VLDB2012 istanbul

The MADlib Analytics Library

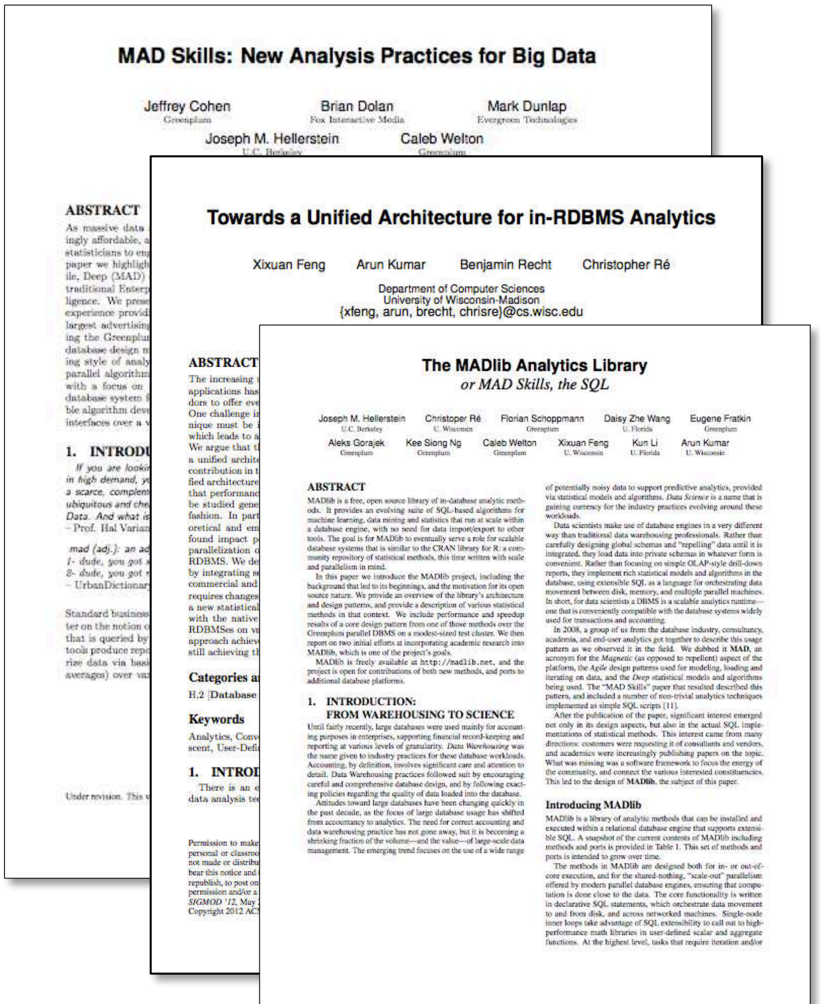
or MAD Skills, the SQL

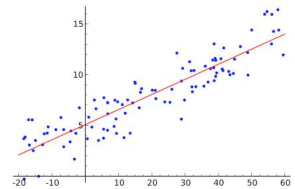
Caleb Welton Florian Schoppmann Christopher Ré

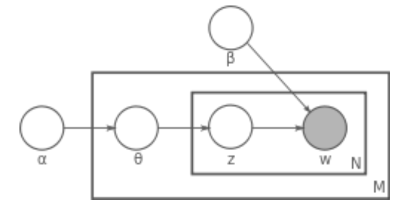
  

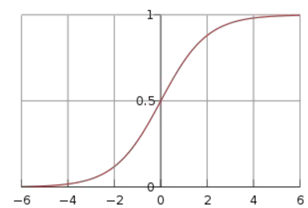
MADlib

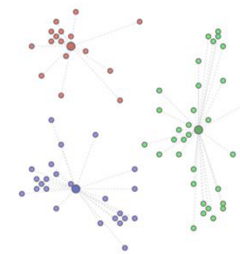
Scalable Machine Learning for BigData

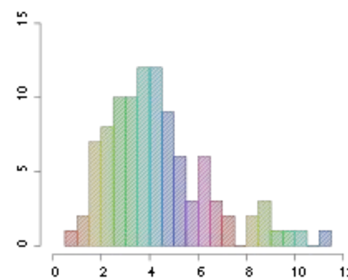


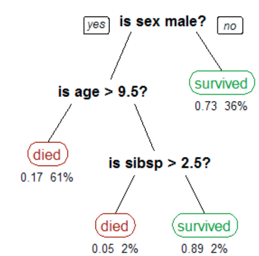




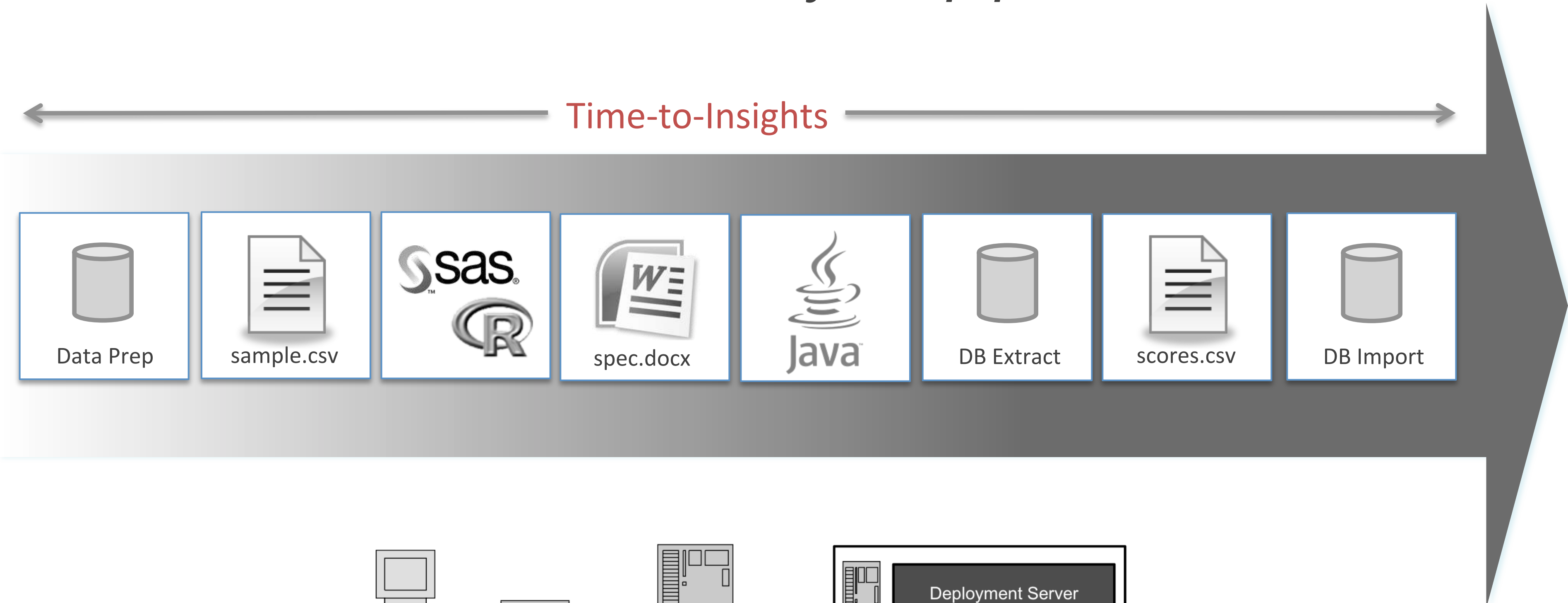


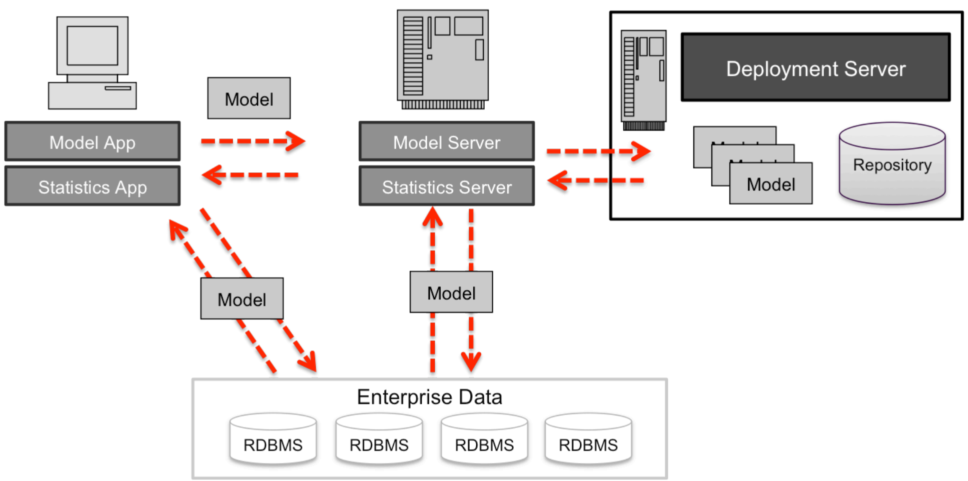




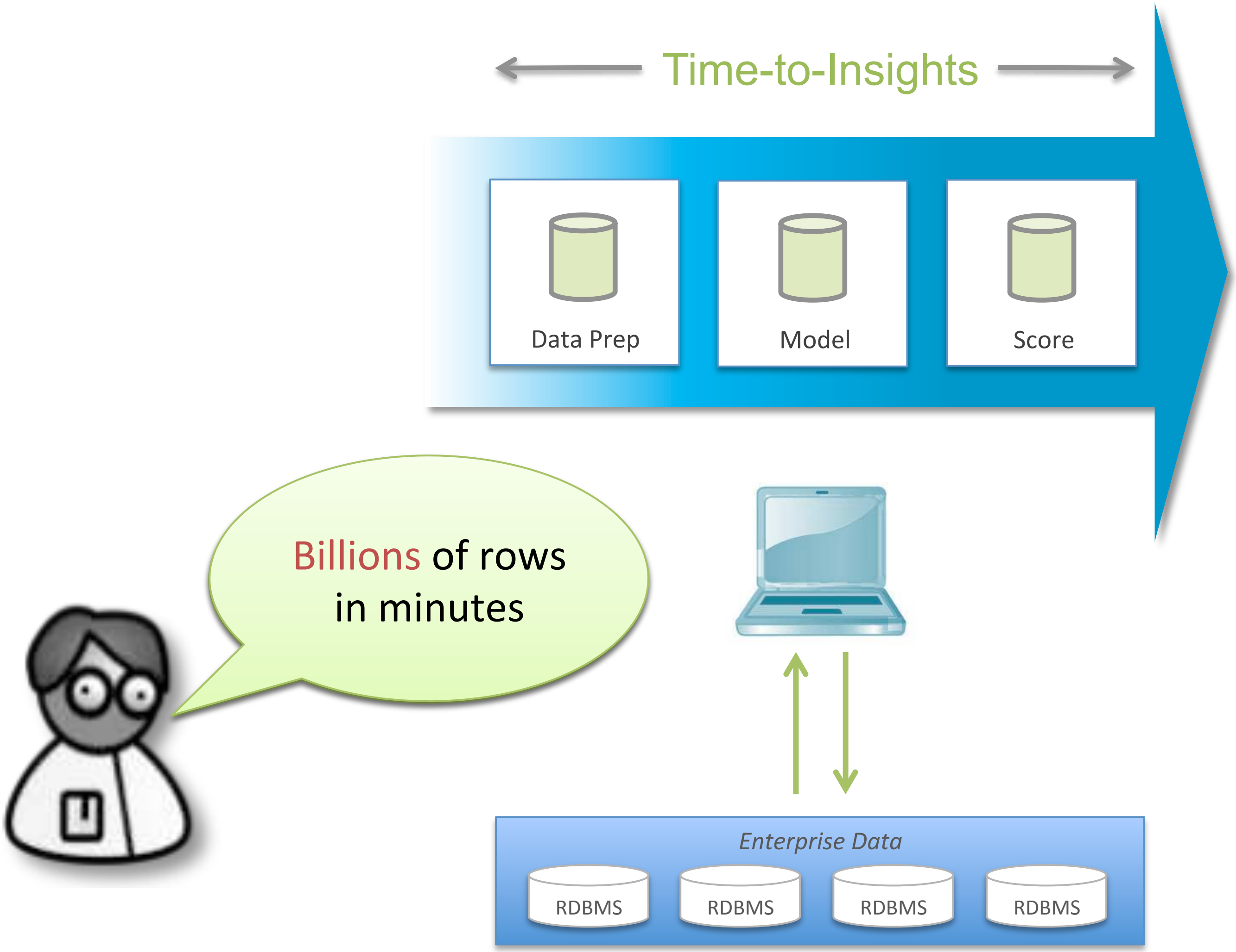


Traditional analytics pipeline





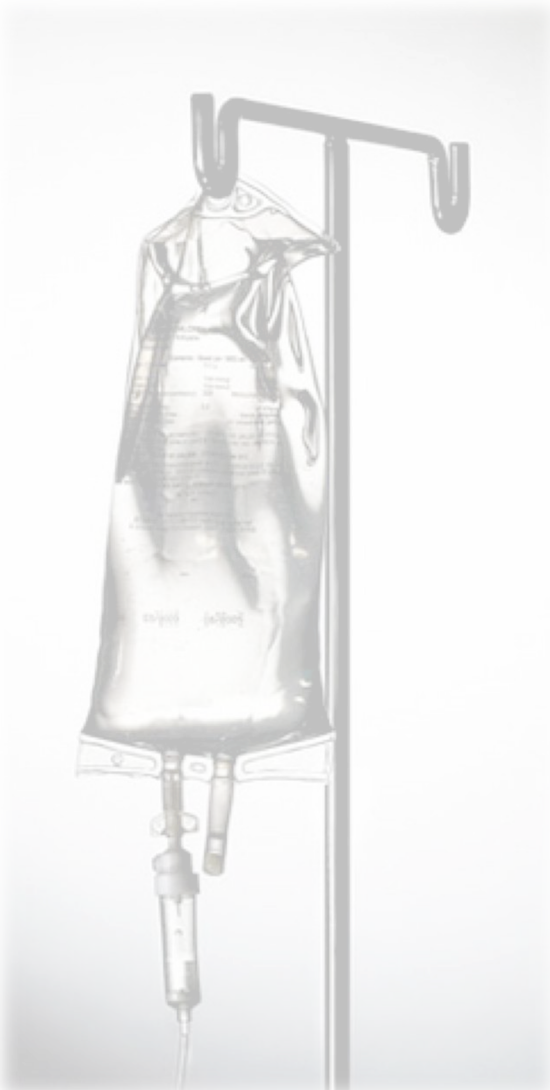
The MAD approach



MADlib in *Action*

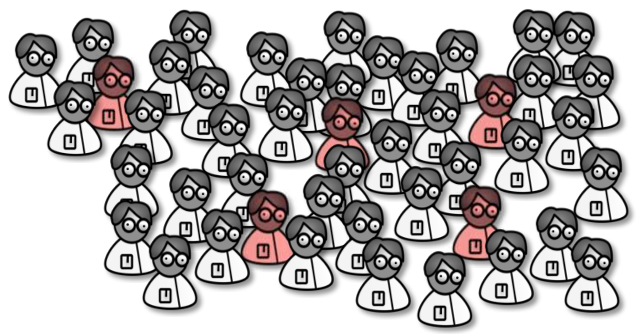
Hospital Admittance Case Study

MADlib in *Action*



Step 1:

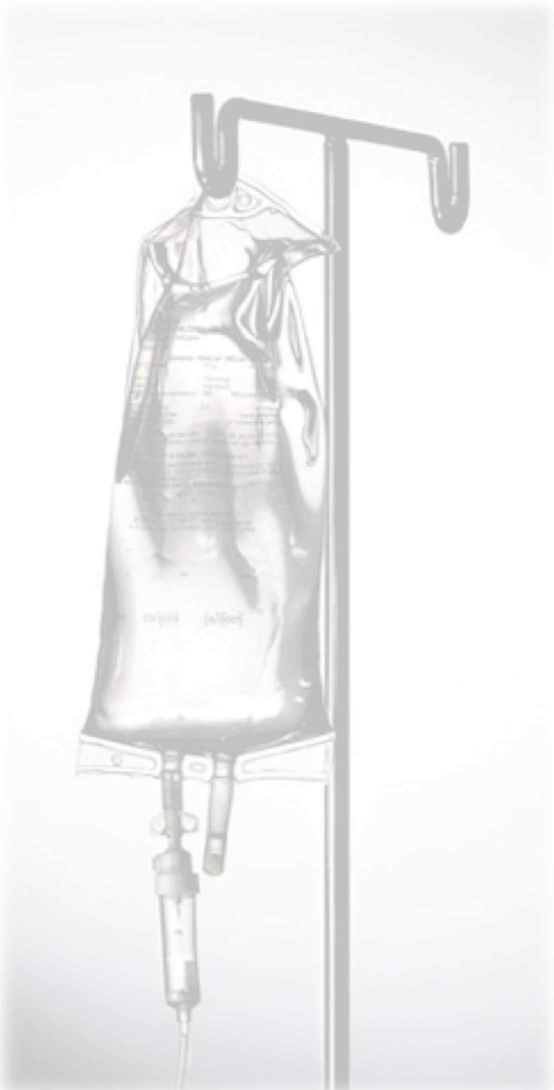
- Identify high risk patients



Goal:

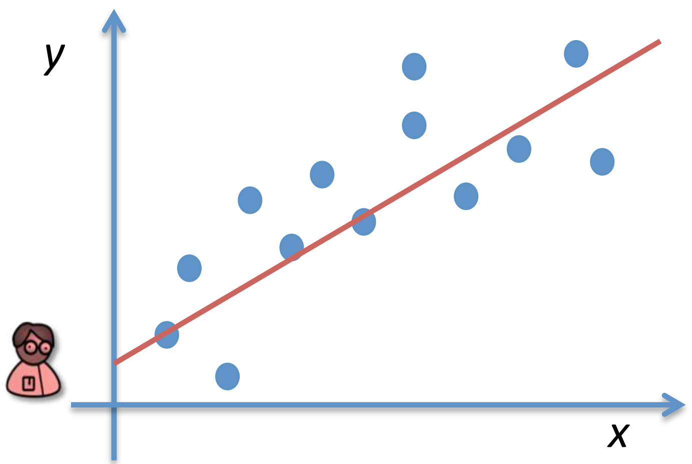
- High risk patients will be eligible for early admittance and be administered preemptive antibiotics

MADlib in *Action*



Step 2:

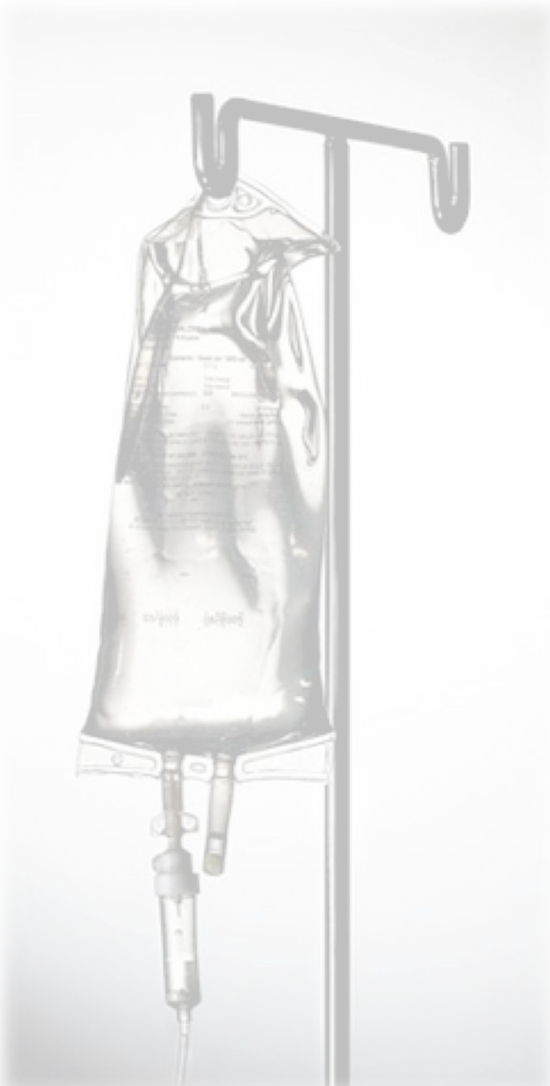
- Build cost model for treatment



Goal:

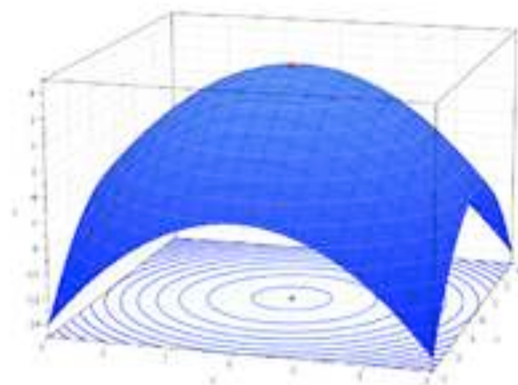
- Predict expected cost of treatment
- With and without early admittance.

MADlib in *Action*



Step 3:

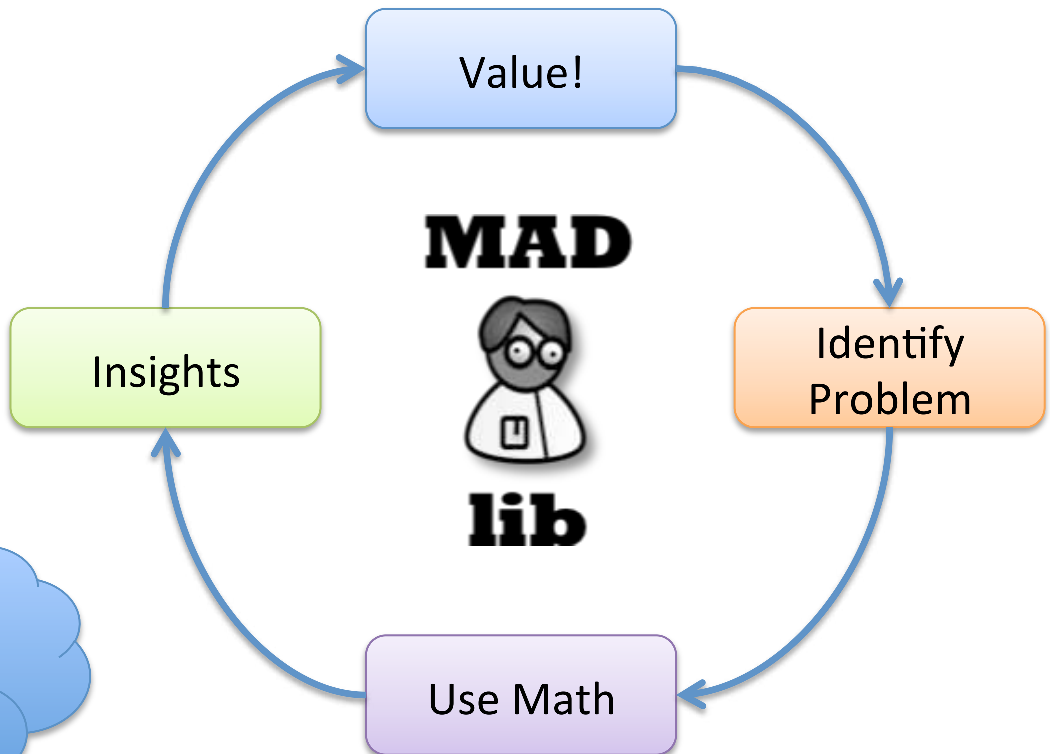
- Optimize early admittance based on risk and cost model

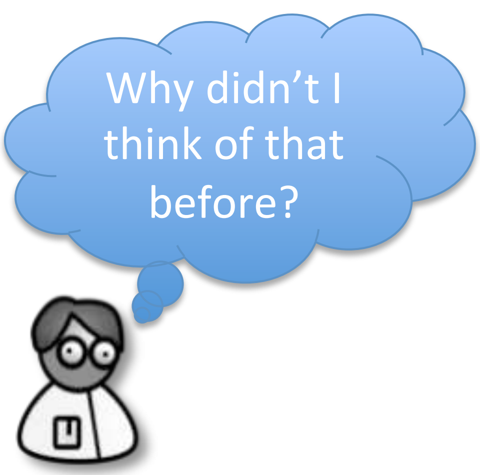


Goal:

- Overall hospital costs will be minimized and patients will receive better care.

MADlib cycle of success





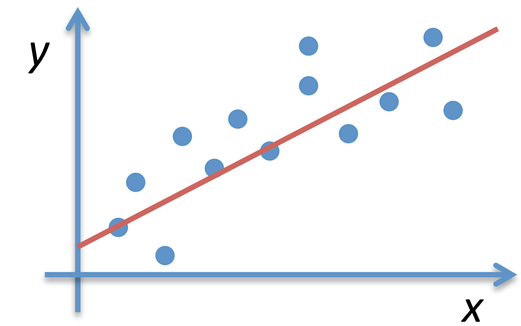
The MADlib Vision

- Academic and industry contributions
- Think of "CRAN for databases"
  - Repository of open-source ML algorithms
  - This time with data parallelism in mind
- Open-Source Framework

BSD License

Eigen

Simple Example: Ordinary Least Squares



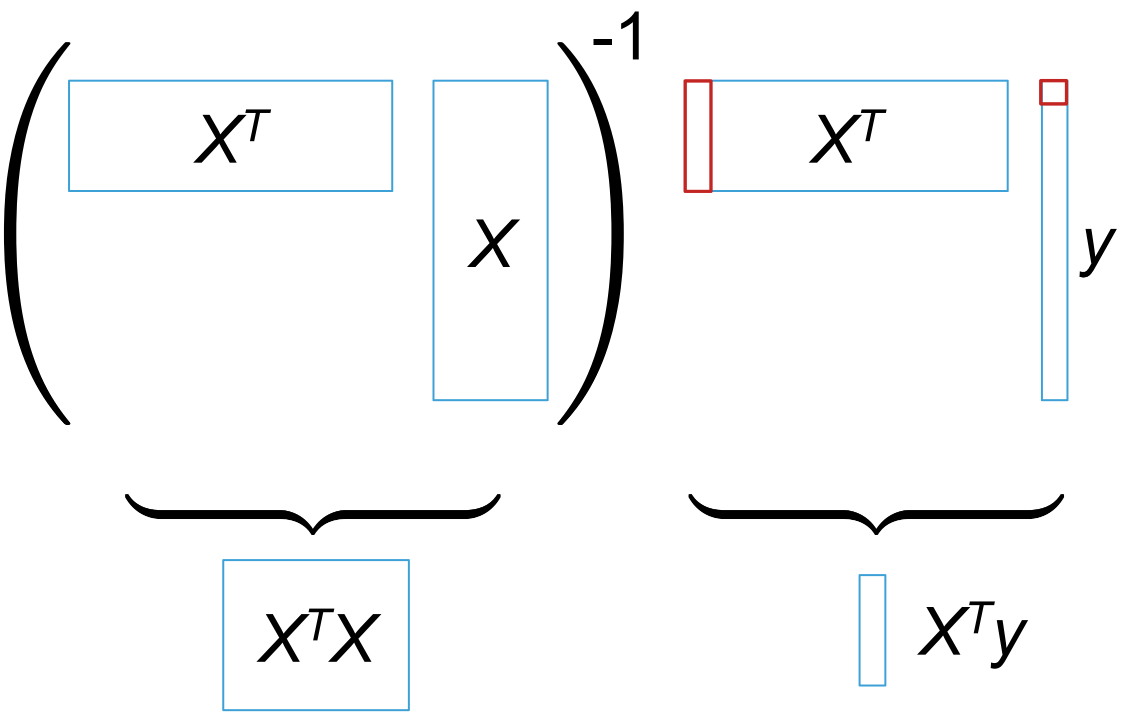

```
# SELECT y, x[1] AS x1, x[2] AS x2 FROM data
    y    |   x1   | x2
---------+--------+------
  10.14  |      0 | 0.3
  11.93  |   0.69 | 0.6
  13.57  |    1.1 | 0.9
  14.17  |   1.39 | 1.2
  15.25  |   1.61 | 1.5
  16.15  |   1.79 | 1.8
```

y *X*

```
# SELECT (linregr(y, x)).* FROM data;
-[ RECORD 1 ]+-------------------------
coef         | {1.7307,2.2428}
r2           | 0.9475
std_err      | {0.3258,0.0533}
t_stats      | {5.3127,42.0640}
p_values     | {6.7681e-07,4.4409e-16}
condition_no | 169.5093
```

Linear Algebra in the Database

$$\hat{\beta} = (X^T X)^{-1} X^T \boldsymbol{y}$$



$$\sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i^T \qquad \sum_{i=1}^{n} \boldsymbol{x}_i y_i$$

Basic Building Block: User-Defined Aggregates

$\boldsymbol{x}$	y
(1,0,3,…,5)	3
(-2,4,5,…,2)	2
…	…

Aggregation phase 1 on each node:

1. Initialize: $(A,\boldsymbol{b}) = (0,0)$
2. Transition for all rows:

   $$(A,\boldsymbol{b}) = (A,\boldsymbol{b}) \underset{\text{reduce}}{+} \underbrace{(\boldsymbol{x}\cdot\boldsymbol{x}^T, \boldsymbol{x}\cdot y)}_{\text{map}}$$

3. Send $(A,\boldsymbol{b})$

$(A,\boldsymbol{b})$
…

Aggregation phase 2 on master node:

1. Merge: $(\bar{A},\bar{\boldsymbol{b}}) = (\bar{A},\bar{\boldsymbol{b}}) + (A,\boldsymbol{b})$
2. Finalize: $\hat{\beta} = \text{solve}(\bar{A},\bar{\boldsymbol{b}}) = \bar{A}^{-1}\cdot\bar{\boldsymbol{b}}$

Problem solved?

No – not yet.

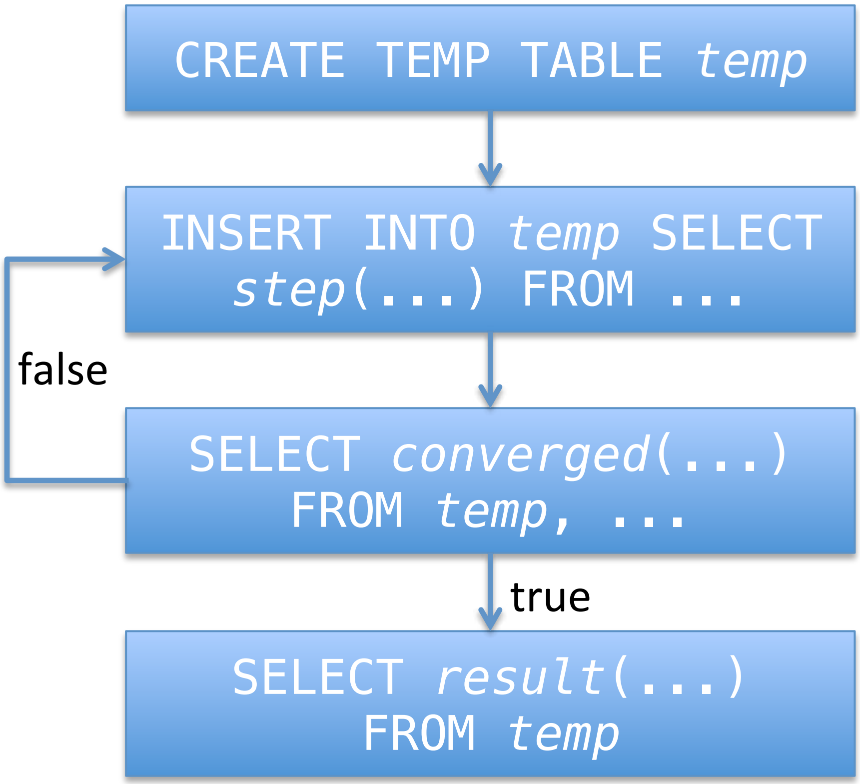
ML Algorithms Based on SQL?

- Four Representative Challenges
  1. Lack of portable multi-pass iterations
  2. Roots in first-order logic
  3. Lack of language support for linear algebra
  4. Extensible SQL limited to small working sets

Need:
- Abstraction Layers
- A few compromises for user interface

1. Lack of portable multi-pass iterations

- WITH RECURSIVE not reliable basis for portability
- User-defined driver functions in Python
  - Outer loops not performance-critical
- Compromise: Different user interface



2. Roots in first-order logic

- Queries need be cognizant of database objects
- Emulate higher-order logic by:
  - dynamic execution of templated SQL
  - abstraction-layer support

```
FunctionHandle dist
    = args[0].getAs<FunctionHandle>();
return dist(x, y);
```

- Example: Distance or kernel functions
- On PostgreSQL, use of type REGPROC

3. Lack of language support for linear algebra

- C++ Abstraction Layer uses Eigen
- (Dense) Vectors and matrices: `DOUBLE PRECISION[]`
- Example:

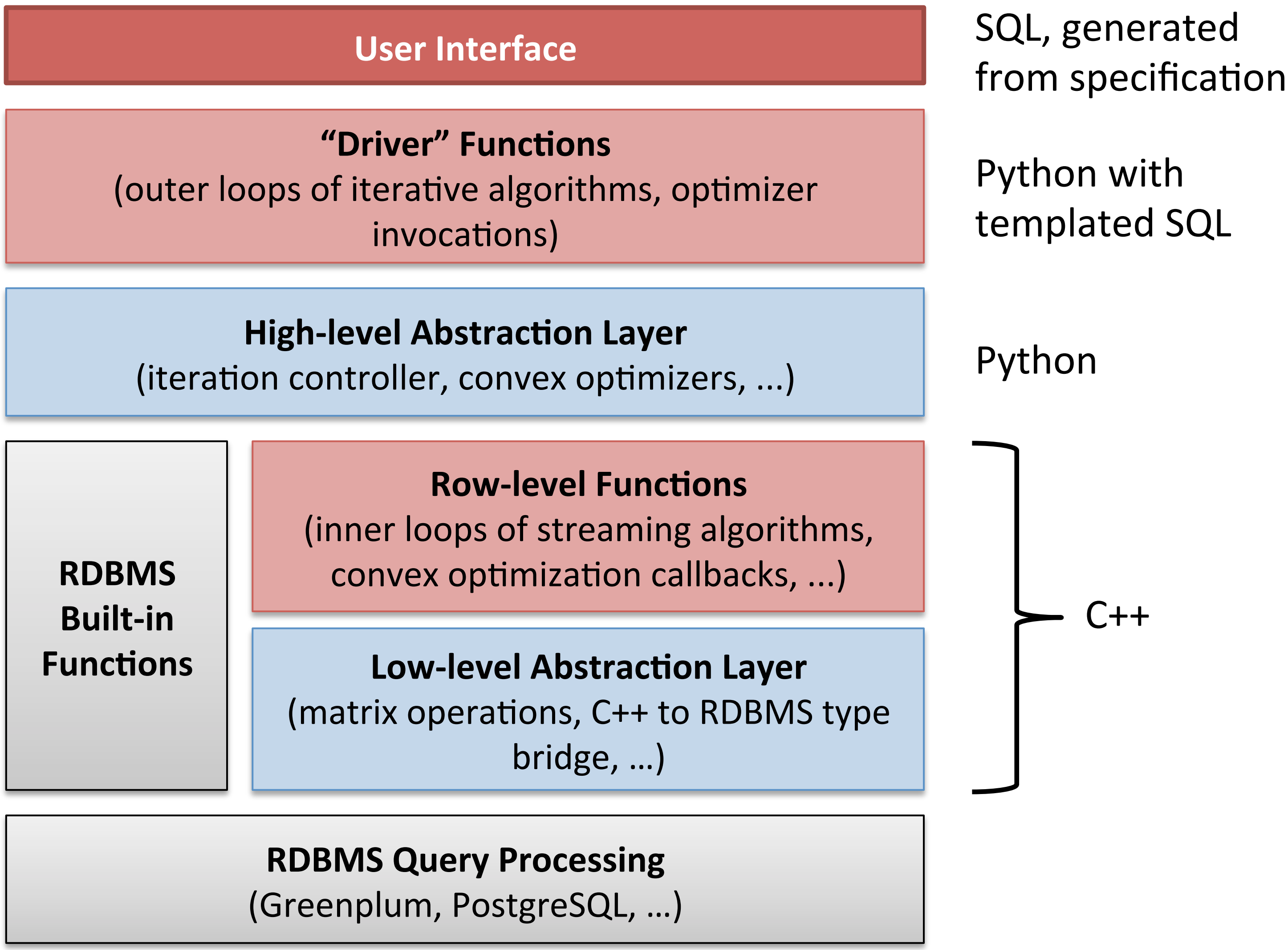
```
AnyType
solve::run(AnyType& args) {
    MappedMatrix A = args[0].getAs<MappedMatrix>();
    MappedColumnVector b = args[1].getAs<MappedColumnVector>();

    MutableMappedColumnVector x = allocateArray<double>(A.cols());
    x = A.colPivHouseholderQr().solve(b);
    return x;
}
```

Performance:
- No unnecessary copying
- No internal type conversion

4. Extensible SQL limited to small working sets

- Tables only portable option for large states
- Access from UDAs slow or impossible
- Example: *k*-means benefits from explicit point-to-centroid assignments
  - Problematic:
    ```
    UPDATE points SET centroid_id =
    closest(state, coords)
    ```
  - Requires own pass
  - Not allowed in subqueries
  - PostgreSQL legacy

MADlib Architecture



Anatomy of an iterative MADlib module

interState = Start(args) — User-defined Function

Repeat — Python Driver Function

    In parallel for each segment:

        intraState = Initialize(interState)

        For each row

            intraState = Transit(intraState, row) — User-defined Aggregate

    For each intraState:

        intraState = Merge(oldIntraState, intraState)

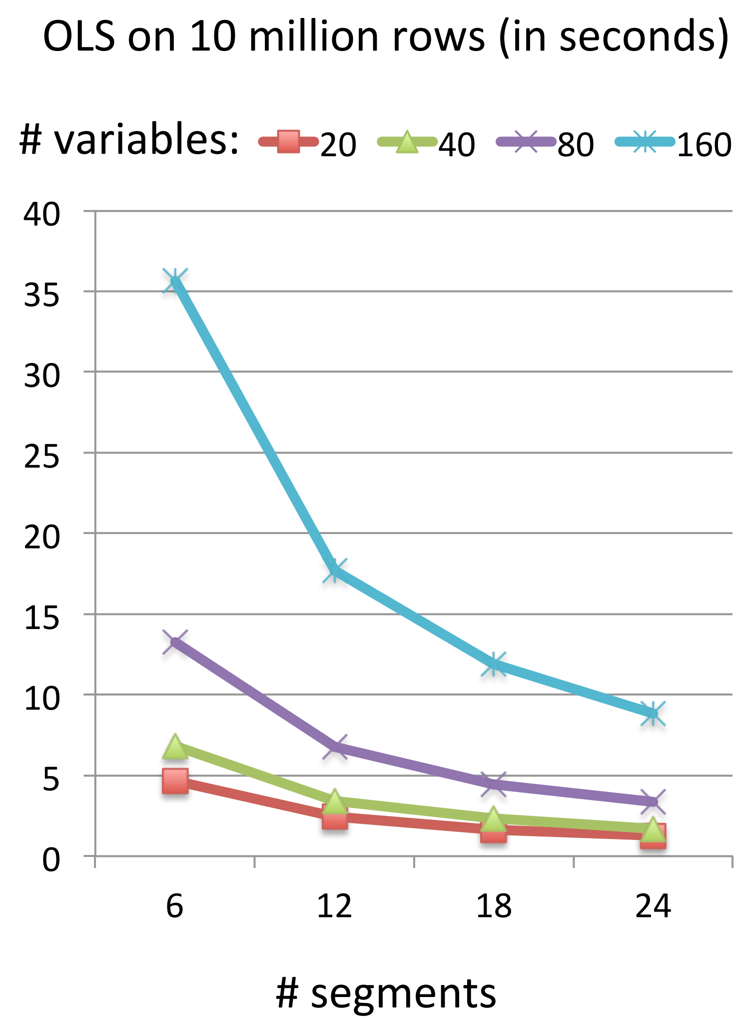
    interState = Finalize(intraState)

Until Converged(interState)

Return End(interState) — User-defined Function

Performance Trends

- Disk I/O is not always the bottleneck
  - Performance tuning is essential
- Overhead for single query very low (fraction of a second)
- Greenplum achieves nearly perfect speedup



Current Modules

Data Modeling

Supervised Learning

- Naive Bayes Classification
- Linear Regression
- Logistic Regression
- Decision Tree
- Random Forest
- Support Vector Machines

Unsupervised Learning

- Association Rules
- k-Means Clustering
- SVD Matrix Factorization
- Parallel Latent Dirichlet Allocation

Descriptive Statistics

Sketch-based Estimators

- CountMin (Cormode-Muthukrishnan)
- FM (Flajolet-Martin)
- MFV (Most Frequent Values)

Profile

Quantile

Inferential Statistics

Hypothesis tests

Support

- Array Operations
- Conjugate Gradient
- Sparse Vectors
- Probability Functions
- Feature Extraction

My MADlib Experience:
A Testimonial.

Christopher Ré, Wisconsin

Refining Ideas and Code

Conversations with GP (and Oracle) lead us to better position our SIGMOD12 paper

Towards a Unified Architecture for in-RDBMS Analytics

Xixuan Feng   Arun Kumar   Benjamin Recht   Christopher Ré

Department of Computer Sciences
University of Wisconsin-Madison
{xfeng, arun, brecht, chrisre}@cs.wisc.edu

ABSTRACT

The increasing use of statistical data analysis in enterprise applications has created an arms race among database vendors to offer ever more sophisticated in-database analytics.

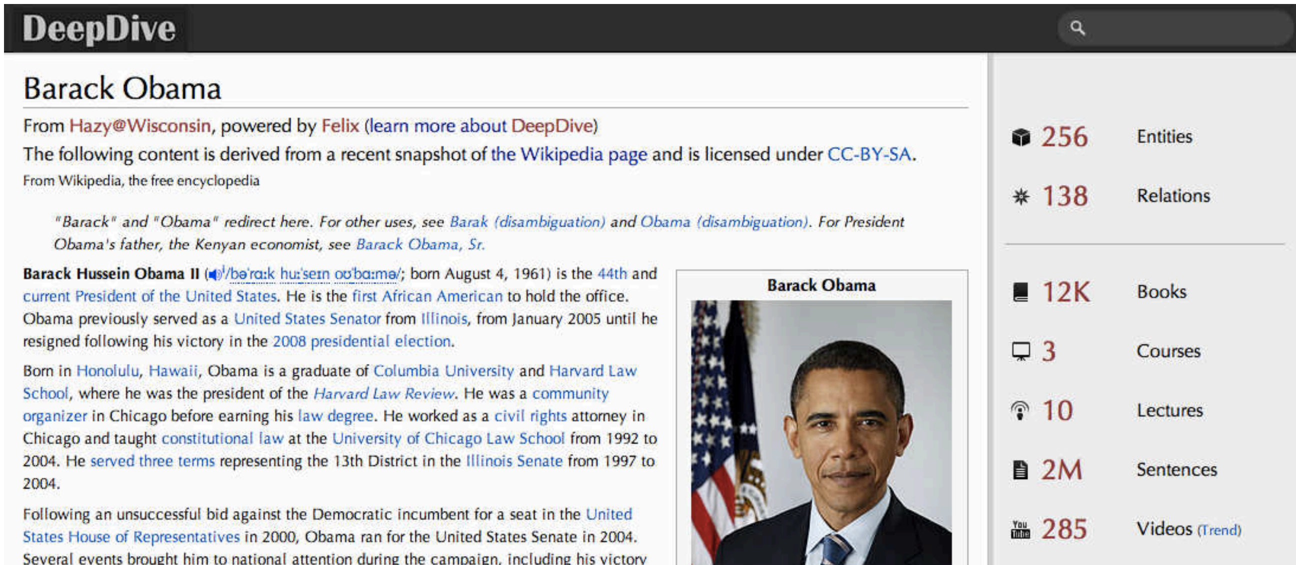
late 1990s and early 2000s, this brought a wave of data mining toolkits into the RDBMS. Several major vendors are again making an effort toward sophisticated in-database analytics with both open source efforts, e.g., the MADlib pla

QA from GP help to transition from *paper* to *deployed code*.

MADlib

MAGNETIC AGILE DEEP

MADlib is Open Source



Enhance Wikipedia with extracted facts from the Web (50+TB of data)

hazy.cs.wisc.edu & www.youtube.com/HazyResearch

Learning & Inference run on (GP or Postgres) + MADLib

Critical: it's free, open, and we can modify it

Testimonial Summary

MADlib

MADlib is open to contributions and open source

http://madlib.net

Caleb Welton
Caleb.Welton@emc.com

Florian Schoppmann
Florian.Schoppmann@emc.com

Christopher Ré
chrisre@cs.wisc.edu

Fork me on GitHub