FPO

Enhancing Selectivity in Big Data

Mathias Lecuyer, Riley Spahn, and Roxana Geambasu | Columbia University
Tzu-Kuo Huang | Uber Advanced Technologies Group
Siddhartha Sen | Microsoft Research

Today's companies collect immense amounts of personal data, exposing it to external hackers and privacy-transgressing employees. This study shows that only a fraction of the data is needed to approach state-of-the-art accuracy. We propose selective data systems designed to pinpoint the data that is valuable for a company's workloads.

Driven by the immense perceived potential of "big data," Internet companies, advertisers, and governments are accumulating vast quantities of personal data: clicks, locations, social interactions, and more. While data offers unique opportunities to improve personal and business effectiveness, its aggressive collection and long-term archival pose significant risks for organizations. Hacking and exploiting sensitive corporate and governmental information have become commonplace. Privacy-transgressing employees have been discovered snooping into data stores to spy on friends and family. Although organizations strive to restrict access to particularly sensitive data (such as passwords, SSNs, emails, banking data), properly managing access controls for diverse and potentially sensitive information is an open problem.

We hypothesize that not all data that is collected and archived by today's organizations is—or may ever be—actually needed to satisfy their workloads. We ask whether it is possible to architect data-driven systems, such as machine learning–based targeting and personalization systems, to permit a clean separation between data that is truly needed by an organization's current and evolving workload, from data that is collected for potential future needs. The former, called *in-use data*, should be minimized in size, timespan, and sensitivity. The latter, called *unused data*, should be set aside and tapped only in exceptional circumstances (see Figure 1). The separation should permit day-to-day evolutions of an organization's workload, by accessing just the in-use data and without the need to tap into the unused data. A system that achieves these goals without damaging functional properties, such as scalability, performance, and accuracy, is called a *selective data system*.

Selective Data Systems

Selective data systems can be used to improve data protection (see Figure 1). The ability to distinguish data needed now or in the likely future, from data collected "just in case," can help organizations restrict the latter's exposure to attacks. For example, one could ship unused data to a tightly controlled store, whose read accesses are carefully audited and mediated. Intuitively, data that is accessed day-to-day is less amenable to certain kinds of protection (such as auditing or case-by-case access control decisions) than data

 Copublished by the IEEE Computer and Reliability Societies

accessed only for exceptional situations (such as launching a new application).

Turning selective data systems into a reality requires achieving two conflicting goals: (1) minimizing the in-use data while (2) avoiding the need to access the unused data to meet both current and evolving workload needs. This tension is traditional in operating systems, where many algorithms (for instance, caching) rely on processes having a *working set* of limited size that captures their data needs for a period of time. However, the context of modern, data-driven ecosystems brings new challenges that likely make traditional working set algorithms ineffective. For example, many of today's big data applications involve machine learning (ML) workloads that are periodically retrained to incorporate new data, by accessing all of the data. How can we determine a minimal *training set*, the "working set" for emerging ML workloads? And how can we ensure this training set is sufficient even when workloads evolve?

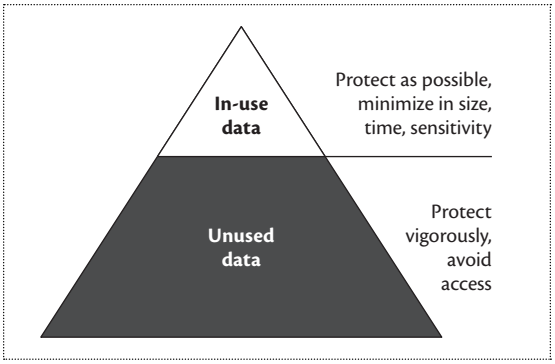

Figure 1. Selectivity concept.

Approach Highlights

We observe that for ML workloads, significant research is devoted to limiting the amount of data required for training. The reasons are many but typically do not involve data protection. Rather, they include increasing performance, dealing with sparsity, and limiting labeling effort. Techniques such as dimensionality reduction, feature hashing,[1] vector quantization,[2] and count featurization[3] are routinely applied in practice to reduce data dimensionality so models can be trained on manageable training sets. Active learning[4] reduces the amount of labeled data needed for training when labeling requires manual effort. Can such mechanisms also be used to limit exposure of the data being collected? How can an organization that already uses these methods architect a selective data system around them? What kinds of protection guarantees can this system provide?

As a first step to answering these questions, we present Pyramid,[5] a selective data system built around a specific training set minimization method called *count featurization*.[3] (Pyramid was first introduced at the 2017 IEEE Symposium on Security and Privacy.) Count featurization is a widely used technique for reducing training sets by feeding ML algorithms with a small subset of the collected data combined (or *featurized*) with historical aggregates from much larger amounts of data. Pyramid builds upon count featurization to: keep a small, rolling window of accessible in-use data (the *hot window*); summarize the history with privacy-preserving aggregates (called *counts*); and train application models using hot window data featurized with counts. The counts are infused with differentially private noise[6] to protect individual observations that are no longer in the hot window. Counts can support a variety of models that fall within the important class of supervised classification tasks. Historical raw data, which may be needed for workloads not supported by count featurization, such as unsupervised learning or regression tasks, is kept in an encrypted store whose decryption requires special access.

Our evaluation with two representative workloads—targeted advertising on the Criteo dataset and movie recommendation on the MovieLens dataset—reveals that: (1) historical counts let ML models approach state-of-the-art accuracy by training on under 1 percent of the data, (2) protecting historical counts with differential privacy has only 2 percent impact on accuracy, and (3) Pyramid works well for an important class of ML algorithms—supervised classification tasks—and can support workload evolution within that class.

Example Use Case

MediaCo, a media conglomerate, collects *observations* of user behavior from its hundreds of affiliate news and entertainment sites. Observations include the articles users read and share, the ads they click, and so on. MediaCo integrates all of this data into one repository and uses it to optimize many processes, such as article recommendation and ad targeting. Because the data is needed by all of its engineering teams, MediaCo wants to provide them with wide access to the repository, but it worries about the risks of doing so given recent external hacking and insider attacks affecting other companies.

MediaCo decides to use Pyramid to limit the exposure of historical observations in anticipation of an attack. For MediaCo's main workloads—targeting and personalization—the company already uses count featurization to address sparsity challenges; hence, Pyramid is directly applicable. It configures Pyramid by keeping its hot window of raw observations and its noise-infused historical counts in the widely accessible repository, allowing all engineers to train their models, tune them, and explore new algorithms. Pyramid

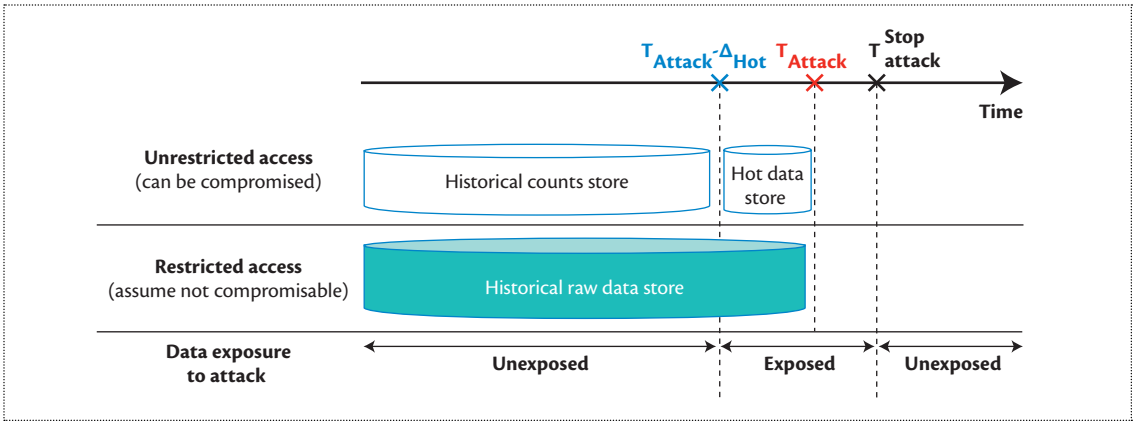

Figure 2. Threat model.

absorbs current and evolving workload needs as long as the algorithms draw on the same user data to predict the same outcome (for instance, whether a user will click on an ad). In addition, MediaCo stores all raw observations in an encrypted store whose read accesses are disabled by default. Access to this store is granted temporarily and on a case-by-case basis to engineers who demonstrate the need for statistics beyond those that Pyramid maintains. With this configuration, MediaCo minimizes data access (and hence exposure) to a needs basis.

Threat Model

Figure 2 illustrates Pyramid's threat model and guarantees. Pyramid ensures that a one-time compromise will not allow an adversary to access past data. Attacks are assumed to have a well-defined start time, T_{attack}, when the adversary gains access to the machines charged with running Pyramid, and a well-defined end time, T^{stop}_{attack}, when administrators discover and stop the intrusion. Adversaries are assumed to not have had access to the system before T_{attack}, nor to have performed any action in anticipation of their attack (for example, monitoring external predictions, the hot window, or the models' state), nor to have continued access after T^{stop}_{attack}. The attacker aims to exfiltrate individual observations of user activities (for instance, to know if a user clicked on a specific ad). Historical raw data is assumed to be protected through independent means and not compromised in this attack. Pyramid aims to limit the hot data and protect the historical counts, both of which are accessible to the attacker.

After compromising Pyramid's internal state, the attacker gains access to data in three representations: the hot data store containing raw observations, the historical counts, and the trained models. The raw observations are not protected in any way. The historical counts consist of differentially private count tables of the recent past. The attacker learns some aggregate information from the count tables but individual records will be protected with differential privacy. Pyramid forces models to be retrained when observations are removed from the hot raw data store to avoid past information leaking through the models. We assume that no out-of-bound copies of the hot data exist.

Selectivity Requirements

Four requirements define selective data systems:

- R1: *Reduce in-use, exposed data*. The hot data window is exposed to attackers; hence, Pyramid must limit its size and timespan subject to application-level functional requirements, such as the accuracy of models trained with it.
- R2: *Protect unused data from in-use data structures*. Any state reflecting past, unused data and retained by Pyramid for prolonged periods of time (such as count tables) must be protected with strong, differential privacy guarantees.
- R3: *Limit impact on accuracy and performance*. Pyramid must preserve the functional properties of applications, such as model accuracy. This requirement is at odds with the preceding two, hence Pyramid must find a balance between functionality and protection.
- R4: *Allow workload evolution*. The in-use data must support as many current and evolving workload needs as possible to limit access to, and therefore exposure of, the historical raw data.

The Pyramid Architecture

Pyramid combines and augments two known methods—count featurization from ML and differential privacy from cryptography—to meet the preceding selectivity requirements for a specific class of workloads: classification tasks such as targeting and personalization. Figure 3 shows Pyramid's architecture. Pyramid is a

data management component to be deployed alongside a model management system. It acts as an interface between the model manager and the organization's datasets. Pyramid controls what data is exposed to the models and the format of this data.

Pyramid maintains two data structures—the hot raw data store and the historical count tables—and leverages two functional building blocks: count featurization (CF) and differential privacy (DP). The hot data store is a cache of recent data containing raw observations (feature vector, label pairs) that are transformed using CF and used for training. It is a sliding window cache that we expect to contain on the order of days or weeks of data. The historical count tables (or *count tables* for short) store the number of times each feature value has been observed with each label. We leverage DP to protect the past observations used to construct the count tables. We expect the count tables to be populated with months' or years' worth of data, and erase observations from count tables past a given retention period.

Pyramid exposes a small API, which the model manager uses to interact with the organization's data: `featurize(`$\vec{x}$`)` and `getTrainSet()`. On prediction requests, the model manager calls `featurize(`$\vec{x}$`)` to obtain a count-featurized version of the raw feature vector $\vec{x}$, denoted $\vec{x'}$. Pyramid also manages new observations added to the system. Given a new observation, consisting of a label (l) and feature vector ($\vec{x}$), Pyramid updates the appropriate count tables, adds the observation to the hot data store, and submits it to the historical raw data store. To train its models, the model manager calls `getTrainSet()` to obtain a count-featurized version of all observations in the hot data store.

The two building blocks—CF and DP—address the first two selectivity requirements from the previous section. CF, a known training set minimization method, reduces the amount of in-use data needed to train application-level models (selectivity requirement R1). It works by augmenting recent hot data with historical counts. DP protects the unused, phased-out observations from the historical count tables and other in-use data structures (selectivity requirement R2). Unfortunately, these methods raise substantial accuracy, scalability, and evolution challenges. We address the challenges by augmenting CF and DP with a set of novel mechanisms to meet the remaining selectivity requirements R3 and R4. Following are four technical sections that describe our base CF and DP processes, plus two of our augmentation mechanisms (for others, see our symposium paper[5]). These sections provide a blueprint for how to build a selective data system based on training set minimization.

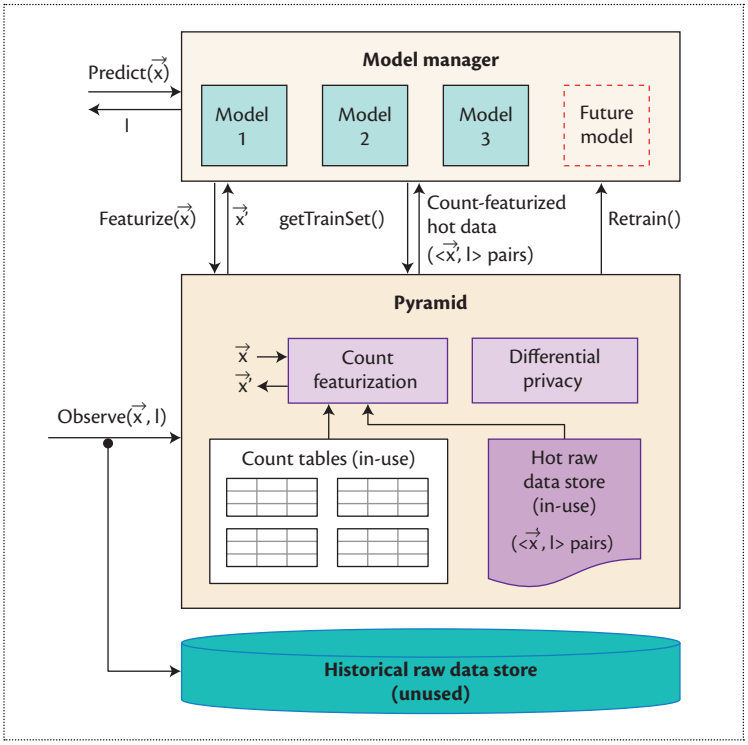

Figure 3. The Pyramid architecture.

Count Featurization (R1: Reduce In-Use Data)

CF is a popular technique for handling high-cardinality categorical variables, such as unique identifiers, when training classification models.[3] CF replaces each feature vector value with the number of times that feature value has been observed with each label and the conditional probability of each label given that feature. This leads to dramatic dimensionality reduction over standard one-hot encoding. In a dataset where each observation contains d features with average feature cardinality K and labels of cardinality L, where $K \gg L$, a one-hot encoded feature vector would have size dK. CF results in a feature vector of size dL. In many cases this will be a dramatic reduction. For example, in click prediction, we expect L to be 2—click or nonclick—while K is very large, for instance, millions of user identifiers. Although no theoretical analysis exists for CF, intuitively, this dimensionality reduction should allow more efficient learning and reduce training data size without a loss in predictive power.

In Pyramid, we use CF to reduce the in-use data needed to train classification models (selectivity requirement R1). Figure 4 shows an example of count tables constructed by Pyramid to count-featurize observations as well as a sample count-featurized observation.

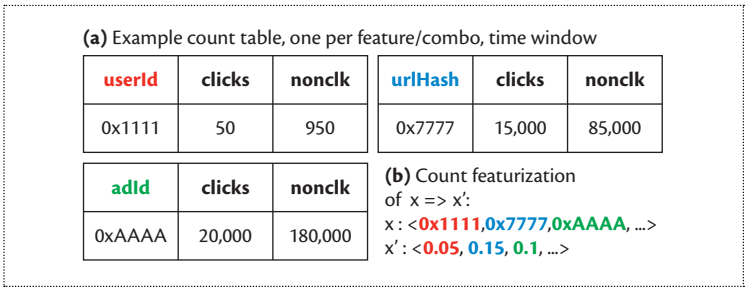

Figure 4. Count featurization example: (a) count tables constructed by Pyramid to count-featurize observations and (b) a sample count-featurized observation.

In more detail, an observation's feature vector $\vec{x}$ might consist of user features (for instance, ID, gender, age, preferences) and contextual information (for instance, the URL of the article or the ad shown to the user). The label l might indicate whether the user clicked on the article or ad.

Once an observation stream of the preceding type is registered with Pyramid, the system creates a number of count tables. Each count table is stored as a count sketch data structure (described shortly). For example, the *userId* table encodes the user's propensity to click on ads by maintaining for each user the total number of clicks and nonclicks on any ad shown.

To count-featurize a feature vector $\vec{x} = \langle x_1, x_2, ..., x_d \rangle$, Pyramid first replaces each of its features with the conditional probabilities computed from the count tables, for example, $\vec{x'} = \langle P(click|x_1), P(click|x_2), ..., P(click|x_d) \rangle$, where $P(click|x_i) = \frac{clicks}{clicks + nonclicks}$ from the row matching the value of x_i in the table corresponding to x_i. Pyramid also appends to $\vec{x'}$ the conditional probabilities for any feature combinations it maintains.

Figure 4b shows an example of feature vector $\vec{x}$ and its count-featurized version $\vec{x'}$. Suppose a boosted-tree model is trained on a count-featurized dataset ($\langle \vec{x}, l \rangle$ pairs). It might find that for users with a click propensity over 0.04, the chances of a click are high for ads whose clickability exceeds 0.05 placed on websites with ad-clickability over 0.1. Then, for the example feature vector in Figure 4, the model would predict a "click" label.

Differential Privacy (R2: Protect Unused Data from In-Use Structures)

DP comprises a family of techniques that randomize function outputs to protect function inputs. We use DP to protect unused, past observations from exposure through count tables and other in-use data structures (selectivity requirement R2). Let D_1 be the database of past observations, D_2 be a database that differs from D_1 by exactly one observation (that is, D_2 adds or removes one observation), and S the range of all possible count tables that can result from a randomized query $Q()$ that builds a count table from a window of observations. The count table query $Q()$ is ϵ-*differentially private* if $P[Q(D_1) \in S] \leq e^{\epsilon} \times P[Q(D_2) \in S]$.[6] Adding or removing an observation in D_1 does not significantly change the probability distribution of possible count tables. ϵ is the query's privacy budget.

In Pyramid, we apply DP as shown in Figure 5. We split time into windows and maintain count tables separately for each window of time. Upon rolling over the hot window, we seal the window's count tables and create new count tables for the new hot window. To create a count table, we initialize each cell in that table with a random draw from a Laplace distribution.[6] As observations arrive in the hot window, the count tables are updated by incrementing the appropriate cells. To count-featurize a feature x_i, we sum the corresponding entries in the feature's count tables across the past time windows, excluding the under-construction count tables for the current hot window. The sum constitutes a noisy version of x_i's count over the data retention period and is used to compute the conditional probabilities.

With this mechanism, Pyramid ensures that past observations, which have been phased out of the in-use, hot window, are protected from the count tables. To protect past observations from the trained models, Pyramid additionally forces retraining of all application models when rolling over the hot window.

Count-Median Sketch (R3: Limit Impact on Accuracy and Performance)

Although CF and DP are known mechanisms, their integration raises substantial accuracy challenges, which we have addressed by designing a number of mechanisms (selectivity requirement R3). As an example, we find that DP interacts poorly with count-min sketches, which are routinely used in CF implementations to keep the count table storage overhead practical. For a categorical variable of cardinality K and a label of cardinality L, the count table is of size $O(LK)$, an impractical prospect if one were to store the exact table. Count-min sketches store approximate counts in sublinear space by using a 2D array with an independent hash function for each row. Adding an entry to the sketch involves using the hash function associated with each row to map the value to a column in the row and incrementing that cell; reading an entry from the sketch involves taking the minimum of those cells. Without noise, taking the minimum reduces the chance of overcounting from collisions and results in tight error bounds for the counts. With Laplacian noise, which is centered around zero, taking the minimum across multiple draws of the

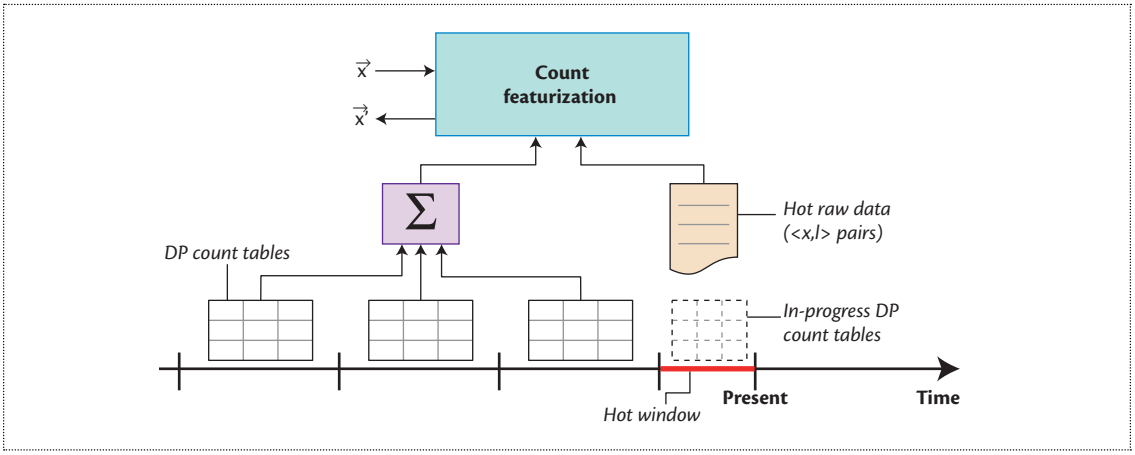

Figure 5. Windowed DP count tables.

Laplacian introduces substantial negative bias in the counts, breaking the sketch's error bounds.

Our solution is to instead use *count-median sketches* to store count tables compactly.[7] These differ from the count-min sketches in two ways: (1) each row i has a second hash function s_i that maps the key to a random sign s_i (key) $\epsilon \{+1,-1\}$, with each cell updated with s_i (key) h_i (key), and (2) the estimate is the median of all counts multiplied by its sign. Without noise, count-median sketches offer worse error bounds than count-min sketches for typical distributions. With Laplacian noise, the signed update of the count-median sketch means the expected impact of collisions is zero, because they have an equal chance of being negative or positive. This removes the bias and improves the quality of the count estimator. Our evaluation shows that for small ϵ, it is worth trading the count-min sketch's better guarantees for reduced noise impact with the count-median sketch.

Count Table Selection (R4: Allow Workload Evolution)

Two aspects of Pyramid's design enable workload evolution without tapping into the historical raw data store (selectivity requirement R4). First, CF is a model-independent preprocessing step, allowing Pyramid to support small model changes, such as hyperparameter tuning or learning algorithm changes, without accessing the historical raw data store. Second, Pyramid includes an automatic process of *count table selection* that inspects the data to identify *feature groups* worth counting together. This is important because CF does not capture relationships between features. For example, a *userId* and *adId* together may be more predictive than either of the individual features. That information could be inferred by a learning algorithm from the raw data, but it is lost through CF unless we explicitly maintain a count table for the (*userId*, *adId*) group.

Our goal in count table selection is to identify feature groups that provide more information about the label than individual features. For each feature x_i, we find all other features x_j such that x_i and x_j together exhibit higher mutual information (MI)—a general measure of dependence between two random variables—with the label than x_i alone. From these groups, we select a configurable number with the highest MIs. To find promising groups of larger sizes, we apply this process greedily, trying out new features with existing groups. For each selected group, Pyramid creates and maintains a count table. This exploration of promising groups operates periodically on the hot window. Count table selection must be performed differentially privately to ensure that the groups selected for a particular hot window do not leak information about that window in the future.

Evaluation

We evaluate how Pyramid meets the four selectivity requirements. We use two public datasets: (1) Criteo, which consists of 45M points of ad-click data that was part of a Kaggle competition and (2) MovieLens,[8] which consists of 22M ratings in the range 0–5 on 34K movies from 240K users. As baselines, we use a feed-forward neural network for Criteo and collaborative filtering for MovieLens, both trained using Vowpal Wabbit on 80/20 percent train/test splits. We highlight four results:

- R1: *CF reduces in-use data by two orders of magnitude while incurring less than 3 percent loss in accuracy*. Figure 6a shows that for MovieLens, the CF boosted-tree algorithms perform within 4 percent of the collaborative filtering baseline with only

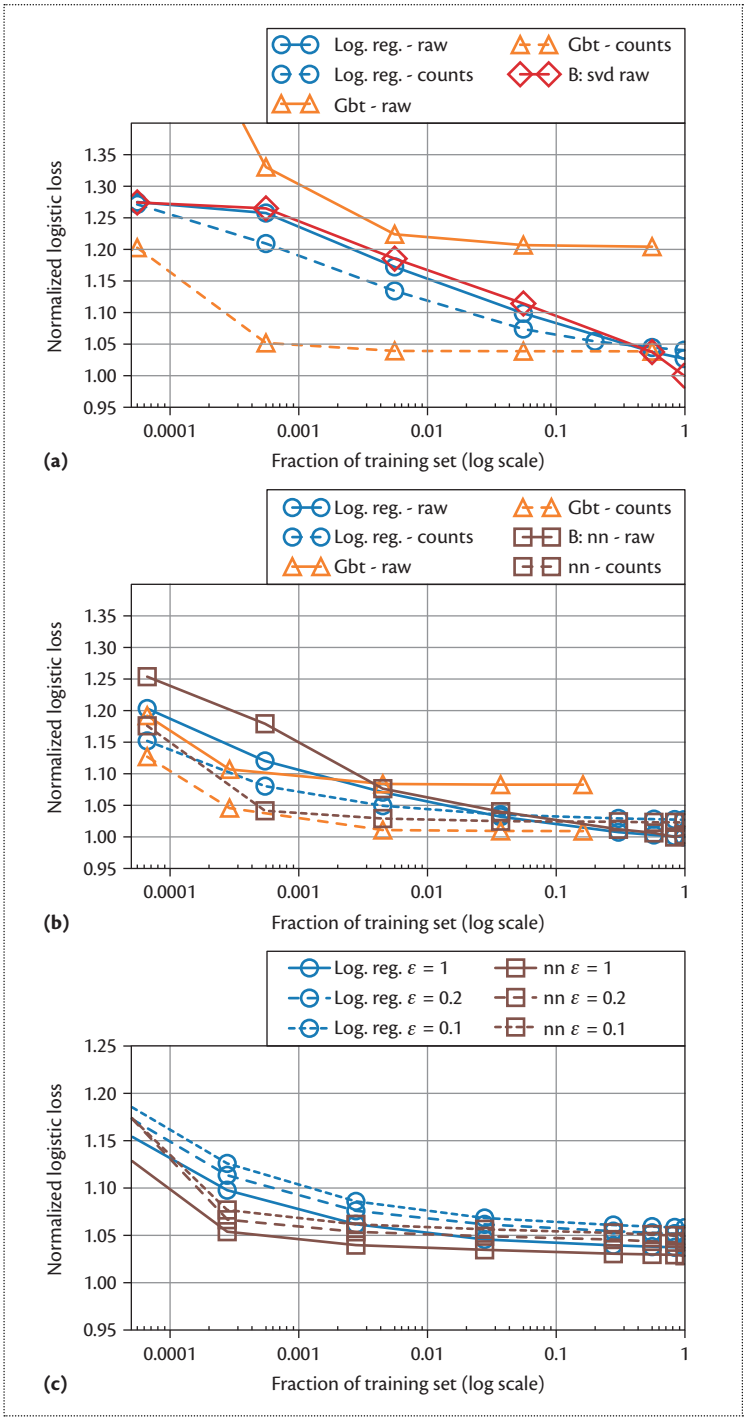

Figure 6. Normalized performance for raw and count algorithms: (a) MovieLens algorithms, (b) Criteo algorithms, and (c) Criteo protection (performance at different levels of protection). B indicates baseline: neural network for Criteo, collaborative filtering for MovieLens.

0.8 percent of the training set. This is on par with the raw logistic regression model. Figure 6b shows similar results with the Criteo dataset. The CF neural network performs within 3 percent of the baseline raw neural network when trained on 0.4 percent of the training set. CF fulfills its mission of reducing training set exposure and provides a good basis for selectivity. However, Pyramid must also protect past data using DP which may have a negative impact on accuracy.

- R2: *DP provides meaningful protection of unused data for up to 2 percent additional loss in accuracy*. Figure 6c shows the Criteo models' accuracy when count tables are protected with DP using various privacy budgets. A lower value of ϵ corresponds to higher levels of noise and increased protection. All of the Criteo models perform within 5 percent of the baseline when using $\epsilon = 0.2$ as a privacy budget, which is considered in the DP literature to give high-quality protection. MovieLens is less resilient to noise but still performs within 5 percent of the baseline when the privacy budget is at $\epsilon = 1$, a value still considered to offer meaningful protection. Thus, in both cases, Pyramid provides meaningful protection through data reduction with CF and past-data protection with DP. These results are obtained with the complete Pyramid implementation, which includes several mechanisms that augment CF and DP to address substantial accuracy challenges.[5] We next show the importance of one mechanism, the count-median sketch.
- R3: *Count-median sketch helps mitigate the negative impact on accuracy*. The count-median sketch resolves a tension arising when applying DP to standard implementations of CF, which rely on count-min sketches to compactly store count tables for high-dimensional features. Count-min sketches exhibit a strong downward bias when initialized with DP noise because they take the minimum, whereas count-median sketches take the (unbiased) median. For our workloads (MovieLens and Criteo), count-median sketches bring us 0.5 percentage points closer to the baseline losses than count-min sketches when training with 0.8 percent of the data and DP parameters $\epsilon = 1$. By contrast, without noise, count-min sketches perform better than count-median sketches, particularly for MovieLens, where the loss difference is about 2 percentage points. This result provides a broader lesson for anyone aiming to protect a sketch with DP: one should choose an unbiased sketch implementation even if it is suboptimal in no-noise scenarios.
- R4: *Supported workloads and evolution*. Our choice of CF as the core building block to address the first selectivity requirement—minimizing in-use data—exhibits both benefits and limitations. As a model-independent featurization layer, CF allows some common model changes without accessing historical raw data. Developers can fine-tune model hyperparameters, try different learning algorithms on the count-featurized data, change their learning

framework (for example, TensorFlow versus Vowpal Wabbit), and add/remove features that are already counted to/from their models. Indeed, our evaluation applied various algorithms and hyperparameters using the same count tables. Augmented with count table selection, CF can additionally allow developers to incorporate more complex count features into their models. For example, for MovieLens, the boosted-tree algorithm using Pyramid selected feature groups gets within 3 percent of the baseline loss, compared to within 4 percent without the groups, training on the same 0.8 percent of the raw data.

However, CF restricts the workloads we can support. CF is designed for *supervised classification tasks*, which are commonly used in industry for targeting and personalization. The preceding evaluations exercised such tasks. Regression and unsupervised classification tasks, also popular in industry, are not easily modeled with CF. In regression, the predicted label is continuous, while CF requires a discrete label to count its occurrences with various feature values. Regression tasks can be handled by binning the label value, but the outcome is poor. For example, we performed a regression for MovieLens that predicted a continuous 0–5 movie rating. Using as baseline a linear regression running on the full raw data, count models get at best within 16 percent of the baseline loss using linear regression and within 13 percent using gradient boosted-tree regression. Our vision for supporting a wider range of workloads is to incorporate into Pyramid other training set minimization and modeling mechanisms that support complementary workloads. The key challenge will be to preserve Pyramid's protection semantics and selectivity requirements.

Related Work

Multiple data minimization methods exist in ML and non-ML literature. Some are sketches, which compute compact representations of the data that support queries of summary statistics;[9] streaming/online algorithms[10] process data using bounded memory, retaining only the information relevant to the problem at hand; hash featurization[1] condenses high-cardinality categorical variables; autoencoders attempt to learn a compressed identity function for deep learning workloads;[11] and active learning[4] reduces the amount of labeled data needed for training. These methods have been shown to improve performance, robustness, or labeling cost, but have never been used for protection. This article gives a blueprint for how to retrofit one training set minimization method—CF—for protection. Applying this blueprint to other methods is a fruitful avenue for constructing selective data systems.

Pyramid applies differential privacy[6] to its count tables to protect individual observations. However, selectivity differs from the standard threat model used by systems that enforce differential privacy:[12–14] these systems guarantee privacy of released results, but assume that there is a trusted party to mediate access to the entire raw dataset. Pan privacy[15] has a similar threat model to ours, allowing a one-time compromise of the system, but to our knowledge no such system has been implemented or evaluated in practice. A major difference between Pyramid and all of this work is the exposure of raw data through the hot window. This brief exposure is what enables ML models to be trained with state-of-the-art performance.

Today's indiscriminate data collection and archival practices are risky and unsustainable. It is time for a more selective approach to big data collection and protection. Our vision involves architecting data systems to allow a clean separation of data needed by current and evolving workloads, from data collected and archived for possible future needs. The former should be minimized; the latter should be set aside, protected vigorously, and only tapped under exceptional circumstances. This article has formulated the requirements for selective data systems and presented Pyramid, an initial selective data system that combines and augments two known methods—count featurization and differential privacy—to meet these requirements for supervised classification workloads. Experiments with Pyramid show that data can be trimmed by two orders of magnitude without disrupting performance. ■

References

1. Q. Shi et al., "Hash Kernels for Structured Data," *The Journal of Machine Learning Research*, vol. 10, 2009, pp. 2615–2637.
2. A. Gersho and R.M. Gray, *Vector Quantization and Signal Compression*, vol. 159, Springer Science & Business Media, 2012.
3. A. Srivastava, A.C. Konig, and M. Bilenko, "Time Adaptive Sketches (Ada-Sketches) for Summarizing Data Streams," *ACM SIGMOD Conference*, 2016.
4. B. Settles, "Active Learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, no. 1, 2012, pp. 1–114.
5. M. Lecuyer et al., "Pyramid: Enhancing Selectivity in Big Data Protection with Count Featurization," *Proc. IEEE Symposium on Security and Privacy* (SP), 2017.
6. C. Dwork et al., "Calibrating Noise to Sensitivity in Private Data Analysis," *Proceedings of the Third Conference on Theory of Cryptography* (TCC 06), Springer-Verlag, 2006, pp. 265–284.

7. M. Charikar, K. Chen, and M. Farach-Colton, "Finding Frequent Items in Data Streams," *Proceedings of the 29th International Colloquium on Automata, Languages and Programming* (ICALP 02), Springer-Verlag, 2002, pp. 693–703.
8. F.M. Harper and J.A. Konstan, "The MovieLens Datasets: History and Context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, 2015, pp. 19:1–19:19.
9. G. Cormode and S. Muthukrishnan, "An Improved Data Stream Summary: The Count-Min Sketch and Its Applications," *Journal of Algorithms*, vol. 55, no. 1, 2005, pp. 58–75.
10. S. Shalev-Shwartz, "Online Learning and Online Convex Optimization," *Foundations and Trends in Machine Learning*, vol. 4, no. 2, 2011, pp. 107–194.
11. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, Cambridge, Massachusetts: The MIT Press, 2016.
12. F.D. McSherry, "Privacy Integrated Queries: An Extensible Platform for Privacy-Preserving Data Analysis," *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data* (SIGMOD 09), 2009, pp. 19–30.
13. I. Roy et al., "Airavat: Security and Privacy for MapReduce," *NSDI*, vol. 10, 2010, pp. 297–312.
14. F. McSherry and I. Mironov, "Differentially Private Recommender Systems: Building Privacy into the Netflix Prize Contenders," *Proceedings of the 15th ACM SIGKDD International Conference Knowledge Discovery and Data Mining*, ACM, 2009, pp. 627–636.
15. C. Dwork et al., "Pan-Private Streaming Algorithms," *ICS*, 2010, pp. 66–80.

Mathias Lecuyer is a PhD candidate at Columbia University. Before joining Columbia's PhD program, he received a BS and an MS from Ecole Polytechnique in France. In his research, Lecuyer builds tools and designs mechanisms that leverage statistics and machine learning to: increase the current web's transparency by revealing how personal data is being used, and enable a more rigorous and selective approach to big data collection, access, and protection, to reap its benefits without imposing undue risks.

Riley Spahn is a PhD candidate in computer science at Columbia University. He is interested in building systems and tools that enable machine learning workloads to be run more securely and privately. Spahn received his MS in computer science from Columbia University and his BS in software engineering from Auburn University. He received the 2015 Google PhD Fellowship in privacy.

Roxana Geambasu is an associate professor of computer science at Columbia University and a member of Columbia's Data Sciences Institute. She joined Columbia in fall 2011 after finishing her PhD at the University of Washington. For her work in cloud and mobile data privacy, she received an Alfred P. Sloan Faculty Fellowship, a Microsoft Research Faculty Fellowship, an NSF CAREER award, a "Brilliant 10" Popular Science nomination, an Early Career Award in Cybersecurity from the University of Washington Center for Academic Excellence, the Honorable Mention for the 2013 inaugural Dennis M. Ritchie Doctoral Dissertation Award, a William Chan Dissertation Award, two best paper awards at top systems conferences, and the first Google PhD Fellowship in cloud computing.

Tzu-Kuo (TK) Huang is a senior engineer at Uber Advanced Technologies Group working on self-driving technology. He has worked on various topics in machine learning, including active learning, dynamic model learning, and ranking. He completed his PhD in machine learning at Carnegie Mellon University, followed by post-doc research at Microsoft Research, New York City.

Siddhartha Sen is a researcher at Microsoft Research in New York City. He creates distributed systems that use novel data structures and algorithms to deliver unprecedented functionality or performance. Recently, he has generalized this approach by using contextual machine learning to optimize various decisions in Microsoft's cloud infrastructure. Sen received his BS degrees in computer science and mathematics and his MEng degree in computer science from MIT. From 2004–2007 he worked as a developer at Microsoft and built a network load balancer for Windows Server. He returned to academia and completed his PhD from Princeton University in 2013. Sen received the first Google Fellowship in Fault-Tolerant Computing in 2009, the best student paper award at PODC 2012, and the best paper award at ASPLOS 2017.