L_1)} & \textbf{Partition 2 } (L_2) \\ \hline (1,0,1,1) & (1,0,1,0) & (0,1,1,0) \\ \downarrow & \downarrow & \downarrow \\ \multicolumn{2}{c}{\langle\ 1,\ \ 0,\ \ 1,\ \ \frac{1}{2}\ \rangle} & \langle\ 0,\ \ 1,\ \ 1,\ \ 0\ \rangle \end{array}$$

Although there are now two encodings, the encodings are not ambiguous. The feature `status = ?` appears in exactly half of the log entries, and is indeed independent of the other features. All other attributes in each encoding appear in all queries in their respective partitions. Furthermore, the maximum entropy distribution induced by each encoding is exactly the distribution of queries in the partitioned log. Hence, the Reproduction Error is zero for both encodings.

5.2 Generalized Encoding Fidelity

We next generalize our definitions of Reproduction Error and Verbosity from pattern-based to pattern mixture encodings. Suppose query log L has been partitioned into K clusters with L_i, $\mathcal{E}_i$, $\overline{\rho}_{\mathcal{E}_i}$ and ρ^*_i (where $i \in [1, K]$) representing the log of queries, encoding, maximum entropy distribution, and true distribution (respectively) for the ith cluster. First, observe that the distribution for the whole log (i.e., ρ^*) is the sum of distributions for each partition (i.e., ρ^*_i) weighted by the proportion (i.e., $\frac{|L_i|}{|L|}$) of queries:

$$\rho^*(\mathbf{q}) = \sum_{i=1,\ldots,K} w_i \cdot \rho^*_i(\mathbf{q}) \qquad \text{where } w_i = \frac{|L_i|}{|L|}$$

Generalized Reproduction Error. Similarly, the maximum entropy distribution $\overline{\rho}_\mathcal{E}$ for the whole log is:

$$\overline{\rho}_\mathcal{E}(\mathbf{q}) = \sum_{i=1,\ldots,K} w_i \cdot \overline{\rho}_{\mathcal{E}_i}(\mathbf{q})$$

We define the *Generalized Reproduction Error* of a pattern mixture encoding similarly, as the weighted sum of Reproduction Error for each partition:

$$e(\mathcal{E}) = \mathcal{H}(\overline{\rho}_\mathcal{E}) - \mathcal{H}(\rho^*) = \sum_i w_i (\, \mathcal{H}(\overline{\rho}_{\mathcal{E}_i}) - \mathcal{H}(\rho^*_i)\,) = \sum_i w_i e(\mathcal{E}_i)$$

When it is clear from context, we refer to Generalized Reproduction Error as Error in the rest of this paper. As in the base case, a pattern mixture encoding with low Error indicates a high-fidelity representation of the original log. A process can infer the frequency of any query $p(Q = \mathbf{q} \mid L)$ drawn from the original distribution, simply by inferring its frequency in each cluster i (i.e., $p(Q = \mathbf{q} \mid L_i)$) and taking a weighted average over all inferences.

Generalized Verbosity. We generalize verbosity to pattern mixture encodings as the *Total Verbosity* ($\sum_i |S_i|$), or the total size of the encoded representation.

6. PATTERN MIXTURE COMPRESSION

We are now ready to describe the LogR compression scheme. Broadly, LogR attempts to identify a pattern mixture encoding that optimizes for some target trade-off between Total Verbosity and Error. A naive — though impractical — approach to finding such an encoding would be to search the entire space of possible pattern mixture encodings. Instead, LogR approximates the same outcome by first identifying the naive mixture encoding that is closest to optimal for the desired trade-off. As we show experimentally, this naive mixture encoding is competitive with more complicated, slower techniques for summarizing query logs. We also explore a hypothetical second stage, where LogR refines the naive mixture encoding to further reduce Error. The outcome of this hypothetical stage has a slightly lower Error and Verbosity, but does not admit efficient computation of database statistics.

6.1 Constructing Naive Mixture Encodings

LogR searches for a naive mixture encoding that best optimizes for a requested tradeoff between Total Verbosity and Error. As a way to make this search efficient, we observe that a log (or log partition) uniquely determines its naive (or naive mixture) encoding. Thus the problem of searching for a naive mixture encoding reduces to searching for the corresponding log partitioning. We further observe that the Error of a naive mixture encoding is proportional to the diversity of the queries in the log being encoded: The more uniform the log (or partition), the lower the Error. Hence, the partitioning problem further reduces to clustering queries in the log by feature overlap. To identify a suitable clustering scheme, we next evaluate four commonly used clustering schemes with respect to their ability to create naive mixture encodings with low Error and Verbosity: (1) KMeans [22] with Euclidean distance (i.e., l_2-norm) and Spectral Clustering [26] with (2) Manhattan (i.e., l_1-norm), (3) Minkowski (i.e., l_p-norm) with $p = 4$, and (4) Hamming distances[1].

Experiment Setup. Spectral and KMeans clustering algorithms are implemented by *sklearn* [40] in Python. We gradually increase K (i.e., the number of clusters) for each clustering scheme to mimic the process of continuously sub-clustering the log, tolerating higher Total Verbosity for lower Error. To reduce randomness in clustering, we run each of them 10 times for each K and averaging the Error of the resulting encodings. We used two datasets: "US Bank" and "PocketData". We describe both datsets and the data preparation process in detail in Section 7. All results for our clustering experiments are shown in Figure 2.

[1]We also evaluated Spectral Clustering with Euclidean, Chebyshev and Canberra distances; These did not perform better and we omit them in the interest of conciseness.

6.1.1 Clustering

We next show that clustering is an effective way to consistently reduce Error, although no one clustering scheme is ideal for all three of Error, Verbosity, and runtime.

More clusters reduces Error. Figure 2a compares the relationship between the number of clusters (x-axis) and Error (y-axis), showing the varying rates of convergence to zero Error for each clustering scheme. We observe that adding more clusters does consistently reduce Error for both data sets, regardless of clustering algorithm or distance measure. We note that the US Bank dataset is significantly more diverse than the PocketData dataset, with respect to the total number of features (See Table 1) and that more than 30 clusters may be required for reaching near-zero Error. In general, Hamming distance converges faster than other distance measures on PocketData. Minkowski distance shows faster convergence rate than Hamming within 14 clusters on the US bank dataset.

Adding more clusters increases Verbosity. Figure 2b compares the relationship between the number of clusters (x-axis) and Verbosity (y-axis). We observe that Verbosity increases with the number of clusters. This is because when a partition is split, each feature common to both partitions increases the Verbosity by one.

Hierarchical Clustering. The clustering schemes produce non-monotonic cluster assignments. That is, Error can occasionally grow as clusters are added (Figure 2a). An alternative is to use hierarchical clustering [22], which forces monotonic assignments and offers more dynamic control over the Error/Verbosity tradeoff.

Run Time Comparison. The total runtime (y-axis) in Figure 2c includes both distance matrix computation time (if any) and clustering time. Note the log-scale: K-Means is orders of magnitude faster than the others.

Take-Aways. For time-sensitive applications, KMeans algorithm is preferred to Spectral Clustering. With respect to distance measures, minkowski (i.e., l_p-norm) with $p = 4$ provides the best tradeoff between Error and runtime.

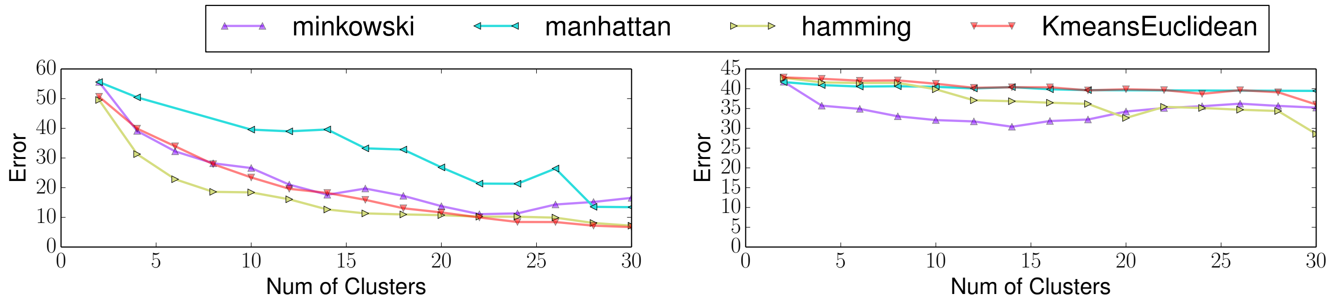
Visualizing Naive Mixture Encoding. As with normal pattern summaries, naive mixture summaries are also interpretable. For example a visualization like that of Figure 1a can be repeated, once for each cluster. For more details, see our accompanying technical report [45].

6.2 Approximating Log Statistics

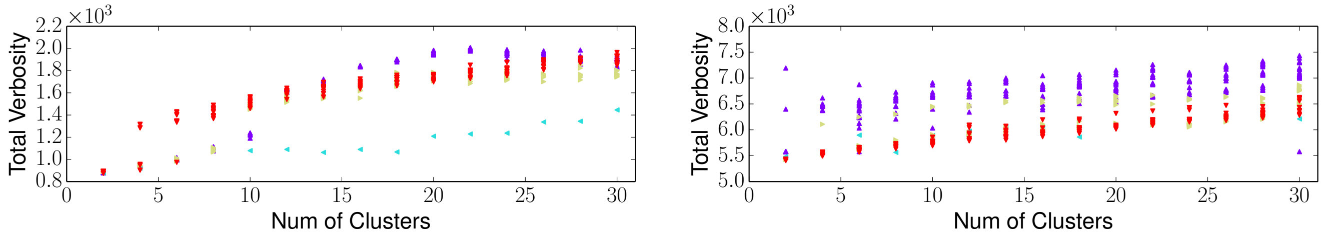
Recall that our primary goal is estimating statistical properties. In particular, we are interested in counting the occurrences $\Gamma_{\mathbf{b}}(L)$ (i.e., $p(Q \supseteq \mathbf{b}) \cdot |L|$) of some pattern $\mathbf{b}$ in the log. Recall that a naive encoding $\ddot{\mathcal{E}}$ includes only single-feature patterns (i.e., patterns exactly encoding $p(X_i \geq x_i)$) and that the closed-form representation for the maximum entropy distribution $\overline{\rho}_{\ddot{\mathcal{E}}}$ arises by independence between features (i.e., $\overline{\rho}_{\ddot{\mathcal{E}}}(Q = \mathbf{q}) = \prod_i p(X_i = x_i)$). Similarly, we use the independence assumption to estimate:

$$est[\Gamma_{\mathbf{b}}(L) \mid \ddot{\mathcal{E}}] = \overline{\rho}_{\ddot{\mathcal{E}}}(Q \supseteq \mathbf{b}) \cdot |L| = \prod_i p(X_i \geq x_i) \cdot |L|$$

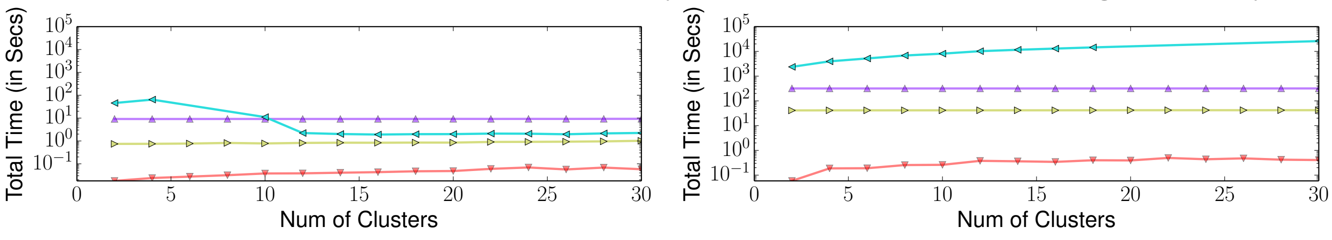
This process trivially generalizes to naive pattern mixture encodings by mixing distributions. Specifically, given a set of partitions $L_1 \cup \ldots \cup L_K = L$, the estimated counts for $\Gamma_{\mathbf{b}}(L)$ under each individual partition L_i can be computed



(a) **Error v. Number of Clusters (Left: PocketData, Right: US Bank)**



(b) **Total Verbosity v. Number of Clusters (Left: PocketData, Right: US Bank); Each point is the verbosity of one of ten trials. The Y-axis' lower bound is the verbosity at 1 cluster to better show the change in verbosity.**



(c) **Runtime v. Number of Clusters (Left: PocketData, Right: US Bank)**

Figure 2: **Clustering Schemes Comparison**

based on the partition's naive encoding $\ddot{\mathcal{E}}_i$, and we then sum up the estimated counts in each partition:

$$est[\, \Gamma_{\mathbf{b}}(L_i) \mid \ddot{\mathcal{E}}_1, \ldots, \ddot{\mathcal{E}}_K \,] = \sum_{i \in [1,K]} est[\, \Gamma_{\mathbf{b}}(L_i) \mid \ddot{\mathcal{E}}_i \,]$$

6.3 Pattern Synthesis & Frequency Estimate

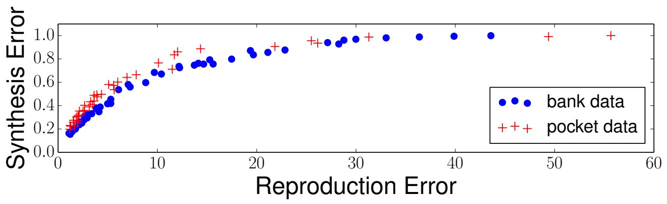
In this section, we empirically verify the effectiveness of naive mixture encodings in approximating log statistics from two related perspectives. The first perspective focuses on *synthesis error*. It measures whether patterns synthesized by the naive mixture encoding actually appear in the log. From the second perspective, we further investigate the *frequency deviation* of patterns contained in the log. This evaluates whether a naive mixture encoding computes the correct frequency for patterns of interest to client applications. Experimental results are shown in Figure 3. Both synthesis error and frequency deviation consistently decrease given more clusters. Furthermore, as we vary the number of clusters, both measures correlate with Reproduction Error.

Synthesis Error is measured by $1 - \frac{m}{n}$ where m out of n randomly synthesized patterns actually appear in the log. Intuitively, when synthesis error grows, it is more likely that a pattern from the synthesized log will not appear in the original log (i.e., smaller values are better). Figure 3a shows synthesis error (y-axis) versus Reproduction Error (x-axis). The figure is generated by synthesizing $n = 10000$ patterns from each cluster of the log. Note that different values of n give similar observations. The overall synthesis error is measured by the average of synthesis errors for all clusters, weighted by the proportion of queries in each cluster.

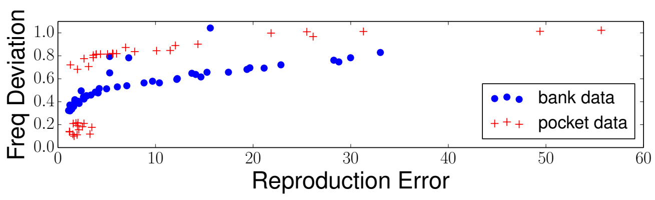
Frequency Deviation is measured for a pattern by $\frac{|est-t|}{t}$ where t stands for true frequency of a pattern and est is the one estimated by the naive mixture encoding. Since frequency deviation is smaller when evaluated on a pattern contained in the other, as an alternative, we treat each distinct query in the log as a pattern and the frequency deviation on it will be the worst case for all patterns that it contains. Intuitively, this value captures the percentage error of frequency estimates (i.e., smaller values are better). For each cluster, we sum frequency deviations on all of its distinct queries and the final frequency deviation for the whole log is an weighted average (same as synthesis error) over all clusters. Figure 3b shows frequency deviation (y-axis) versus Reproduction Error (x-axis).

6.4 Naive Encoding Refinement

Naive mixture encodings can already achieve close to near-zero Error (Figure 2a), have low Total Verbosity, and admit efficiently computable log statistics $\Gamma_{\mathbf{b}}(L)$. Doing so makes estimating statistics more computationally expensive. However, as a thought experiment we consider a hypothetical second pass to enrich naive mixture encodings with non-naive patterns. We start by considering the simpler problem of identifying the *individual* non-naive pattern that maximally reduces the Reproduction Error of a naive encoding.



(a) **Synthesis Error v. Reproduction Error**



(b) **Frequency Deviation v. Reproduction Error**

Figure 3: **Effectiveness of Naive Mixture Encoding**

Feature-Correlation Refinement. Recall that under naive encodings, we have a closed-form estimation $\overline{\rho}_{\ddot{\mathcal{E}}}(Q \supseteq \mathbf{b})$ of pattern frequencies $p(Q \supseteq \mathbf{b})$. We thus define the *feature-correlation* of pattern $\mathbf{b}$ as the log-difference from its actual frequency to the estimate.

$$fc(\mathbf{b}, \ddot{\mathcal{E}}) = |\log(p(Q \supseteq \mathbf{b})) - \log(\overline{\rho}_{\ddot{\mathcal{E}}}(Q \supseteq \mathbf{b}))|$$

Intuitively, patterns with higher feature correlations carry more information content of the log that its naive encoding ignores, making them ideal candidates for addition to the naive encoding. For two patterns with the same feature-correlation, the one that occurs more frequently [19] will have greater impact on Reproduction Error. As a result, we compute an overall score for ranking individual patterns:

$$corr\_rank(\mathbf{b}) = p(Q \supseteq \mathbf{b}) \cdot fc(\mathbf{b}, \ddot{\mathcal{E}})$$

We show in Section 7.1 that *corr_rank* closely correlates with Reproduction Error. That is, a higher *corr_rank* value indicates that a pattern produces a greater reduction in Reproduction Error if introduced into the naive encoding.

Pattern Diversification. In general, we would like to identify a *set* of patterns. The greedy approach that adds patterns one by one based on their ranking scores *corr_rank* is unreliable, as modifying the naive encoding invalidates the closed-form estimation $\overline{\rho}_{\ddot{\mathcal{E}}}(Q \supseteq \mathbf{b})$ that score *corr_rank* relies on. In other words, we can not sum up *corr_rank* scores of patterns in a set to rank its overall contribution to Reproduction Error reduction, as information content carried by patterns may overlap. To counter such overlap, or equivalently to *diversify* patterns, a search through the space of pattern-sets is needed. This type of diversification is commonly used in pattern mining applications, but can quickly become expensive. As we show experimentally in Section 7.2, the benefit of diversification is minimal.

7. EXPERIMENTS

In this section, we design experiments to empirically (1) validate that Reproduction Error correlates with Deviation and (2) evaluate the effectiveness of LogR compression.

We use two specific datasets in the experiment: (1) SQL query logs of the Google+ Android app extracted from the PocketData public dataset [27] and (2) SQL query logs that capture all query activity on the majority of databases at a major US bank over a period of approximately 19 hours. A summary of these two datasets is given in Table 1.

The PocketData-Google+ query log. The dataset consists of SQL logs that capture all database activities of 11 Android phones. We selected the Google+ application for our study since it is one of the few applications where all users created a workload. This dataset is a stable workload of exclusively machine-generated queries.

Table 1: **Summary of Data sets**

Statistics	PocketData	US bank
# Queries	629582	1244243
# Distinct queries	605	188184
# Distinct queries (w/o const)	605	1712
# Distinct conjunctive queries	135	1494
# Distinct re-writable queries	605	1712
Max query multiplicity	48651	208742
# Distinct features	863	144708
# Distinct features (w/o const)	863	5290
Average features per query	14.78	16.56

The US bank query log. This log is an anonymized record of queries processed by multiple relational database servers at a major US bank [30] over a period of 19 hours. Of the nearly 73 million database operations captured, 58 million are not directly queries, but rather invocations of stored procedures. A further 13 million used non-standard SQL features not supported by our SQL parser. Of the remaining of the 2.3 million parsed SQL queries, we base our analysis on the 1.25 million conjunctive `SELECT` queries. This dataset can be characterized as a diverse workload of both machine- and human-generated queries.

Common Experiment Settings. Experiments were performed on a 2.8 GHz Intel Core i7 CPU with 16 GB 1600 MHz DDR3 memory and a SSD running macOS Sierra.

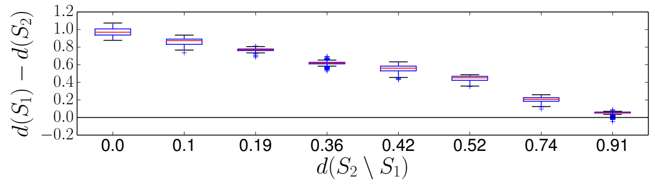
Constant Removal. A number of queries in the US bank query log differ only in hard-coded constant values. Table 1 shows the total number of queries, as well as the number of distinct queries if we ignore constants. By comparison, queries in PocketData all use JDBC parameters. For these experiments, we ignore constant values in queries.

Query Regularization. We apply query rewrite rules (same as [31]) to regularize queries into equivalent conjunctive forms, where possible. Table 1 shows that $\frac{135}{605}$ and $\frac{1494}{1712}$ of distinct queries are in conjunctive form for PocketData and US bank respectively. After regularization, all queries in both data sets can be either simplified into conjunctive queries or re-written into a `UNION` of conjunctive queries compatible with feature scheme of Aligon et al. [3].

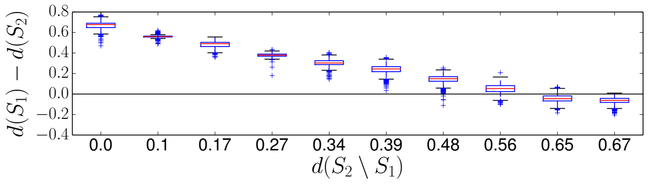
Convex Optimization Solving. All convex optimization problems for measuring Reproduction Error and Deviation are solved by the *successive approximation heuristic* implemented by the CVX toolbox [18] with the Sedumi solver.

7.1 Validating Reproduction Error

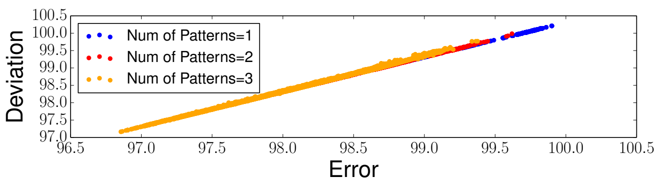
In this section, we validate that Reproduction Error is a practical alternative to Deviation. In addition, we also offer measurements on its correlation with Deviation and score *corr_rank* in Section 6.4. As it is impractical to enumerate



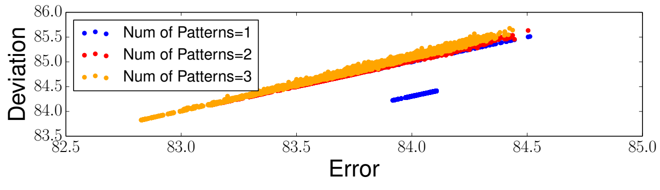
(a) **Containment captures Deviation (US bank)**



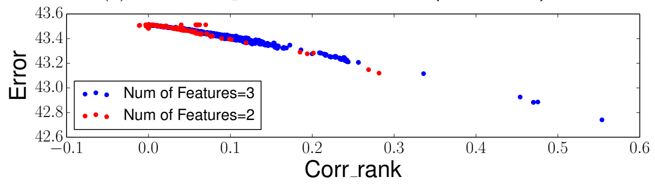
(b) **Containment captures Deviation (PocketData)**



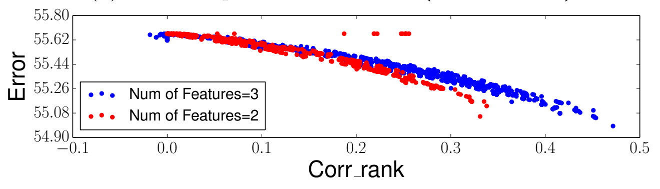
(c) **Error captures Deviation (US bank)**



(d) **Error captures Deviation (PocketData)**



(e) **Error captures Correlation (US bank)**



(f) **Error captures Correlation (PocketData)**

Figure 4: **Validating Reproduction Error**

all possible encodings, we choose a subset of encodings for both datasets. Specifically, we first select all features with frequencies in the range $[0.01, 0.99]$ and use these features to construct patterns. We then enumerate combinations of K (up to 3) patterns as our chosen encodings.

Containment Captures Deviation. Here we empirically verify that containment (Section 4.2) captures Deviation (i.e., $\mathcal{E}_1 \leq_\Omega \mathcal{E}_2 \to d(\mathcal{E}_1) \leq d(\mathcal{E}_2)$) to complete the chain of reasoning that Reproduction Error captures Deviation. Figures 4a and 4b show all pairs of encodings where $\mathcal{E}_2 \supset \mathcal{E}_1$. The y-axis shows the difference in Deviation values (i.e., $d(\mathcal{E}_2) - d(\mathcal{E}_1)$). Deviation $d(\mathcal{E})$ is approximated by drawing 1 million samples from the space $\Omega_\mathcal{E}$ induced by the encoding $\mathcal{E}$. For clarity, we bin pairs of encodings by the degree of overlap between them, measured by the Deviation of the set-difference $d(\mathcal{E}_2 \setminus \mathcal{E}_1)$; Higher $d(\mathcal{E}_2 \setminus \mathcal{E}_1)$ implies less overlap. Y-axis values are grouped into bins and visualized by boxplot where the boxes represent ranges within standard deviation and crosses are outliers. Intuitively, *points above zero* on the y-axis (i.e., $d(\mathcal{E}_2) - d(\mathcal{E}_1) > 0$) are pairs of encodings where the Deviation order agrees with containment order. This is the case for virtually all encoding pairs.

Additive Separability of Deviation. We also observe from Figures 4a and 4b that agreement between Deviation and containment order is correlated with overlap: More similar encodings are more likely to have agreement. Combined with Proposition 1, this shows first that for similar encodings, Reproduction Error is likely to be a reliable indicator of Deviation. This also suggests that Deviation is additively separable: The information loss (i.e., $d(\mathcal{E}_2) - d(\mathcal{E}_1)$) caused by excluding the encoding $\mathcal{E}_2 \setminus \mathcal{E}_1$ from $\mathcal{E}_2$ correlates with the quality (i.e., $d(\mathcal{E}_2 \setminus \mathcal{E}_1)$) of the encoding $\mathcal{E}_2 \setminus \mathcal{E}_1$ itself:

$$\mathcal{E}_2 \supset \mathcal{E}_1 \to d(\mathcal{E}_2) - d(\mathcal{E}_1) < 0 \quad \text{and} \quad d(\mathcal{E}_2 \setminus \mathcal{E}_1) \propto d(\mathcal{E}_2) - d(\mathcal{E}_1)$$

Error correlates with Deviation. As a supplement, Figures 4c and 4d empirically confirm that that Reproduction Error (x-axis) indeed closely correlates with Deviation (y-axis). Mirroring our findings above, correlation between them is tighter at lower Reproduction Error.

Error and Feature-Correlation. Figure 4e and 4f show the relationship between Reproduction Error (y-axis) and score *corr_rank* (x-axis), as discussed in Section 6.4. Values of y-axis are Reproduction Error of the naive encodings extended by a non-naive pattern **b** containing multiple features (up to 3 for illustrative purposes). One can observe that Reproduction Error of extended naive encodings almost linearly correlates with *corr_rank*(**b**). In addition, one can also observe that *corr_rank* becomes higher when the pattern **b** encodes more correlated features.

7.2 Feature-Correlation Refinement

In this section, we design experiments serving two purposes: (1) Evaluating the potential reduction of Error from refining naive mixture encodings through state-of-the-art pattern-based summarizers, and (2) Evaluating whether we can replace naive mixture encodings by the encodings created from summarizers that we have plugged-in.

Experiment Setup. To serve both purposes, we construct pattern mixture encodings under three configurations: (1) Naive mixture encodings; (2) Pattern-based encodings and (3) Naive mixture encodings refined into pattern-based encodings. Naive mixture encodings are constructed by K-Means clustering. Pattern-based encodings are generated by two state-of-the-art pattern-based summarizers: (1) *Laserlight* [16] that summarizes multi-dimensional data in order to predict an augmented binary variable and (2) *MTV* [35] that aims at mining maximally informative patterns that summarize binary multi-dimensional data.

The experimental results are shown in Figure 5 that contains 3 sub-figures sharing the same x-axis, i.e., the number of clusters. Figure 5a compares the Error (y-axis) between naive mixture encodings and pattern mixture encodings that consist of patterns mined from *MTV* or *Laserlight*. Figure 5b evaluates the change in Error (y-axis) through refining naive mixture encodings by adding patterns from *MTV* or *Laserlight*. Figure 5c compares the runtime (y-axis) between constructing naive mixture encodings and applying

MTV or *Laserlight*. We only show the results for US bank query log as results for PocketData give similar observations.

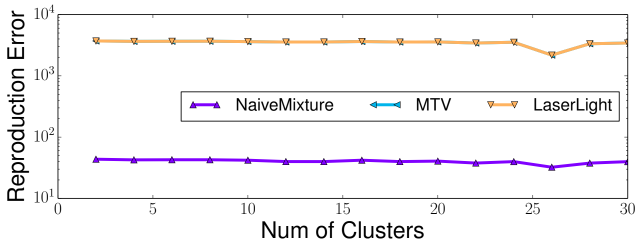
7.2.1 Pattern-based vs Naive Mixture Encodings

Figure 5a and 5c suggest that naive mixture encodings outperform pattern-based encodings in two ways.

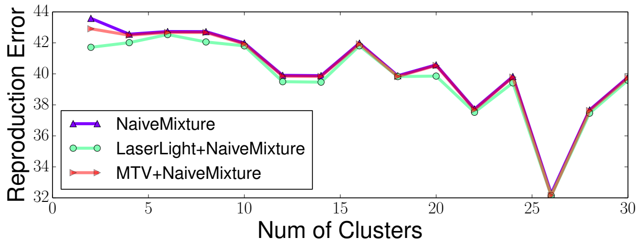
Reproduction Error. We observe from Figure 5a that the Reproduction Error of naive mixture encodings are orders of magnitude lower than pattern-based encodings generated by *Laserlight* or *MTV* alone.

Computation Efficiency. From Figure 5c we observe that the runtime of constructing naive mixture encodings is significantly lower than that of *Laserlight* and *MTV*.

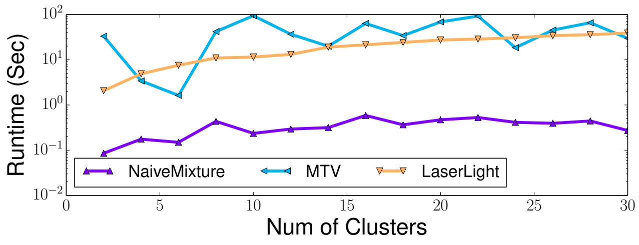
The one way where pattern-based encodings outperform naive mixture encodings is in Total Verbosity. *Laserlight* and *MTV* produce encodings with significantly fewer patterns, as the naive mixture encoding requires at least one pattern for each feature (e.g., 5290 patterns in the US bank query log). Conversely, mining this number of patterns is computationally infeasible (Figure 5c).



(a) **Naive Mixture v. LaserLight/MTV alone. Note that y-axis is in log scale.**



(b) **Naive Mixture v. Naive Mixture+LaserLight/MTV. Note that we offset y-axis (non-zero start).**



(c) **Runtime Comparison (y-axis in log scale)**

Figure 5: **Feature-correlation refinement (US bank)**

7.2.2 Refining Naive Mixture Encodings

The experiment result is shown in Figure 5b. Note that we offset y-axis to show the change in Error. We observe from the figure that reduction of Error contributed by plugging-in pattern-based summarizers is small for both algorithms.

Dimensionality Restriction. For *Laserlight*, the observation is partially due to the fact that we only keep top 100 features (in terms of variability) of the data as its input, since *Laserlight* is implemented in PostgresSQL 9.1 which has a threshold of 100 arguments (one argument for each feature) that can be passed to a function.

Pattern Restriction. For *MTV*, this is due to a runtime error that limits us to 15 or less patterns. We refer the reader to Section 4.5 in [35] that explains the difficulty in inferring the maximum entropy distribution constrained by a large number of non-naive patterns.

8. ALTERNATIVE APPLICATIONS

To fairly evaluate *Laserlight* and *MTV*, we incorporate their own data sets and empirically evaluate them against *naive mixture encoding* under their own applications.

Data Sets. Specifically, we choose *Mushroom* data set used in *MTV* [35] which is obtained from FIMI dataset repository and U.S. Census data on Income or simply *Income* data set, which is downloaded from IPUMS-USA at *https://usa.ipums.org/usa/* and used in *Laserlight* [16]. The basic statistics of the data sets are given in Table 2.

Table 2: **Data Sets of Alternative Applications**

Statistics	Income	Mushroom
# Distinct data tuples	777493	8124
# Features per tuple	9	21
Feature Binary-valued?	no	no
# Distinct features	783	95
Binary Classification Feature	$> 100,000$?	Edibility
Assumed data tuple multiplicity	1	1

8.1 Experiments

All experiments involving *Laserlight* and *MTV* will be evaluated under their own Error measures and data sets, unless otherwise stated. The experiments are organized as follows: First, we establish baselines by evaluating classical *Laserlight* and *MTV* on their original data; Then we show that classical *Laserlight* and *MTV* can be generalized to partitioned data and that the generalization improves on their Error measures and also runtime; At last, we compare their generalized versions with *naive mixture encoding* to show that *naive mixture encoding* is a reasonable alternative.

8.1.1 Error Measures

We first explain how *naive mixture encoding* is evaluated based on Error defined by *Laserlight* and *MTV*.

Evaluating Naive Encoding on Laserlight Error. Algorithm *Laserlight* summarizes data D which consists of feature vectors t augmented by some binary feature v. Denote the valuation of the binary feature v for each feature vector t as $v(t)$. The goal is to mine a summary encoding $\mathcal{E}$, which is a set of patterns contained in $t \in D$ that offer predictive power on $v(t)$. Denote the estimation (based on $\mathcal{E}$) of $v(t)$ as $u_{\mathcal{E}}(t) \in [0, 1]$, the *Laserlight* Error is measured by

$$\sum_t (v(t) \log(\frac{v(t)}{u_{\mathcal{E}}(t)}) + (1 - v(t)) \log(\frac{1 - v(t)}{1 - u_{\mathcal{E}}(t)}))$$

Since *naive encoding* $\ddot{\mathcal{E}}$ assumes feature independence, estimation of $v(t)$ is independent of t, namely $u_{\ddot{\mathcal{E}}}(t) = u_{\ddot{\mathcal{E}}} =$

$|\{\tau|v(\tau) = 1, \tau \in D\}|/|D|$. Consequently, the *Laserlight* Error of *naive encoding* is

$$-|D|(u_{\ddot{\mathcal{E}}} \log u_{\ddot{\mathcal{E}}} + (1 - u_{\ddot{\mathcal{E}}}) \log(1 - u_{\ddot{\mathcal{E}}}))$$

Evaluating Naive Encoding on MTV Error. Given binary feature vectors D, the *MTV* Error of encoding $\mathcal{E}$ is

$$-|D|H(\overline{\rho}_{\mathcal{E}}) + 1/2|\mathcal{E}| \log |D|$$

where $H(\overline{\rho}_{\mathcal{E}})$ is the entropy of maximum entropy distribution $\overline{\rho}_{\mathcal{E}}$ defined in Section 4.1. The second term in *MTV* Error penalizes Verbosity of the encoding $\mathcal{E}$. Since naive encoding assumes feature independence, we can first compute entropy of the marginal distribution of each individual feature. Entropy $H(\overline{\rho}_{\mathcal{E}})$ is simply the sum of feature entropies.

Evaluating Naive Mixture Encoding. Evaluation of *naive encoding* can be generalized to *naive mixture* by taking a weighted average over resulting clusters (See Section 5.2).

8.1.2 Classical Laserlight and MTV

Establishing Baselines. To establish baselines, we evaluate *Laserlight* and *MTV* on their own data sets. The takeaways from related experiments are that (1) *naive encoding* is faster and more accurate than classical *Laserlight* and *MTV*; (2) the runtime increases superlinearly with the number of patterns mined from both *Laserlight* and *MTV*. For detailed experiment results, we refer the reader to [45].

Anti-correlation and Dimentionality Reduction. Recall in Section 7.2.2 that *Laserlight* is restricted to 100 features. For its own *Income* data set, *Laserlight* can be applied with its full set of 783 features. This is due to the prior knowledge that the 783 features belong to 9 groups. In each group, features are mutually anti-correlated which can be reduced to a single feature. Similarly, *Mushroom* data set can be reduced from 95 to 21 features.

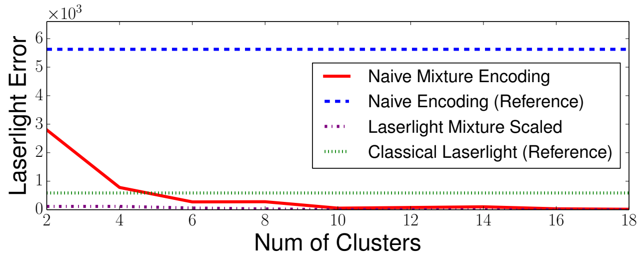
8.1.3 Generalizing Laserlight and MTV

We generalize *Laserlight* and *MTV* on partitioned data by applying them on each cluster. We then combine Errors on all clusters by taking a weighted average, as described in Section 5.2. Depending on how many patterns are mined from each cluster, *Laserlight* and *MTV* can be generalized into two types: (1) The number of patterns mined from each cluster is scaled to be equal to Verbosity of the *naive encoding*; and (2) The total number of patterns mined from all clusters is fixed to a given number. We name the first type *Laserlight (MTV) Mixture Scaled*, which is comparable to *naive mixture encoding*. We name the second type *Laserlight (MTV) Mixture Fixed*, which is comparable to the classical *LaserLight (MTV)* algorithm.

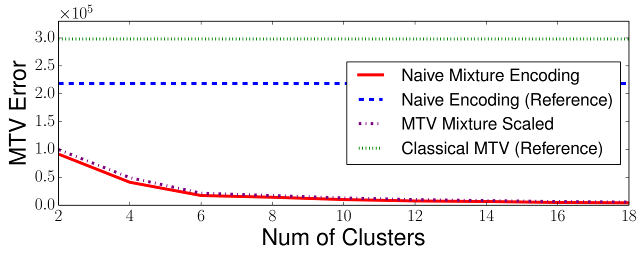
Take-away. As the data is partitioned into more clusters, both runtime and Error of *Laserlight (MTV) Mixture Fixed* exponentially decrease. This observation can be potentially generalized to other pattern mining algorithms. For experiment details, we refer the reader to [45].

8.1.4 Comparison with Naive Mixture Encoding

At last, we compare *Laserlight (MTV) Mixture Scaled* with *naive mixture encoding*. Note that it is time-consuming for *Laserlight* to mine the same number of patterns as *naive encoding* on *Income* data (See runtime analysis in [45]), we choose *Mushroom* data for *Laserlight Mixture Scaled* instead. The experiment results are given in Figure 6. The x-axis for



(a) **Laserlight Error v. # of Clusters on Mushroom data**



(b) **MTV Error v. # of Clusters on Mushroom data**

Figure 6: **Naive Mixture v. Laserlight/MTV Mixture**

all sub-figures in Figure 6 represents the number of clusters and the y-axes stands for *Laserlight* and *MTV* Error respectively. We incorporate baselines (i.e., *naive encoding*, classical *Laserlight* and *MTV*) as reference lines in Figure 6a and 6b respectively. We also experienced a limitation of 15 patterns for *MTV*. Hence the comparison between *MTV Mixture Scaled* and naive mixture encoding is not strictly on equal footing as *MTV Mixture Scaled* is not able to reach the same Total Verbosity as *naive mixture encoding*. Note that their difference in Verbosity is mitigated by the fact that *MTV* Error measure penalizes encoding Verbosity.

Figure 6a shows that both *naive mixture encoding* and *Laserlight Mixture Scaled* have lower Error than their baselines. In addition, *Laserlight Mixture Scaled* has lower Error than *naive mixture encoding* when the number of clusters is less than 4 and they become close after 6 clusters. In other words, *Laserlight* is more accurate on lightly partitioned data. As the data is further partitioned, clusters become easier to summarize, and *naive encoding* becomes more similar to *Laserlight*. Figure 6b shows that *naive mixture encoding* marginally outperforms *MTV Mixture Scaled*.

Take-away. *Naive mixture encoding* is faster and has similar (lower) Error than *Laserlight (MTV) Mixture Scaled*.

9. RELATED WORK

We aim at compressing query logs for accurately and efficiently computing workload statistics. Before the discussion of compression, we first review usecases and related work for workload analysis.

9.1 Workload Analysis

Existing approaches related to workload analysis usually aim at specific tasks like query recommendation [36, 17, 28, 46, 3], performance optimization [7, 11], outlier detection [25] or visual analysis [34].

Query Recommendation. This task aims at tracking historical querying behavior and generating query recommendations. Related approaches [36, 28] flatten a query *abstract syntax tree* as a set of *fragments* [36] or *snippets* [28].

User profiles are then built by grouping and summarizing queries of specific users in order to make personalized recommendation. Under OLAP systems, profiles are also built for workloads of similar OLAP sessions [3].

Performance Optimization. Index selection [13, 15] and materialized view selection [2, 7, 11] are typical performance optimization tasks. The configuration search space is usually large, but can be reduced with appropriate summaries.

Outlier Detection. Kamra *et al.* [25] aim at detecting anomalous behavior of queries in the log by summarizing query logs into profiles of normal user behavior.

Visual Analysis. Makiyama *et al.* [34] provide a set of visualizations that facilitate further workload analysis on Sloan Digital Sky Survey (SDSS) dataset. QueryScope [20] aims at finding better tuning opportunities by helping human experts to identify patterns shared among queries.

In these approaches, queries are commonly encoded as feature vectors or bit-maps where a bit array is mapped to a list of features with 1 in a position if the corresponding feature appears in the query and 0 otherwise. Workloads under the bit-map encoding must then be compressed before they can be efficiently queried or visualized for analysis.

9.2 Workload Compression Schemes

Run-length Encoding. *Run-length encoding (RLE)* is a loss-less compression scheme commonly used in *Inverted Index Compression* [43, 49] and *Column-Oriented Compression* [1]. RLE-based compression algorithms include but not limited to: Byte-aligned Bitmap Code (BBC) used in Oracle systems [6], Word-aligned Hybrid (WAH) [44] and many others [37, 4, 5]. In general, RLE-based methods focus on column-wise compression and requires additional heavyweight inference on frequencies of cross-column (i.e., row-wise) patterns used for workload analysis.

Lempel-Ziv Encoding. Lempel-Ziv [47, 48] is the loss-less compression algorithm used by gzip. It takes variable sized patterns (row-wise in our case) and replaces them with fixed length codes, in contrast to Huffman encoding [21]. Lempel-Ziv encoding does not require knowledge about pattern frequencies in advance and builds the pattern dictionary dynamically. There are many other similar schemes for compressing files represented as sequential bit-maps, e.g. [41].

Dictionary Encoding. *Dictionary encoding* is a more general form of Lempel-Ziv. It has the advantage that patterns with frequencies stored in the dictionary can be interpreted as workloads statistics useful for analysis. In this paper, we extend dictionary encoding and focus on using a dictionary to infer frequencies of patterns not in it. Mampaey *et al.* proposed *MTV* algorithm [35] that finds the dictionary (of given size) having optimal *Bayesian Information Criterion(BIC)* score. Gebaly *et al.* proposed *Laserlight* algorithm [16] that builds a pattern dictionary for correctly inferring the truth-value of some augmented binary feature.

Generative Models. A generative model is a lossy compressed representation of the original log. Typical generative models are *probabilistic topic models* [8, 42] and *noisy-channel* model [29]. Generative models can infer pattern frequencies but they lack a model-independent measure for efficiently evaluating overall inference accuracy.

Matrix Decomposition. Matrix decomposition methods including Principal Component Analysis (PCA) [24] and Non-negative matrix factorization (NMF) [33] offer lossy data compression. But the resulting matrices after decomposition are not suited for inferring workload statistics.

10. CONCLUSIONS

In this paper, we introduced the problem of log compression and defined a family of pattern-based log encodings. We precisely characterized the information content of logs and offered three principled and one practical measures of encoding quality: Verbosity, Ambiguity, Deviation and Reproduction Error. To reduce the search space of pattern-based encodings, we introduced the idea of log partitioning, which induces the family of pattern mixture as well as its simplified form: naive mixture encodings. Finally, we experimentally showed that naive mixture encodings are more informative and can be constructed more efficiently than state-of-the-art pattern-based summarization techniques. We expect that making accurate and efficient inference on pattern frequencies will enable a range of more powerful database tuning and intrusion detection systems.

11. FUTURE WORK

Multiplicity-aware clustering. As the number of feature vectors can be millions or more, practically we only keep *distinct* feature vectors as input of clustering schemes. We can store feature vector frequencies in a separate column called *multiplicities.* A multiplicity-ignorant clustering scheme assumes a uniform distribution of queries in the log. However, query distributions $p(Q)$ of production database logs are usually skewed. For example, routine queries repeat themselves overwhelmingly in the log but contribute to a minority of distinct queries. We plan to improve naive mixture encodings by exploring *multiplicity-aware* clustering schemes such that distinct feature vectors can be clustered *as if they have been replicated.* The use of mixture models for summarization has potential implications for work on pattern mining; As we show, existing techniques can be substantially improved both in runtime and Error.

Feature Clustering. For the usecase of materialized view selection, computing pattern frequencies may not be enough. We may need to summarize a query log as a limited set of *basis* views such that queries in the log can be represented by a simple join of a subset of basis views. Capturing basis views is not only relevant to data tuning tasks, but also facilitates human inspection of workloads in the log. To achieve the goal, in addition to partitioning queries into separate workload clusters, for each cluster we need to further partition its features into separate clusters where each cluster is equivalent to a *basis view.*

12. ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their helpful comments and contributions. This work was supported by NSF Awards SaTC-1409551 and IIS-1750460. The conclusions and opinions in this work are solely those of the authors and do not represent the views of the National Science Foundation.

13. REFERENCES

[1] D. J. Abadi, S. Madden, and M. Ferreira. Integrating compression and execution in column-oriented database systems. In S. Chaudhuri, V. Hristidis, and N. Polyzotis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 671–682. ACM, 2006.

[2] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K. Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 496–505. Morgan Kaufmann, 2000.

[3] J. Aligon, M. Golfarelli, P. Marcel, S. Rizzi, and E. Turricchia. Similarity measures for OLAP sessions. *Knowl. Inf. Syst.*, 39(2):463–489, 2014.

[4] S. Amer-Yahia and T. Johnson. Optimizing queries on compressed bitmaps. In A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K. Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 329–338. Morgan Kaufmann, 2000.

[5] G. Antoshenkov. Byte-aligned bitmap compression. In *Proceedings of the Conference on Data Compression*, DCC '95, pages 476–, Washington, DC, USA, 1995. IEEE Computer Society.

[6] G. Antoshenkov and M. Ziauddin. Query processing and optimization in oracle rdb. *VLDB J.*, 5(4):229–237, 1996.

[7] K. Aouiche, P. Jouve, and J. Darmont. Clustering-based materialized view selection in data warehouses. In Y. Manolopoulos, J. Pokorný, and T. K. Sellis, editors, *Advances in Databases and Information Systems, 10th East European Conference, ADBIS 2006, Thessaloniki, Greece, September 3-7, 2006, Proceedings*, volume 4152 of *Lecture Notes in Computer Science*, pages 81–95. Springer, 2006.

[8] D. M. Blei. Probabilistic topic models. *Commun. ACM*, 55(4):77–84, 2012.

[9] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[10] N. Bruno and S. Chaudhuri. Automatic physical database tuning: A relaxation-based approach. In F. Özcan, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 227–238. ACM, 2005.

[11] N. Bruno, S. Chaudhuri, and L. Gravano. Stholes: A multidimensional workload-aware histogram. In S. Mehrotra and T. K. Sellis, editors, *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, pages 211–222. ACM, 2001.

[12] G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, N. Polyzotis, and J. S. V. Varman. The querie system for personalized query recommendations. *IEEE Data Eng. Bull.*, 34(2):55–60, 2011.

[13] S. Chaudhuri and V. R. Narasayya. An efficient cost-driven index selection tool for microsoft SQL server. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 146–155. Morgan Kaufmann, 1997.

[14] C. Dwork. Differential privacy. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2006.

[15] S. J. Finkelstein, M. Schkolnick, and P. Tiberio. Physical database design for relational databases. *ACM Trans. Database Syst.*, 13(1):91–128, 1988.

[16] K. E. Gebaly, P. Agrawal, L. Golab, F. Korn, and D. Srivastava. Interpretable and informative explanations of outcomes. *PVLDB*, 8(1):61–72, 2014.

[17] A. Giacometti, P. Marcel, E. Negre, and A. Soulet. Query recommendations for OLAP discovery-driven analysis. *IJDWM*, 7(2):1–25, 2011.

[18] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, Mar. 2014.

[19] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: current status and future directions. *Data Min. Knowl. Discov.*, 15(1):55–86, 2007.

[20] L. Hu, K. A. Ross, Y. Chang, C. A. Lang, and D. Zhang. Queryscope: visualizing queries for repeatable database tuning. *PVLDB*, 1(2):1488–1491, 2008.

[21] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, Sept 1952.

[22] A. K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[23] E. T. Jaynes. Prior probabilities. *IEEE Trans. Systems Science and Cybernetics*, 4(3):227–241, 1968.

[24] I. T. Jolliffe. Principal component analysis. In M. Lovric, editor, *International Encyclopedia of Statistical Science*, pages 1094–1096. Springer, 2011.

[25] A. Kamra, E. Terzi, and E. Bertino. Detecting anomalous access patterns in relational databases. *VLDB J.*, 17(5):1063–1077, 2008.

[26] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *J. ACM*, 51(3):497–515, 2004.

[27] O. Kennedy, J. A. Ajay, G. Challen, and L. Ziarek. Pocket data: The need for TPC-MOBILE. In R. Nambiar and M. Poess, editors, *Performance Evaluation and Benchmarking: Traditional to Big Data to Internet of Things - 7th TPC Technology Conference, TPCTC 2015, Kohala Coast, HI, USA, August 31 - September 4, 2015. Revised Selected Papers*, volume 9508 of *Lecture Notes in Computer Science*, pages 8–25. Springer, 2015.

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

[29] K. Knight and D. Marcu. Summarization beyond

sentence extraction: A probabilistic approach to sentence compression. *Artif. Intell.*, 139(1):91–107, 2002.

[30] G. Kul, D. Luong, T. Xie, P. Coonan, V. Chandola, O. Kennedy, and S. J. Upadhyaya. Ettu: Analyzing query intents in corporate databases. In J. Bourdeau, J. Hendler, R. Nkambou, I. Horrocks, and B. Y. Zhao, editors, *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11-15, 2016, Companion Volume*, pages 463–466. ACM, 2016.

[31] G. Kul, D. T. A. Luong, T. Xie, V. Chandola, O. Kennedy, and S. Upadhyaya. Similarity metrics for sql query clustering. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2408–2420, Dec 2018.

[32] G. Kul, S. J. Upadhyaya, and V. Chandola. Detecting data leakage from databases on android apps with concept drift. In *IEEE TrustCom*, pages 905–913, 2018.

[33] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788, 1999.

[34] V. H. Makiyama, J. Raddick, and R. D. C. Santos. Text mining applied to SQL queries: A case study for the SDSS skyserver. In J. A. Lossio-Ventura and H. Alatrista-Salas, editors, *Proceedings of the 2nd Annual International Symposium on Information Management and Big Data - SIMBig 2015, Cusco, Peru, September 2-4, 2015.*, volume 1478 of *CEUR Workshop Proceedings*, pages 66–72. CEUR-WS.org, 2015.

[35] M. Mampaey, J. Vreeken, and N. Tatti. Summarizing data succinctly with the most informative itemsets. *TKDD*, 6(4):16:1–16:42, 2012.

[36] S. Mittal, J. S. V. Varman, G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Querie: A query recommender system supporting interactive database exploration. In W. Fan, W. Hsu, G. I. Webb, B. Liu, C. Zhang, D. Gunopulos, and X. Wu, editors, *ICDMW 2010, The 10th IEEE International Conference on Data Mining Workshops, Sydney, Australia, 13 December 2010*, pages 1411–1414. IEEE Computer Society, 2010.

[37] A. Moffat and J. Zobel. Compression and fast indexing for multi-gigabyte text databases. *Australian Computer Journal*, 26(1):1–9, 1994.

[38] B. O'Donoghue, E. Chu, N. Parikh, and S. P. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *J. Optimization Theory and Applications*, 169(3):1042–1068, 2016.

[39] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. C. Mowry, M. Perron, I. Quah, S. Santurkar, A. Tomasic, S. Toor, D. V. Aken, Z. Wang, Y. Wu, R. Xian, and T. Zhang. Self-driving database management systems. In *CIDR*, 2017.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[41] P. Skibinski and J. Swacha. Fast and efficient log file compression. In Y. E. Ioannidis, B. Novikov, and B. Rachev, editors, *Communications of the Eleventh East-European Conference on Advances in Databases and Information Systems, Varna, Bulgaria, September 29 - October 3, 2007*, volume 325 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[42] D. Wang, S. Zhu, T. Li, and Y. Gong. Multi-document summarization using sentence-based topic models. In *ACL 2009, Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2-7 August 2009, Singapore, Short Papers*, pages 297–300. The Association for Computer Linguistics, 2009.

[43] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, 1994.

[44] K. Wu, E. J. Otoo, and A. Shoshani. Compressing bitmap indexes for faster search operations. In *Proceedings of the 14th International Conference on Scientific and Statistical Database Management, July 24-26, 2002, Edinburgh, Scotland, UK*, pages 99–108. IEEE Computer Society, 2002.

[45] T. Xie, O. Kennedy, and V. Chandola. Query log compression for workload analytics. *CoRR*, abs/1809.00405, 2018.

[46] X. Yang, C. M. Procopiuc, and D. Srivastava. Recommending join queries via query log analysis. In Y. E. Ioannidis, D. L. Lee, and R. T. Ng, editors, *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 964–975. IEEE Computer Society, 2009.

[47] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Information Theory*, 23(3):337–343, 1977.

[48] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Information Theory*, 24(5):530–536, 1978.

[49] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2):6, 2006.