RAPIDS
GPU POWERED MACHINE LEARNING

Miguel Martínez
WHAT IS RAPIDS
RAPIDS is a set of open source libraries for GPU accelerating data preparation and machine learning.

OSS website: rapids.ai
RAPIDS LIBRARIES

**cuDF**
- GPU-accelerated lightweight in-GPU memory database used for data preparation
- Accelerates loading, filtering, and manipulation of data for model training data preparation
- Python drop-in Pandas replacement built on CUDA C++

**cuML**
- GPU accelerated traditional machine learning libraries
- XGBoost, PCA, Kalman, K-means, k-NN, DBScan, tSVD ...

**cuGRAPH**
- Collection of graph analytics libraries.
HOW TO SETUP AND START USING RAPIDS
HOW? DOWNLOAD AND DEPLOY

Source available on GitHub
https://github.com/rapidsai

Container available on NGC and Docker Hub
https://ngc.nvidia.com
https://hub.docker.com/u/rapidsai

Conda and PIP
https://anaconda.org/rapidsai
https://pypi.org/project/cudf/
https://pypi.org/project/cuml/

Pascal GPU architecture or better
CUDA 9.2 or 10.0
Ubuntu 16.04 or 18.04
RUNNING RAPIDS CONTAINER IN THE CLOUD
A step-by-step installation guide (MS Azure)

1. Create a **NC6s_v2** virtual machine instance on **Microsoft Azure Portal** using **NVIDIA GPU Cloud Image for Deep Learning and HPC** as image.

2. Start the virtual machine.

3. Connect to the virtual machine using the following command:
   
   ```
   $ ssh -L 8080:localhost:8888
   -L 8787:localhost:8787
   username@public_ip_address
   ```

4. Pull the **RAPIDS container** from **NGC**. Run it.
   
   ```
   $ docker pull nvcr.io/nvidia/rapidsai/rapidsai:cuda10.0-runtime-ubuntu18.04
   $ docker run --runtime=nvidia \
     --rm -it \
     -p 8888:8888 \
     -p 8787:8787 \
     -p 8786:8786 \
     nvcr.io/nvidia/rapidsai/rapidsai:cuda10.0-runtime-ubuntu18.04
   ```

5. Run JupyterLab:
   
   (rapids)$ bash /rapids/notebooks/utils/start-jupyter.sh

6. Open your browser, and navigate to **http://localhost:8080**.

7. Navigate to:
   
   - **cuml** folder for cuML IPython examples.
   - **mortgage** folder for XGBoost IPython examples.

8. Enjoy!
1. Create a p3.8xlarge machine instance on Amazon Web Services using NVIDIA Volta Deep Learning AMI as image.

2. Start the virtual machine.

3. Connect to the virtual machine using the following command:
   ```
   $ ssh -L 8080:localhost:8888 -L 8787:localhost:8787 ubuntu@public_ip_address
   ```

4. Pull the RAPIDS container from NGC. Run it.
   ```
   $ docker pull nvcr.io/nvidia/rapidsai:rapidsai:cuda10.0-runtime-ubuntu18.04
   ```

5. Run JupyterLab:
   ```
   (rapids)$ bash /rapids/notebooks/utils/start-jupyter.sh
   ```


7. Navigate to:
   - cuml folder for cuML IPython examples.
   - mortgage folder for XGBoost IPython examples.

8. Enjoy!
HOW TO PORT EXISTING CODE
Training results:
- CPU: 57.1 seconds
- GPU: 4.28 seconds
\textbf{CPU vs GPU PORTING EXISTING CODE KNN}

Training results:
- CPU: ~9 minutes
- GPU: 1.12 seconds

\textbf{Training results:}

\textbf{Before...}

\textbf{Specific: Import CPU algorithm}

\texttt{from sklearn.neighbors import KNeighborsClassifier}

\textbf{Common: Helper functions}

\texttt{# Timer, load_data...}

\texttt{from helper import *}

\textbf{Common: Data loading and algo params}

\texttt{# Data loading
\texttt{data = ...}
\texttt{X_train, X_test, y_train, y_test = train_test_split(data, ...)}

\texttt{# Algorithm parameters
\texttt{k_neighbors = 10
\texttt{}}

\textbf{Specific: Training}

\texttt{print(knn.score(X_train, y_train))}

\texttt{CPU time: user 24.9 s, sys: 0.03 s, total: 25.0 s
\texttt{Wall time: 25.0 s

\textbf{...Now!}

\textbf{Specific: Import GPU algorithm}

\texttt{from cuml import KNeighborsClassifier}

\textbf{Common: Helper functions}

\texttt{# Timer, load_data...}

\texttt{from helper import *}

\textbf{Common: Data loading and algo params}

\texttt{# Data loading
\texttt{data = ...}
\texttt{X_train, X_test, y_train, y_test = train_test_split(data, ...)}

\texttt{# Algorithm parameters
\texttt{k_neighbors = 10
\texttt{}}

\textbf{Specific: Training}

\texttt{print(knn.score(X_train, y_train))}

\texttt{CPU time: user 0.07 s, sys: 0.03 s, total: 0.10 s
\texttt{Wall time: 0.10 s

\textbf{Training results:}

- CPU: ~9 minutes
- GPU: 1.12 seconds
The bigger the dataset is, the higher the training performance difference is between CPU and GPU.

### Dataset size trained in 15 minutes.
- **CPU**: ~130,000 rows.
- **GPU**: ~5,900,000 rows.

### Specs
<table>
<thead>
<tr>
<th></th>
<th>NC6s_vs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>6</td>
</tr>
<tr>
<td>GPU</td>
<td>1 x P100</td>
</tr>
<tr>
<td>Memory</td>
<td>112 GB</td>
</tr>
<tr>
<td>Local Disk</td>
<td>~700 GB SSD</td>
</tr>
<tr>
<td>Network</td>
<td>Azure Network</td>
</tr>
</tbody>
</table>

---

The bigger the dataset is, the higher the training performance difference is between CPU and GPU.
WHAT IS XGBOOST
XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

It is a powerful tool for solving classification and regression problems in a supervised learning setting.
PREDICT: WHO ENJOYS COMPUTER GAMES

Example of Decision Tree

Input: age, gender, occupation, ...

Does the person like computer games

age < 15

is male?

Y N

prediction score in each leaf

+2 +0.1 -1

Source: https://goo.gl/C6WKIf
COMBINE TREES FOR STRONGER PREDICTIONS

Example of Using Ensembled Decision Trees

**tree1**
- age < 15
  - Y: +2
  - N: +0.1
  - N: -1

**tree2**
- Use Computer Daily
  - Y: +0.9
  - N: -0.9

\[ f(\text{boy}) = 2 + 0.9 = 2.9 \]
\[ f(\text{girl}) = -1 - 0.9 = -1.9 \]

Source: https://goo.gl/C6WKiF
Models fit to the *Boston Housing* Dataset.

Source: https://goo.gl/GWhQff
WHY XGBoost
A STRONG HISTORY OF SUCCESS

On a Wide Range of Problems

Winner of Caterpillar Kaggle Contest 2015
- Machinery component pricing

Winner of CERN Large Hadron Collider Kaggle Contest 2015
- Classification of rare particle decay phenomena

Winner of KDD Cup 2016
- Research institutions' impact on the acceptance of submitted academic papers

Winner of ACM RecSys Challenge 2017
- Job posting recommendation
WHICH ML ALGORITHM PERFORMS BEST
Average rank across 165 ML datasets

Source: https://goo.gl/aztMh2
WHY RAPIDS WITH XGBOOST
Multi-GPU, Multi-Node, Scalability

▷ XGBoost:
  ▷ Algorithm tuned for eXtreme performance and high efficiency
  ▷ Multi-GPU and Multi-Node Support

▷ RAPIDS:
  ▷ End-to-end data science & analytics pipeline entirely on GPU
  ▷ User-friendly Python interfaces
  ▷ Faster results helps hyperparameter tuning
  ▷ Relies on CUDA primitives, exposes parallelism and high-memory bandwidth
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
5x DGX-1 on InfiniBand network
# cuML ROADMAP

<table>
<thead>
<tr>
<th>cuML Algorithms</th>
<th>Available Now</th>
<th>Q2-2019</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGBoost GBDT</td>
<td>MGMN</td>
<td></td>
</tr>
<tr>
<td>XGBoost Random Forest</td>
<td>MGMN</td>
<td></td>
</tr>
<tr>
<td>K-Means Clustering</td>
<td>MG</td>
<td></td>
</tr>
<tr>
<td>K-Nearest Neighbors (KNN)</td>
<td>MG</td>
<td></td>
</tr>
<tr>
<td>Principal Component Analysis (PCA)</td>
<td>SG</td>
<td></td>
</tr>
<tr>
<td>Density-based Spatial Clustering of Applications with Noise (DBSCAN)</td>
<td>SG</td>
<td></td>
</tr>
<tr>
<td>Truncated Singular Value Decomposition (tSVD)</td>
<td>SG</td>
<td></td>
</tr>
<tr>
<td>Uniform Manifold Approximation and Projection (UMAP)</td>
<td>SG</td>
<td>MG</td>
</tr>
<tr>
<td>Kalman Filters (KF)</td>
<td>SG</td>
<td></td>
</tr>
<tr>
<td>Ordinary Least Squares Linear Regression (OLS)</td>
<td>SG</td>
<td></td>
</tr>
<tr>
<td>Stochastic Gradient Descent (SGD)</td>
<td>SG</td>
<td></td>
</tr>
<tr>
<td>Generalized Linear Model, including Logistic (GLM)</td>
<td>MG</td>
<td></td>
</tr>
<tr>
<td>Time Series (Holts-Winters)</td>
<td>SG</td>
<td></td>
</tr>
<tr>
<td>Autoregressive Integrated Moving Average (ARIMA)</td>
<td>SG</td>
<td></td>
</tr>
</tbody>
</table>

Last updated 29.03.19
LEARN MORE ABOUT RAPIDS
RAPIDS
Open GPU Data Science
GET STARTED
https://rapids.ai
CUDF
CODE SAMPLES
LOADING DATA INTO A GPU DATAFRAME

Create an empty DataFrame, and add a column.

```python
import cudf
gdf = cudf.DataFrame()
gdf['my_column'] = [6, 7, 8]
print(gdf)
```

```
   my_column
0    6
1    7
2    8
```

Create a DataFrame with two columns.

```python
import cudf
gdf = cudf.DataFrame({'a': [3, 4, 5], 'b': [6, 7, 9]})
print(gdf)
```

```
b   a
0  6  3
1  7  4
2  9  5
```

Load a CSV file into a GPU DataFrame.

```python
import cudf

path = './apartments.csv'
names = ['city', 'zipcode', 'price_per_m2', 'year_built', 'population', 'median_income', 'date']

# Note: dtypes detection is not yet supported.
dtypes = ['category', 'int64', 'float64', 'float64', 'int64', 'int64', 'date']

gdf = cudf.read_csv(path, names=names, dtypes=dtypes, delimiter=';'
, skiprows=1, skipfooter=1)
```

Use Pandas to load a CSV file, and copy its content into a GPU DataFrame.

```python
import pandas as pd
import cudf

# Load a CSV file using pandas.
pdf = pd.read_csv(path, delimiters=';')

# Convert data types to ones supported by cudf.
pdf['city'] = pdf['city'].astype('category')
pdf['date'] = pdf['date'].astype('datetime64[ns]')

# Create a cudf dataframe from a pandas dataframe.
gdf = cudf.DataFrame.from_pandas(pdf)
```
Return the first three rows as a new DataFrame.

```python
print(gdf.head(3))
```

<table>
<thead>
<tr>
<th>city</th>
<th>zipcode</th>
<th>price_per_m2</th>
<th>year_built</th>
<th>population</th>
<th>median_income</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>espo</td>
<td>2100</td>
<td>5444.02222</td>
<td>2085</td>
<td>4332</td>
<td>26167</td>
<td>2018-09-06T00:00:00.000</td>
</tr>
<tr>
<td>espo</td>
<td>2130</td>
<td>3768.0</td>
<td>1972</td>
<td>5083</td>
<td>25570</td>
<td>2018-08-02T00:00:00.000</td>
</tr>
<tr>
<td>espo</td>
<td>2240</td>
<td>2778.0</td>
<td>1977</td>
<td>3689</td>
<td>26447</td>
<td>2018-12-10T00:00:00.000</td>
</tr>
</tbody>
</table>

Find the mean and standard deviation of a column.

```python
print(gdf['population'].mean())
print(gdf['population'].std())
```

8014.397849462365
4373.122998945762

Change the data type of a column.

```python
import numpy as np

print('Median income dtype used to be: ', gdf['median_income'].dtype)
gdf['median_income'] = gdf['median_income'].astype(np.float64)
print('Median income dtype is now: ', gdf['median_income'].dtype)
```

Median income dtype used to be: int64
Median income dtype is now: float64

Row slicing with column selection.

```python
print(gdf.loc[2:5, ['zipcode', 'year_built']])
```

<table>
<thead>
<tr>
<th>zipcode</th>
<th>year_built</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2140</td>
</tr>
<tr>
<td>3</td>
<td>2160</td>
</tr>
<tr>
<td>4</td>
<td>2170</td>
</tr>
<tr>
<td>5</td>
<td>2180</td>
</tr>
</tbody>
</table>

Count number of occurrences per value, and number of unique values.

```python
print(gdf['city'].value_counts())
print(gdf['city'].unique_count()) # unique() in pandas.
```

helsinki 65
espo 28

Transform column values with a custom function.

```python
def double_income(median_income):
    return 2*median_income
gdf['median_income'] = gdf['median_income'].applymap(double_income)
print(gdf.head(2))
```

<table>
<thead>
<tr>
<th>city</th>
<th>zipcode</th>
<th>price_per_m2</th>
<th>year_built</th>
<th>population</th>
<th>median_income</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>espo</td>
<td>2100</td>
<td>5444.02222</td>
<td>2085</td>
<td>4332</td>
<td>52316.0</td>
<td>2018-09-06T00:00:00.000</td>
</tr>
<tr>
<td>espo</td>
<td>2130</td>
<td>3768.0</td>
<td>1972</td>
<td>5083</td>
<td>51160.0</td>
<td>2018-08-02T00:00:00.000</td>
</tr>
</tbody>
</table>
Query the columns of a DataFrame with a boolean expression.

```python
query = gdf.query("year_built < 1938")
print(query.head(3))
```

<table>
<thead>
<tr>
<th>city</th>
<th>zipcode</th>
<th>price_per_sq</th>
<th>year_built</th>
<th>population</th>
<th>median_income</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>helsinki</td>
<td>1102</td>
<td>7816.0</td>
<td>1971</td>
<td>9254</td>
<td>7510.0</td>
<td>2020-02-17T00:00:00.000000</td>
</tr>
<tr>
<td>helsinki</td>
<td>1102</td>
<td>7816.0</td>
<td>1971</td>
<td>9254</td>
<td>7510.0</td>
<td>2020-02-17T00:00:00.000000</td>
</tr>
<tr>
<td>helsinki</td>
<td>1102</td>
<td>7816.0</td>
<td>1971</td>
<td>9254</td>
<td>7510.0</td>
<td>2020-02-17T00:00:00.000000</td>
</tr>
</tbody>
</table>

Sort a column by its values.

```python
gdf = gdf.sort_values(by='population', ascending=False)
print(gdf.head(3))
```

<table>
<thead>
<tr>
<th>city</th>
<th>zipcode</th>
<th>price_per_sq</th>
<th>year_built</th>
<th>population</th>
<th>median_income</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>helsinki</td>
<td>1102</td>
<td>7816.0</td>
<td>1971</td>
<td>9254</td>
<td>7510.0</td>
<td>2020-02-17T00:00:00.000000</td>
</tr>
<tr>
<td>helsinki</td>
<td>1102</td>
<td>7816.0</td>
<td>1971</td>
<td>9254</td>
<td>7510.0</td>
<td>2020-02-17T00:00:00.000000</td>
</tr>
<tr>
<td>helsinki</td>
<td>1102</td>
<td>7816.0</td>
<td>1971</td>
<td>9254</td>
<td>7510.0</td>
<td>2020-02-17T00:00:00.000000</td>
</tr>
</tbody>
</table>

Return the first ‘n’ rows ordered by ‘columns’ in ascending order.

```python
three_smallest = gdf.nsmallest(n=3, columns=['population'])
print(three_smallest)
```

<table>
<thead>
<tr>
<th>city</th>
<th>zipcode</th>
<th>price_per_sq</th>
<th>year_built</th>
<th>population</th>
<th>median_income</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>helsinki</td>
<td>1102</td>
<td>7816.0</td>
<td>1971</td>
<td>9254</td>
<td>7510.0</td>
<td>2020-02-17T00:00:00.000000</td>
</tr>
<tr>
<td>helsinki</td>
<td>1102</td>
<td>7816.0</td>
<td>1971</td>
<td>9254</td>
<td>7510.0</td>
<td>2020-02-17T00:00:00.000000</td>
</tr>
<tr>
<td>helsinki</td>
<td>1102</td>
<td>7816.0</td>
<td>1971</td>
<td>9254</td>
<td>7510.0</td>
<td>2020-02-17T00:00:00:00.000000</td>
</tr>
</tbody>
</table>

Join columns with other DataFrame on index.

```python
left = grouped
right = cudf.DataFrame({'zipcode': [28, 65], 'feature1': [1,2]})

# join() uses the index.
join_left = left.set_index('count.zipcode')
join_right = right.set_index('zipcode')

# Different join styles are supported.
joined = join_left.join(join_right, how='right')
```

Join columns with other DataFrame on index.

Group by column with aggregate function.

```python
# Differences to pandas:
# - aggregated column names are prefixed with the
#   aggregated function name.
# - 'city' becomes index in pandas but not in cudf.
grouped = gdf.groupby(['city']).agg({'zipcode': 'count'})
```

Merge two DataFrames.

```python
# Only inner join is supported currently.
merged = left.merge(right, on='zipcode')
```

One-hot encoding.

```python
gdf["city_codes"] = gdf.city.cat.codes
codes = gdf.city_codes.unique()

# get_dummies() in pandas.
encoded = gdf_one_hot_encoding(column='city_codes', cats=categories,
                               prefix='city_codes_dummy', dtype='int8')
```
GPU Accelerated Data Science

RAPIDS is a set of open source libraries for GPU accelerating data preparation and machine learning.

Visit www.rapids.ai
ONE MORE THING
MESSAGE TO DATA SCIENTISTS

FIND A NEW ARGUMENT

THE #1 DATA SCIENTIST EXCUSE FOR LEGITIMATELY SLACKING OFF: "MY MODEL'S TRAINING."

HEY! GET BACK TO WORK!

TRAINING!

OH. CARRY ON.