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
• 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

Data Growth and Source in Exabytes

Source: IDC and EMC Digital Universe Report

- Enterprise Data
- VOIP
- Social Media & Web
- Sensor + Devices
Problem

Data sizes continue to grow
Problem

Data sizes continue to grow
Problem

Data sizes continue to grow
Problem
Data sizes continue to grow

- Histograms / Distributions
- Dimension Reduction
  - Feature Selection
- Remove Outliers
- Sampling
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

- 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.
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.
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.
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.
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.
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.
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.

Iterate. Cross Validate & Grid Search. Iterate some more.

Meet reasonable speed vs accuracy tradeoff
Data sizes continue to grow

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

Hours?

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.

Meet reasonable speed vs accuracy tradeoff

Iterate. Cross Validate & Grid Search. Iterate some more.
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

Data Preparation → Model Training → Visualization

Pandas-like
ScikitLearn-like
NetworkX-like
HIGH-LEVEL APIs

Python
- Dask Multi-GPU ML
- Scikit-Learn-Like

CUDA/C++
- ML Primitives
- ML Algorithms

Multi-Node & Multi-GPU Communications

Host 1
- GPU1
- GPU3
- GPU2
- GPU4

Host 2
- GPU1
- GPU3
- GPU2
- GPU4
**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
- Bayesian Inference
- Gaussian Mixture Models
- Hidden Markov Models

Statistical Inference
- K-Means
- DBSCAN
- Spectral Clustering
- Principal Components
- Singular Value Decomposition
- UMAP
- Spectral Embedding
- ARIMA
- Holt-Winters

Clustering
- Recommendations
- Implicit Matrix Factorization

Decomposition & Dimensionality Reduction
- Cross Validation
- Hyper-parameter Tuning
- Timeseries Forecasting
- More to come!
HIGH-LEVEL APIs

- **Python**
  - Dask Multi-GPU ML
  - Scikit-Learn-Like

- **CUDA/C++**
  - ML Algorithms
  - ML Primitives

- **Multi-Node / Multi-GPU Communications**
  - Host 1: GPU1, GPU3, GPU2, GPU4
  - Host 2: GPU1, GPU3, GPU2, GPU4

- **Data Distribution**
- **Model Parallelism**
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

Python

Cython

cuML Algorithms

cuML Prims

CUDA Libraries

CUDA

Dask cuML
Dask cuDF
cuDF
Numpy

Thrust
Cub
cuSolver
nvGraph
CUTLASS
cuSparse
cuRand
cuBlas
"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

\[ X = \text{pd.read_csv('data.csv')} \]

cuDF

\[ X\_\text{cudf} = \text{cudf.read_csv('data.csv')} \]
Linear Regression (OLS)

Python Layer

cuDF

```python
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

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

```python
from cuml import LinearRegression as cumlOLS
```
Linear Regression (OLS)

Python Layer

Scikit-Learn

```python
reg_sk = sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=True)
result_sk = reg_sk.fit(X, y)
```

cuML

```python
reg_cuml = cuml.OLS(fit_intercept=True, normalize=True, algorithm=algorithm)
result_cuml = reg_cuml.fit(X_cudf, y_cudf)
```
Linear Regression (OLS)

Python Layer

Scikit-Learn

\[ y_{sk} = \text{reg}_sk.predict(X) \]

cuML

\[ y_{cuml}' = \text{reg}_cuml.predict(X_{cudf}) \]
Linear Regression (OLS)

cuML Algorithms CUDA C++ Layer

```c
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

```cpp
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

```cpp
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

```cpp
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

Matrix A

\[ \begin{bmatrix} c_1 & c_2 & c_3 & \ldots & c_N \end{bmatrix} \]

Vector b

\[ \begin{bmatrix} b_1 & b_2 & b_3 & \ldots & b_N \end{bmatrix} \]
Benchmarks
ALGORITHMS

Benchmarked on DGX1
UMAP
Released in 0.6!

UMAP Speedup: cuML vs SKLearn

Dimensions
- 64
- 128
- 256
- 512

Speedup

Training Examples (2^x)
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

<table>
<thead>
<tr>
<th>cuML</th>
<th>SG</th>
<th>MG</th>
<th>MGMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Boosted Decision Trees (GBDT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GLM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logistic Regression</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Forest (regression)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Means</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-NN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBSCAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMAP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARIMA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kalman Filter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Holts-Winters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principal Components</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singular Value Decomposition</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DASK-CUML
OLS, tSVD, and KNN in RAPIDS 0.6

<table>
<thead>
<tr>
<th>cuML</th>
<th>SG</th>
<th>MG</th>
<th>MGMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Boosted Decision Trees (GBDT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GLM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logistic Regression</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Forest (regression)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Means</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-NN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBSCAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMAP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARIMA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kalman Filter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Holts-Winters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principal Components</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singular Value Decomposition</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# DASK-CUML

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

<table>
<thead>
<tr>
<th>cuML</th>
<th>SG</th>
<th>MG</th>
<th>MGMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Boosted Decision Trees (GBDT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GLM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logistic Regression</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Forest (regression)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Means</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-NN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBSCAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMAP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARIMA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kalman Filter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Holts-Winters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principal Components</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singular Value Decomposition</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• 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 hyperparameter 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