Data Management in Machine Learning: Challenges, Techniques, and Systems

Arun Kumar  
UC San Diego  
La Jolla, CA, USA

Matthias Boehm  
IBM Research – Almaden  
San Jose, CA, USA

Jun Yang  
Duke University  
Durham, NC, USA

SIGMOD 2017
Who We Are

Arun Kumar
UC San Diego
La Jolla, CA, USA

Matthias Boehm
IBM Research – Almaden
San Jose, CA, USA

Jun Yang
Duke University
Durham, NC, USA

Bismarck
Columbus
Orion
Hamlet

Apache SystemML™

Cumulon
RIOT
Motivation: A Data-Centric View of ML

- **Application Perspective**
  - Machine learning / advanced analytics / deep analytics
  - Modern data-driven applications (e.g., BI, e-commerce, healthcare)

- **Workload Perspective**
  - Repetitive ML workflows
  - Often iterative ML algorithms
  - Often I/O-bound operations (e.g., matrix-vector multiplications)

- **Systems Perspective**
  - ML in data systems
  - DB-inspired ML systems
  - ML Lifecycle Systems

Diagram showing Operational Intensity (FLOPs/Byte) vs. Peak Compute, with Mem Bandwidth and Attainable GFLOPs highlighted.
Motivation: Systems Landscape
Motivation: Tutorial Goals

- **Overall Goal:** Comprehensive review of systems and techniques that tackle data management challenges in the context of ML workloads

- **#1 Categorize Existing Systems**
  - ML in data systems, DB-inspired ML systems, ML lifecycle systems

- **#2 Survey State-of-the-Art Techniques**
  - Query gen, UDFs, factorized learning, deep DBMS integration
  - Optimization and runtime techniques, incl. resource elasticity
  - Model selection and model management

➡️ **Intended Takeaways**
- Awareness of existing systems and techniques
- Survey of effective optimization and runtime techniques
- Overview of open research problems
What this Tutorial is \textbf{NOT}

- Introduction to Machine Learning

- \textbf{Tutorial on General-Purpose Systems}
  - Dataflow systems
  - Graph-focused systems

- \textbf{Tutorial on Deep Learning}
  - Deep learning algorithms
  - Deep learning systems (e.g., Torch, Theano, BigDL, TensorFlow, MXNet, CNTK, Singa, Keras, Caffe, DL4J)

- \textbf{Tutorial on ML for RDBMS Internals}
  - Cost models
  - Workload prediction (e.g., in Peloton)
Tutorial Outline

ML in Data Systems

- 2 Query Generators and UDFs 14min JY
- 3 Factorized Learning and Deep RDBMS Integration 8min AK

DB-Inspired ML Systems

- 4 Rewrites, Operator Selection, and Fusion 14min MB
- 5 Compression, Scan Sharing, and Index Structures 10min MB
- 6 Cloud Resource Elasticity 10min JY

ML Lifecycle Systems

- 7 Feature Engineering, Model Selection/Management 16min AK

Open Problems and Q&A
Part 2: ML with SQL & UDF

“I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail.”

Abraham Maslow, 1966

Jun Yang
Duke University
Durham, NC, USA

SIGMOD 2017
ML in Database – Why?

- **Convenience**
  - “Elephants” (octopi?) have shown remarkable flexibility
  - A single platform for not only data management, transformation, and querying, but also ML and application of insights

- **Efficiency**
  - Move the analysis, not data
  - Can co-optimize various steps involved in the “big data pipeline”

- **Declarativeness**
  - Simplifies development
  - Enables effective automatic optimization, which helps scalability/efficiency
  - One area where the DB community has plenty to offer
Roadmap

- First, examples of what SQL can do for ML, at various levels of abstraction:
  - Matrix multiply
  - Ordinary least squares
  - Gradient descent
  (See backup slides for
   - $k$-means
   - Markov-chain Monte-Carlo)

- Then, a brief discussion of approaches to using SQL for ML
Matrix Multiply: Take 1

- **Data:** $A(i,j,\text{val}), B(i,j,\text{val})$
  - Basically a sparse representation

- **SELECT** $A.i$, $B.j$, $\text{SUM}(A.\text{val}*B.\text{val})$
  **FROM** $A$, $B$
  **WHERE** $A.j = B.i$
  **GROUP BY** $A.i$, $B.j$;

- Works pretty well for sparse matrices
- Not so good for dense matrices, but still beats “small-data” platforms when data doesn’t fit in memory

**MAD Skills** [VLDB'09]
Matrix Multiply: Take 2

- **Data:** $A(i, \text{row}), B(j, \text{col})$
  - $\text{row}$ and $\text{col}$ are ARRAY types or user-defined vector types
  - Basically a row-/column-major representation

- **UDF (user-defined function):** $\text{dotproduct}(v_1, v_2)$ computes the dot product of two vectors

  ```
  SELECT A.i, B.j, dotproduct(A.row, B.col)
  FROM A, B;
  ```

- Works fine for dense matrices

- But still suboptimal in terms of compute-to-I/O ratio

  ![Diagram](image)

  Computation: $O(\ell mn)$, or volume
  I/O: $O(ml + \ell n + nm)$, or surface

  Want instead “blocky” units to maximize compute-to-I/O ratio

- Also note the change in representation (from input to output)
Matrix Multiply: Take 3

- Data: $A(i,j,V), B(i,j,V)$
  - $V$ represents a submatrix; assume the dimensions are compatible
  - Basically a blocked representation

- UDFs
  - $\text{matmult}(V_1, V_2)$ computes the product of two matrices
  - $\text{matsum}(V)$ is a UDA (user-defined aggregate) that sums up input matrices

  ```sql
  SELECT A.i, B.j, matsum(matmult(A.V, B,V))
  FROM A, B
  WHERE A.j = B.i
  GROUP BY A.i, B.j;
  ```

- Choose a “big enough” $V$ with good aspect ratio
  - E.g., square $V$’s beat skinny $V$’s

- UDFs can use optimized libraries like BLAS
Ordinary Least Squares

- To fit data \((X, y)\) to a linear model
  \[ y = X\beta + \epsilon \]
  \[ \beta^* = (X^T X)^{-1} X^T y \]

- Computation involves basic matrix operators expressible in SQL with help of UDFs
  - Inverse is tougher, but assuming the input matrix is small:
    - Code it as a UDF with memory-resident input
    - Processing won’t benefit from DBMS though

\textit{MAD [VLDB'09, '12]}
\textit{SimSQL [ICDE'17]}
Observation

- How far can UDF and UDA go? Surprisingly very!

- **UDF** (oftentimes coded in other languages, e.g., Python and R)
  - Either on the tuple-level (invoked by SQL queries),
  - Or like an application program (invoking SQL queries)

- **UDA**
  - `Init(state)` initializes the state
  - `Accumulate(state, data)` computes updated state with new data
  - [optional] `Merge(state, state)` merges intermediate results computed over disjoint input subsets
  - `Finalize(state)` computes the final result from the state

☞ This pattern covers lots of iterative computation in ML, e.g.
  - $k$-means (backup slides) *GLADE [LADIS'11, SIGMOD'12], MADlib [VLDB'12]*
  - Gradient descent (next)
Gradient Descent (GD)

- Given a model with parameters $w$, we want to learn from data $D$, i.e., minimize a loss function $F(w; D)$
  - E.g., sum of loss over all training data + a regularization term

- Start with some guess $w_0$
- In each step $t + 1$, update $w$ in the direction of the gradient of the loss function at $w_t$, i.e., $F'(w_t)$
- Rinse and repeat

- Under certain (commonly held) conditions, GD converges to a local minimum
  - If $F$ is convex, that’s its global minimum

https://zh.wikipedia.org/wiki/File:Coordinate_descent.svg
Stochastic GD (SGD)

- **Suppose** $F(w; D)$ is linearly separable over $D$
  - I.e., $F(w; D) = \sum_i f_i(w; d_i)$, where $i$ iterates over the data points $D = \{d_i\}_i$

- **Instead of updating** $w$ **using the “full gradient” computed over** $D **in each GD step, just choose a single point in** $D$
  - I.e., use $f'_i(w)$ to approximate $F'(w)$

- **Remarkably, for convex** $F(w)$, SGD also converges to the global minimum, even if we pick points from $D$ in a fixed, arbitrary order
  - Albeit at a slower rate
GD/SGD in SQL

- **GD (full gradient)**
  - Computation of full gradient over $D$ can be done by a query using UDA
  - Several options for driving outer loop
    - *MADlib [VLDB'12]* uses Python UDF
    - *ScalOps [DeBull'12]* uses Datalog
      - Underlying implementation is MapReduce instead of SQL

- **SGD** *Bismarck [SIGMOD'12]*
  - The entire procedure can be written as a query over $D$ using UDA—each `Accumulate()` corresponds to one step
MCMC in SQL

- MCMC (Markov-Chain Monte-Carlo) is a key method in Bayesian ML
- Bayesian ML comes down to analyzing the “posterior” distribution
  \[ P(\text{parameters, hidden variables} \mid \text{observations}) \]
- Direct analysis is often hard, so we use Monte-Carlo simulation
  - Repeatedly sample from the posterior, and analyze the samples
- But sampling directly from the posterior is often hard, so we use MCMC
  - A sampler generates a Markov chain of samples, whose stationary distribution is the target posterior

You can do Gibbs sampling (a form of MCMC) in SimSQL [SIGMOD'13]
  - With user-defined “value-generating” functions that draw samples
  - See backup slides for details
Approaches to SQL+ML

Backend choices

- “On top of” (e.g., RIOT-DB [CIDR'09], MAD [VLDB'09,VLDB'12]) vs. “inside” DBMS (e.g., SimSQL [ICDE'17])

- Not DBMS, but still inspired by or rooted in DBMS
  - General-purpose “big-data” platform (e.g., SystemML [ICDE'11,VLDB'16], Cumulon [SIGMOD'13])
  - Specialized system from ground up (e.g., RIOT [ICDE'10], SciDB [CSE'13])

Interface choices

- SQL + libraries or extensions (e.g., MAD [VLDB'09,VLDB'12], SimSQL [ICDE'17], Oracle Data Mining, …)

- ML-oriented languages on top of SQL (e.g., RIOT-DB [CIDR'09], BUDS/SimSQL [SIGMOD'17], Oracle R Enterprise, …)
Interface: SQL + Libraries/Extensions

- Especially nice with integrated model management, e.g.,
  **Oracle Data Mining**
  - Can create, store, update, and apply models in SQL

```sql
-- Create model settings:
CREATE TABLE svm_settings(
  setting_name VARCHAR2(30),
  setting_value VARCHAR2(30));
INSERT INTO svm_settings VALUES(
  dbms_data_mining.algo_name,
  dbms_data_mining.algo_support_vector_machines);
-- ...
-- Build model:
DBMS_DATA_MINING.CREATE_MODEL(
  model_name => 'svm_model',
  mining_function => dbms_data_mining.classification,
  data_table_name => 'mining_data_build_v',
  case_id_column_name => 'cust_id',
  target_column_name => 'affinity_card',
  settings_table_name => 'svm_settings');
-- Apply model:
DBMS_DATA_MINING.APPLY(
  model_name => 'svm_model',
  data_table_name => 'mining_data_apply_v',
  case_id_column_name => 'cust_id',
  result_table_name => 'svm_apply_result');
```
Interface: no SQL

- Let user write whatever they are comfortable with (R, Python, etc.)
  - Provide a library of data manipulation and ML functions implemented by
    the underlying system; can pre-compile user code
    - SQL underneath: RIOT [CIDR'09,ICDE'10], BUDS/SimSQL [SIGMOD'17],
      Oracle R Enterprise, etc.
  - Other "big-data" platforms underneath: SystemML [ICDE'11,VLDB'16],
    Spark R, Mahout Samsara, etc.

Bayesian LASSO in BUDS... in Mahout Samsara... in SystemML

(Examples from BUDS/SimSQL [SIGMOD'17])
Summary

- You can get a lot of mileage for machine learning with SQL+UDF (octopus + hammer)

- Deep roots in
  - DBMS extensibility research
  - Array DBMS, e.g., SciDB [CSE'13]; see Rusu & Cheng [arXiv 2013] for survey

- Next: more opportunities for deeper ML+DB integration
References for Part 2: ML with SQL & UDF

- **MADlib** [VLDB'12] Hellerstein et al. “The MADlib Analytics Library or MAD Skills, the SQL.” PVLDB 5(12), 2012
- **RIOT-DB** [CIDR'09] Zhang et al. “RIOT: I/O-efficient numerical computing without SQL.” CIDR 2009
- **RIOT** [ICDE'10] Zhang et al. “I/O-efficient statistical computing with RIOT.” ICDE 2010
- **SimSQL** [ICDE'17] Luo et al. “Scalable Linear Algebra on a Relational Database System.” ICDE 2017
- **SystemML** [VLDB'16] Boehm et al. “SystemML: Declarative machine learning on Spark.” PVLDB 9(13), 2016
Part 2 Backup/Extra Slides
$k$-Means Clustering

- Given $n$ points, find $k$ centroids to minimize sum of squared distances between each point and its closest centroid.

- EM-style iterative algorithm:
  1. Pick initial $k$ candidate centroid locations.
  2. Assign each point to the closest candidate.
  3. Reposition each candidate as the centroid of its assigned points.
  4. Repeat 2-3 above until assignment changes no more (or very little).
$k$-Means as UDA

- **State:** $k$ candidates with locations + cluster info
  \{\langle loc_i, sum_i, cnt_i \rangle \}_{1 \leq i \leq k}

- **Init:** given centroid locations, with sum and count of 0

- **Accumulate:** given a data point $p$, find the candidate $i$ closest to $p$; increment $sum_i$ by $p$’s coordinates and $cnt_i$ by one

- **Merge:** merge $\langle loc, sum, cnt \rangle$ records by $loc$; add $sum$ and $cnt$

- **Finalize:** for each $i$, compute new $loc_i$ as $sum_i / cnt_i$

One SQL query with this UDA gives one iteration of the EM algorithm

- For the next iteration, the UDA will be initialized with the $k$ locations computed from the previous
- Can use a UDF to drive overall iterations
- Termination condition can be evaluated in SQL too (see MADlib)

---

GLADE [LADIS'11, SIGMOD'12]

MADlib [VLDB'12]
Markov-Chain Monte-Carlo (MCMC)

- Bayesian ML comes down to analyzing the “posterior” distribution
  \[ P(\text{parameters, hidden variables} \mid \text{observations}) \]
- **Direct analysis is often hard, so we use Monte-Carlo simulation**
  - Repeatedly sample from the posterior, and analyze the samples
- **But sampling directly from the posterior is often hard, so we use MCMC**
  - A sampler generates a Markov chain of samples, whose stationary distribution is the target posterior
Example: Gibbs Sampling

- Suppose we have an \( n \)-variate distribution, but the conditional distributions are easier to sample from
- Begin with some initial sample \( \mathbf{z}^{(0)} \)
- For the \((t + 1)\)-th sample \( \mathbf{z}^{(t+1)} \), sample each component \( z_i^{(t+1)} \) conditioned on all other components sampled most recently, i.e.,
  \[
p \left( z_i^{(t+1)} \Big| z_1^{(t+1)}, \ldots, z_{i-1}^{(t+1)}, z_{i+1}^{(t)}, \ldots, z_n^{(t)} \right)
\]
- Rinse and repeat
MCMC in SimSQL

SimSQL [SIGMOD’13]

- Think of each sample as a table (tables)
- Write UDF to define “VG” (value-generating) functions that draw samples
- Write SQL with VG functions to define how to generate $T[t]$ (instance of table $T$ in the $t$-th sample) from $T[t - 1]$
- Write SQL to simulate multiple MCMC chains, and to compute distributional properties for variables of interest from $T[t]$’s across $T$’s, $t$’s, and chains

An example of staying true to the declarative roots of databases

- But also need new techniques not in traditional DBMS, e.g.:
  - Plans are huge—cut them into “frames”; observe execution stats of last frame and to optimize the next
  - Use “tuple bundles” to instantiate/process multiple possible worlds simultaneously
Part 3: Learning Over Joins, SRL, and Deep RDBMS Integration

Arun Kumar
UC San Diego
La Jolla, CA, USA

SIGMOD 2017
Overview: Learning Over Joins

Problem: Many datasets are multi-table → ML toolkits assume single-table inputs → ML after joining tables

Overheads:
- Extra storage
- Computational redundancy
- Join time
- Maintenance headaches

Learning Over Joins: “Push Down” ML through joins
1) Over standard data systems: Orion, Santoku, Morpheus
2) Over a “factorized database” system: FDB-F
3) Special-purpose tools: libFM, TensorDB, Compressed ML

Related but orthogonal: Statistical relational learning (DeepDive, etc.)
Learning Over Joins

Over standard data systems: Orion, Morpheus, Santoku

Example: GLMs with gradient descent (GD)

\[ L(w) = \sum_{i=1}^{n} f(w' x_i, y_i) \quad \nabla L(w) = \sum_{i=1}^{n} g(w' x_i, y_i) x_i \]

Orion [SIGMOD’15]:
Introduced the scalable “factorized learning” idea
Easy UDA implementation on existing data systems (RDBMS/Hive/Spark)

Morpheus [VLDB’17]:
Generalizes factorized learning to any ML algorithm in linear algebra
“Push down” rewrites for matrix-vector mult., gramian, ginv, etc.

Santoku [VLDB’15]: Discrete features (Naive Bayes, trees, etc.)
Learning Over Joins

Over a “factorized database” system: FDB-F [SIGMOD’16]
Generalized semiring-based aggregates over “factorized joins”
SRL; Deep RDBMS Integration

SRL combines statistical learning with logic-based rules/constraints

“Non-IID” ML models (MVDs, EMVDs, JDs) NIPS’12 tutorial by Lise Getoor Book with Ben Taskar

Inference and learning often perform joins internally!

Scalable grounding using RDBMS: Tuffy [VLDB’10]
Incremental maintenance: IncrementalDeepDive [VLDB’15]

Increasing interest in deeper integration of ML into DBMS kernel!

SAP HANA SLACID: Linear algebra kernels in an RDBMS [SSDBM’14]
New compressed sparse row/col. representations
Integrated API for basic access patterns and lin. alg. ops
OpenMP-based shared memory parallelism in DBMS task scheduler
References: Part 3

Columbus [SIGMOD’14]: Materialization Optimizations for Feature Selection Workloads
DeepDive [DataEng’14]: Feature Engineering for Knowledge Base Construction
FDB-F [SIGMOD’16]: Learning Linear Regression Models over Factorized Joins
IncrementalDeepDive [VLDB’15]: Incremental Knowledge Base Construction Using DeepDive
Morpheus [VLDB’17]: Towards Linear Algebra over Normalized Data
Orion [SIGMOD’15]: Learning Generalized Linear Models Over Normalized Data
Santoku [VLDB’15]: Demonstration of Santoku: Optimizing Machine Learning over Normalized Data
SLACID [SSDBM’14]: SLACID - Sparse Linear Algebra in a Column-Oriented In-Memory Database System
Tuffy [VLDB’10]: Tuffy: Scaling up Statistical Inference in Markov Logic Networks using an RDBMS
Backup Slides
Statistical Relational Learning Systems

Captures logical dependencies between entities/variables

“Non-IID” ML models
(MVDs, EMVDs, JDs)

PODS tutorial by Lise Getoor on Tue!
(also NIPS’12; book with Taskar)

Example: Markov Logic Network (MLN); used by DeepDive

MLN inference (MAP) computes “most probable world” by plugging values of variables to predict

Scalable grounding using RDBMS: Tuffy [VLDB’10]
Scalable Gibbs sampling: Elementary [SIGMOD’13]
Incremental maintenance: IncrementalDeepDive [VLDB’15]
Deep RDBMS Integration

Integrating linear algebra kernels into an RDBMS: SAP HANA

**SLACID [SSDBM’14]:** Mutable columnar layout for sparse matrices
- Compressed sparse row/col. representation + incr. delta
- Integrated API for basic access patterns and lin. alg. ops
- OpenMP-based shared memory parallelism in DBMS task scheduler

Time series-specific systems: Fa, F2DB

**Fa [VLDB’07]:** “Declarative forecasting” queries for time series
- Projection and shift-based time series feature transformations
- Feature ranking and subset selection heuristics
- Lin. reg., Bayesian networks, SVM, CART, Random Forest
- Both one-time and continuous forecasting
Part 4: Rewrites, Operator Selection, and Operator Fusion

Matthias Boehm
IBM Research – Almaden
San Jose, CA, USA

SIGMOD 2017
Overview Optimizing Compilers for ML Algorithms

- **Comparison Query Optimization**
  - Rule- and cost-based rewrites and operator ordering
  - Physical operator selection and query compilation
  - Linear algebra / other ML operators, DAGs, control flow, sparse/dense formats

- **#1 Interpretation** (operation at-a-time)
  - Examples: Morpheus [PVLDB’17]

- **#2 Lazy Expression Compilation** (DAG at-a-time)
  - Examples: RIOT [CIDR’09], Mahout Samsara [MLSystems’16]
  - Examples w/ control structures: Weld [CIDR’17], OptiML [ICML’11], Emma [SIGMOD’15]

- **#3 Program Compilation** (entire program)
  - Examples: SystemML [PVLDB’16], Cumulon [SIGMOD’13], Tupleware [PVLDB’15]

**Optimization Scope**

```
1: X = read($1); # n x m matrix
2: y = read($2); # n x 1 vector
3: maxi = 50; lambda = 0.001;
4: intercept = $3;
5: ...
6: r = -(t(X) %*% y);  
7: norm_r2 = sum(r * r); p = -r;
8: w = matrix(0, ncol(X), 1); i = 0;
9: while(i<maxi & norm_r2>norm_r2_trgt) {  
10:   q = (t(X) %*% X %*% p)+lambda*p;       
11:   alpha = norm_r2 / sum(p * q);  
12:   w = w + alpha * p;
13:   old_norm_r2 = norm_r2;
14:   r = r + alpha * q;
15:   norm_r2 = sum(r * r);
16:   beta = norm_r2 / old_norm_r2;
17:   p = -r + beta * p; i = i + 1;
18: }
19: }
20: write(w, $4, format="text");
```
Logical Simplification Rewrites

- **Traditional PL Rewrites** (e.g., TensorFlow, OptiML, SystemML)
  - CSE, constant folding, branch removal

- **Algebraic Simplification Rewrites** (e.g., SystemML, FAQ [PODS’16])
  - \( t(X) \%\% y \rightarrow t(t(y) \%\% X) \)
  - \( \text{trace}(X \%\% Y) \rightarrow \text{sum}(X * t(Y)) \)
  - \( \text{sum}(X + Y) \rightarrow \text{sum}(X) + \text{sum}(Y) \)
  - \( \text{sum}(X^2) \rightarrow t(X) \%\% X, \text{iff ncol}(X)=1 \)

- **Loop Vectorization** (e.g., OptiML, SystemML)
  
  ```
  for(i in a:b)
  X[i,1] = Y[i,2] + Z[i,1]
  ```

  \( X[a:b,1] = Y[a:b,2] + Z[a:b,1] \)

- **Incremental Computations**
  - Delta update rules (e.g., LINVIEW [SIGMOD’14], factorized [CoRR’17])
  - Incremental iterations (e.g., Flink)
    - \( A = t(X) \%\% X + t(\Delta X) \%\% \Delta X \)
  - Update-in-place (e.g., SystemML)
    - \( b = t(X) \%\% y + t(\Delta X) \%\% \Delta y \)
Logical Simplification Rewrites
Matrix Multiplication Chain Optimization

- **Optimization Problem**
  - Matrix multiplication chain of \( n \) matrices \( M_1, M_2, ... M_n \) (associative)
  - Optimal parenthesization of the product \( M_1M_2 ... M_n \)

- **Search Space Characteristics**
  - Naïve exhaustive: Catalan numbers \( \Omega(4^n / n^{3/2}) \)
  - DP applies: (1) optimal substructure, (2) overlapping subproblems
  - Textbook DP algorithm [MIT Press’09]: \( \Theta(n^3) \) time, \( \Theta(n^2) \) space
    - Examples: SystemML [Data Eng. Bull. ’14], RIOT (including I/O costs), SpMachO (including sparsity for intermediates) [EDBT’15],
  - Best known algorithm: \( O(n \log n) \)
Matrix Multiplication Chain Optimization

<table>
<thead>
<tr>
<th></th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>10x7</td>
<td>7x5</td>
<td>5x1</td>
<td>1x3</td>
<td>3x9</td>
</tr>
</tbody>
</table>

Cost matrix

\[
m_{1,3} = \min(m_{1,1} + m_{2,3} + p_1p_2p_4, m_{1,2} + m_{3,3} + p_1p_3p_4)
\]

\[
m_{1,3} = \min(0 + 35 + 10 \times 7 \times 1, 350 + 0 + 10 \times 5 \times 1)
\]

\[
m_{1,3} = \min(105, 400) = 105
\]
Matrix Multiplication Chain Optimization

<table>
<thead>
<tr>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
</tr>
</thead>
<tbody>
<tr>
<td>10x7</td>
<td>7x5</td>
<td>5x1</td>
<td>1x3</td>
<td>3x9</td>
</tr>
</tbody>
</table>

Cost matrix $m$

Optimal split matrix $s$

$\Rightarrow$ Open questions: DAGs; other operations, joint opt w/ rewrites, CSE, fusion, and physical operators

$((M1 (M2 M3)) (M4 M5))$
Physical Rewrites and Optimizations

- **Distributed Caching**
  - Redundant compute vs. memory consumption and I/O
  - #1 Cache intermediates w/ multiple refs (Emma)
  - #2 Cache initial read and read-only loop vars (SystemML)

- **Partitioning**
  - Many frameworks exploit co-partitioning for efficient joins
  - #1 Partitioning-exploiting operators (SystemML, Emma, Samsara)
  - #2 Inject partitioning to avoid shuffle per iteration (SystemML)
  - #3 Plan-specific data partitioning (SystemML, Dmac [SIGMOD’15], Kasen [PVLDB’16])

- **Other Data Flow Optimizations** (Emma)
  - #1 Exists unnesting (e.g., filter w/ broadcast → join)
  - #2 Fold-group fusion (e.g., groupByKey → reduceByKey)

- **Physical Operator Selection**
Physical Operator Selection

- **Common Selection Criteria**
  - **Data and cluster characteristics** (e.g., data size/shape, memory, parallelism)
  - **Matrix/operation properties** (e.g., diagonal/symmetric, sparse-safe ops)
  - **Data flow properties** (e.g., co-partitioning, co-location, data locality)

- **#0 Local Operators**
  - SystemML mm, tsmm, mmchain; Samsara/Mllib local linalg

- **#1 Special Operators** (often fused operators)
  - Special patterns (SystemML tsmm, tsmm2, mapmmchain, pmm; Samsara AtA)
  - Sparsity exploiting (SystemML wdivmm, wsloss, wcemm; Cumulon maskMult)

- **#2 Broadcast-Based Operators** (aka broadcast join)
  - SystemML mapmm, mapmmchain

- **#3 Co-Partitioning-Based Operators** (aka improved repartition join)
  - SystemML zipmm; Emma, Samsara OpAtB

- **#4 Shuffle-Based Operators** (aka repartition join)
  - SystemML cpmm, rmm; Samsara OpAB
Example Physical Operators

- Example Linear Regression Direct Solve
  - Transpose-self for t(X) %*% X
  - Broadcast-based for t(X) %*% y
  - Logical and physical rewrites
  - E.g., Samsara, SystemML

\[ A = t(X) \%*\% X \]
\[ b = t(X) \%*\% y \]
\[ w = \text{solve}(A, b) \]

Input Matrices

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_{1,1} )</td>
<td>( y_1 )</td>
<td></td>
</tr>
<tr>
<td>( X_{2,1} )</td>
<td>( y_2 )</td>
<td></td>
</tr>
<tr>
<td>( X_{m,1} )</td>
<td>( y_m )</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- \( t(b) \)
- \( \text{fold(sum)} \)
- \( \text{map}(\text{mapmm}) \)
- \( \text{map(tsmm)} \)
- \( \text{broadcast()} \)
- \( \text{persist(MEM_DISK)} \)
- \( t(y) \)
- \( X \)
Fused Operators

- **Motivation**
  - Problem: Memory-bandwidth-bound operations (I/O)
  - Goal: Reduce number of scans and intermediates

- **Matrix-Vector Chains:** \( t(X) \%\% (X\%\%v) \)
  - Fused single-pass operator: `mmchain` [PPoPP’15]
  - Row-aligned creation/consumption

- **Ternary Aggregates:** `sum(X*Y*Z)`
  - Fused aggregation operator
  - Avoid materialized intermediates

- **Other ML-Specific Operators**
  - Sample proportion: \( X \ * \ (1-X) \)
  - Sigmoid: \( \frac{1}{1 + \exp(-X)} \)
  - Axpy: \( X + s*Y, X - s*Y \)
Sparsity-Exploiting Fused Operators

- **Goal:** Avoid dense intermediates and unnecessary computation

- **#1 Fused Physical Operators**
  - E.g., SystemML [PVLDB’16]
    - wsloss, wcemm, wdivmm
  - Selective computation over non-zeros of “sparse driver”

- **#2 Masked Physical Operators**
  - E.g., Cumulon MaskMult [SIGMOD’13]
  - Create mask of “sparse driver”
  - Pass mask to single masked matrix multiply operator

-open questions: NaN handling, automatic operator fusion (codegen)
Automatic Operator Fusion

- **Motivation**
  - Large development effort for hand-coded fused operators
  - UDF-centric systems w/o pre-defined operators

- **General Approach: Fuse by Access Pattern**
  - #1 Loop fusion (OptiML, Tupleware, Weld, TensorFlow XLA [github'17])
  - #2 Templates (Kasen, SPOOF [CIDR’17])
  - Scope: expression or program compilation

- **Additional Techniques**
  - Tupleware: Micro optimizations (tile-at-a-time, predicates, result allocation)
  - Weld: Cross-library optimizations (via common IR of basic operations)
  - SystemML-SPOOF: sparsity-exploiting fused operators

→ Open question: Optimization of fusion plans for DAGs (redundant compute vs materialization, access patterns)
Runtime Adaptation (see AQP)

- **Problem of Unknown/Changing Size Information**
  - Dimensions/sparsity required for cost comparisons/valid plans
  - Unknowns → conservative fallback plans

- **Challenges**
  - Conditional control flow, function call graphs, UDFs
  - Data-dependent ops (e.g., sampling, group by classes, output sparsity)
  - Computed size expressions, changing dimensions/sparsity

- **Approaches**
  - **#1 Lazy expression optimization** (RIOT, OptiML, Emma, Weld, Samsara)
    - Optimize on triggering actions (unconditional scope)
  - **#2 Dynamic inter-DAG recompilation** (SystemML)
    - Split/mark DAGs, recompile DAGs/functions w/ exact stats

- **Open questions:**
  - Estimating the size and sparsity of intermediates
  - Adaptive query processing and storage
References for Part 4

Part 5: Compression, Scan Sharing, and Index Structures

Matthias Boehm
IBM Research – Almaden
San Jose, CA, USA

SIGMOD 2017
Motivation: Workload Characteristics

- **Memory-Bandwidth-Bound Operations**
  - Iterative ML algorithms w/ read-only data access
  - **#1:** I/O-bound matrix vector products
    - Crucial to fit matrix into memory (single node, distributed, GPU)
    - Avoid unnecessary scans
  - **#2:** Matrix and vector intermediates
    - Reduce number of reads and writes

- **Common Data Characteristics**
  - Tall & skinny matrices (#row >> #columns)
  - Non-uniform sparsity
  - Low column cardinality
  - Column correlations

```c
while(!converged) {
    ... q = X %*% v ...
}
```

![Graphs showing data characteristics for Covertype, ImageNet, and Mnist8m](image)
Motivation: Workload Characteristics

- **Single Node**: 2x6 E5-2440 @2.4GHz–2.9GHz, DDR3 RAM @1.3GHz (ECC)
  - Peak memory bandwidth: $2 \times 32\text{GB/s}$ (local), $2 \times 12.8\text{GB/s}$ (remote QPI)
  - Peak compute bandwidth: $2 \times 115.2\text{GFlops/s}$

- **Roofline Analysis**
  
  ![Graph showing the relationship between memory bandwidth and compute bandwidth. The graph illustrates the attainable GFlops/s and operational intensity (Flops/Byte) for different workloads. The diagram highlights the 36x IO-bound matrix-vector multiplication optimized by SystemML.]
Background: Block Partitioning and Layouts

- Blocked Matrix Representations
  - Blocks, a.k.a. “tiles”, “chunks”, or “pages”
  - #1 Logical (fixed-size) blocking (→ var. physical size)
  - #2 Physical blocking (→ fixed physical size)
  - Blocks encoded independently (dense/sparse)
  - Local matrices → single block

- Common Block Representations
  - Dense (linearized arrays)
  - CSR (compressed sparse rows)
  - CSC (compressed sparse columns)
  - MCSR (modified CSR)
  - COO (Coordinate matrix)
  - ...

Example 3x3 Matrix

Dense (row-major)

```
.7 0 .1 2 .4 0 0 .3 0
```

Logical blocking 3,400x2,700 matrix (w/ $B_c=1,000$)

<table>
<thead>
<tr>
<th></th>
<th>(1,1)</th>
<th>(1,2)</th>
<th>(1,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2,1)</td>
<td>(2,2)</td>
<td>(2,3)</td>
<td></td>
</tr>
<tr>
<td>(3,1)</td>
<td>(3,2)</td>
<td>(3,3)</td>
<td></td>
</tr>
<tr>
<td>(4,1)</td>
<td>(4,2)</td>
<td>(4,3)</td>
<td></td>
</tr>
</tbody>
</table>

CSR

```
0 .7 0 0 .1 2 .4 0 0 .3 0
```

MCSR

```
0 0 .7 2 .1 0 .1 0 .2 1 .3 0
```

COO

```
0 0 .7 0 2 .1 0 1 .4 1 1 .4 1 1 .3 2 1 .3
```
Overview Techniques for Data-Intensive Machine Learning

- **#1 (Distributed) Caching**
  - Keep read only feature matrix in (distributed) memory

- **#2 Compression**
  - Fit larger datasets into available memory

- **#3 Scan Sharing (and operator fusion)**
  - Reduce the number of scans as well as read/writes

- **#4 Index Structures**
  - Out-of-core data, I/O-aware ops, updates

- **#5 NUMA-Aware Partitioning and Replication**
  - Matrix partitioning / replication → data locality

- **#6 Buffer Pool Management**
  - Graceful eviction of intermediates, out-of-core ops
Compression Techniques

- **#1 Block-Level General-Purpose Compression**
  - Heavyweight or lightweight compression schemes
  - Decompress matrices block-wise for each operation
  - E.g.: Spark RDD compression (Snappy/LZ4), SciDB SM [SSDBM’11], TileDB SM [PVLDB’16]

- **#2 Block-Level Matrix Compression**
  - Compress matrix block with common encoding scheme
  - Perform LA ops over compressed representation
  - E.g.: CSR-VI (dict) [CF’08], cPLS (grammar) [KDD’16], TOC (LZW w/ trie) [CoRR’17]

- **#3 Column-Group-Level Matrix Compression**
  - Compress column groups w/ heterogenous schemes
  - Perform LA ops over compressed representation
  - E.g.: SystemML CLA (RLE, OLE, DDC, UC) [PVLDB’16]
Scan Sharing Techniques

- **#1 Batching**
  - One-pass evaluation of multiple configurations
  - Use cases: EL, CV, feature selection, hyper parameter tuning
  - E.g.: TUPAQ [SoCC’16], Columbus [SIGMOD’14]

- **#2 Fused Operator DAGs**
  - Avoid unnecessary scans, (e.g., part 4 mmchain)
  - Avoid unnecessary writes / reads
  - Multi-aggregates, redundancy
  - E.g.: SystemML codegen
    - \[ a = \text{sum}(X^2) \]
    - \[ b = \text{sum}(X*Y) \]
    - \[ c = \text{sum}(Y^2) \]

- **#3 Runtime Piggybacking**
  - Merge concurrent data-parallel jobs
  - “Wait-Merge-Submit-Return”-loop
  - E.g.: SystemML parfor [PVLDB’14]
    - \[ \text{parfor}( i \ \text{in} \ 1:\text{numModels} ) \]
    - \[ \text{while}( \ !\text{converged} ) \]
    - \[ q = X \times \times \times v; \ldots \]
Index Structures and NUMA Awareness

- **Goals:** Out-of-core operations and data placement

- **Index Structures**
  - Tree structures of blocks w/ user-defined/fixed linearization functions
  - **LAB-Tree** (Linearized Array B-tree, RIOT) [PVLDB’11]
    - Leaf-splitting strategies, and update batching via flushing policies
  - **TileDB Storage Manager** [PVLDB’16]
    - Two-level blocking and update batching via fragments
  - **AT MATRIX** (Adaptive Tile Matrix, SAP HANA) [ICDE’16]
    - Two-level blocking and NUMA-aware range partitioning

- **NUMA-Aware Model/Data Replication**
  - **DimmWitted**: HW vs statistical efficiency [PVLDB’14]
  - Model: PerCore, PerNode, PerMachine
  - Data: partitioning (sharding), full replication

⇒ Open questions: Heterogenous hardware, cache coherence, etc
References for Part 5

- Stavros Papadopoulos et al. The TileDB Array Data Storage Manager. PVLDB 10(4), 2016.
Backup: Compressed Linear Algebra (CLA)

- **Overview compression framework**
  - Column-wise matrix compression (values + offset lists / references)
  - Column co-coding (column groups encoded as single unit)
  - Heterogeneous column encoding formats (OLE, RLE, DDC, UC)

- **Experiments**
  - **LinregCG**, 10 iterations, SystemML 0.14
  - 1+6 node cluster, Spark 2.1

### Compression Ratios

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Gzip</th>
<th>Snappy</th>
<th>CLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Higgs</td>
<td>1.93</td>
<td>1.38</td>
<td>2.17</td>
</tr>
<tr>
<td>Census</td>
<td>17.11</td>
<td>6.04</td>
<td>35.69</td>
</tr>
<tr>
<td>Covtype</td>
<td>10.40</td>
<td>6.13</td>
<td>18.19</td>
</tr>
<tr>
<td>ImageNet</td>
<td>5.54</td>
<td>3.35</td>
<td>7.34</td>
</tr>
<tr>
<td>Mnist8m</td>
<td>4.12</td>
<td>2.60</td>
<td>7.32</td>
</tr>
<tr>
<td>Airline78</td>
<td>7.07</td>
<td>4.28</td>
<td>7.44</td>
</tr>
</tbody>
</table>

### End-to-End Performance [sec]

- Uncompressed
- Snappy (RDD Compression)
- CLA

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Mnist40m</th>
<th>Mnist240m</th>
<th>Mnist480m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agg mem</td>
<td>89</td>
<td>765</td>
<td>5663</td>
</tr>
<tr>
<td>Clusters</td>
<td>135</td>
<td>463</td>
<td>2730</td>
</tr>
<tr>
<td>90GB</td>
<td>93</td>
<td>998</td>
<td>1.1TB</td>
</tr>
<tr>
<td>540GB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1TB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Backup: Index Structures

- **Overview Common Indexing Techniques**
  - Physical blocking w/ leaf splitting strategies
  - Dense and sparse leaf blocks w/ contiguous ranges of cells
  - Batching of updates (deferred insertion)

- **LAB-Tree (Linearized Array B-tree, RIOT)** [PVLDB 2011]
  - Operations: get, scan (iterator w/ given order), left/right indexing (on disk)
  - B-tree w/ physical blocking (sparse/dense), leaves have assigned ranges
  - Array linearization via UDFs (e.g., row/column major, Z-order, etc)
  - **Leaf splitting strategies:** split-in-middle, split-aligned, split-off-dense, split-defer-next, split-balanced-ratio
  - **Flushing policies for update batching:** flush-all, least-recently-used, smallest-page, largest-page, largest-page-probabilistically, largest-group
Backup: Index Structures, cont.

- **AT MATRIX (Adaptive Tile Matrix, SAP HANA) [ICDE 2016]**
  - Operations: matrix multiplication ATMult (in-memory)
  - Two-level blocking: **Adaptive variable-sized tiles** (dense or sparse w/ CSR), composed of atomic squared blocks
  - Two-dimensional quad-tree, w/ Z-order as linearization function
  - Horizontal partitioning across NUMA nodes

- **TileDB Storage Manager [PVLDB 2016]**
  - Operations: init, write, read, consolidate, finalize (on disk)
  - Two-level blocking: space tiles (fixed size), data tiles (variable size for sparse)
  - Two-level linearization: cell order and tile order (row/column major)
  - **Fragments for update batching**
    ("a timestamped snapshot of a batch of array updates")
Backup: NUMA-Aware Partitioning and Replication

- **AT MATRIX (Adaptive Tile Matrix)**
  - Recursive NUMA-aware partitioning into dense/sparse tiles
  - Inter-tile (worker teams) and intra-tile (threads in team) parallelization
  - Job scheduling framework from SAP HANA (horizontal range partitioning, socket-local queues with task-stealing)

- **NUMA-Aware Model and Data Replication**
  - DimmWitted: HW vs statistical efficiency
  - Model Replication
    - PerCore, PerMachine
    - PerNode (hybrid)
  - Data Replication
    - Partitioning (sharding)
    - Full replication

⇒ Open questions: Heterogeneous hardware, cache coherence, etc.
Backup: Buffer Pool Management

#1 Intermediates of LA Programs
- Hybrid runtime plans of in-memory and distributed operations
- Graceful eviction of intermediates at granularity of variables
- Example: SystemML
  - Soft references for in-memory matrices and broadcasts
  - LRU, FIFO buffer replacement strategies

#2 Operation/Algorithm-Specific Buffer Management
- Operations/algorithms over out-of-core matrices and factor graphs
- Page-level storage layout and buffer replacement policies
- Example #2a: RIOT
  - Chains of matrix multiplications
  - Operation-aware I/O schedules
- Example #2b: Elementary
  - LR, CRF, LDA over out-of-core factor graphs
  - Materialization strategies and MRU/LFU buffer replacement
Part 6: Resource Elasticity

“Intelligence is the ability to adapt to change.”

Stephen Hawking (?)

Jun Yang
Duke University
Durham, NC, USA

SIGMOD 2017
Rise of Cloud

- Cluster computing for big data is easier than ever
  - Clouds allow you to get a cluster on demand, and pay as you go
  - There is a growing ecosystem of platforms and tools for data analysis

Challenges

- Maddening array of “knobs”
  - Hardware provisioning, software configuration, program tuning

- “Elastic” environment
  - Multi-tenant clusters, fluctuating markets, failures
  - Particularly hard for large-scale, long-running ML workloads
Roadmap

- Provisioning (& scheduling): what do I need (& when)?
- Recovery: what do I do when what I need fails?
- Working with markets

These problems are not limited to DB & ML workloads, but we shall see how DB & ML add twists
Provisioning: Example Decisions

- Given an ML program, what types of machines to acquire, and how many
  - A bigger cluster may get results faster, but cost more
  - No perfect speedup, so big clusters may not give good cost/time trade-off
  - Cumulon [SIGMOD'13+follow-up]

- Given a cluster, how to configure the execution of an ML program
  - What’s the appropriate degree of parallelism for an execution step?
    - Overhead of parallelism isn’t always justified
  - How much memory do we allocate to master and work processes?
    - Optimal allocation depends on computation and data access patterns
  - SystemML [DEBull'14,PVLDB'16]
  - ScalOps [DeBull'12]
  - SystemML [SIGMOD'15]

촉 Decisions interact with optimizations discussed earlier
  - Cluster configuration affects degree of parallelism and memory allocation, as well as optimal execution strategies
Provisioning/Scheduling: Techniques

Depend on the level of abstraction:

- **Program is a black box**
  - First observe, and then decide; can leverage past execution profiles

- **Program is broken down into a workflow with clear input/output for each unit, e.g., MapReduce, Spark**
  - More effective profiling and optimization on a per-unit basis

- **Program is specified declaratively, DB-style**
  - Reusable and composable cost models
  - Bigger search space through rewrites
  - Cost-based what-if analysis
  - *SystemML* [ICDE'11+follow-up]
  - *Cumulon* [SIGMOD'13+follow-up]

- **Program follows a specific template**
  - Even more opportunities arise; e.g., scheduling parameter updates/synchronization in *parameter servers* [VLDB'10,OSDI'14] + resource provisioning in *Dolphin* [MLSys'16] + adapting learning rate by update staleness in *DynSGD* [SIGMOD'17]

☞ Adaptation is always key, regardless of abstraction level
Recovery: General Techniques

Depend on the level of abstraction:

- **Program is a black box**
  - Checkpointing VM state in reliable/redundant storage

- **Program is a workflow with clear input/output for each unit**
  - Write input/output to reliable storage + rerun failed units, e.g., *Hadoop/MapReduce*
  - Intermediate results can be in memory and lost + recover using lineage *Spark RDD [NSDI'12]*

- **Program is specified declaratively, DB-style**
  - Finer-grained lineage-based recovery using knowledge of operators + intelligent selective checkpointing *Cümülön [PVLDB'15]*
Recovery: Algorithm-Specific

- Many ML algorithms can tolerate missing input or errors by design
  - Instead of recovering to a state where as if failures never occurred, convert failures into “soft” ones that algorithms can handle themselves

- **Example: distributed batch gradient descent**

  Narayanamurthy+ *(REEF)* [BigLearn'13]

  - In an iteration, if a task fails to calculate the contribution from one partition of data, simply use an approximation (from the previous iteration)
  - Algorithm still converges

  Generalized to user-defined, algorithm-specific “compensations”

  Schelter+ [CIKM'13]
Working with Markets

- “On-demand” (regular) instances: fixed price, guaranteed
- “Spot” instances: availability/price vary over time; e.g; on Amazon:
  - You set a bid price, and get instances if bid price \( \geq \) market price
  - You pay market price (@hour start), by hours
  - You lose the instances if market price rises above your bid, but your last hour will be free
- Price can depend on machine type, region, and time

☞ How do we leverage markets effectively?
  - Pop quiz: would you ever bid higher than the fixed price?
  - Yes! Less chance of losing them, yet still lower cost on average
Working with Markets: Techniques

- **Diversify your portfolio:** consider instances with different types, across regions
  - If one market is too expensive, turn to others, e.g., *Dyna* [TCC'16]
  - A heterogeneous cluster may be best for mixed workloads, e.g., *Zhang*+ [PER'15]

- **Minimizing expected cost is often not enough; need to control risk**
  - Model the market to quantify uncertainty, e.g., *Cümülön(-D)* [PVLDB'15,'17]

- **Zafer**+ [Cloud'12] squeezes entire execution on spots in an hour; retries with a higher bid price if you lose them
  - Losing spots within an hour incurs no cost with Amazon

- **Dyna** [TCC'16] tries faster spots before falling back to on-demand
  - But only if doing so improves the execution time distribution

- **Cümülön** [PVLDB'15] picks the optimal mix of spot/on-demand instances
  - To minimize expected cost while meeting deadline/budget with high probability
  - Recovers and re-optimizes if spots are lost

- **Cümülön-D** [PVLDB'17] adapts proactively dynamically and proactively
  - Bids/terminates as needed, based on execution progress and market condition
  - Solves the optimization problem as a Markov Decision Process (MDP) and pre-compiles a “cookbook” to apply at run time
Summary

- Large-scale ML is increasingly being done in a cloud
- Challenges of elasticity are not unique to DB & ML

- Lots of uncertainty, but adaption & stochastic modeling come to rescue

- Different levels of abstraction lead to different opportunities—declarative (DB-style) ML enables smarter, more effective solutions

References for Part 6: Resource Elasticity

- **Cümülön** [PVLDB'15] Huang et al. “Cümülön: matrix-based data analytics in the cloud with spot instances.” PVLDB 2015
- **Cümülön-D** [PVLDB'17] Huang & Yang. “Cümülön-D: data analytics in a dynamic spot market.” PVLDB 2017
- **Parameter Server** [ODSI'14] Li et al. “Scaling Distributed Machine Learning with the Parameter Server.” OSDI 2014
- **SystemML** [VLDB'16] Boehm et al. “SystemML: Declarative machine learning on Spark.” PVLDB 9(13), 2016
Part 7: ML Lifecycle Systems

Arun Kumar

UC San Diego
La Jolla, CA, USA

SIGMOD 2017
Overview: ML Lifecycle Issues

Data sourcing

Feature engineering and model selection

Model serving
Tighter loop between inference and learning

Model management
Feature Engineering

Q: What is feature engineering (FE)?

The process of obtaining a formal representation of the data-generating process as structured signals (features) for an ML model.

Q: Why is it important from a data management perspective?

High-quality features are the “secret sauce” of applied ML.

FE operations are basically data transformations!

Often “brushed under the carpet” by ML community.

Q: What sort of operations constitute feature engineering?

Depends on the data type!

Structured data: Whitening, feature selection/ranking, joins, PCA, etc.

Text: Bag-of-words, Parsing-based, Domain-specific, Word2Vec, etc.

Deep CNNs and RNNs for images, audio, video, time series, etc.
Feature Engineering Systems

**Feature selection:**
Obtain a subset of features to improve accuracy and/or interpretability

**Columbus [SIGMOD’14]:**
Often not a single algorithm but a human-in-the-loop dialogue process
Data scientist explores multiple subsets based on domain insights

Understanding customer churn

<table>
<thead>
<tr>
<th>CustID</th>
<th>Churn?</th>
<th>Age</th>
<th>Income</th>
<th>Gender</th>
<th>City</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

*Evaluate error with all features in chosen set*
*Drop demographic features and re-evaluate*
*Add Gender back in and so on …*

A few such common steps encoded as “declarative” ops in DSL
Impl. on top of R/Python; optimizing code-gen middleware
Batching/materialization; QR decomposition; coresets; warm start
Feature Engineering Systems

Treating FE as a dataflow-oriented process; DB-style optimizations:
  Brainwash [CIDR’13] / DeepDive [DataEng’14]
  Workflows of UDFs; feature recommendations
KeystoneML [ICDE’17]
  Alternative phy. impl. of solvers; cost-based op. selection

Reducing amount of work for feature coding/evaluation:
  Zombie [ICDE’16]
  Index structure to sub select relevant data; bandit techniques

Applying learning theory to skip features and help with sourcing tables:
  Hamlet [SIGMOD’16]

More open questions remain in systematizing feature engineering
Overview: ML Lifecycle Issues

Data sourcing
Feature engineering and model selection
Model serving
Tighter loop between inference and learning
Model management
Model Selection

Q: What is model selection (MS)?

The process of obtaining a prediction function to capture a data-generating process using data generated by that process

MSMS [SIGMODRec’15]
Model Selection Triple (MST)
(FE, AS, PT)

FE: Feature Engineering
AS: Algorithm Selection
PT: (Hyper-)Parameter Tuning

Q: Why is it important from a data management perspective?

FE, AS, and PT often access the dataset (or subsets) repeatedly.
A lot of opportunities to improve efficiency with DB-style opt.
FE, AS, and PT are inter-dependent and together constitute MS
MSMS [SIGMODRec’15]  Model Selection Triple (MST): (FE, AS, PT)

Data scientists typically think at a higher level of abstraction

Automation essentially groups MSTs en masse

MS abstractions can help capture intermediate points
Model Selection Process

MSMS [SIGMODRec’15]  Model SelectionTriple (MST): (FE, AS, PT)

1. **“Declarative” interfaces**
   - Group a set of “logically related” MSTs

\[
\{\text{FE1, FE2}\} \times \text{AS1} \times \{\text{PT1, PT2}\} \ldots
\]

2. **Code Generation**
   - Evaluate models using system
   - Optimization
   - Manage results

3. **Provenance management**
   - Next iteration

Many old and recent MS abstractions can be “retro-fitted”
Several new MS abstractions can be introduced to co-exist
Model Selection Management Systems (MSMS)

**MSMS [SIGMODRec’15]**

**The Higher Layers: Declarative Interfaces (some in hindsight!)**

**Autotuned functions**
- E.g., glmnet() in R
  - $\{FE\} \times AS \times \{PT\}$

**Columbus**
- E.g., StepAdd()
  - $\{FE\} \times AS \times PT$

**MLBase**
- E.g., doClassify()
  - $FE \times \{AS1 \times \{PT\}, AS2 \times \{PT\}\}$

**New Abstractions**
- $\{FE\} \times \{AS \times PT\}$, ...

**The Narrow Waist:**
A set of logically related Model Selection Triples (MST)

**The Lower Layers: Optimized Implementations**

- In-memory
- In-RDBMS
- Spark
- Others
Model Selection Systems

Automation of AS and PT search with pre-defined search space:

**MLbase [CIDR’13] / TuPAQ [SoCC’15]**
- Declarative ML tasks (e.g., “DoClassify”); fixed set of algorithms
- Data batching; bandit techniques for explore-exploit search

**Hemingway [MLSys’16]**
- Joint AS and cluster sizing for optimization algorithms
- Observe-and-adapt approach for convergence properties

DB-style optimizations for PT and general meta-learning:

**SystemML [ICDE’15]; GLADE [DanaC’12]**

Many open questions remain on optimizing/improving model selection
- Interactions of PT with AS and FE
- Exploiting redundancy across and within MSTs; cost models
- Incorporating constraint/approximations and visualizations, etc.
Overview: ML Lifecycle Issues

Data Scientist/ML Engineer

Data sourcing
Feature engineering and model selection

Model serving
Tighter loop between inference and learning

Model management
Model Management Systems

Q: What is model management?
Treating trained models as data themselves (store, query, debug, etc.)

Integrating ML models with SQL querying: LongView [CIDR’11]

Iterative ML debugging: MindTagger [VLDB’15], PALM [HILDA’17]

Specialized storage engines and custom optimizations:
  ModelHub [ICDE’17]
    Versioned storage/retrieval of CNNs (sets of float matrices)
    Optimizations for reducing storage footprint

Many open questions on managing large space of MSTs, especially for large models (DNNs/trees); ML provenance and debugging
Other ML Lifecycle Issues

Model Serving: High-throughput/low-latency inference/re-learning
- MacroBase [SIGMOD’17]
- Clipper [NSDI’17] / Velox [CIDR’15]
Integrating data-driven applications with reinforcement learning

Data Sourcing: Modeling labeling process; ML+cleaning; ML+pricing
- Snorkel [NIPS’16]
- ActiveClean [VLDB’16]
- Model-Based Pricing [DEEM’17]

Interactive Model Building: Human-in-the-loop interfaces
- Ava [CIDR’17]
- Vizdom [VLDB’15]
References: Part 7

ActiveClean [VLDB’16]: ActiveClean: Interactive Data Cleaning For Statistical Modeling
Ava [CIDR’17]: Ava: From Data to Insights Through Conversation
Brainwash [CIDR’13]: Brainwash: A Data System for Feature Engineering
Clipper [NSDI’17]: Clipper: A Low-Latency Online Prediction Serving System
Hamlet [SIGMOD’16]: To Join or Not to Join? Thinking Twice about Joins before Feature Selection
Hemingway [MLSys’16]: Hemingway: Modeling Distributed Optimization Algorithms
KeystoneML [ICDE’17]: KeystoneML: Optimizing Pipelines for Large-Scale Advanced Analytics
Longview [CIDR’11]: The Case for Predictive Database Systems: Opportunities and Challenges
MacroBase [SIGMOD’17]: MacroBase: Prioritizing Attention in Fast Data
MindTagger [VLDB’15]: MindTagger: A Demonstration of Data Labeling in Knowledge Base Construction
MLbase [CIDR’13]: MLbase: A Distributed Machine-learning System
Model-Based Pricing [DEEM’17]: Model-based Pricing: Do Not Pay for More than What You Learn!
ModelDB [HILDA’16]: MODELDB: A System for Machine Learning Model Management
ModelHub [ICDE’17]: ModelHub: Towards Unified Data and Lifecycle Management for Deep Learning
MSMS [SIGMODRec’15]: Model Selection Management Systems: The Next Frontier of Advanced Analytics
Palm [HILDA’17]: PALM: Machine Learning Explanations For Iterative Debugging
Snorkel [NIPS’16]: Data Programming: Creating Large Training Sets, Quickly
SystemML [ICDE’15]: Efficient Sample Generation for Scalable Meta Learning
TuPAQ [SoCC’15]: Automating Model Search for Large Scale Machine Learning
Velox [CIDR’15]: The Missing Piece in Complex Analytics: Low Latency, Scalable Model Management and Serving with Velox
Vizdom [VLDB’15]: Vizdom: Interactive Analytics through Pen and Touch
Zombie [ICDE’16]: Input Selection for Fast Feature Engineering
Part 8: Open Problems and Conclusions

Arun Kumar
UC San Diego
San Diego, CA, USA

Matthias Boehm
IBM Research – Almaden
San Jose, CA, USA

Jun Yang
Duke University
Durham, NC, USA

SIGMOD 2017
Open Problems: Optimizer and Runtime

- **#1 Size and Sparsity Estimation**
  - Fundamental building block for cost comparisons / valid plan generation
  - Issues: function calls, UDFs, data-dependent operators, changing sizes

- **#2 Convergence Estimation**
  - Number of iterations until convergence unknown
  - Required for cost comparisons and progress estimation

- **#3 Adaptive Query Processing and Storage**
  - Unknown or changing workloads → adaptive query processing
  - Currently limited to inter-DAG recompilation and expression optimization

- **#4 Automatic Rewrites and Operator Fusion**
  - Huge potential for simplification rewrites and operator fusion
  - Challenging in presence of new access methods, compression, etc.

- **#5 Special Value Handling**
  - Special values such as NaN, INF, -0 ignored by most systems → incorrect results
  - Support these special values w/o sacrificing performance
Open Problems: End-to-End Lifecycle

- **#6 Integrating Relational and Linear Algebra**
  - Seamless optimizer / runtime integration in holistic framework
  - Including data transformations, training and prediction

- **#7 Seamless Feature Engineering and Model Selection**
  - (Semi-)automating feature engineering and model selection
  - Including abstractions, meta-algorithms, and system architectures

- **#8 ML System Benchmarks**
  - Existing benchmarks limited to ML tasks in terms of reference implementations of large-scale ML libraries or SQL-centric workloads
  - Broader range of benchmarks at various abstraction levels
Conclusions

Summary
- Compelling arguments for integrating ML $\rightarrow$ DB and DB $\rightarrow$ ML
- ML in data systems, DB-inspired ML systems, ML lifecycle systems

#1 Existing Work to Build Upon
- Awareness of existing systems and techniques
- Survey of effective optimization and runtime techniques

#2 Where the Data Management Community Can Help
- Integrating ML into existing data systems
- Optimizer and runtime techniques for large-scale ML systems
- Tools and systems to simplify/improve the end-to-end ML lifecycle
  ➔ Many open technical problems