Data Lifecycle Challenges in Production Machine Learning: A Survey *

Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang †, Martin Zinkevich

Google Research KAIST†

{npolyzotis, sudipr, martinz}@google.com swhang@kaist.ac.kr†

ABSTRACT

Machine learning has become an essential tool for gleaning knowledge from data and tackling a diverse set of computationally hard tasks. However, the accuracy of a machine learned model is deeply tied to the data that it is trained on. Designing and building robust processes and tools that make it easier to analyze, validate, and transform data that is fed into large-scale machine learning systems poses data management challenges.

Drawn from our experience in developing data-centric infrastructure for a production machine learning platform at Google, we summarize some of the interesting research challenges that we encountered, and survey some of the relevant literature from the data management and machine learning communities. Specifically, we explore challenges in three main areas of focus – data understanding, data validation and cleaning, and data preparation. In each of these areas, we try to explore how different constraints are imposed on the solutions depending on where in the lifecycle of a model the problems are encountered and *who* encounters them.

1. INTRODUCTION

Machine learning (ML) has become essential in modern computing. More and more organizations are adopting ML to glean knowledge from data and tackle a diverse set of computationally hard tasks, ranging from machine perception and text understanding to health care and genomics. As a striking example, deep learning techniques can be used to detect diabetic eye diseases with an accuracy on-par with ophthalmologists [1].

However, developing reliable, robust, and understandable ML models requires much more than a good training algorithm. Specifically, it is necessary to build the model using high-quality training data. Moreover, this training data needs to be translated into a set of features that can expose the underlying signal to the training algorithm. And finally, the data fed to the model at serving time must be similar in distribution (and in features) to the training data, otherwise the model's accuracy will decrease. Ensuring that each of these steps is done in a consistent manner becomes even more challenging in a setting where new training data arrives continuously and accordingly triggers the training and deployment of updated models.

To further illustrate the previous points, we consider a scenario where a software error in a data source causes a feature in the training data to get pinned to an error value (e.g., -1). Training on such corrupted data will typically lead to reduced model accuracy that may only be noticed after a few days. The predictions obtained using the poor model will persist in the logged serving data (new data on which the model runs on). Typically this logged data is fed back as training data for the next training cycle. This can therefore cause the data error to percolate through the system and taint downstream data, which can make recovery painful. Depending on the impact of the error on the model accuracy, it can at best cause a small reduced model accuracy and at worst cause a hard to recover service outage. This scenario illustrates the importance of catching errors early and reasoning about their propagation within the data flow of an ML pipeline.

In this article, driven from our experience in building data management infrastructure for a large-scale ML platform [11], we identify several core challenges in the management of ML data that are relevant to [11] and other ML platforms [17]. We organize these challenges around the following broad themes: data understanding, data validation and cleaning, and data preparation. We draw connections to existing work in the data management literature and outline open problems that remain to be solved.

*This article extends a tutorial the authors delivered at the SIGMOD conference in 2017.

†Corresponding author, work done at Google and KAIST

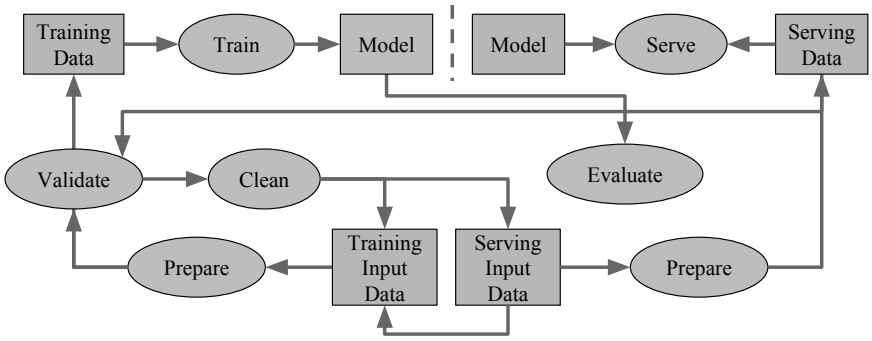

Figure 1: An overview of an end-to-end machine learning pipeline with a data point of view.

The rest of the paper is structured as follows:

- We provide an overview of large-scale ML pipelines through the lens of data management (Section 2).
- We focus on the following data management themes and study existing work: data understanding (Section 3), data validation and cleaning (Section 4), and data preparation (Section 5). At the end of each section, we identify open research challenges.
- We summarize lessons learned through our experience in building data management infrastructure for large-scale ML (Section 6).

The article aims to inform both database researchers and practitioners about the class of problems that exist in the intersection of production ML pipelines and data management, and to motivate further research in this area. We believe that the database community is well positioned to tackle these problems in the context of ML.

2. OVERVIEW OF PRODUCTION ML

The data management community has explored several interesting problems around the optimization of ML pipelines as data flows, and this line of work has resulted in the development of novel system architectures such as Velox [24], Weld [54], and SystemML [15]. In comparison this survey takes a *data centric point of view* and focuses on the challenges that arise in the management of ML data, which are largely separate from the efficiency issues of large-scale data flows.

More recently model understanding has become a critical issue especially when using deep learning or representative learning on semi-structured or unstructured data. The challenges range from understanding the state at different layers of deep architectures, interpreting results [28], to finding minimal architectures without reducing model accuracy. Model understanding deserves a survey on its own, and this survey complements by focusing more on training and serving data.

In this section, we introduce two dimensions that help us characterize data management challenges. The first dimension derives from the different classes of users of an ML pipeline. The second dimension stems from the data's lifecycle through an ML pipeline and the corresponding activities performed by the users. The following subsections discuss these dimensions in more detail.

2.1 Users Interacting with ML Platforms

An often overlooked aspect of large-scale ML at bigger companies is that multiple people play different roles [41] in the development and maintenance of an ML pipeline. The best person for coming up with new features is unlikely to be the best person to develop the ML architecture or to handle emergencies with the production system: moreover, people will approach the pipeline with different priorities. Borrowing from marketing and UX research, we identify three personas [37] representative of groups that would use an ML pipeline based on our experience:

- ML Expert: has a broad knowledge of ML, knows how to create models and how to use statistics, and can advise multiple pipelines.
- Software Engineer: understands the problem domain and has the most engineering expertise for a specific product.
- Site Reliability Engineer: maintains the health of many ML pipelines simultaneously, but cannot afford to know the application details.

As an example of how these personas play different roles, suppose that the pipeline is experiencing new errors due to an out-of-range feature value (e.g., the price of an item is higher than expected). The ML expert could fix the quantization of the price for model training. The software engineer could implement the quantization and run backfilling (explained in Section 2.2). The site reliability engineer, on the other hand, may want to rollback the pipeline to a working state first.

Finally, the three personas play different roles in an ML pipeline's lifecycle. During the experiments phase, the ML expert and software engineer are mostly involved. During the launch, the site reliability engineer becomes involved. The subsequent refinement of the pipeline is done by the software engineer while the maintenance of the overall pipeline is done by the site reliability engineer. A key takeaway is that many people with radically different backgrounds will have to handle a variety of tasks to keep the pipeline running smoothly.

2.2 Data Lifecycle in an ML Pipeline

The data lifecycle of an ML pipeline starts with generating *Training data*. Specifically, we distinguish the raw *input data* that is fetched from a variety of sources including databases, key-value stores, and logs, to name a few. Depending on the type of the problem at hand and available data, this data can be structured, semi-structured, or unstructured, and may correspond to different degrees of curation. In many cases, a few invariants can be asserted over the data [62].

As a running example, suppose a team is developing an app store. The initial raw input data is illustrated on the left side of Figure 2, and the goal is to predict app purchases based on app store user and product features. (Note that the "app store users" are not the same as "users" of the ML pipeline.) While the example is intentionally simplistic, the input data can be large and generated by joining heterogeneous data sources with different data qualities and trust issues.

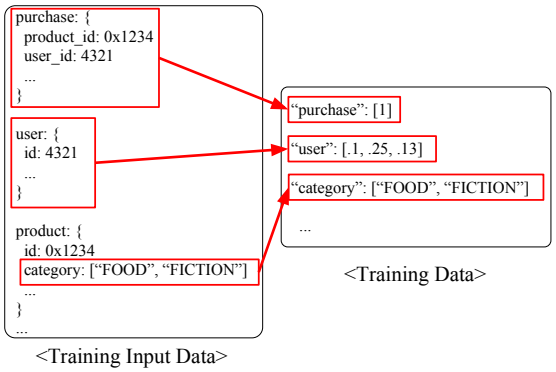

Figure 2: Input to Training Data for the App Store ML pipeline. The same preparation must also be done for serving data as shown in Figure 1.

Prepare. The input data is transformed to the training data through the *Prepare* module. Figure 2 shows how the raw input data is converted to a format of features and values, which can be used for training by the *Train* module. For example, the user information has been mapped to a vector of three values, which is called an embedding. The key questions to ask for preparation are: what features can be generated from data, what are the properties of the feature values, and what are the best practices to transcode values.

Train and Evaluate. Once the training data is ready, it is fed into the *Train* module, which can be frameworks including TensorFlow [5], Keras [3], and Apache MXNet [4]. The trained model can be evaluated by an *Evaluate* module, which checks if the model has an acceptable accuracy, whether the data should be encoded differently, and whether there should be more data or features. Note that an ML pipeline may train an ensemble of models using either the same or different input data where the predictions are intersected or unioned to increase accuracy.

Validate. The *Validate* module is necessary to make sure the training data does not contain errors that may propagate down to the model training. To illustrate why validating data is important, suppose that an engineer is refactoring a backend that generates a feature for the app store ML pipeline, but introduces a bug that results in the generation of wrong values. Note that in this case there are no newly introduced features or data, and the training and serving logic remain the same. Once the erroneous code rolls out to production, it causes the feature to acquire erroneous values. The bad feature values then cause the model training accuracy to decrease and result in significant production issues during model serving where the model is executed for the app store users. Overall, data validation is a key element of production ML infrastructure: by detecting an issue during training time, we can avoid the rollout of a broken model and thus prevent a significant negative impact on the app store users and revenue.

Validating data is a complex problem that needs to be solved for various parts of the pipeline. In the *Prepare* module, the key questions to ask are which data properties affect significantly the model accuracy and whether there are any dependencies to other data and infrastructure. Validation is also required between the *Training Data* and *Serving Data* (depicted as the arrow from *Serving Data* to *Validate* in Figure 1). If there are deviations between the two types of data, then the model trained on the training data will not perform consistently during serving. Hence, the questions to ask are what are possible deviations between the two types of data and when they can be problematic.

Any detected data errors must be forwarded to the user as alerts. An important challenge is to formulate alerts so they are understandable and actionable. As an example, consider the detection of a missing feature Country from the app store user data and the following alternatives to formulate the alert: (a) feature Country is missing, (b) feature Country is missing from 18% of the input examples, or (c) feature Country is missing from the examples that correspond to "gender=female". Clearly, some of these formulations are more actionable and allow the user to understand better both the scope of the error and its potential effect on model accuracy. Another issue is how sensitive the alerts should be. If there are too many false positives, the user may end up ignoring alerts altogether. On the other hand, being too strict on alerts will result in failing to detect critical errors.

Clean. The user may also decide to fix the data based on the alerts, after understanding whether cleaning the data will improve the model, which part of the data is to be fixed, and how should the fix be reflected to all the input data until now (this operation is known as backfilling).

Serve. After a model is trained and deployed, the *Serve* module is responsible for receiving the *serving input data* (in our example, app store user impressions and clicks) and preparing it as *serving data* that can be processed through the model. The serving input data is typically generated a single example at a time, and has stringent latency constraints. The serving input data needs the same preparation as was applied to the raw training time before being sent to the model.

It is often the case that serving data is logged and channelled back as training data for the next training epoch through some bulk data processing stages, thereby completing the data lifecycle.

In summary, there are various data management challenges that arise at different stages of the data lifecycle, which we broadly classify as follows:

- Data Understanding: analyzing and knowing what to expect from the data.
- Data Validation and Cleaning: identifying and fixing any errors in the data.
- Data Preparation: engineering features and gathering examples for training.

3. DATA UNDERSTANDING

The first step of ML is to understand your data. There are largely two parts in an ML pipeline for data understanding. First, sanity checks are important when training a model for the first time. Next, more advanced analysis and diagnosis are needed during the launch and iterate cycles where a model is iteratively improved with new training data.

3.1 Sanity Checks

When the user performs sanity checks, the challenge is to see if the data has the expected "shape" before training the first model. The following are some examples of sanity checks:

- A continuous feature's minimum, maximum, most common values, and histogram are reasonable (e.g., latitude values must be within the range [-90, 90] or $[-\frac{\pi}{2}, \frac{\pi}{2}]$, and not all values are in one bucket).
- The distribution of a categorical value is as expected (e.g., it has the expected domain, and the more common values are what you would expect).
- A feature is present in enough examples (e.g., the country code must be in $\geq$70% of the examples).
- A feature has the right number of values (e.g., there cannot be more than one age of a person).
- Labels from external services may have trust issues and must be verified with known labels.

The key ML challenge here is how to set such expectations of the data. For example, how do we know a distribution is "right"? If we know exactly what we need, then one can use any SQL tool to perform sanity checks. However, the requirements are often unclear because there may be no ownership of the feature. In this case, visualization tools can help us understand the data shape by discovering surprising properties of data and thus develop better sanity checks.

SeeDB [70] recommends data-driven visualizations using deviation-based metrics (e.g., Earth Mover's distance, Euclidean distance, Kullback-Leibler divergence, and Jenson-Shannon distance). The recommendations can provide insights to users on what to expect of the training data and subsequent ones. For example, suppose there are two histograms that show Desktop versus Mobile usage between two groups of people. If the two groups are female versus male, the two histograms may not differ much. However, if the groups are users in emerging versus mature markets, there may be a relatively higher mobile usage in emerging markets. This difference in usage makes the latter histogram more interesting and thus more likely to be recommended. ZenVisage [66] is a follow-up work on interactive visual analytics using the ZQL query language.

Another recent line of work is to control false discovery rates for recommending visualizations. Con-

tinuing our example above, as more visual recommendations are made, there is bound to be more meaningless ones as well. The QUDE system [14, 75] takes a statistical approach and provides an interactive data exploration framework that uses multiple hypothesis testing to control the false positives. Traditional methods for controlling family-wise error rates (e.g., Bonferroni correction) or false-discovery rates (e.g., Benjamini-Hochberg procedure) assume "static" hypotheses and do not work for interactive data exploration. QUDE proposes α-investing to control the marginal false discovery rate, which is the expected value of the false discovery rate. Intuitively, recommending good visualizations will be rewarded with more budget to explore while bad recommendations will result in losing it.

3.2 Analyses for Launch and Iterate

The next part of data understanding is to do more analyses during the launch and iterate cycles.

3.2.1 Feature-based Analysis

There are major ML challenges that involve feature analysis. One is analyzing features in conjunction with a trained model where the goal is to find interesting training data slices (based on features) that lead to high/low model accuracy. For example, an app recommendation model may perform poorly for people in certain countries. Another challenge is detecting training-serving skew, which was briefly mentioned in Section 1. For instance, if the model was trained on data that had an even gender ratio, but the actual serving logs have a completely different ratio for people in the age range [20, 40], then the model may be biased due to the skew. As another example, there may be unseen features that appear on serving data, but not in training data. Skew can be fixed by debugging data generation, which is usually the culprit, or possibly making the model training more robust to skew.

Data cube analysis can be applied to analyze slices of data, which are defined with features or feature crosses. For example, MLCube [39] is a tool for visually exploring ML results that enable users to define slices using feature conditions to compute aggregate statistics and evaluation metrics over the slices. The tool can be used to help understand and debug a model or compare two models. Another interesting work is prediction cubes [21], which summarize models trained on individual cubes.

While such manual exploration is useful, an interesting research question is how to automatically prioritize user attention and identify what are the "important" slices. While we are not aware of any recent data cube research that directly addresses this problem, intelligent roll-ups in multi-dimensional OLAP data [60] are relevant and have been proposed to automatically generalize from a specific problem case in detailed data and return the broadest context in which the problem occurs. Similarly, smart drill-downs [38] discover and summarize interesting slices of the entire data. The roll-ups or drill-downs can be used to find problematic slices in training data that positively (or negatively) affect model metrics (e.g., log loss, AUC, and calibration).

3.2.2 Data Lifecycle Analysis

Another important analysis is to track the lifecycle of data. A common analysis is to identify dependencies of features. For example, a label feature must not "leak" into any other feature where some of its information is duplicated or encoded in the other feature, and the model trained on that information makes unrealistically-accurate predictions, but generalizes poorly. Another useful analysis is to identify sources of data errors. For example, a subset of the training data may have been dropped because a data source was unavailable. The tools to address these analyses largely fall into two categories: coarse-grained and fine-grained tracking.

The advantage of coarse-grained tracking is that it is general and not tied to a particular system. Goods [35] gathers metadata from tens of billions of datasets (including provenance) within Google and implements services on top of this metadata. A key design choice is to gather this data in a post-hoc fashion where dataset owners do not have to do any registration, and the metadata is crawled afterwards in a non-intrusive manner. As a result, while Goods can track which dataset was generated from which process, it cannot extend the tracking to individual features.

Fine-grained tracking, on the other hand, can analyze individual features, but tends to be tightly coupled with the underlying system. ProvDB [49] provides a unified provenance and metadata management system to support lifecycles of complex collaborative data science workflows. The metadata consists of artifacts, which include version lineages of data, scripts, results, data provenance among artifacts, and workflow metadata on derivations and dependencies among artifact snapshots. Other relevant systems include ModelDB [69], ModelHub [50], and Amazon's ML experiments system [61], which provide lifecycle management for various models, and Ground [36], which has a goal similar to that of ProvDB, but with a simple, flexible metamodel that is model agnostic.

3.3 Open Challenges

There are open questions for ML analysis that are not covered by the previous techniques. Recently, determining if a trained model is "fair" [59] has become a critical issue. For example, we would like to know if a model is prejudiced against certain classes of data. Since a model is only as good as its training data, we need to understand if the data reflects reality. Identifying new kinds of "spam" [34] is an open challenge as well. For example, are users abusing the system in an adversarial way? Here, we need to apply adversarial testing on the training data. While using general SQL-based systems [48, 6] is an "escape hatch" for analysis, we may need more specialized tools to address the above issues.

4. DATA VALIDATION AND CLEANING

Since ML largely depends on its data, it requires data validation to perform well. Models cannot answer questions they are not asked. For example, suppose a model uses the feature Country and understands when its value is "US". However, if in the next batch of training data the value becomes "us", then without any validation or preprocessing, the model will simply think that there is a new country. As another example, a feature may suddenly change its unit (e.g., age changes from days to hours) or even disappear. Unfortunately, model training is resilient to such errors, and instead of crashing, the model accuracy may simply decrease. Hence, data must be validated early on to avoid errors from propagating to model training.

How do we deal with these problems? If a feature value is not consistent, we could insert automatic corrections (e.g., capitalize all countries). If a feature appears for the first time, we can create a new field. If a feature disappears, we can find where it disappeared using provenance or root cause analysis [53, 72, 9]. While some fixes can be done automatically, in many cases, we need to notify users to solve the problems by providing "playbooks," which are manuals that contain concrete actions to take for addressing each alert. Although many data cleaning [68, 40, 45] techniques are relevant, in production ML it is equally important to design actionable alerts with humans in mind.

4.1 Alert Tradeoffs

When alerting users to validate and fix data errors, there is a tradeoff to make between recall and precision. It is not rare for some ML data issue to cause a minor emergency for a particular product. A common response is to overcompensate, by setting alerts for every conceivable issue with the data. Then, these alerts fire every day, annoying users and making them insensitive to real issues. As a result, all the alerts are ignored, and the vicious cycle starts again. Hence, it is important to balance recall (i.e., what fraction of problems we catch) and precision (what fraction of alerts lead to good catches).

An alert is considered a good catch if it is actionable and eventually leads to a fix. For example, alerting that a feature is missing is clearly actionable. However, alerting that there was a distributional shift in a feature's value may be less actionable. Imagine an ML expert saying that the age should have a Kolmogorov distance of less than 0.1 from the previous day and then leaves and works on a different system. Later on, another engineer may be alerted that the age has a Kolmogorov distance of 0.11. While the alert may indicate a real problem, the engineer may not know how to resolve it. Hence, the question is not whether something is wrong if an alert fires, but whether it gets fixed.

In some cases, a data fix may encompass fixing a constraint. For example, suppose a Country feature is known to contain four values, but from some point a new value "SS" is introduced in the training data. While this value may indeed be incorrect, "SS" could also be a valid country (say South Sudan), which means the constraint should now include five countries. Existing work [22, 33] provides opportunities to fix both data and constraints.

If there are many alerts, finding alerts that are related and combining them becomes useful. In the literature, cost-based models [16, 42] and conflict hypergraphs [23] are proposed.

Not all anomalies are equally important, and the ones that result in worse model accuracy in production must be alerted first. In large-scale ML, features with different qualities co-exist. Often, an "alpha" or completely new and untested features will co-exist with more established production features that the system relies on. An untested feature is evaluated in an experimental model, which may become the next production model. At some point, a feature may also be deprecated. Hence, fixing features that are used in production is more useful than fixing features for experimental models. An interesting open question is whether the improvements of correcting a feature can be estimated without having to make the correction itself.

4.2 Alert Categories

General alerts are hard to design and depend on the training data. For example, predicting car accidents, house prices, social connections, or clicks on a web page will have very different data, and it

is hard to predict the expected data shape for all these applications. To handle a variety of domains with minimal effort, there needs to be some commonalities among them.

One common setting is where data arrives continuously, say in web applications. As new data arrives, old data is thrown away, and newer data is given more priority. The data validation can be done by comparing data with its previous versions, and alerts can be raised based on accumulated evidence to date [71]. In case there is a concept drift, the expected data shape can be updated based on the user's judgement.

In this setting, basic alerts are motivated by engineering problems. For example, missing fields can be detected by checking if a field that was present is now absent. RPC timeouts can be detected by checking if the most common value is not more common than before. Format changes can be detected by checking if the domain of values has increased.

It is also worth mentioning alerts that are based on statistics including homogeneity tests, analysis of variance (e.g., ANOVA [30, 31]), time series analysis, and change detection. For example the chi-squared test can be used to check homogeneity [55] by rejecting the null hypothesis for the distributions being the same. One problem with a chi-squared test is that statistically significant changes may be common on all fields if the data is large enough. Other metrics that can be used include the L_1 and L_∞ metrics or Earth Mover's distance [32, 70]. Some metrics including the number of examples, the number of positive labels, or the total number of clicks may fluctuate, but are nonetheless very important. Time series analysis [10, 27, 18] can help track these statistics.

4.3 Open Challenges

Selecting alerts that lead to the most impact in production is an open question. Ideally, we can perform impact analysis that will estimate how the system would improve if an error were fixed. Automatically generating fixes and playbooks is also an open-ended challenge. There is an interesting connection between the notion of alerts and fixes to existing work on automatic database repairs. Similarly, the notion of minimizing the number of alerts to users is analogous to active learning, which tries to minimize the number of labelings.

5. DATA PREPARATION

One of the largely "black art" aspects of ML is data preparation. Similar to the other data management challenges, specific facets of the data preparation problem arise in different forms at different points in the ML lifecycle. For instance, during the initial development of a model, data preparation boils down to engineering a set of features that are most predictive of the task label. Once the models are more mature, the focus may shift to resource optimization and latency reduction by selecting a subset of all the available features while still retaining the same accuracy. This is typically referred to as the *feature selection* problem. Finally, often the original data that was available for the task may simply be incomplete or partially complete. A third aspect of data preparation is enriching the training data by importing information from other data sources.

5.1 Feature Engineering and Selection

Feature engineering is a well-studied problem [7, 44, 43, 74, 58, 46, 8, 67] with a suite of techniques that are largely designed based on experience of ML experts. Consider the following specific ML task. Starting with the census data on housing, we would like to predict the median housing prices at the granularity of city blocks. For this task, reasonable features include location of blocks (possibly specified using latitude and longitude coordinates), number of households per block, crime rate, and so on. While some of these features may directly be available in the census data, others may require some queries over the data to extract. Furthermore, the *goodness* of a feature is typically based on the predictive power of the feature. While the predictive power is hard to estimate upfront, a good proxy is to understand the correlation of the feature with the label. Analyses techniques discussed in Section 3 that easily present these correlations to experts can be invaluable here.

Even once the raw features are designed, often a suite of transformations are applied before they are fed into the ML pipeline. Some of the typical transformations applied include normalization, bucketization, winsorizing, one-hot encoding, feature crosses, and using a pre-trained model or embeddings (mappings from values such as words to real numbers) to extract features [51]. For example, the crime rate of a city block could be transformed into a one-hot encoding using three categories: low, medium, and high. The exact feature transform to perform depends on both the data as well as the ML training algorithm. Some algorithms that natively perform transformation include Lasso (regularize unimportant features to zero) and on-the-fly scaling and shifting (avoid turning sparse data into dense data).

An interesting research direction is to learn feature engineering itself. Feeding training data directly to a deep neural network and letting it figure out the features is referred to as "representation learning" in the ML community. Some promising techniques include autoencoders and restricted Boltzmann Machines [12]. However, learning both the representations and the objective may require significant resources and data, so manual feature engineering is still used in most cases. A middle ground between completely automated feature engineering and manual feature engineering is data-driven feature engineering where based on certain data characteristics we can automatically infer the best set of transformations to apply. While this is relatively simple to do in many cases, determining the best embedding that should be used on a feature by searching over an available set of pre-trained embeddings is still an open research problem.

As models become more mature, developers often experiment with addition and removal of new features. This is one instance of a problem that highlights the different views that users with different roles have for the same problem. For example, an ML expert could be choosing features that improve the model accuracy the most. However, the software engineer may have to also worry about how to actually add the feature into the existing pipeline. The check list includes making sure the feature is available at serving time, whether one is allowed to even use the feature, and what the return of investment is for the feature. The site reliability engineer may have to worry about introducing new dependencies and making sure the pipeline is robust. Another concern is whether the feature will affect the model size and prediction latency.

5.2 Data Enrichment

Data enrichment [29, 56, 64] refers to the augmentation of the training and serving data with information from *external data sources* in order to improve the accuracy of the generated model. A common form of enrichment is to join in a new data source in order to augment the existing features with new signals. Another form is using the same signals with different transformations, e.g., using a new embedding for text data.

A first step for data enrichment is cataloging and contextualizing all the available data. Different systems have been designed that solve this problem within enterprises as well as over the web. For example, systems including Goods [35], Ground [36], and Datahub [13] can be used to explore datasets siloed within product areas of organizations. On the web, tools including Webtables [19], Kaggle [2], and Data Civilizer [20] can be used to search scientific datasets published independently by organizations.

The next step is to extract knowledge and acquire labels, which can be challenging when labeling is expensive and/or heterogeneous sources can have different label qualities and labeling costs. Crowdsourcing frameworks like Alfred [25] help moderately skilled crowd workers in extracting knowledge from a corpus of unstructured documents. DeepDive [73] uses incomplete knowledge bases and rules for distant supervision to minimize expensive human annotations. In active learning [26, 63, 52], the learning procedure decides how best to enrich the data iteratively. Transfer learning is a way to leverage previously acquired knowledge from one domain to improve the model accuracy of a different domain. Weak supervision is used in Snorkel [57, 56] where hand labeling is avoided altogether, and workers can programmatically generate lower-quality labels, which are then denoised with generative models. Finally, label hierarchies can be used to predict unseen labels.

While data enrichment often leads to model accuracy improvements, this is not always the case. Sheng et al. [65] shows how improving the quality of the already available labels can better improve model accuracy than collecting more examples. Hence, there may be a tradeoff between obtaining more data and improving its data. Similary, a recent paper [47] has looked into understanding the impact of adding new features by joining with other data sources for a specific class of algorithms. It would be interesting to consider extensions to other cases (e.g., training with black-box learning algorithms that are hard to approximate, or using different transformations on existing signals).

5.3 Open Challenges

Given input features and an ML training algorithm, automatically generating feature transforms that result in the highest model accuracy is an open question. From our experience, this step is a pain point for users who do not necessarily understand the nuances of transforms.

6. LESSONS LEARNED

In building a production ML platform for Google [11], we encountered a host of the challenges that we have presented in this survey. We summarize some of the over-arching lessons that we learnt on the way.

- *Interesting data management challenges beyond optimizing data flow.* The data management community has focused more on optimizing data flow

for large scale data processing (specifically for ML). However, as we discussed in this article, there are data management problems beyond optimizing data flow. Complementing the traditional data flow point of view, we have provided a data point of view for ML pipelines and identified challenges in understanding, validating, and preparing data. As shown in the previous sections, many prior techniques from the data management literature are relevant to building robust large-scale ML systems. The data management and ML communities have a lot to learn from each other through closer collaboration.

- *Make realistic assumptions when developing solutions.* In developing research solutions, we must be careful about the assumptions that we make. For instance, it is unreasonable to assume that data lives in a single source (e.g., a DBMS). Instead, most enterprise data often resides in multiple storage systems (e.g., Spanner, BigTable, Dremel, and CNS, to name a few) that have different characteristics. Typically, there needs to be an ingestion step that converts this data to become compatible with the trainer. Similarly, it is important to stay abreast of the state-of-art developments in the ML community and ensure that the data management solutions complete them.
- *Be aware of the diverse needs of different users.* Many of the key design decisions in our infrastructure were based on diverse needs of different set of users that interact with such a system. In addition to the personas covered in this paper, ML systems may have a wide spectrum of end users as well, starting from ML novices who have yet to train their first models up to experts with extensive modeling experience. Building a large-scale ML system must be flexible enough to accommodate all these users as much as possible.
- *Ensure that your solution integrates smoothly into the development workflow.* The launch and iterate cycle time for ML pipelines is small, and users will not use tools unless they are necessary for their critical development workflows. To ensure the adoption of data management tools, it is thus critical to integrate them into workflows smoothly and make the benefits of using them obvious.

7. CONCLUSION

Data management in large-scale ML systems will only get more important as the amount of data continues to increase rapidly. In this survey, we have described large-scale ML pipelines in a data point of view. We then focused on three data management challenges – understanding, validation and cleaning, and preparation – and surveyed relevant techniques from the data management literature. Finally, we summarized lessons learned from building data management infrastructure for a large-scale ML platform. We believe data management research in ML systems has plenty of open challenges that can be solved with close collaboration between the data management and ML communities.

8. REFERENCES

[1] Deep learning for detection of diabetic eye disease. https://research.googleblog.com/2016/11/deep-learning-for-detection-of-diabetic.html.

[2] Kaggle. https://www.kaggle.com/.

[3] Keras. https://keras.io/.

[4] Mxnet. https://mxnet.incubator.apache.org/.

[5] Tensorflow. https://www.tensorflow.org/.

[6] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Eurosys*, pages 29–42, 2013.

[7] M. R. Anderson, D. Antenucci, V. Bittorf, M. Burgess, M. J. Cafarella, A. Kumar, F. Niu, Y. Park, C. Ré, and C. Zhang. Brainwash: A data system for feature engineering. In *CIDR*, 2013.

[8] M. R. Anderson and M. J. Cafarella. Input selection for fast feature engineering. In *ICDE*, pages 577–588, 2016.

[9] P. Bailis, E. Gan, S. Madden, D. Narayanan, K. Rong, and S. Suri. Macrobase: Prioritizing attention in fast data. In *SIGMOD*, pages 541–556, 2017.

[10] M. Basseville and I. V. Nikiforov. *Detection of Abrupt Changes: Theory and Application.* Prentice-Hall, Inc., 1993.

[11] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, C. Y. Koo, L. Lew, C. Mewald, A. N. Modi, N. Polyzotis, S. Ramesh, S. Roy, S. E. Whang, M. Wicke, J. Wilkiewicz, X. Zhang, and M. Zinkevich. Tfx: A tensorflow-based production-scale machine learning platform. In *SIGKDD*, pages 1387–1395, 2017.

[12] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *TPAMI*, 35(8):1798–1828, 2013.

[13] A. P. Bhardwaj, S. Bhattacherjee, A. Chavan, A. Deshpande, A. J. Elmore, S. Madden, and A. G. Parameswaran. Datahub: Collaborative

data science & dataset version management at scale. *CoRR*, abs/1409.0798, 2014.
[14] C. Binnig, L. D. Stefani, T. Kraska, E. Upfal, E. Zgraggen, and Z. Zhao. Toward sustainable insights, or why polygamy is bad for you. In *CIDR*, 2017.
[15] M. Boehm, M. W. Dusenberry, D. Eriksson, A. V. Evfimievski, F. M. Manshadi, N. Pansare, B. Reinwald, F. R. Reiss, P. Sen, A. C. Surve, and S. Tatikonda. Systemml: Declarative machine learning on spark. *PVLDB*, 9(13):1425–1436, 2016.
[16] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, pages 143–154, 2005.
[17] J.-H. Böse, V. Flunkert, J. Gasthaus, T. Januschowski, D. Lange, D. Salinas, S. Schelter, M. Seeger, and Y. Wang. Probabilistic demand forecasting at scale. *PVLDB*, 10(12):1694–1705, 2017.
[18] K. H. Brodersen, F. Gallusser, J. Koehler, N. Remy, and S. L. Scott. Inferring causal impact using bayesian structural time-series models. *Annals of Applied Statistics*, 9:247–274, 2015.
[19] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: Exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.
[20] R. Castro Fernandez, D. Deng, E. Mansour, A. A. Qahtan, W. Tao, Z. Abedjan, A. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. A demo of the data civilizer system. In *SIGMOD*, pages 1639–1642, 2017.
[21] B.-C. Chen, L. Chen, Y. Lin, and R. Ramakrishnan. Prediction cubes. In *PVLDB*, pages 982–993, 2005.
[22] F. Chiang and R. J. Miller. A unified model for data and constraint repair. In *ICDE*, pages 446–457, 2011.
[23] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469, 2013.
[24] D. Crankshaw, P. Bailis, J. E. Gonzalez, H. Li, Z. Zhang, M. J. Franklin, A. Ghodsi, and M. I. Jordan. The missing piece in complex analytics: Low latency, scalable model management and serving with velox. In *CIDR*, 2015.
[25] V. Crescenzi, P. Merialdo, and D. Qiu. Crowdsourcing large scale wrapper inference. 33:1–28, 2014.
[26] S. Dasgupta and J. Langford. Tutorial summary: Active learning. In *ICML*, page 18, 2009.
[27] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: Experimental comparison of representations and distance measures. *PVLDB*, 1(2):1542–1552, 2008.
[28] F. Doshi-Velez and B. Kim. A roadmap for a rigorous science of interpretability. *CoRR*, abs/1702.08608, 2017.
[29] R. C. Fernandez, Z. Abedjan, S. Madden, and M. Stonebraker. Towards large-scale data discovery: Position paper. In *ExploreDB*, pages 3–5, 2016.
[30] R. A. Fisher. On the probable error of a coefficient of correlation deduced from a small sample. *Metron*, 1:3–32, 1921.
[31] R. A. Fisher. *Statistical Methods for Research Workers*, pages 66–70. Springer New York, 1992.
[32] A. L. Gibbs and F. E. Su. On choosing and bounding probability metrics. *International Statistical Review*, 70(3):419–435, 2002.
[33] L. Golab, I. F. Ilyas, G. Beskales, and A. Galiullin. On the relative trust between inconsistent data and inaccurate constraints. In *ICDE*, pages 541–552, 2013.
[34] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2014.
[35] A. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. Goods: Organizing google's datasets. In *SIGMOD*, pages 795–806, 2016.
[36] J. M. Hellerstein, V. Sreekanti, J. E. Gonzales, Sudhansku, Arora, A. Bhattacharyya, S. Das, A. Dey, M. Donsky, G. Fierro, S. Nag, K. Ramachandran, C. She, E. Sun, C. Steinbach, and V. Subramanian. Establishing common ground with data context. In *CIDR*, 2017.
[37] A. Jenkinson. Beyond segmentation. *Journal of Targeting, Measurement and Analysis for Marketing*, (1):60–72, 1994.
[38] M. Joglekar, H. Garcia-Molina, and A. G. Parameswaran. Interactive data exploration with smart drill-down. In *ICDE*, pages 906–917, 2016.
[39] M. Kahng, D. Fang, and D. H. P. Chau. Visual exploration of machine learning results using data cube analysis. In *HILDA*, pages 1:1–1:6, 2016.
[40] Z. Khayyat, I. F. Ilyas, A. Jindal, S. Madden,

M. Ouzzani, P. Papotti, J.-A. Quiané-Ruiz, N. Tang, and S. Yin. Bigdansing: A system for big data cleansing. In *SIGMOD*, pages 1215–1230, 2015.

[41] M. Kim, T. Zimmermann, R. DeLine, and A. Begel. Data scientists in software teams: State of the art and challenges. *TSE*, PP(99):1–1, 2017.

[42] S. Kolahi and L. V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, pages 53–62, 2009.

[43] P. Konda, A. Kumar, C. Ré, and V. Sashikanth. Feature selection in enterprise analytics: A demonstration using an r-based data analytics system. *PVLDB*, 6(12):1306–1309, 2013.

[44] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. Mlbase: A distributed machine-learning system. In *CIDR*, 2013.

[45] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *PVLDB*, 9(12):948–959, 2016.

[46] A. Kumar, R. McCann, J. Naughton, and J. M. Patel. Model selection management systems: The next frontier of advanced analytics. *SIGMOD Rec.*, 44(4):17–22, 2016.

[47] A. Kumar, J. F. Naughton, J. M. Patel, and X. Zhu. To join or not to join?: Thinking twice about joins before feature selection. In *SIGMOD*, pages 19–34, 2016.

[48] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: Interactive analysis of web-scale datasets. *PVLDB*, 3(1-2):330–339, 2010.

[49] H. Miao, A. Chavan, and A. Deshpande. Provdb: A system for lifecycle management of collaborative analysis workflows. *CoRR*, abs/1610.04963, 2016.

[50] H. Miao, A. Li, L. S. Davis, and A. Deshpande. Towards unified data and lifecycle management for deep learning. In *ICDE*, pages 571–582, 2017.

[51] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[52] F. Olsson. A literature survey of active machine learning in the context of natural language processing. volume T2009 of *SICS Technical Report*. Swedish Institute of Computer Science, 2009.

[53] C. Olston and B. Reed. Inspector gadget: A framework for custom monitoring and debugging of distributed dataflows. In *SIGMOD*, pages 1221–1224, 2011.

[54] S. Palkar, J. J. Thomas, A. Shanbhag, M. Schwarzkopt, S. P. Amarasinghe, and M. Zaharia. A common runtime for high performance data analysis. In *CIDR*, 2017.

[55] K. Pearson. *On the Criterion that a Given System of Deviations from the Probable in the Case of a Correlated System of Variables is Such that it Can be Reasonably Supposed to have Arisen from Random Sampling*, pages 11–28. Springer New York, 1992.

[56] A. Ratner, S. H. Bach, H. R. Ehrenberg, J. A. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *PVLDB*, 11(3):269–282, 2017.

[57] A. J. Ratner, C. D. Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. In *NIPS*, pages 3567–3575, 2016.

[58] C. Ré, A. A. Sadeghian, Z. Shan, J. Shin, F. Wang, S. Wu, and C. Zhang. Feature engineering for knowledge base construction. *IEEE Data Eng. Bull.*, 37(3):26–40, 2014.

[59] A. Romei and S. Ruggieri. A multidisciplinary survey on discrimination analysis. *Knowledge Eng. Review*, 29(5):582–638, 2014.

[60] G. Sathe and S. Sarawagi. Intelligent rollups in multidimensional olap data. In *VLDB*, pages 531–540, 2001.

[61] S. Schelter, J.-H. Boese, J. Kirschnick, T. Klein, and S. Seufert. Automatically tracking metadata and provenance of machine learning experiments. In *Workshop on ML Systems at NIPS 2017*, 2017.

[62] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden technical debt in machine learning systems. In *NIPS*, pages 2503–2511, 2015.

[63] B. Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool, 2012.

[64] V. Shah, A. Kumar, and X. Zhu. Are key-foreign key joins safe to avoid when learning high-capacity classifiers? *PVLDB*, 11(3):366–379, 2017.

[65] V. S. Sheng, F. Provost, and P. G. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *SIGKDD*, pages 614–622, 2008.

[66] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. Parameswaran. Effortless data

exploration with zenvisage: An expressive and interactive visual analytics system. *PVLDB*, 10(4):457–468, 2016.

[67] E. R. Sparks, S. Venkataraman, T. Kaftan, M. J. Franklin, and B. Recht. Keystoneml: Optimizing pipelines for large-scale advanced analytics. In *ICDE*, pages 535–546, 2017.

[68] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu. Data curation at scale: The data tamer system. In *CIDR*, 2013.

[69] M. Vartak. MODELDB: A system for machine learning model management. In *CIDR*, 2017.

[70] M. Vartak, S. Rahman, S. Madden, A. G. Parameswaran, and N. Polyzotis. SEEDB: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13):2182–2193, 2015.

[71] M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller. Continuous data cleaning. In *ICDE*, pages 244–255, 2014.

[72] X. Wang, X. L. Dong, and A. Meliou. Data x-ray: A diagnostic tool for data errors. In *SIGMOD*, pages 1231–1245, 2015.

[73] C. Zhang. *DeepDive: A Data Management System for Automatic Knowledge Base Construction*. PhD thesis, 2015.

[74] C. Zhang, A. Kumar, and C. Ré. Materialization optimizations for feature selection workloads. *ACM TODS*, 41(1):2:1–2:32, 2016.

[75] Z. Zhao, L. De Stefani, E. Zgraggen, C. Binnig, E. Upfal, and T. Kraska. Controlling false discoveries during interactive data exploration. In *SIGMOD*, pages 527–540, 2017.