RAPIDS

cuML: A Library for GPU Accelerated Machine Learning

Onur Yilmaz, Ph.D. | oyilmaz@nvidia.com | Senior ML/DL Scientist and Engineer
Corey Nolet | cnolet@nvidia.com | Data Scientist and Senior Engineer

About Us

Onur Yilmaz, Ph.D.

Senior ML/DL Scientist and Engineer on the RAPIDS cuML team at NVIDIA

Focuses on building single and multi GPU machine learning algorithms to support extreme data loads at light-speed

Ph.D. in computer engineering, focusing on ML for finance.

Corey Nolet

Data Scientist & Senior Engineer on the RAPIDS cuML team at NVIDIA

Focuses on building and scaling machine learning algorithms to support extreme data loads at light-speed

Over a decade experience building massive-scale exploratory data science & real-time analytics platforms for HPC environments in the defense industry

Working towards PhD in Computer Science, focused on unsupervised representation learning

Agenda

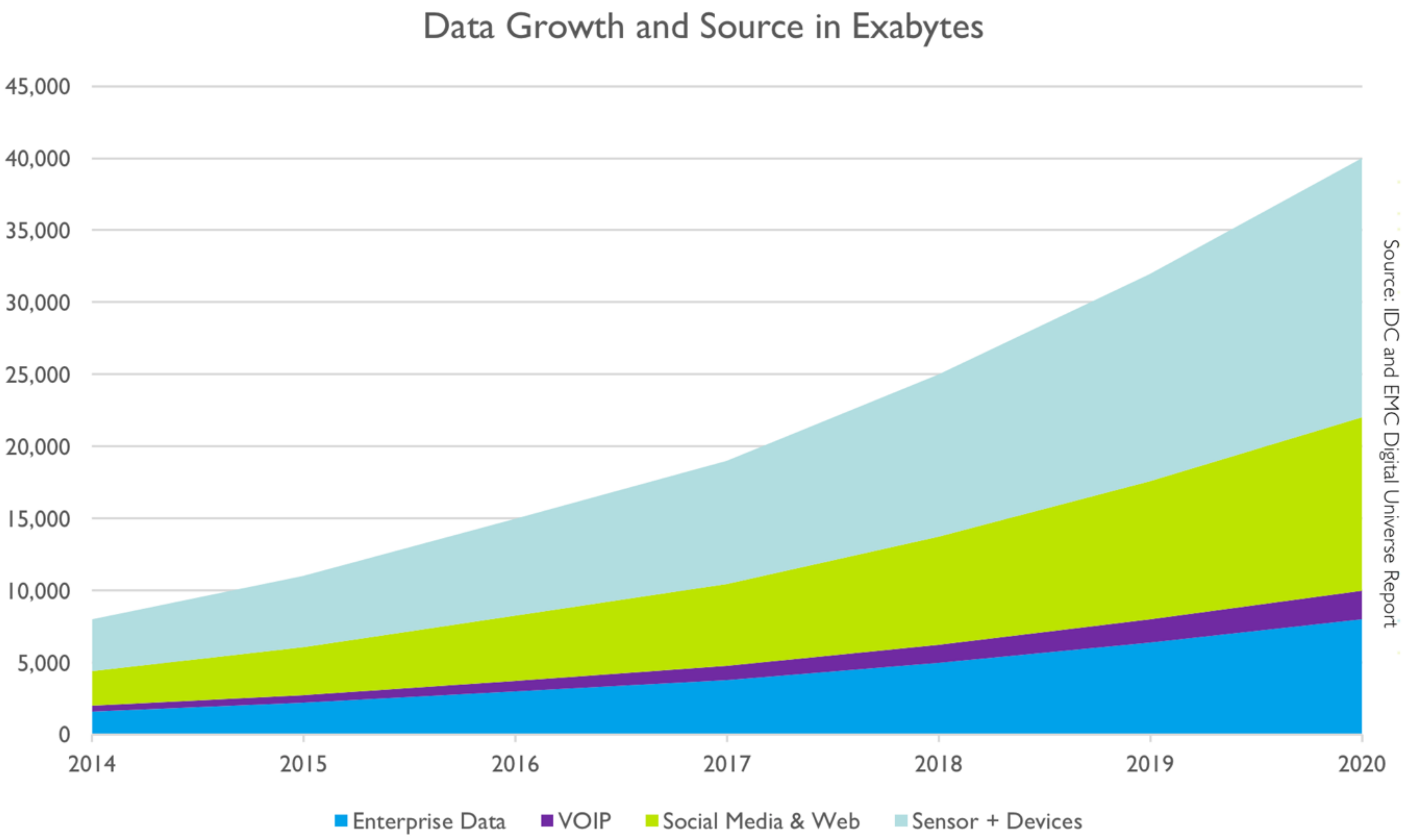
- Introduction to cuML
- Architecture Overview
- cuML Deep Dive
- Benchmarks
- cuML Roadmap

Introduction

“Details are confusing. It is only by selection, by elimination, by emphasis, that we get to the real meaning of things.”

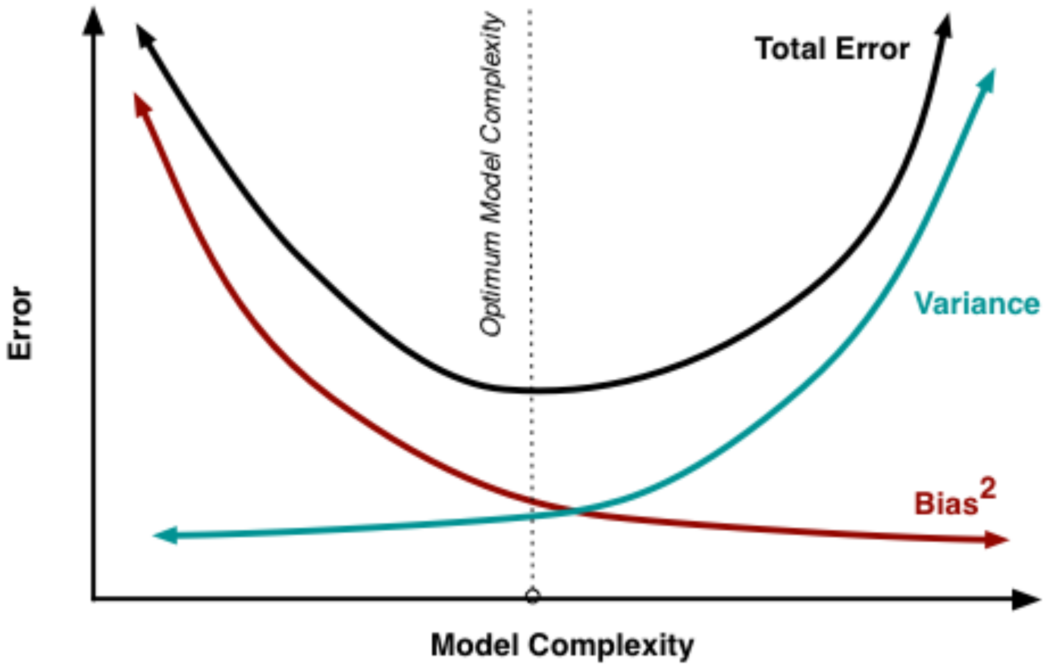
~ Georgia O'Keefe
Mother of American Modernism

Realities of Data



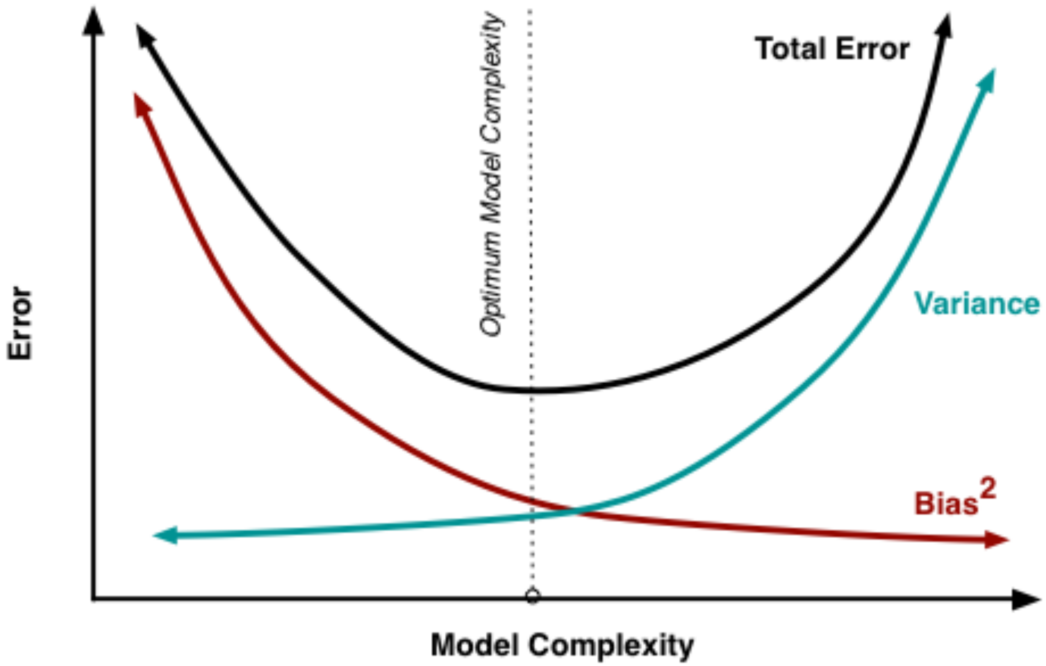
Problem

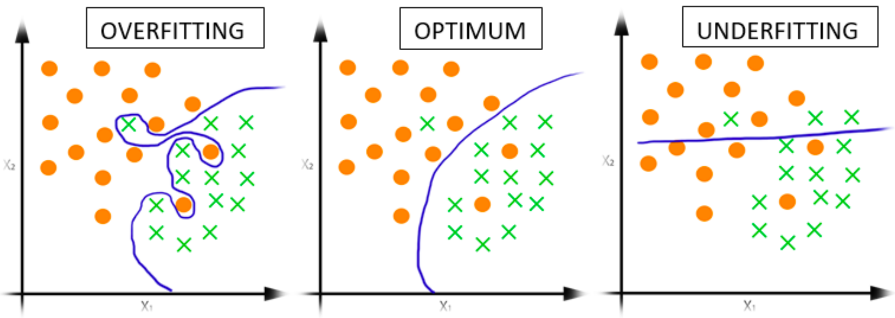
Data sizes continue to grow



Problem

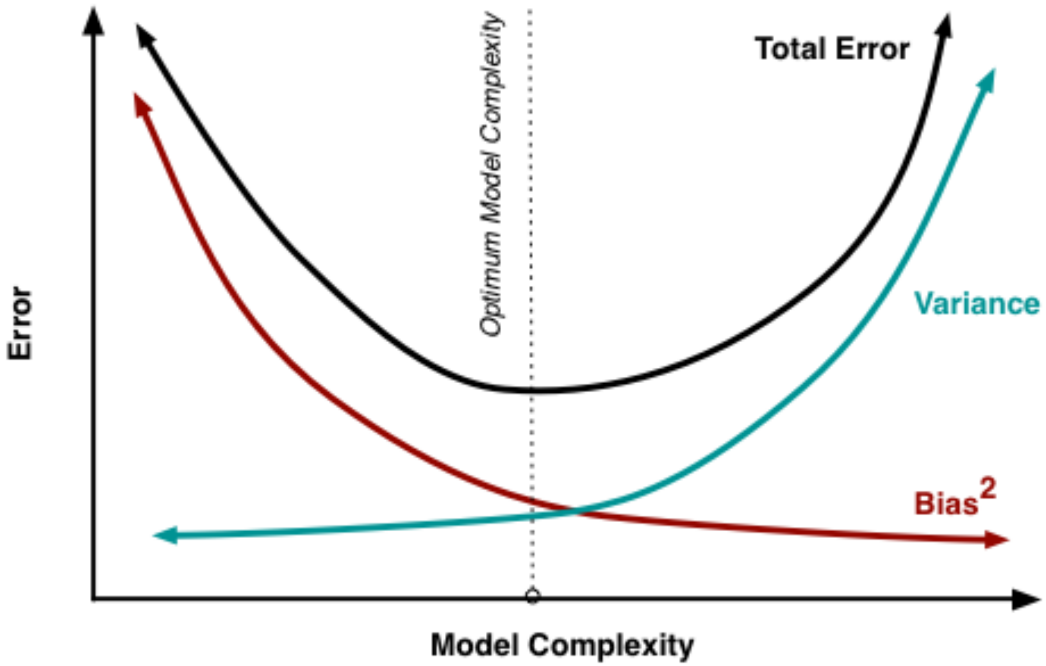
Data sizes continue to grow

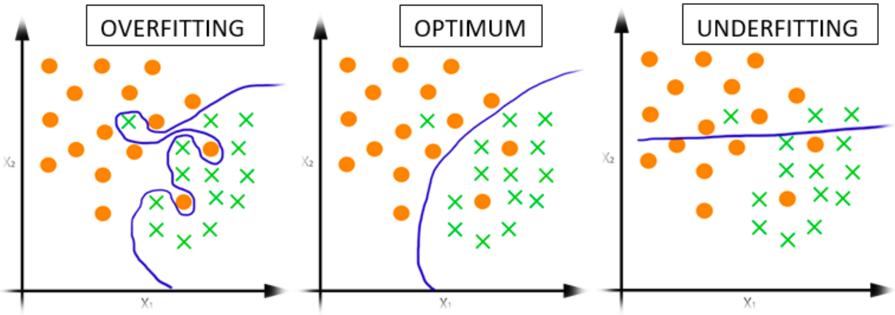




Problem

Data sizes continue to grow





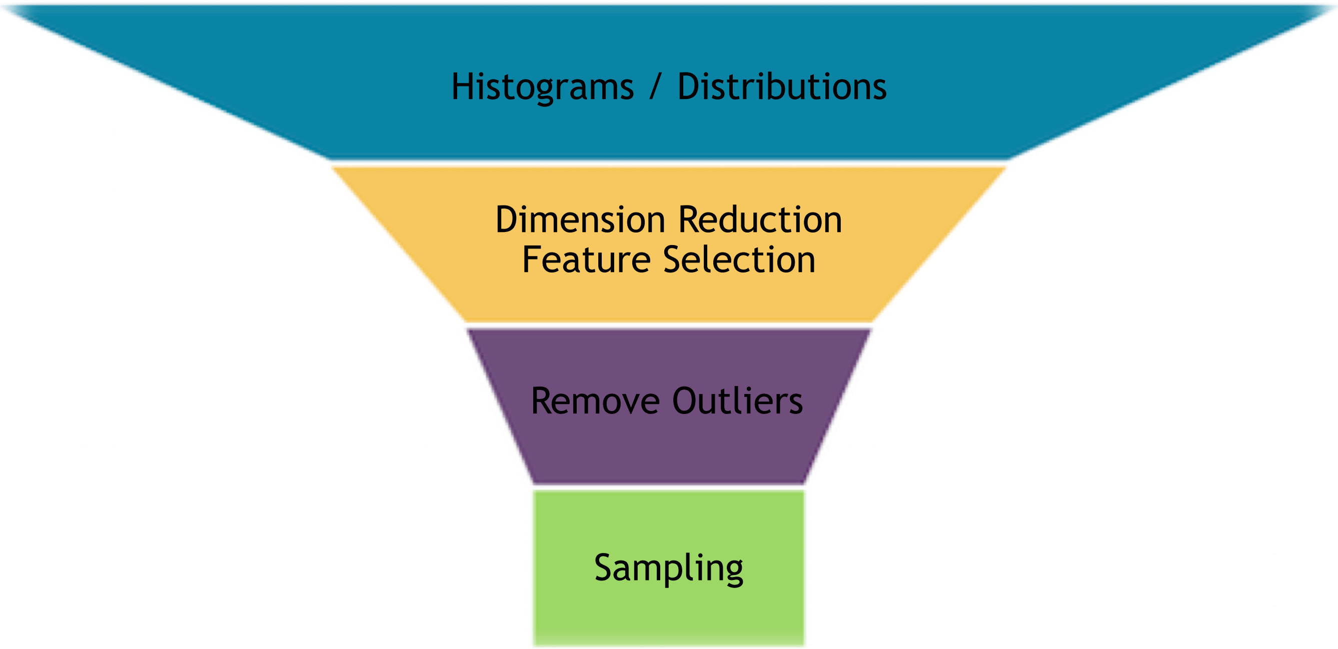
$$\begin{bmatrix} 2 & -1 & 3 \\ 4 & 2 & 1 \\ -6 & -1 & 2 \end{bmatrix}$$

min(variance)

min(bias)

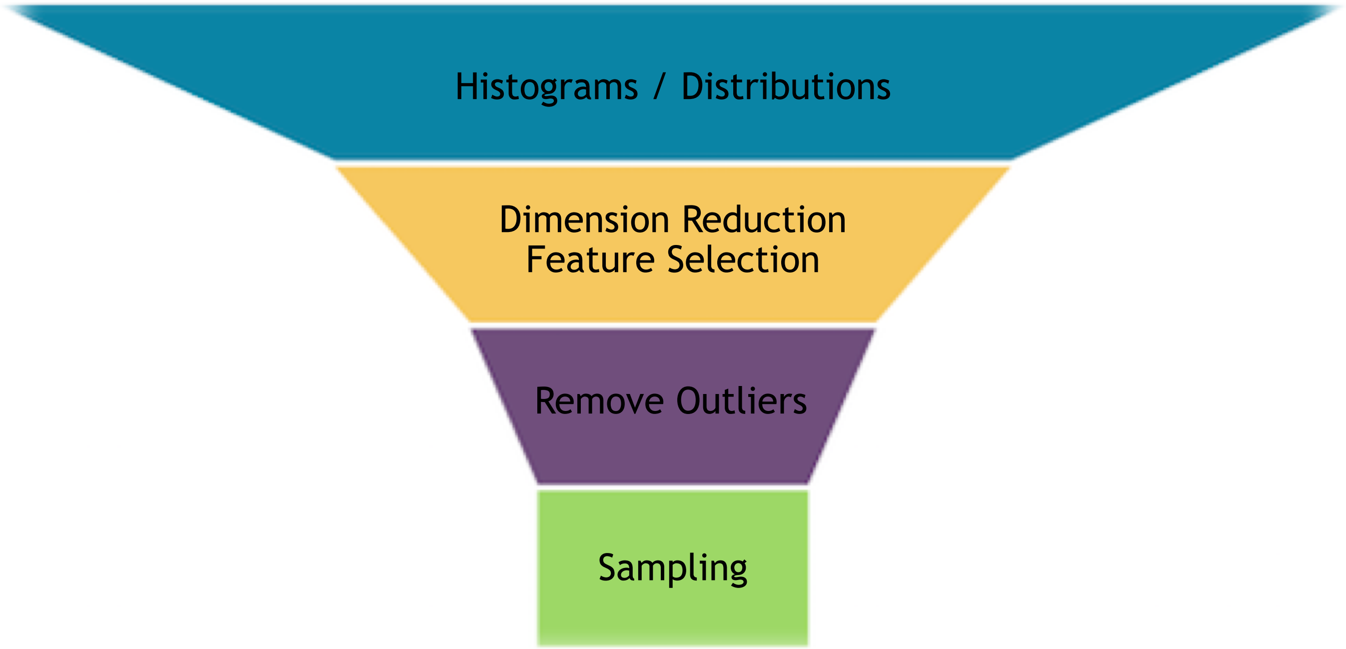
Problem

Data sizes continue to grow



Problem

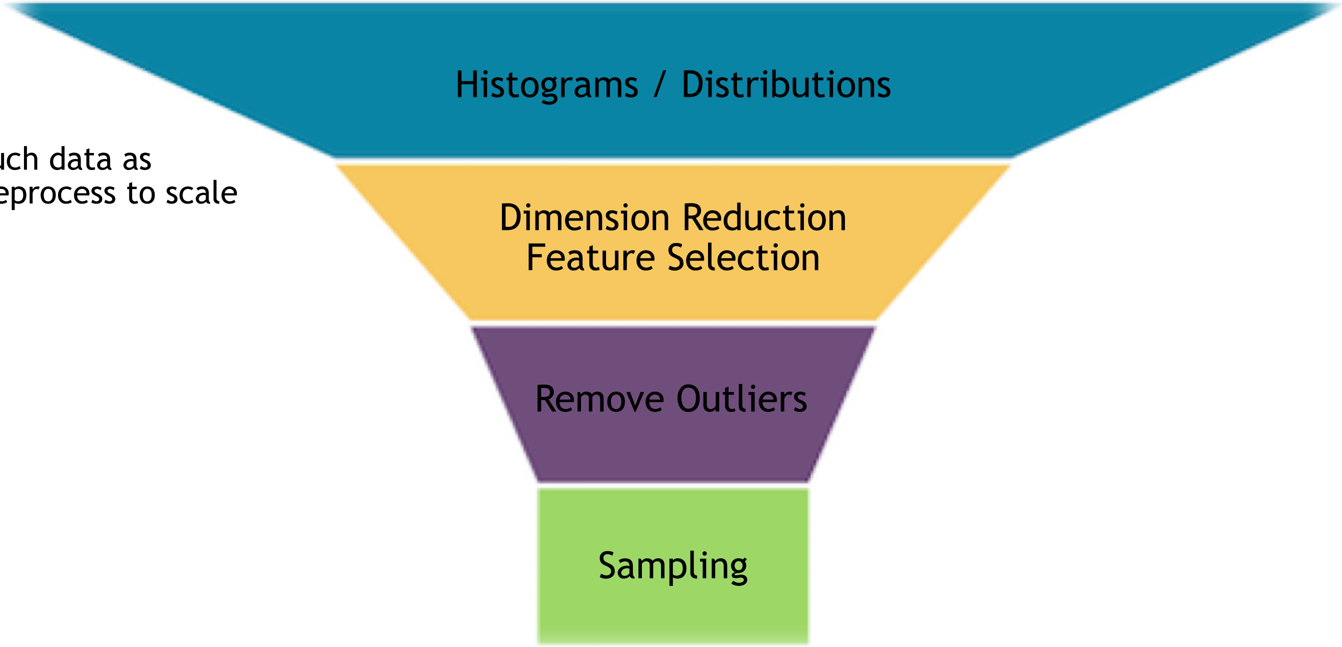
Data sizes continue to grow



Problem

Data sizes continue to grow

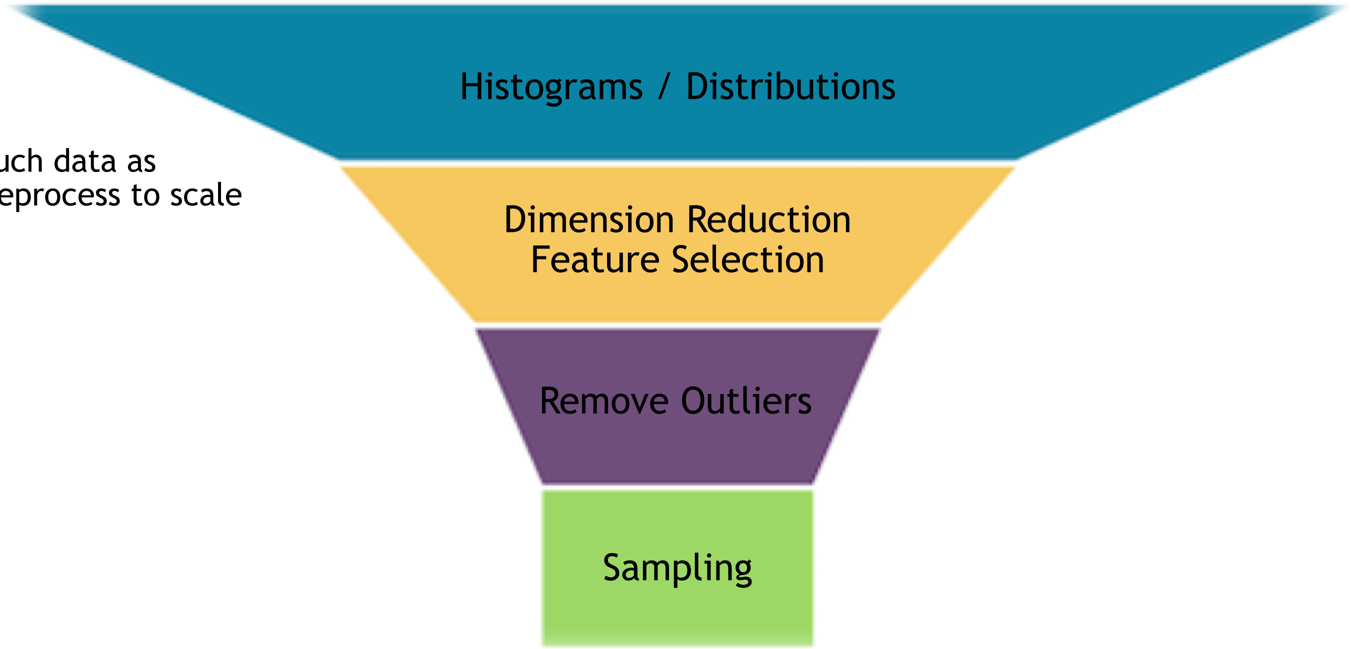
Better to start with as much data as possible and explore / preprocess to scale to performance needs.



Problem

Data sizes continue to grow

Massive Dataset

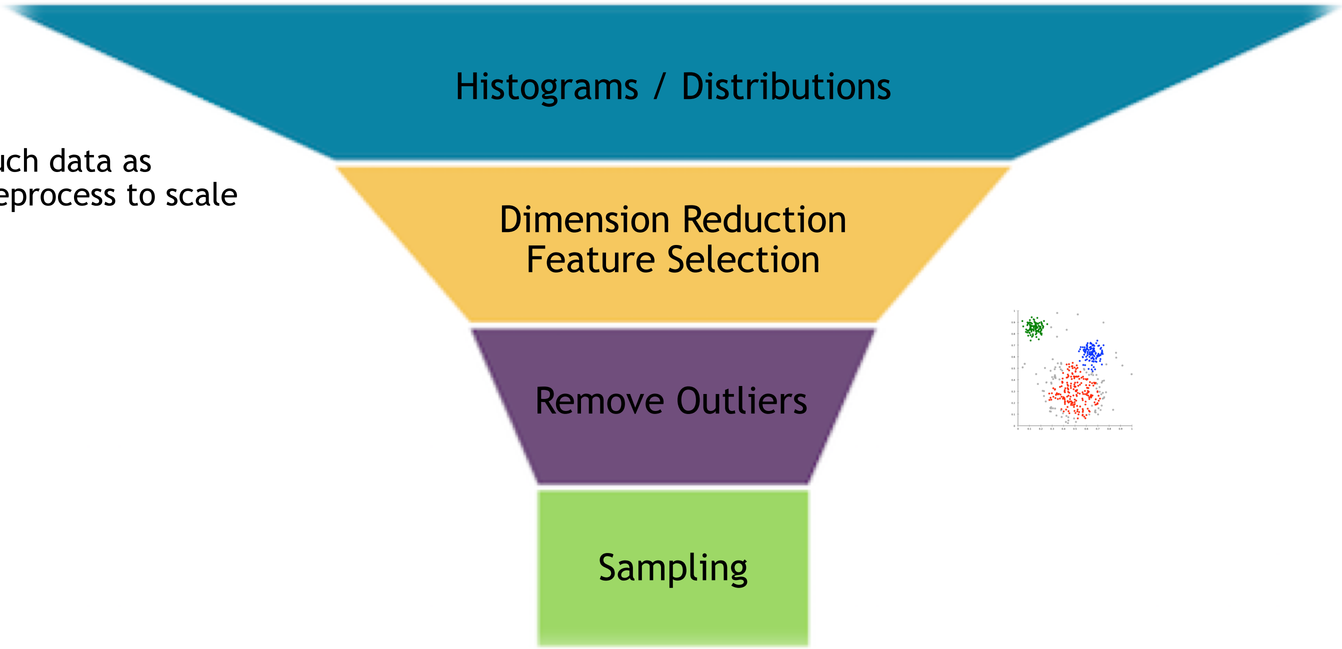


Better to start with as much data as possible and explore / preprocess to scale to performance needs.

Problem

Data sizes continue to grow

Massive Dataset

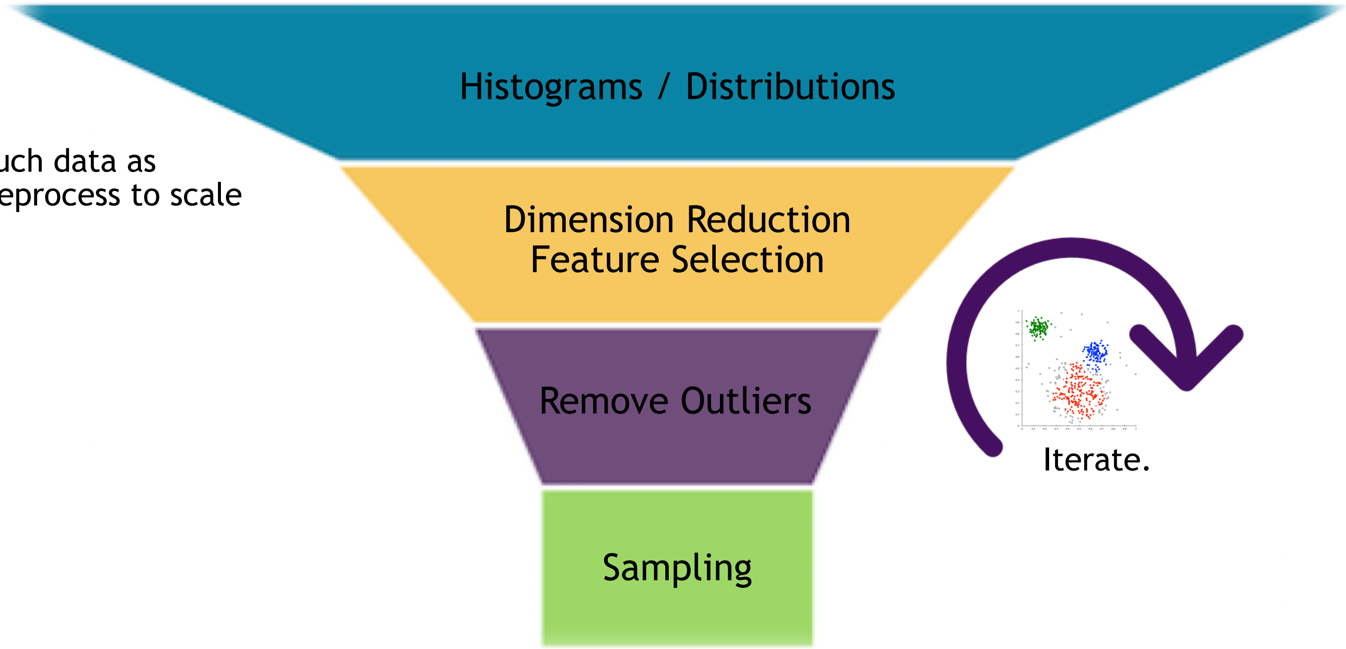


Better to start with as much data as possible and explore / preprocess to scale to performance needs.

Problem

Data sizes continue to grow

Massive Dataset



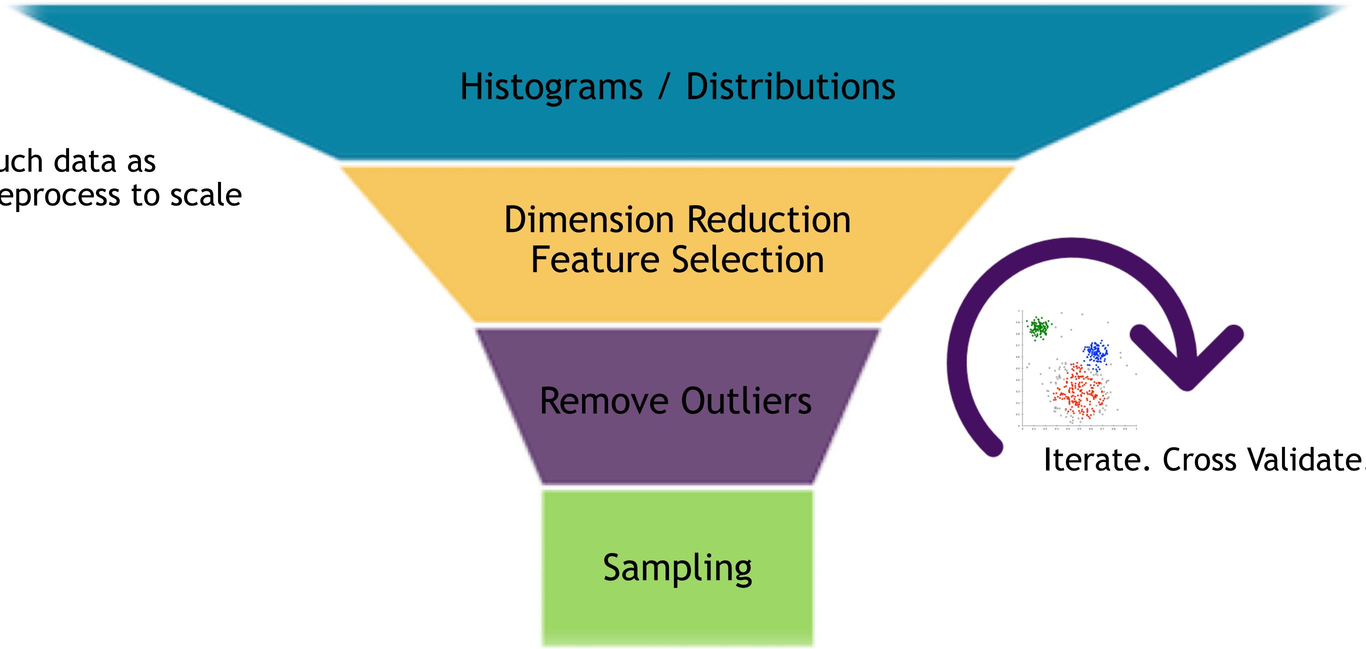
Better to start with as much data as possible and explore / preprocess to scale to performance needs.

Iterate.

Problem

Data sizes continue to grow

Massive Dataset



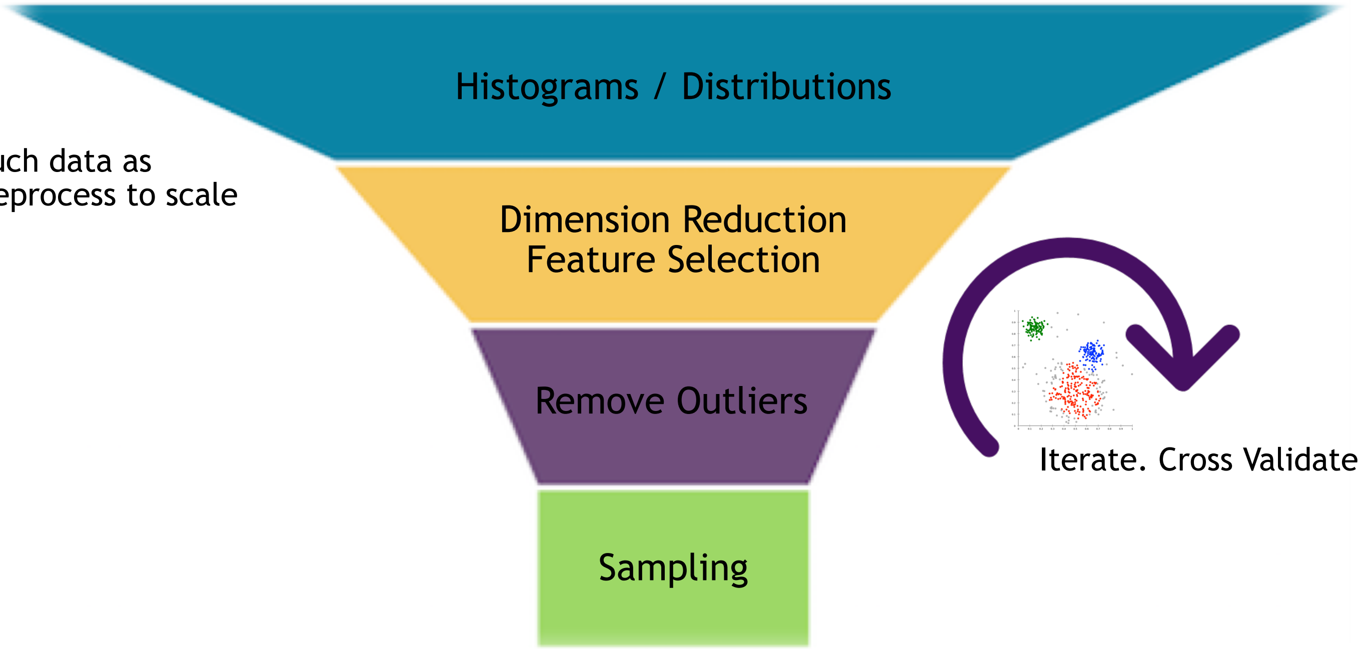
Better to start with as much data as possible and explore / preprocess to scale to performance needs.

Iterate. Cross Validate.

Problem

Data sizes continue to grow

Massive Dataset



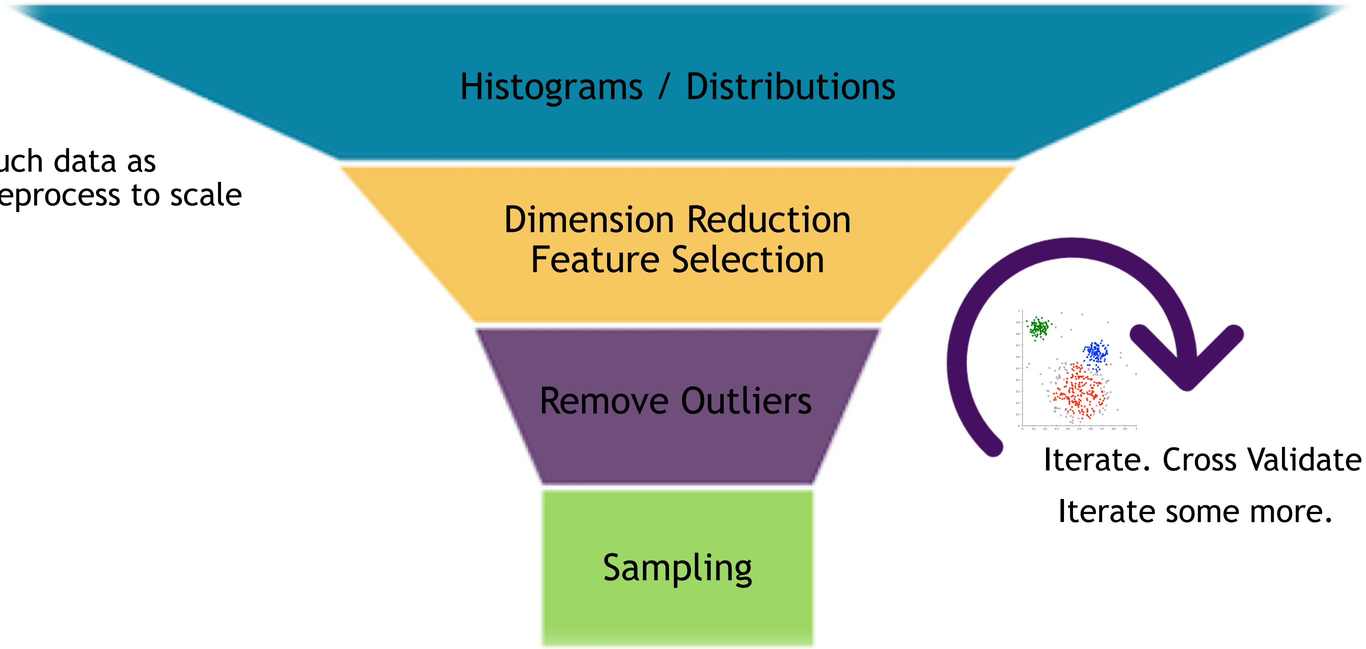
Better to start with as much data as possible and explore / preprocess to scale to performance needs.

Iterate. Cross Validate & Grid Search.

Problem

Data sizes continue to grow

Massive Dataset



Better to start with as much data as possible and explore / preprocess to scale to performance needs.

Iterate. Cross Validate & Grid Search.

Iterate some more.

Problem

Data sizes continue to grow

Massive Dataset

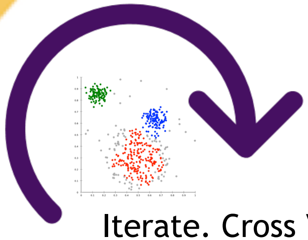
Histograms / Distributions

Dimension Reduction
Feature Selection

Remove Outliers

Sampling

Better to start with as much data as possible and explore / preprocess to scale to performance needs.



Iterate. Cross Validate & Grid Search.

Iterate some more.

Meet reasonable speed vs accuracy tradeoff

Problem

Data sizes continue to grow

Massive Dataset

Histograms / Distributions

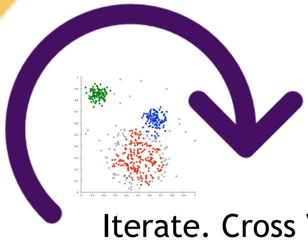
Dimension Reduction
Feature Selection

Remove Outliers

Sampling

Better to start with as much data as possible and explore / preprocess to scale to performance needs.

Time
Increases



Iterate. Cross Validate & Grid Search.

Iterate some more.

Meet reasonable speed vs accuracy tradeoff

Problem

Data sizes continue to grow

Massive Dataset

Histograms / Distributions

Dimension Reduction
Feature Selection

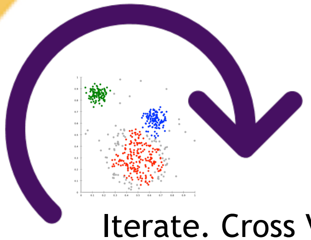
Remove Outliers

Sampling

Better to start with as much data as possible and explore / preprocess to scale to performance needs.

Time Increases

Hours?



Iterate. Cross Validate & Grid Search.

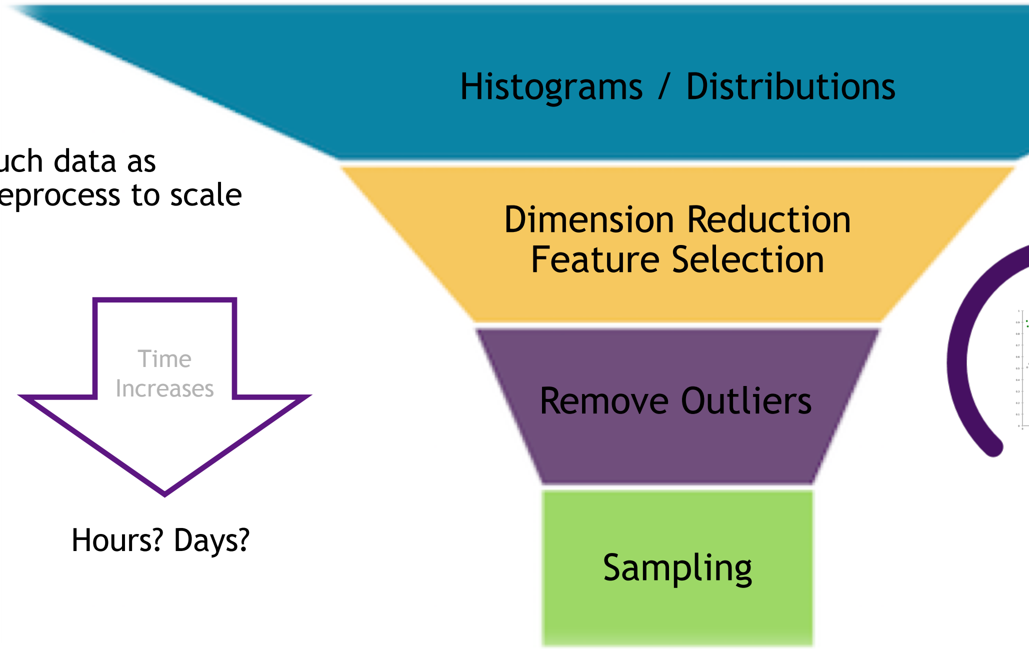
Iterate some more.

Meet reasonable speed vs accuracy tradeoff

Problem

Data sizes continue to grow

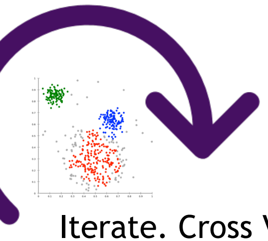
Massive Dataset



Better to start with as much data as possible and explore / preprocess to scale to performance needs.

Time Increases

Hours? Days?



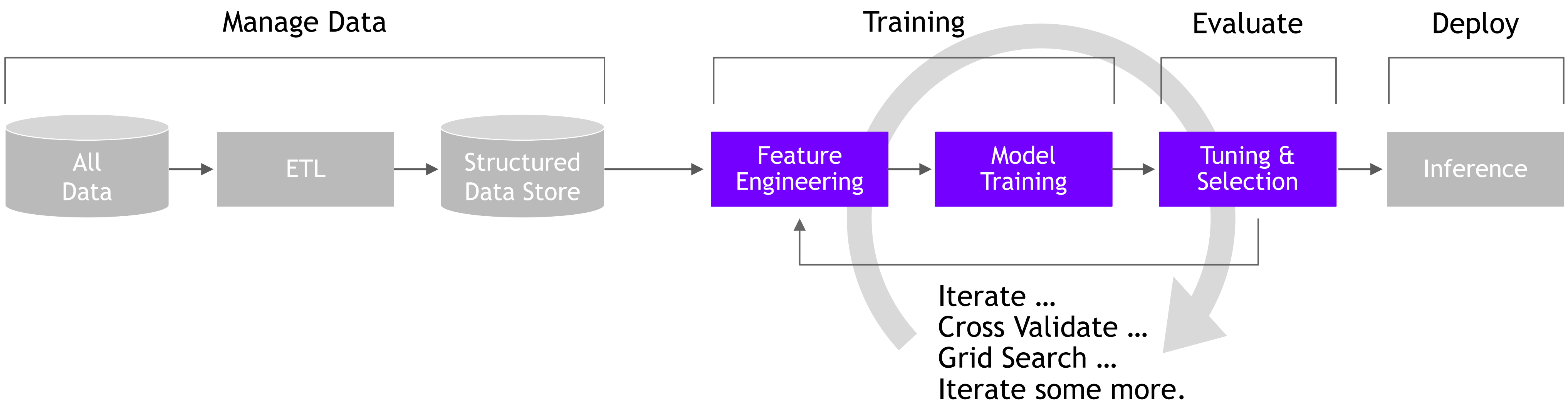
Iterate. Cross Validate & Grid Search.

Iterate some more.

Meet reasonable speed vs accuracy tradeoff

ML Workflow Stifles Innovation

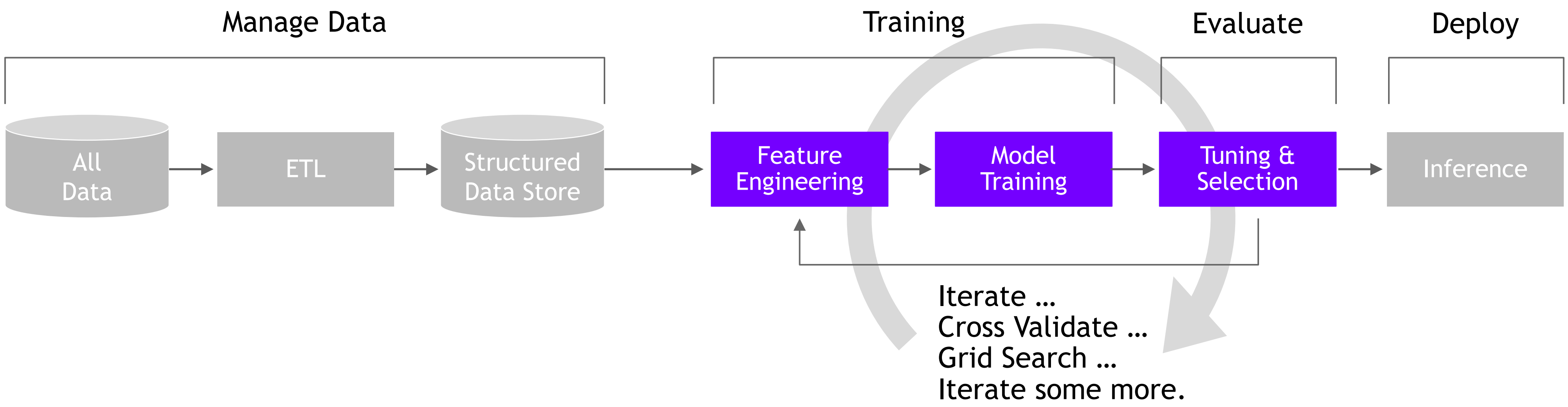
It Requires Exploration and Iterations



Accelerating just `Model Training` does have benefit but doesn’t address the whole problem

ML Workflow Stifles Innovation

It Requires Exploration and Iterations



Accelerating just `Model Training` does have benefit but doesn't address the whole problem

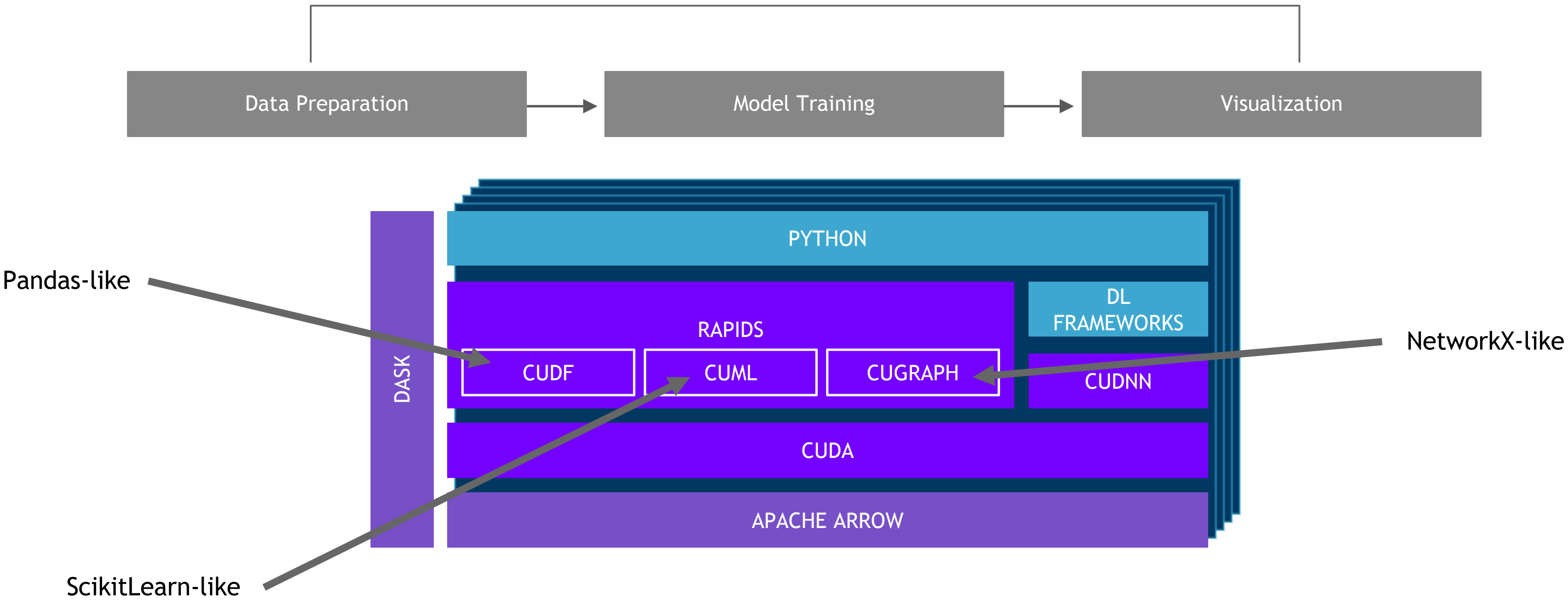
End-to-End acceleration is needed

Architecture

“More data requires better approaches!”
~ Xavier Amatriain
CTO, CurAI

RAPIDS: OPEN GPU DATA SCIENCE

cuDF, cuML, and cuGraph mimic well-known libraries

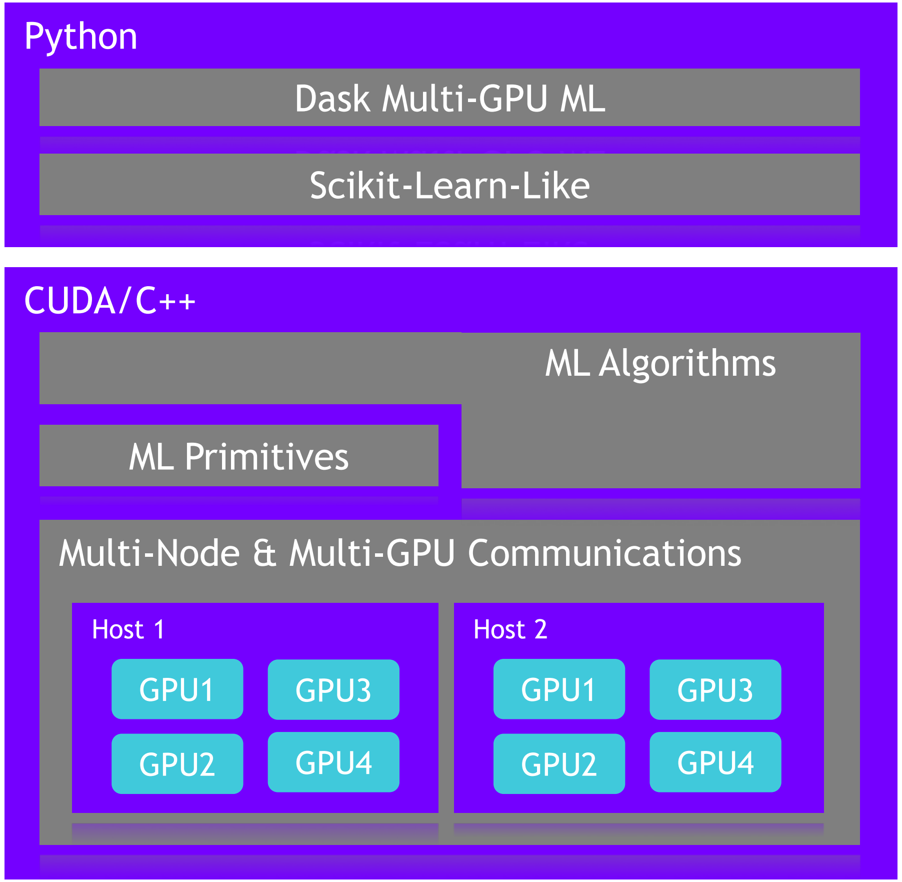


HIGH-LEVEL APIs

Dask-CUML

CuML

libcuml



cuML API

GPU-accelerated machine learning at every layer

Python	Scikit-learn-like interface for data scientists utilizing cuDF & Numpy
Algorithms	CUDA C++ API for developers to utilize accelerated machine learning algorithms.
Primitives	Reusable building blocks for composing machine learning algorithms.

Primitives

GPU-accelerated math optimized for feature matrices

Linear Algebra

- Element-wise operations
- Matrix multiply
- Norms
- Eigen Decomposition
- SVD/RSVD
- Transpose
- QR Decomposition

Statistics

Matrix / Math

Random

Distance / Metrics

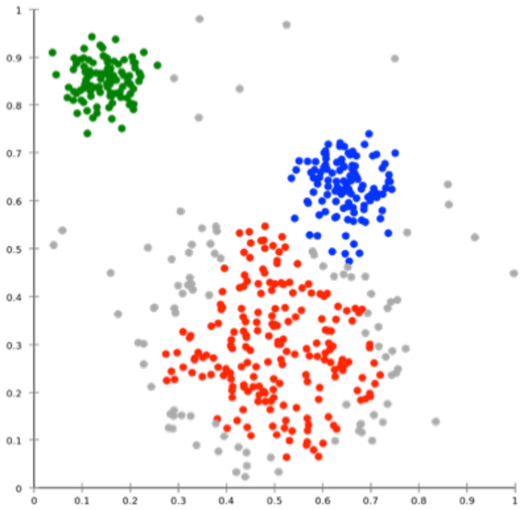
Objective Functions

Sparse Conversions

More to come!

Algorithms

GPU-accelerated Scikit-Learn



Classification / Regression
- Decision Trees / Random Forests
- Linear Regression
- Logistic Regression
- K-Nearest Neighbors
- Kalman Filtering

Statistical Inference
- Bayesian Inference
- Gaussian Mixture Models
- Hidden Markov Models

Clustering
- K-Means
- DBSCAN
- Spectral Clustering

Decomposition & Dimensionality Reduction
- Principal Components
- Singular Value Decomposition
- UMAP
- Spectral Embedding

Timeseries Forecasting
- ARIMA
- Holt-Winters

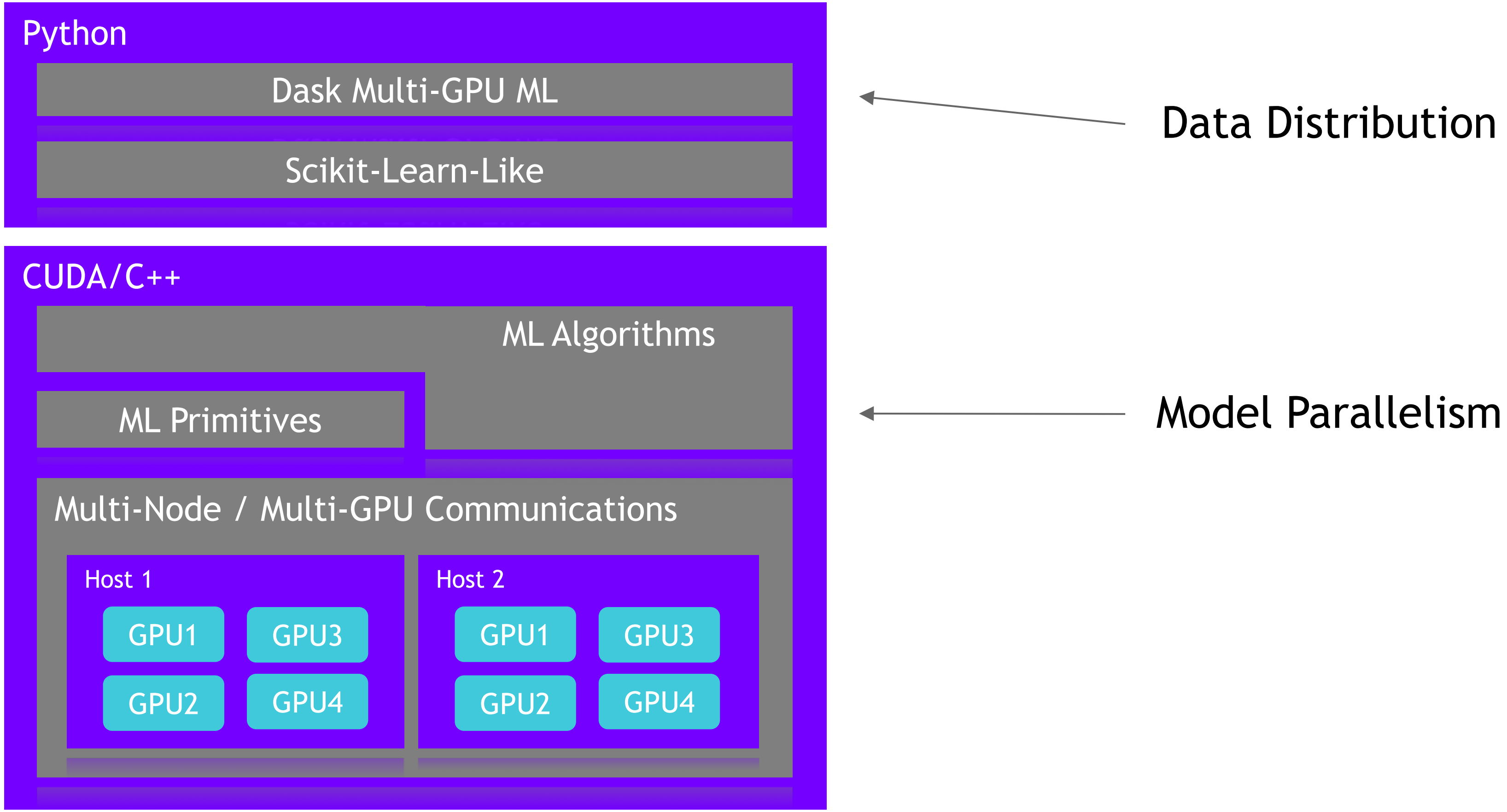
Recommendations
- Implicit Matrix Factorization

Cross Validation

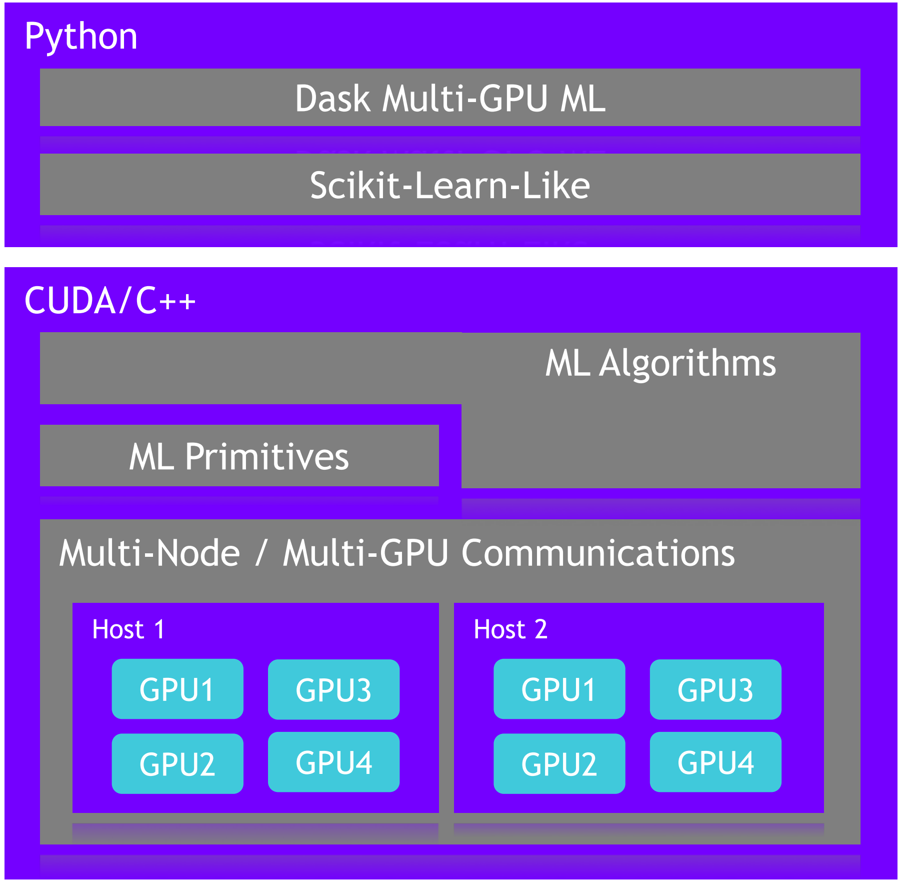
Hyper-parameter Tuning

More to come!

HIGH-LEVEL APIs



HIGH-LEVEL APIs



Data Distribution

Model Parallelism

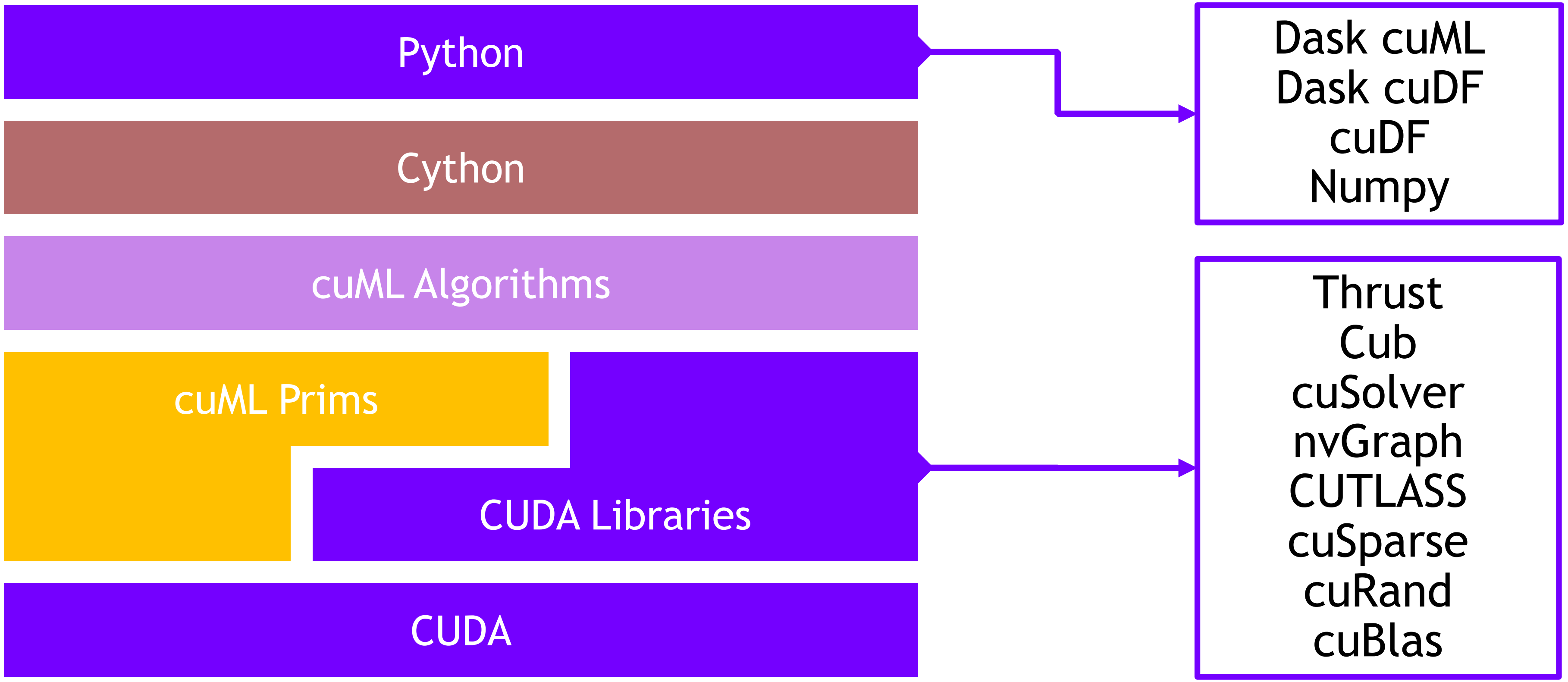
- Portability
- Efficiency
- Speed

Dask cuML

Distributed Data-parallelism Layer

- Distributed computation scheduler for Python
- Scales up and out
- Distributes data across processes
- Enables model-parallel cuML algorithms

ML Technology Stack



cuML Deep Dive

"I would posit that every scientist is a data scientist."
~ Arun Subramaniyan
V.P. of Data Science & Analytics, Baker Hughes, a GE Company

Linear Regression (OLS)

Python Layer

Pandas

```
X = pd.read_csv('data.csv')
```

cuDF

```
X_cudf = cudf.read_csv('data.csv')
```

Linear Regression (OLS)

Python Layer

cuDF

```
X_cudf = cudf.DataFrame.from_pandas(X)
y_cudf = np.array(y.as_matrix())
y_cudf = y_cudf[:,0]
y_cudf = cudf.Series(y_cudf)
```

Linear Regression (OLS)

Python Layer

Scikit-Learn

```
from sklearn.linear_model import LinearRegression as sklGLM
```

cuML

```
from cuml import LinearRegression as cumlOLS
```

Linear Regression (OLS)

Python Layer

Scikit-Learn

```
reg_sk = sklGLM.LinearRegression(fit_intercept=fit_intercept, normalize=normalize)
result_sk = reg_sk.fit(X, y)
```

cuML

```
algorithm = "eig" # eig: eigen decomposition based method, svd: singular value decomposition based method.

reg_cuml = cumlOLS(fit_intercept=fit_intercept, normalize=normalize, algorithm=algorithm)
result_cuml = reg_cuml.fit(X_cudf, y_cudf)
```

Linear Regression (OLS)

Python Layer

Scikit-Learn

```
y_sk = reg_sk.predict(X)
```

cuML

```
y_cuml = reg_cuml.predict(X_cudf)
```

Linear Regression (OLS)

cuML Algorithms CUDA C++ Layer

```
void olsFit(math_t *input,
            int n_rows,
            int n_cols,
            math_t *labels,
            math_t *coef,
            math_t *intercept,
            bool fit_intercept,
            bool normalize,
            cublasHandle_t cublas_handle,
            cusolverDnHandle_t cusolver_handle,
            int algo = 0)
```

Linear Regression (OLS)

cuML Algorithms CUDA C++ Layer

```
if (algo == 0 || n_cols == 1) {
    LinAlg::lstsqSVD(input, n_rows, n_cols, labels, coef, cusolver_handle,
                                          cublas_handle, mgr);
} else if (algo == 1) {
    LinAlg::lstsqEig(input, n_rows, n_cols, labels, coef, cusolver_handle,
                                          cublas_handle, mgr);
}
```

Linear Regression (OLS)

cuML ML-Prims CUDA C++ Layer

```
template<typename math_t>
void lstsqEig(math_t *A, int n_rows, int n_cols, math_t *b, math_t *w,
              cusolverDnHandle_t cusolverH, cublasHandle_t cublasH,
              DeviceAllocator &mgr) {

    math_t *S, *V, *U;
    int U_len = n_rows * n_cols;
    int V_len = n_cols * n_cols;
    allocate(U, U_len);
    allocate(V, V_len);
    allocate(S, n_cols);









}
```

Linear Regression (OLS)

cuML ML-Prims CUDA C++ Layer

```
template<typename math_t>
void lstsqEig(math_t *A, int n_rows, int n_cols, math_t *b, math_t *w,
              cusolverDnHandle_t cusolverH, cublasHandle_t cublasH,
              DeviceAllocator &mgr) {

    math_t *S, *V, *U;
    int U_len = n_rows * n_cols;
    int V_len = n_cols * n_cols;
    allocate(U, U_len);
    allocate(V, V_len);
    allocate(S, n_cols);

    svdEig(A, n_rows, n_cols, S, U, V, true, cublasH, cusolverH, mgr);









}
```

Linear Regression (OLS)

cuML ML-Prims CUDA C++ Layer

```
template<typename math_t>
void lstsqEig(math_t *A, int n_rows, int n_cols, math_t *b, math_t *w,
              cusolverDnHandle_t cusolverH, cublasHandle_t cublasH,
              DeviceAllocator &mgr) {

    math_t *S, *V, *U;
    int U_len = n_rows * n_cols;
    int V_len = n_cols * n_cols;
    allocate(U, U_len);
    allocate(V, V_len);
    allocate(S, n_cols);

    svdEig(A, n_rows, n_cols, S, U, V, true, cublasH, cusolverH, mgr);

    gemv(U, n_rows, n_cols, b, w, true, cublasH);




}
```

Linear Regression (OLS)

cuML ML-Prims CUDA C++ Layer

```
template<typename math_t>
void lstsqEig(math_t *A, int n_rows, int n_cols, math_t *b, math_t *w,
              cusolverDnHandle_t cusolverH, cublasHandle_t cublasH,
              DeviceAllocator &mgr) {

    math_t *S, *V, *U;
    int U_len = n_rows * n_cols;
    int V_len = n_cols * n_cols;
    allocate(U, U_len);
    allocate(V, V_len);
    allocate(S, n_cols);

    svdEig(A, n_rows, n_cols, S, U, V, true, cublasH, cusolverH, mgr);

    gemv(U, n_rows, n_cols, b, w, true, cublasH);

    Matrix::matrixVectorBinaryDivSkipZero(w, S, 1, n_cols, false, true);



}
```

Linear Regression (OLS)

cuML ML-Prims CUDA C++ Layer

```
template<typename math_t>
void lstsqEig(math_t *A, int n_rows, int n_cols, math_t *b, math_t *w,
              cusolverDnHandle_t cusolverH, cublasHandle_t cublasH,
              DeviceAllocator &mgr) {

    math_t *S, *V, *U;
    int U_len = n_rows * n_cols;
    int V_len = n_cols * n_cols;
    allocate(U, U_len);
    allocate(V, V_len);
    allocate(S, n_cols);

    svdEig(A, n_rows, n_cols, S, U, V, true, cublasH, cusolverH, mgr);

    gemv(U, n_rows, n_cols, b, w, true, cublasH);

    Matrix::matrixVectorBinaryDivSkipZero(w, S, 1, n_cols, false, true);

    gemv(V, n_cols, n_cols, w, w, false, cublasH);

    CUDA_CHECK(cudaFree(U));
    CUDA_CHECK(cudaFree(V));
    CUDA_CHECK(cudaFree(S));
}
```

Linear Regression (OLS)

cuML ML-Prims CUDA C++ Layer

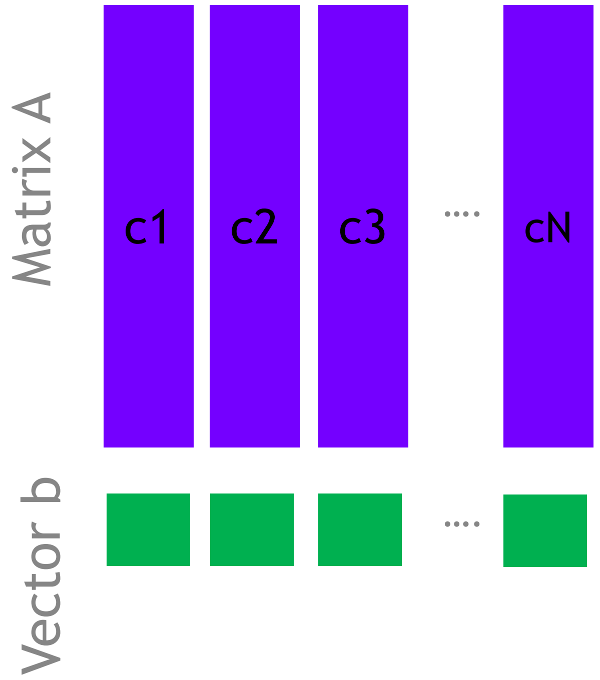
```
template <typename Type, typename IdxType = int, int TPB=256>
void matrixVectorBinaryDivSkipZero(Type* data, const Type* vec, IdxType n_row,
                                   IdxType n_col, bool rowMajor, bool bcastAlongRows,
                                   bool return_zero = false) {

    LinAlg::matrixVectorOp(data, data, vec, n_col, n_row, rowMajor, bcastAlongRows,
                           [] __device__ (Type a, Type b) {
                               if (myAbs(b) < Type(1e-10))
                                   return Type(0);
                               else
                                   return a / b;
                           });

}
```

Linear Regression (OLS)

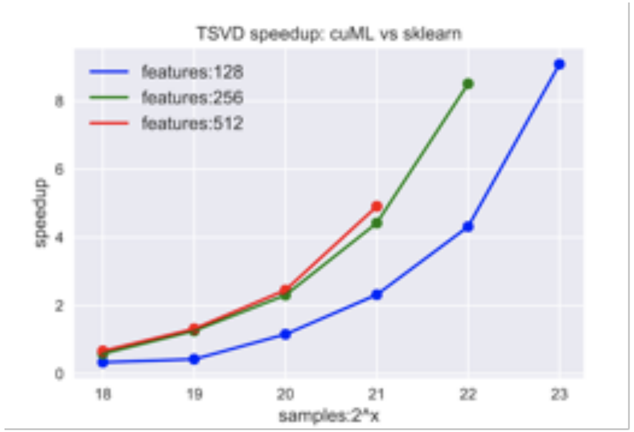
cuML ML-Prims CUDA C++ Layer

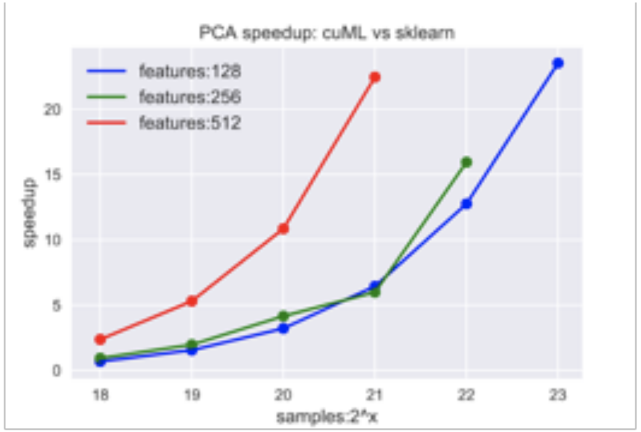


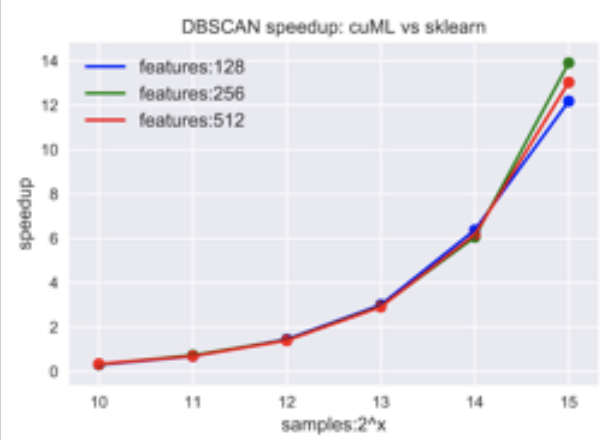
Benchmarks

ALGORITHMS

Benchmarked on DGX1

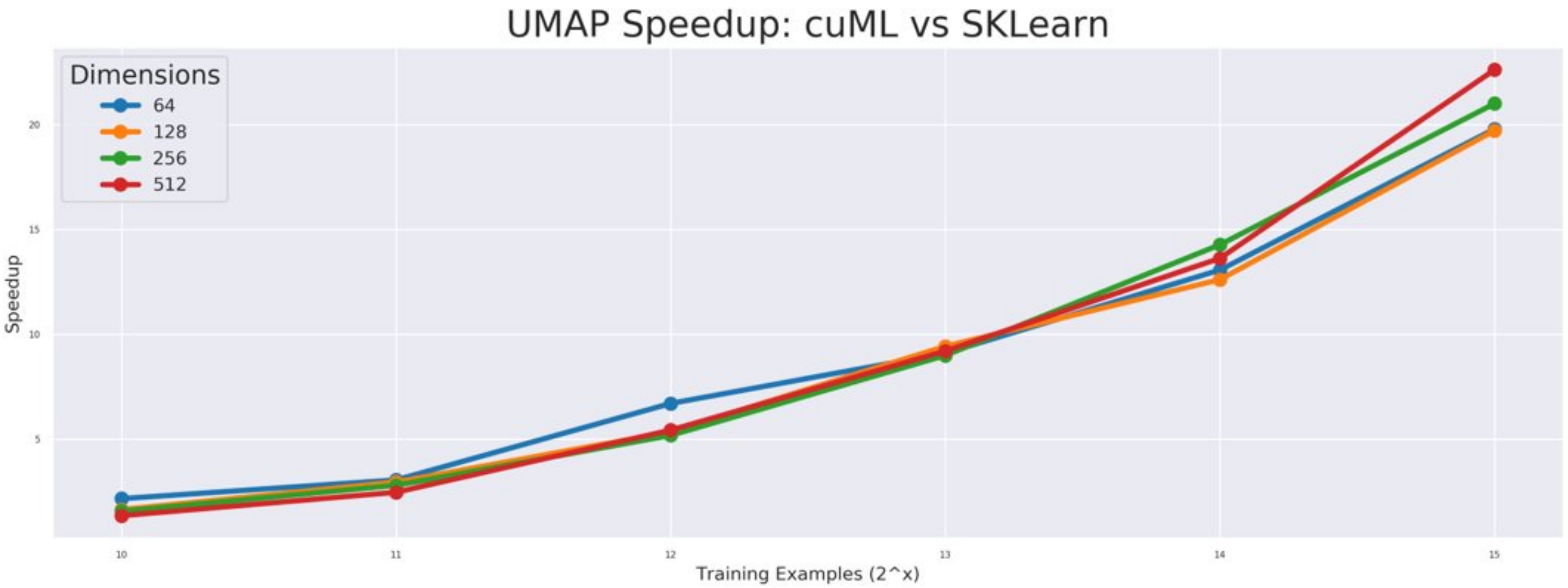






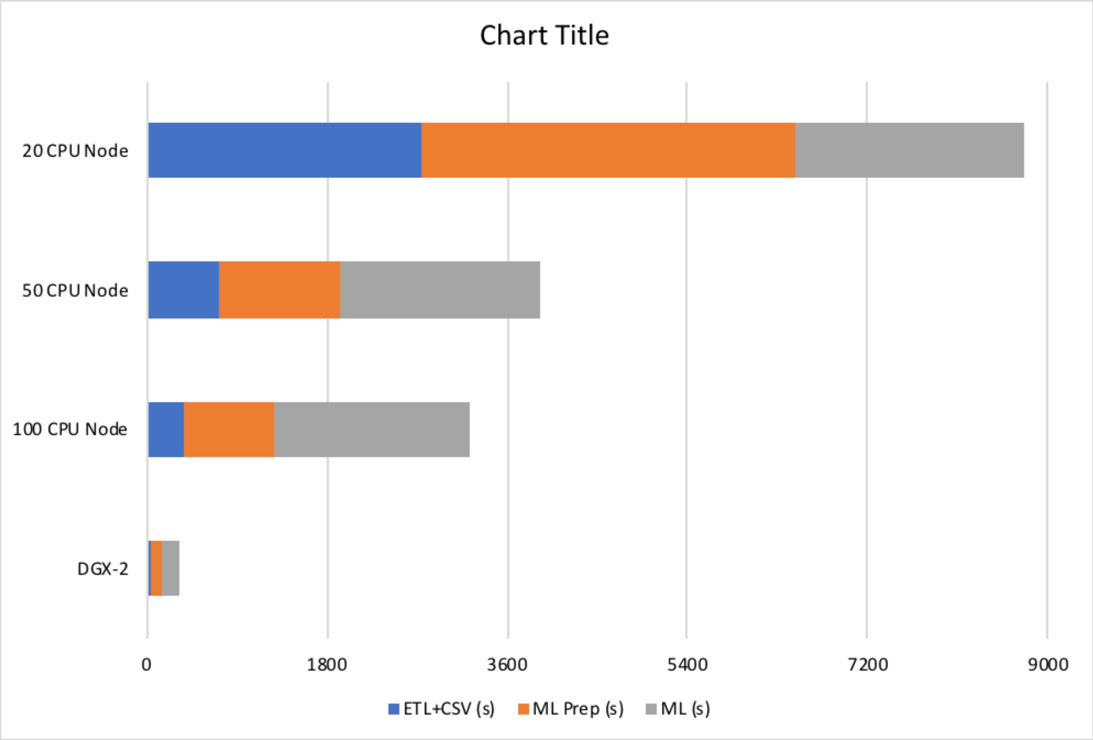
UMAP

Released in 0.6!



cuDF + XGBoost

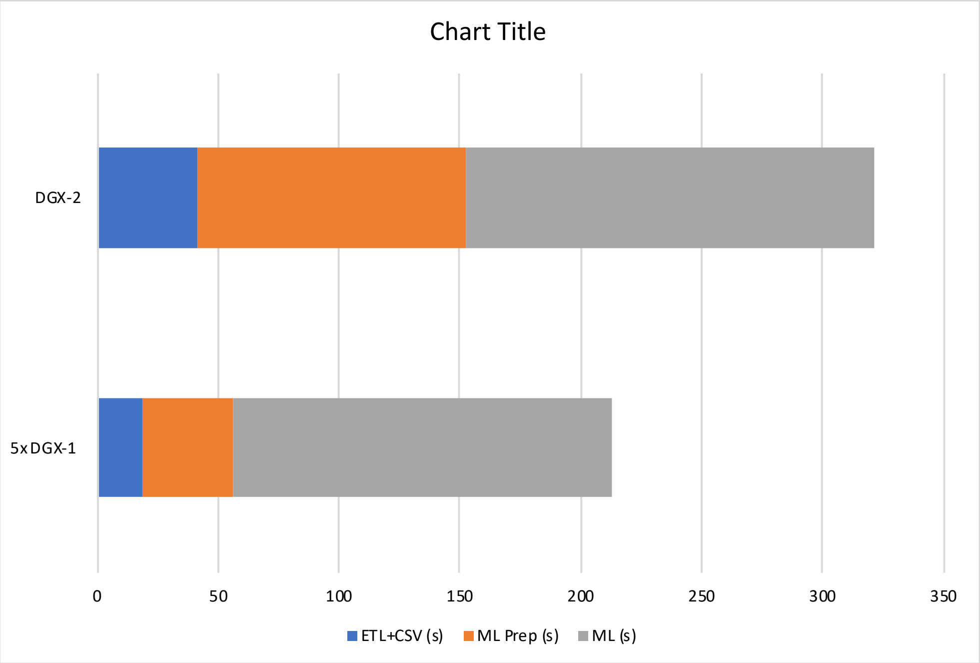
DGX-2 vs Scale Out CPU Cluster



- Full end to end pipeline
- Leveraging Dask + cuDF
- Store each GPU results in sys mem then read back in
- Arrow to Dmatrix (CSR) for XGBoost

cuDF + XGBoost

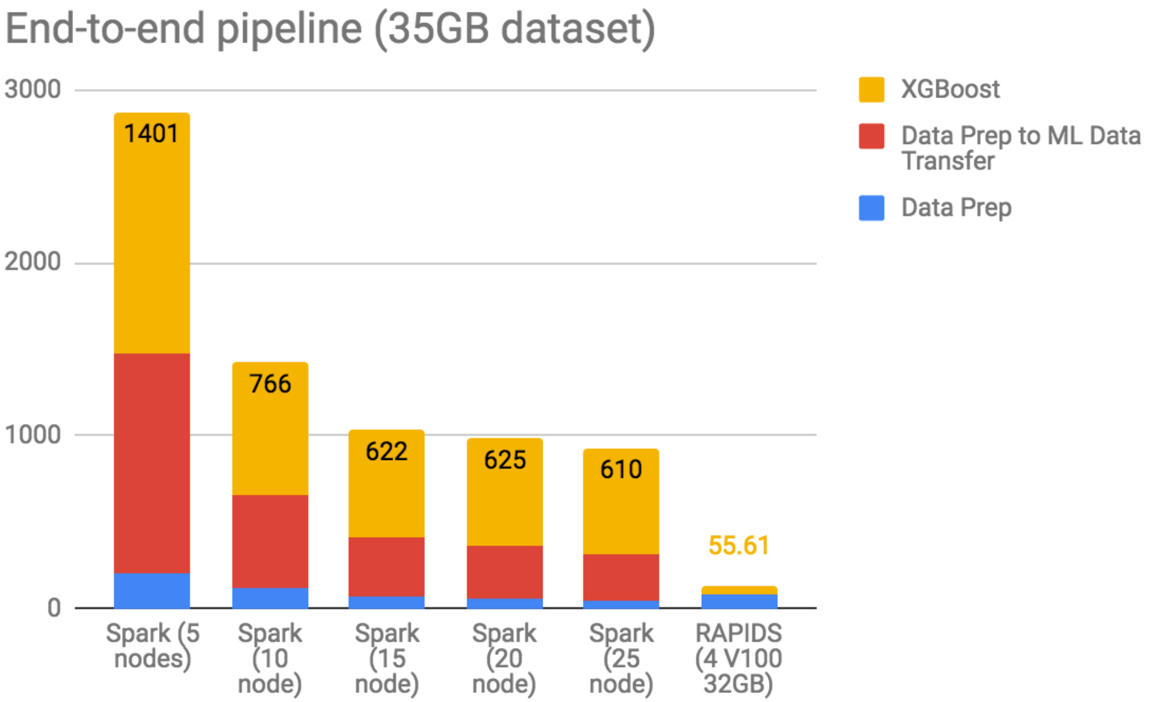
Scale Out GPU Cluster vs DGX-2



- Full end to end pipeline
- Leveraging Dask for multi-node + cuDF
- Store each GPU results in sys mem then read back in
- Arrow to Dmatrix (CSR) for XGBoost

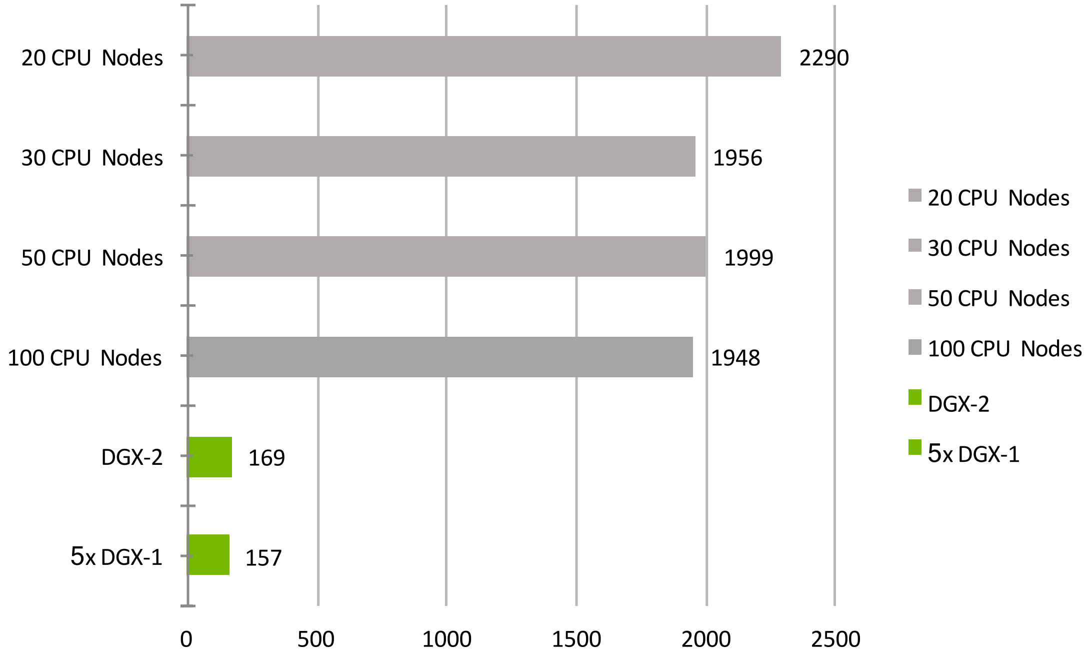
cuDF + XGBoost

Fully In- GPU Benchmarks



- Full end to end pipeline
- Leveraging Dask cuDF
- No Data Prep time all in memory
- Arrow to Dmatrix (CSR) for XGBoost

XGBoost
Multi-node, Multi-GPU Performance



Benchmark

200GB CSV dataset; Data preparation includes joins, variable transformations.

CPU Cluster Configuration

CPU nodes (61 GiB of memory, 8 vCPUs, 64-bit platform), Apache Spark

DGX Cluster Configuration

DGX nodes on InfiniBand network

Single Node Multi-GPU

Will be Released in 0.6

Linear Regression

- Reduction: 40mins -> 1min
- Size: 225gb
- System: DGX2

tSVD

- Reduction: 1.6hrs-> 1.5min
- Size: 220gb
- System: DGX2

Nearest Neighbors

- Reduction: 4+hrs-> 30sec
- Size: 128gb
- System: DGX1

Roadmap

“Data science is the fourth pillar of the scientific method!”
~ Jensen Huang

CUML

Single GPU and XGBoost

cuML	SG	MG	MGMN
Gradient Boosted Decision Trees (GBDT)			
GLM			
Logistic Regression			
Random Forest (regression)			
K-Means			
K-NN			
DBSCAN			
UMAP			
ARIMA			
Kalman Filter			
Holts-Winters			
Principal Components			
Singular Value Decomposition			

DASK-CUML

OLS, tSVD, and KNN in RAPIDS 0.6

cuML	SG	MG	MGMN
Gradient Boosted Decision Trees (GBDT)			
GLM			
Logistic Regression			
Random Forest (regression)			
K-Means			
K-NN			
DBSCAN			
UMAP			
ARIMA			
Kalman Filter			
Holts-Winters			
Principal Components			
Singular Value Decomposition			

DASK

DASK-CUML

K-Means*, DBSCAN & PCA in RAPIDS 0.7/0.8

cuML	SG	MG	MGMN
Gradient Boosted Decision Trees (GBDT)			
GLM			
Logistic Regression			
Random Forest (regression)			
K-Means			
K-NN			
DBSCAN			
UMAP			
ARIMA			
Kalman Filter			
Holts-Winters			
Principal Components			
Singular Value Decomposition			

- Deprecating the current K-means in 0.6 for new K-means built on MLPrims

CuML 0.6

Will be released with RAPIDS 0.6 on Friday!

New Algorithms

- Stochastic Gradient Descent [Single GPU]
- UMAP [Single GPU]
- Linear Regression (OLS) [Single Node, Multi-GPU]
- Truncated SVD [Single Node, Multi-GPU]

Notable Improvements

- Exposing support for hyperparsmeter tuning
- Removing external requirement on FAISS
- Lowered Nearest Neighbors memory requirement

Thank you!

Corey Nolet: @cjnolet
Onur Yilmaz: @Onur02128993

https://rapids.ai
https://github.com/cuml
https://github.com/dask-cuml

RAPIDS