Towards A Unified Graph Model for Supporting Data Management and Usable Machine Learning

Guoliang Li*, Meihui Zhang+, Gang Chen†, Beng Chin Ooi‡
*Department of Computer Science, Tsinghua University, Beijing, China
+Department of Computer Science, Beijing Institute of Technology, Beijing, China
†Department of Computer Science, Zhejiang University, Hangzhou, China
‡School of Computing, National University of Singapore, Singapore

Abstract

Data management and machine learning are two important tasks in data science. However, they have been independently studied so far. We argue that they should be complementary to each other. On the one hand, machine learning requires data management techniques to extract, integrate, clean the data, to support scalable and usable machine learning, making it user-friendly and easily deployable. On the other hand, data management relies on machine learning techniques to curate data and improve its quality. This requires database systems to treat machine learning algorithms as their basic operators, or at the very least, optimizable stored procedures. It poses new challenges as machine learning tasks tend be iterative and recursive in nature, and some models have to be tweaked and retrained. This calls for a reexamination of database design to make it machine learning friendly.

In this position paper, we present a preliminary design of a graph model for supporting both data management and usable machine learning. To make machine learning usable, we provide a declarative query language, that extends SQL to support data management and machine learning operators, and provide visualization tools. To optimize data management procedures, we devise graph optimization techniques to support a finer-grained optimization than traditional tree-based optimization model. We also present a workflow to support machine learning (ML) as a service to facilitate model reuse and implementation, making it more usable and discuss emerging research challenges in unifying data management and machine learning.

1 Introduction

A data science workflow includes data extraction, data integration, data analysis, machine learning and interpretation. For example, consider healthcare analytics. Heterogeneity, timeliness, complexity, noise and incompleteness with big data impede the progress of creating value from electronic healthcare data [10]. We need to extract high quality data from multiple sources, clean the data to remove the inconsistency and integrate the heterogeneous data. Next we can use data analytics techniques to discover useful information and analyze the data to find interesting results (e.g., cohort analysis [7]). We also need to utilize machine learning techniques to

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

learn important features and make decision (e.g., Readmission risk [25]). Finally, we need to present the results in the medical context to enable users to better appreciate the findings.

The data science workflow (as shown in Figure 1) contains both data management components (data extraction, data integration, data analysis) and machine learning component. However, data management and machine learning have been independently studied, be it in industry or academia, even though they are in fact complementary to each other. First, machine learning requires high quality data to guarantee learning results. Thus it needs data management techniques to extract, integrate and clean the data. Second, machine learning relies on data management techniques to enable scalable machine learning. It is widely recognized that machine learning is hard for users without machine learning background to use while database has been featured prominently and used as the underlying system in many applications. Thus machine learning should borrow ideas from the database to make machine-learning algorithms usable and easily deployable. Third, many data management components rely on machine learning techniques to improve the quality of data extraction and integration. For example, we can use machine learning to learn the features that play important roles in data integration. This calls for a new framework to support data management and machine learning simultaneously, and their interaction and dependency in big data analytics.

To address these problems, we propose a unified graph model for modeling both data management and machine learning. We use a graph to model relational data, where nodes are database tuples and edges are foreign key relationships between tuples. A graph can be used to model the unstructured data, semi-structured data and structured data together by linking the tuples in relational database, documents or objects in unstructured data and semi-structured data. More importantly, most machine learning algorithms work on graph. We can therefore use a unified graph model to support both data management and machine learning. Hence, we propose a graph-based framework that integrates data management and machine learning together, which (1) supports data extraction, data integration, data analysis, SQL queries, and machine learning simultaneously and (2) provide machine learning (ML) as a service such that users without machine learning background can easily use machine learning algorithms.

In summary, we cover the following in this position paper.

(1) We sketch a unified graph model to model unstructured data, semi-structured data, and structured data and can support both data management and machine learning.

(2) We propose a user-friendly interface for the users to use data management and machine learning. We provide a declarative query language that extends SQL to support data management and machine learning operators. We also provide visualization tools to assist users in using the machine learning algorithms.

(3) We devise graph optimization techniques to improve the performance, which provide a finer-grained optimization than traditional tree-based optimization model.

(4) We discuss design principles and the important components in providing machine learning as a service workflow.

(5) We discuss research challenges in unifying data management and machine learning.

2 A Unified Graph Model

Graph Model. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ includes a vertex set $\mathcal{V}$ and an edge set $\mathcal{E}$, where a vertex can be a database tuple, a document, or a user, and an edge is a relationship between two vertices, e.g., foreign key between tuples for structured data, hyperlink between documents for unstructured data, or friendship among users in semi-structured data (e.g., social networks).

Graph Model for Relational Data. Given a database $\mathcal{D}$ with multiple relation tables $\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_n$, we can model the relational data as a graph, where each tuple is a vertex and there is an edge between two vertexes if their corresponding tuples have foreign key relationships [12, 14]. Given a SQL query q, we find compact subtrees corresponding to the query from the graph as answers [14] and the details will be discussed in Section 4.

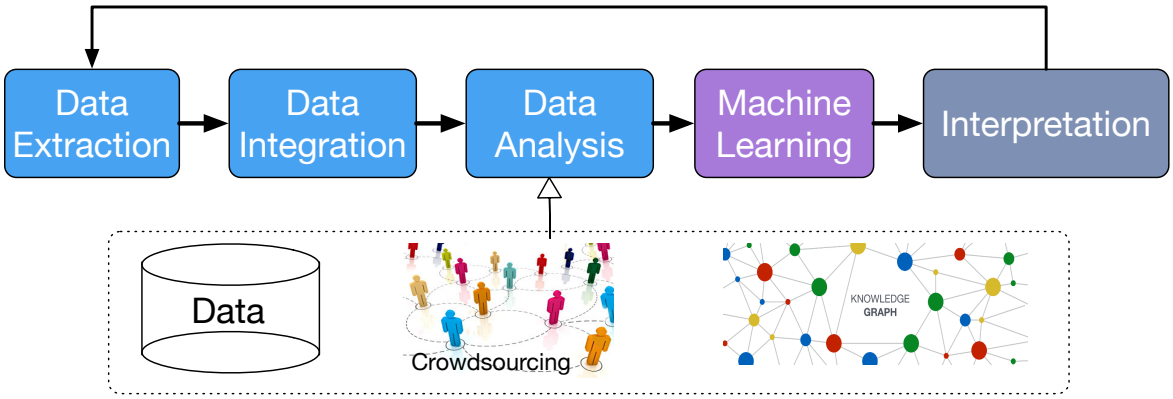

Figure 1: Workflow of Data Science.

Graph Model for Heterogeneous Data. Given heterogeneous data consisting of structured data, semi-structured data and unstructured data, we can also use a unified graph model to represent them. For unstructured data, e.g., html documents, the vertexes are documents and the edges are hyperlinks between documents. For semi-structured data, e.g., Facebook, the vertexes are users and the edges are friendships between users. We can also link the heterogeneous data together using the data extraction and integration techniques [4, 14].

Data Extraction. Data extraction aims to extract entities from the data, including documents, lists, tables, etc. There are some variants of entity extraction. Firstly, given a dictionary of entities and a document, it aims to extract entities from the document that exactly or approximately match the predefined entities in the dictionary [13]. Secondly, given a document and entities in knowledge bases, it aims to link the entities in knowledge base to the entities in the document to address the entity ambiguity problem [19]. Thirdly, it utilizes rules to identify the entities, e.g., a person born in a city, a person with PhD from a university [20]. We can use data extraction techniques to extract entities, based on which we can integrate the heterogeneous data.

Data Integration. We can also link the structured data and unstructured data together and use a unified graph model to represent the data. There are several ways to link the data. First, we can use a similarity-based method [8]. If two objects from different sources have high similarity, we can link them together. Second, we can use crowd-based method to link the data [3]. Given two data from different sources, we ask the crowd to label whether they can be linked. Since crowdsourcing is not free, it is expensive to ask the crowd to check every pair of data and we can use some reasoning techniques to reduce the crowd cost, e.g., transitivity. Third, we can use a knowledge-based method to link the data. We map the data from different sources to entities in knowledge bases and the data mapped to the same entity can be linked [11].

Graph Model for Machine Learning. Most machine learning algorithms adopt a graph model, e.g., PageRank, probabilistic graphical model, neural network, etc.

To summarize, we can utilize a unified graph to model heterogeneous data which can support both data management (including SQL queries, data extraction, data integration) and machine learning (any graph based learning algorithms).

3 A Versatile Graph Framework

We propose a versatile graph framework to support both data management and machine learning simultaneously with a user-friendly interface.

Query Interace. A natural challenge is to utilize the same query interface to support both data management and

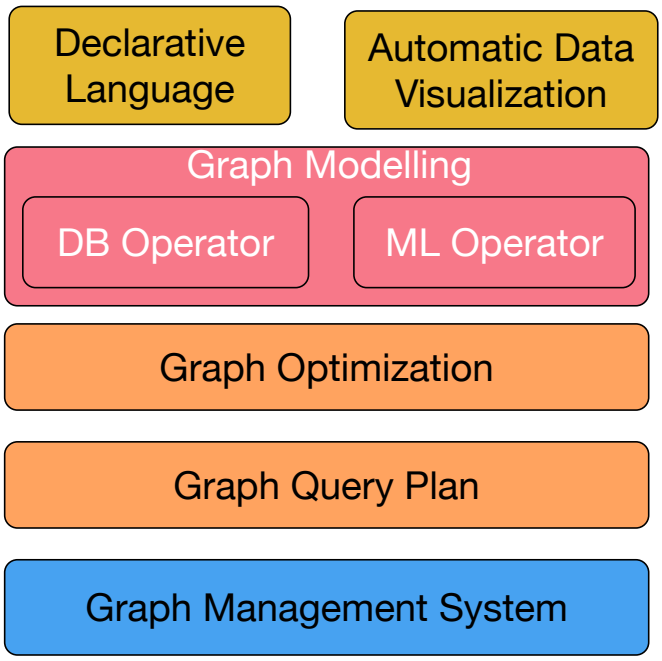

Figure 2: A Unified Graph Model and System.

machine learning. We aim to support two types of query interfaces: declarative query language and visualization tools. The declarative query language extends traditional SQL to support data analysis and machine learning. For example, we can use UDF to support machine learning. We can also add native keywords into declarative query language to enable in-core optimization. Thus users only need to specify what they want to obtain but do not need to know how to get the result. Our framework supports both of the two cases. To support the second case, we need to abstract the machine learning operators, e.g., regression, clustering, classification, correlation. Then our framework supports all the database operators, e.g., selection, join, group, sort, top-k, and machine learning operators, e.g., clustering, classification, correlation, regression, etc. The visualization tool can help users to better understand the data and can also use visualization charts as examples (e.g., line charts, bar charts, pie charts) to provide users with instant feedback, and the framework can automatically compute similar trends or results. More importantly, users can utilize visualization tools, e.g., adding or deleting an attribute like tableau, to visualize the data. Our framework can also automatically analyze the data and suggest the visualization charts. Since the data and features are very complicated, we need to utilize the machine learning techniques to learn the features, decide which charts are interesting and which charts are better than others.

Graph Query Plan. Based on the graph query, we generate a graph query plan. We use a workflow to represent the query plan, which includes multiple query operators (including data management and machine learning operators) and the query operators have some relationships (e.g., precedence order, dependency, decision relation). A straightforward method directly generates the physical plan, executes each query operators based on the relationships, and executes the physical plan without optimization. Note that it is important to select a good query plan to improve the performance and scalability.

Graph Optimization. Traditional database system employs a tree-based optimization model. It employs a table-level coarse-grained optimization, which selects an optimized table-level join order to execute the query. The motivation is to reduce the random access in disk-based setting. However a table-level join order may not be optimal, because different tuples may have different optimal join orders. In disk-based setting, it is hard to get the optimal order for different tuples. However, in memory setting, we have an opportunity to attempt more effective

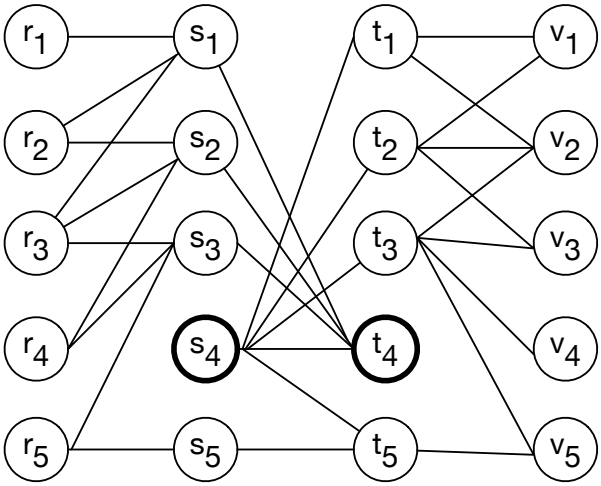

Figure 3: A Graph-Based Optimization Model.

optimizations. To address this problem, we propose a fine-grained tuple-level optimization model, which can find the best order for different tuples (see Section 4). We can also provide graph optimization techniques for other operations.

Graph Management System. There are many graph systems to support graph-based query processing, including disk-based graph systems, in-memory graph systems, and distributed graph systems [6, 5, 9, 18, 6, 24]. Disk-based systems optimize the model to reduce random access. In-memory graph systems utilize the transaction techniques to guarantee high parallelism. Distributed graph systems include synchronized model and asynchronized model, where the former has straggler and imbalance problem while the latter is hard to implement. Our framework should support any graph management system and we focus on automatic suggestion of the best graph execution paths to support different applications.

4 Fine-Grained Graph Processing Model

Given a SQL query, we utilize the graph to directly find the answers of the SQL query. We first consider a simple case that the tables are joined by a chain structure, i.e., each table is joined with at most two other tables and there is no cycle. There are some other join structures, e.g., star join structure and graph join with cycles, while will be discussed later [12].) Suppose there are x join predicates in the SQL. The answers of such SQL query are chains of the graph with x edges.

For example, consider four tables, R, S, T, V in Figure 3 and each table has five tuples. The tuples are connected based on foreign keys. Consider a SQL query with 3 join predicates to join the four tables, e.g., `Select * from R,S,T,V where` $R.A = S.A$, $S.B = T.B$ `and` $T.C = V.C$. The answers of the SQL query are chains in the graph that contain 3 edges and 4 tuples such that the tuples are from different tables. Traditional database systems employ a tree-based optimization model, possibly due to the disk-based optimization requirement. The tree-based optimization model employs a coarse-grained optimization and selects the best table-level join order. For example, the tree model first checks the join predicates between the first two tables and then joins with the third and the fourth tables. The cost of the tree based model is 28. Obviously we can select different best orders for different tuples. For example, we only need to check s_4, t_4 and r_5, s_5, t_5, v_4. The cost is only 6. Thus the graph-based optimization order has much lower cost and can provide finer-grained optimizations.

Graph-Based Optimization Techniques. We can first partition the graph into multiple disjoint connected components. The vertexes in different components cannot form a result. Then we can compute answers from each connected component. We assign each vertex in the graph with a pruning power, i.e., if the vertex is checked, how many tuples will be pruned. For example, the pruning power of s_4 is 8 and the pruning power of t_4 is also 8. Thus we can first check s_4 and t_4. Then we can prune r_1, r_2, r_3, r_4, t_1, t_2, t_3, t_4, s_1, s_2, s_3, s_4, v_1, v_2, v_3, v_4. Then we can get only one result (r_4, s_5, t_5, v_5).

Next we consider other join structures of queries.

Tree Join Structure. The tables are joined by a tree structure and there is no cycle. We can transform it into a chain structure as follows. We first find the longest chain in the tree. Suppose the chain is $T_1, T_2, \cdots, T_x$. Then for each vertex T_i on the chain, which (indirectly) connects other vertices $T'_1, T'_2, \cdots, T'_y$ that are not on the chain, we insert these vertices into the chain using a recursive algorithm. If these vertices are on a chain, i.e., $T_i, T'_1, T'_2, \cdots, T'_y$, then we insert them into the chain by replacing T_i with $T_i, T'_1, T'_2, \cdots, T'_{y-1}, T'_y, T'_{y-1}, \cdots, T_i$. If these vertices are not on a chain, we find the longest chain and insert other vertices not on the chain into this chain using the above method. In this way, we can transform a tree join structure to a chain structure. Note that the resulting chain has some duplicated tables. Hence, joining those tables may result in invalid join tuples (e.g., a join tuple that uses one tuple in the first copy of T_i, and a different tuple in the second copy of T_i). We need to remove those invalid join tuples.

Graph Join Structure. The tables are joined by a graph structure, i.e., there exist cycles in the join structure. We can transform it into a tree structure. For example, given a cycle $(T_1, T_2, \cdots, T_x, T_1)$, we can break the cycle by inserting a new vertex T'_1 and replacing it with T_1. Thus we can transform a cycle to a tree structure, by first finding a spanning tree of the graph using breadth first search, and breaking all non-tree edges.

In summary, given a graph query, a SQL query, a machine learning query, we can use a unified graph model and system to answer the query efficiently. However there are still many open problems in this unified graph based query optimization.

(1) How to build efficient indexes to support various SQL and machine learning queries?

(2) How to support both transactions in data management components and data analytics in machine learning component simultaneously?

(3) How to support concurrency control?

(4) How to support iterative processing?

5 ML As A Service

In 1970s, database was proposed to manage a collection of data. Database has been widely accepted and deployed in many applications because it is easy to use due to its user-friendly declarative query language. In 1990s, search engine was proposed to help Internet users to explore web data, which is also widely used by lay users. Although machine learning is very hot in recent years (every vertical domains claim that they want to use machine learning to improve the performance) and several machine learning systems have been deployed [17, 22, 21, 1, 16, 23], it is still not widely used by non-machine-learning users, because (1) machine learning requires users to understand the underlying model; (2) machine learning requires experts to tune the model to learn the parameters; (3) there is no user-friendly query interface for users to use machine learning; (4) some machine learning algorithms are not easy to explain and users may not appreciate the learning results that are hard to interpret. Stanford also launches a project DAWN on providing infrastructures for machine learning [2]. Different from DAWN, we focus on providing machine learning as a service, which enables ordinary users adopt the machine learning techniques and easily deploy machine learning applications.

Next we shall outline the design requirements.

(1) Scalability. The framework should scale up/out well and scale on volume and dimensionality.

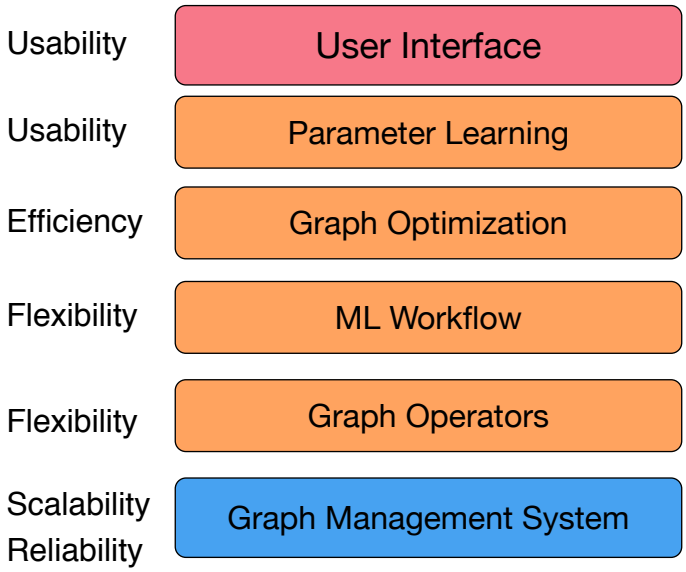

Figure 4: ML as a service

(2) Efficiency. The framework should have low latency and high throughput and we also need to balance the workload.

(3) Reliability. The framework can easily recover for data/machine/network failures.

(4) Flexibility. The framework can support various learning applications.

(5) Usability. The framework should be easy to use for any users. We need to have good abstraction and programming model.

Based on these five factors, we design a new graph-based framework to support ML as a service (as shown in Figure 4).

Encapsulated Operators. We need to encapsulate the machine learning operators so that ordinary users can easily use the operators to accomplish machine learning tasks. For each operator, we also encapsulate multiple algorithms to support the operator. Note that the algorithms are transparent to the end users. For example, we encapsulate clustering, classification, and regression operators. For clustering, we encapsulate multiple clustering algorithms, e.g., K-means, density-based clustering, hierarchical clustering, etc. The users can utilize a single operator or multiple operators to support various applications.

ML Workflow. Each ML operator can support some simple applications (e.g., clustering), and there are some complex applications that require multiple operators (e.g., healthcare). Thus we need to design a model to support applications with multiple operators. We can use a machine learning workflow to describe the model, where each node is an operator and each directed edge is a relationship between two operators. Then we can optimize a machine learning query based on the workflow. (1) We can select the appropriate algorithm for each operator for different applications. (2) We can change the order of two operators to optimize the workflow. (3) We can optimize the structure of the workflow. We can also devise cost-based model and rule-based model to optimize the ML workflow.

Parameter Learning. Machine learning algorithms contain many parameters that significantly affect the performance. Thus it is important to get good parameter values in different applications. However it is tedious and complicated to tune the parameters and therefore, experts are required to be involved to tune the parameters. This is an important reason why users without machine learning background are hard to use the ML algorithms.

Thus it is important to automatically learn the parameter values. In addition, it is important to automate the discovery of data transformation and automatically discover the data drift.

Human-in-the-loop Machine Learning. Machine learning requires to use high-quality labelled data to learn the results. However it is hard to use purely machine learning algorithms. Thus we propose human-in-the-loop machine learning, which utilizes both crowdsourcing and experts to improve machine learning [15]. We use crowd workers to provide high quality labelled data. We can ask the experts to guide the feature selection and parameter learning. Most importantly, we can use active learning techniques to decide when/where to use crowd and when/where to use experts.

6 Research Challenges

Explainable Machine Learning. Most existing machine-learning algorithms function like blackbox and the learning results are not easy to explain, e.g., SVM. As many applications and users require to understand the results to make decision, they require the algorithms to be explainable. For example, in healthcare analytics, the doctors should be convinced why machine-learning results are meaningful and applicable. Thus we require to design explainable techniques to help users better understand machine learning algorithms. We can utilize visualization techniques to visualize the learning process and results. We can also deign new explainable machine learning models.

End-to-end Learning and Deployment. It is expensive to learn the machine learning model and tune the parameters. It is rather hard to deploy the machine learning system for users without machine learning background. Thus it requires to build end-to-end systems that automatically learn the model, tune the parameters, and deploy the system for various applications. The system can also adapt to various domains and involves manual intervention as little as possible.

Incremental Machine Learning. In many applications, machine learning employs a standalone model, which loads the data outside the repository and runs the model out-of-core. However if the data is updated, the method needs to reload the data and run the model from scratch. Obviously this method is not efficient. Thus it calls for incremental learning model that utilizes the original model and the updated data to learn the new model without needing to relearn the model.

Machine Learning on New Hardware. With the development of new hardware, e.g., NVM, GPU, FPGA, RDMA, these new hardware pose new challenges in data management and machine learning. Thus we require to design new techniques and utilize the features of new hardware inherently to improve the performance and scalability of machine learning algorithms. We need to extend the graph model, graph algorithms and graph system to support the new hardware, and devise device-aware optimization techniques.

Graph Processing. Distributed graph processing has straggler and imbalance problems, and we have to design more efficient graph management systems and optimization techniques. Moreover, existing systems focus on iteration graph processing applications, they are expensive for some non-iterative graph applications, e.g., computing shortest path. Thus we need to design new efficient graph systems and framework to support general graph queries.

Machine Learning for Data Integration. We can use machine learning techniques to facilitate data integration. For example, we can use deep learning and embedding techniques to find candidate matching entity/column pairs in data integration. We can also use active learning techniques to reduce the monetary cost. There are two big challenges in using machine learning techniques. The first is to find a large training data to feed the machine learning algorithms. The second is to design new machine learning models for data integration.

7 Conclusion

In this position paper, we sketched a unified graph model for supporting both data management and machine learning. We proposed to provide a user-friendly interface for users to easily use data management and machine learning. We outlined graph-based optimization techniques to improve the performance, which provides a finer-grained optimization than traditional tree-based optimization model. We discussed machine learning as a service and presented a ML workflow with the aim of achieving high flexibility. We discussed emerging research challenges in unifying data management and machine learning.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] P. Bailis, K. Olukotun, C. Ré, and M. Zaharia. Infrastructure for usable machine learning: The stanford DAWN project. *CoRR*, abs/1705.07538, 2017.

[3] C. Chai, G. Li, J. Li, D. Deng, and J. Feng. Cost-effective crowdsourced entity resolution: A partial-order approach. In *SIGMOD*, pages 969–984, 2016.

[4] D. Deng, G. Li, J. Feng, Y. Duan, and Z. Gong. A unified framework for approximate dictionary-based entity extraction. *VLDB J.*, 24(1):143–167, 2015.

[5] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*, pages 17–30, 2012.

[6] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica. Graphx: Graph processing in a distributed dataflow framework. In *OSDI*, pages 599–613, 2014.

[7] D. Jiang, Q. Cai, G. Chen, H. V. Jagadish, B. C. Ooi, K. Tan, and A. K. H. Tung. Cohort query processing. *PVLDB*, 10(1):1–12, 2016.

[8] Y. Jiang, G. Li, J. Feng, and W. Li. String similarity joins: An experimental evaluation. *PVLDB*, 7(8):625–636, 2014.

[9] A. Kyrola, G. E. Blelloch, and C. Guestrin. Graphchi: Large-scale graph computation on just a PC. In *OSDI*, pages 31–46, 2012.

[10] C. Lee, Z. Luo, K. Y. Ngiam, M. Zhang, K. Zheng, G. Chen, B. C. Ooi, and W. L. J. Yip. Big healthcare data analytics: Challenges and applications. In *Handbook of Large-Scale Distributed Computing in Smart Healthcare*, pages 11–41. Springer, 2017.

[11] G. Li. Human-in-the-loop data integration. *PVLDB*, 10(12):2006–2017, 2017.

[12] G. Li, C. Chai, J. Fan, X. Weng, J. Li, Y. Zheng, Y. Li, X. Yu, X. Zhang, and H. Yuan. CDB: optimizing queries with crowd-based selections and joins. In *SIGMOD*, pages 1463–1478, 2017.

[13] G. Li, D. Deng, and J. Feng. Faerie: efficient filtering algorithms for approximate dictionary-based entity extraction. In *SIGMOD*, pages 529–540, 2011.

[14] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*, pages 903–914, 2008.

[15] B. C. Ooi, K. Tan, Q. T. Tran, J. W. L. Yip, G. Chen, Z. J. Ling, T. Nguyen, A. K. H. Tung, and M. Zhang. Contextual crowd intelligence. *SIGKDD Explorations*, 16(1):39–46, 2014.

[16] B. C. Ooi, K. Tan, S. Wang, W. Wang, Q. Cai, G. Chen, J. Gao, Z. Luo, A. K. H. Tung, Y. Wang, Z. Xie, M. Zhang, and K. Zheng. SINGA: A distributed deep learning platform. In *SIGMM*, pages 685–688, 2015.

[17] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J. Crespo, and D. Dennison. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2503–2511, 2015.

[18] Z. Shang, F. Li, J. X. Yu, Z. Zhang, and H. Cheng. Graph analytics through fine-grained parallelism. In *SIGMOD*, pages 463–478, 2016.

[19] Z. Shang, Y. Liu, G. Li, and J. Feng. K-join: Knowledge-aware similarity join. *IEEE Trans. Knowl. Data Eng.*, 28(12):3293–3308, 2016.

[20] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706, 2007.

[21] W. Wang, G. Chen, H. Chen, T. T. A. Dinh, J. Gao, B. C. Ooi, K. Tan, S. Wang, and M. Zhang. Deep learning at scale and at ease. *TOMCCAP*, 12(4s):69:1–69:25, 2016.

[22] W. Wang, M. Zhang, G. Chen, H. V. Jagadish, B. C. Ooi, and K. Tan. Database meets deep learning: Challenges and opportunities. *SIGMOD Record*, 45(2):17–22, 2016.

[23] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu. Petuum: A new platform for distributed machine learning on big data. In *SIGKDD*, pages 1335–1344, 2015.

[24] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[25] K. Zheng, J. Gao, K. Y. Ngiam, B. C. Ooi, and J. W. L. Yip. Resolving the bias in electronic medical records. In *KDD*, pages 2171–2180, 2017.