DLI Accelerated Data Science Teaching Kit

Lecture 15.8 - RAPIDS Acceleration: PCA, UMAP, DBSCAN

PCA Refresher

Principal Component Analysis

- Linear feature extraction method
- Find orthogonal axes that best “fit” the data points
- Typically use only the first few principal components for visualization
- Principal components are eigenvectors of data’s covariance matrix
- Hyperparameter: n_components

Running PCA in cuML

Similar to Scikit-Learn, we can import a PCA model from cuML and instantiate it.

```
import cuml; print('cuML Version:', cuml.__version__)
from cuml.decomposition import PCA as PCA_GPU


pca_gpu = PCA_GPU(n_components=2)
```

```
cuML Version: 0.12.0a+736.gd722488
```

Running PCA in cuML

We can fit our PCA model to the data using the `fit` method and transform the dataset into principle components using the `transform` method.

```
pca_gpu.fit(X_scaled_cudf)
components_gpu = pca_gpu.transform(X_scaled_cudf).to_pandas().values
components_gpu
```

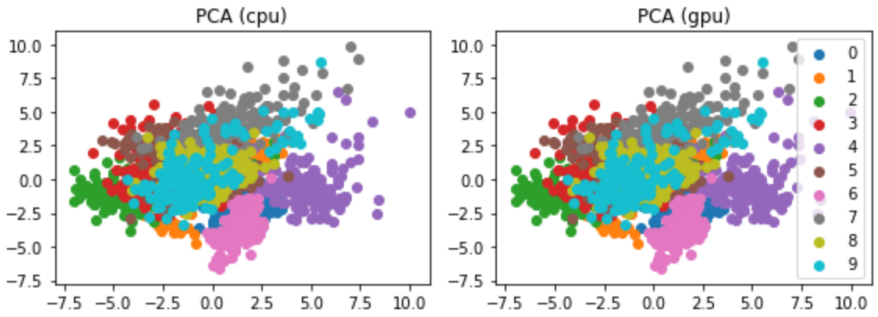
```
array([[ 1.9142139 , -0.9545046 ],
       [ 0.5889803 ,  0.92463815],
       [ 1.3020387 , -0.3171875 ],
       ...,
       [ 1.0225955 , -0.14790928],
       [ 1.0760553 , -0.38090903],
       [-1.2577035 , -2.2275898 ]], dtype=float32)
```

Running PCA in cuML

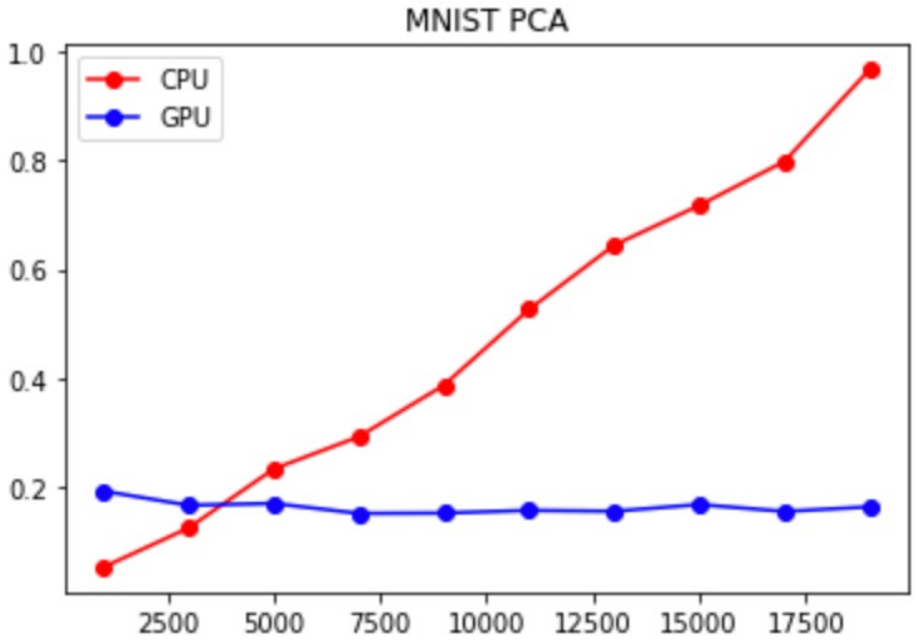
Let's visualize the PCA components generated by the GPU model and compare them to those generated by the CPU model. They should be exactly the same!

```
# create figure
figure = plt.figure()
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 3))

for i in range(10):
    mask = y == i
    ax1.scatter(components[mask, 0], components[mask, 1],
                c=colors[i], label=str(i))
    ax2.scatter(components_gpu[mask, 0], components_gpu[mask, 1],
                c=colors[i], label=str(i))
ax1.set_title('PCA (cpu)')
ax2.set_title('PCA (gpu)')

plt.legend()
plt.tight_layout()
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



Rapids PCA performance



UMAP Overview

Uniform Manifold Approximation Projection

- Dimensional reduction based on neighbor graph
- Project points onto a Uniform Manifold
- Create a topological simplex
- Hyperparameter: n_neighbors, n_components

Running UMAP in cuML

We'll import the `UMAP` class from cuML and instantiate it.

```
from cuml import UMAP as UMAP_GPU


umap_gpu = UMAP_GPU(n_neighbors=10, n_components=2)
```

Running UMAP in cuML

We first use fit_transform() to fit the model and transform the data.

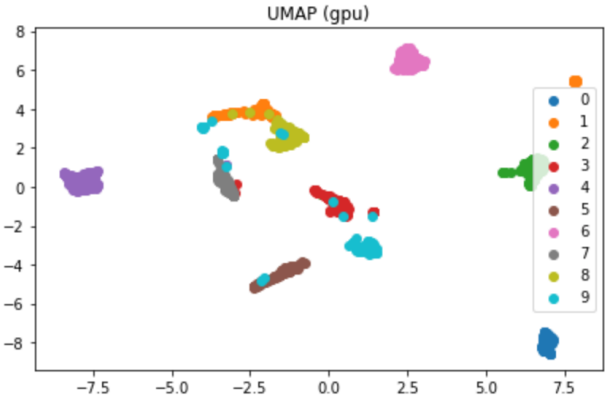
```
components_gpu = umap_gpu.fit_transform(X_cudf).to_pandas().values
components_gpu
```

```
array([[ 6.9769773 , -8.330804  ],
       [-2.398009  ,  4.0054793 ],
       [ 5.586     ,  0.76459444],
       ...,
       [-1.3864897 ,  3.0072045 ],
       [ 1.2171572 , -3.079942  ],
       [-1.1401697 ,  2.5681517 ]], dtype=float32)
```

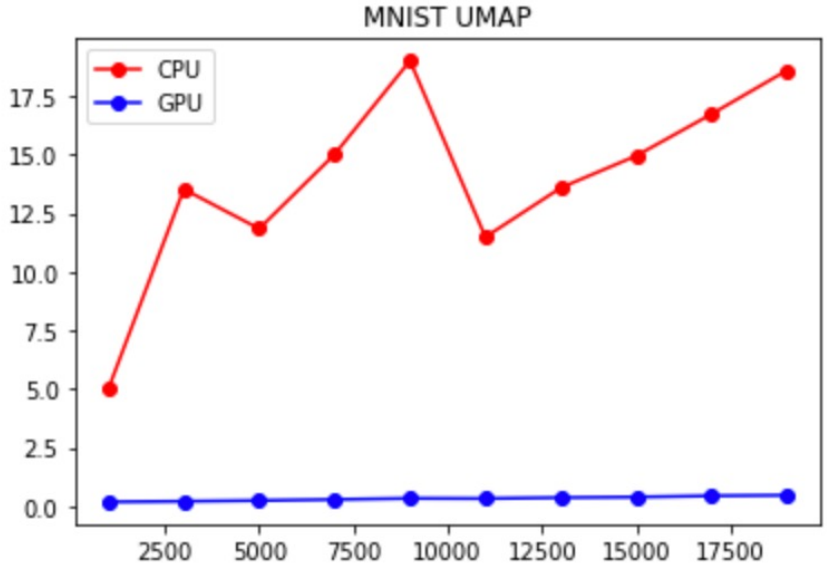
Running UMAP in cuML

Then, we visualize the results

```
# create figure
figure = plt.figure()
axis = figure.add_subplot(111)

for i in range(10):
    mask = y == i
    axis.scatter(components_gpu[mask, 0], components_gpu[mask, 1],
                 c=colors[i], label=str(i))
axis.set_title('UMAP (gpu)')

plt.legend()
plt.tight_layout()
plt.show()
```



RAPIDS UMAP Speed



DBSCAN Refresher

Density-based spatial clustering of applications with noise

- Main idea: closely-packed points (high-density; many neighbors nearby) are grouped
- Points are categorized into three category
  - Core points
  - Reachable points
  - Outlier
- Hyper parameter: eps and minPts

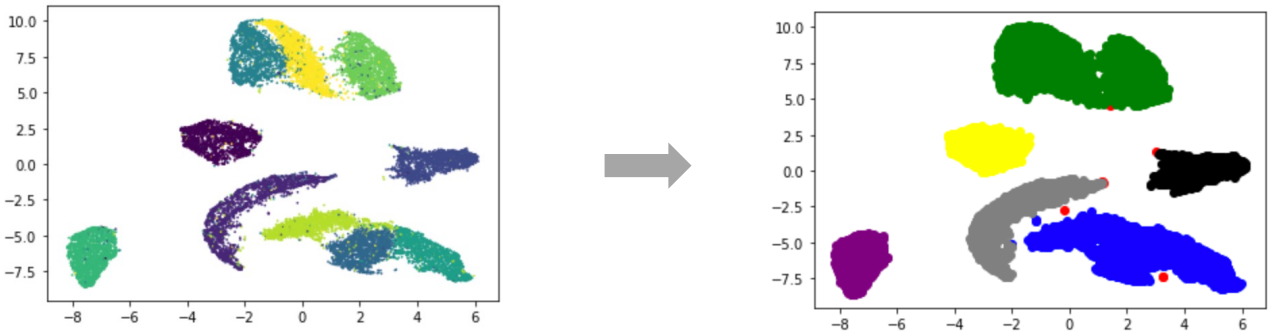
How to run DBSCAN in cuML?

```
# Import Library
from cuml import datasets
import cuml
import matplotlib.pyplot as plt
import numpy as np

# Setup hyperparameter
EPS = 3
MIN_SAMPLE = 300
N_SAMPLES = 1500
```

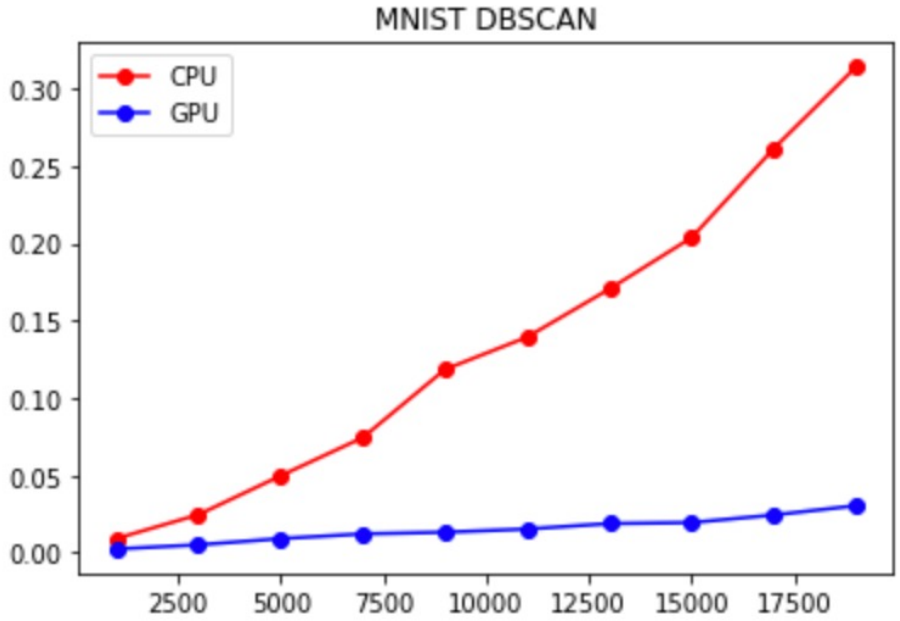
Running DBSCAN in cuML

We use UMAP to first reduce the dimensions of MNIST.
Then, we run DBSCAN on the 2-dimentional data.

```
EPS = 0.75
MIN_SAMPLE = 300
N_SAMPLES = 1500

dbscan_mnist = cuml.DBSCAN(eps=EPS, min_samples=MIN_SAMPLE)
dbscan_mnist.fit(train_test_2D)

colors = ["red", "blue", "green", "black", "yellow", "purple", "gray", "pink", "olive", "indigo", "tan", "honeydew"]

# Visualize data
for ind, label in enumerate(np.unique(dbscan_mnist.labels_)):
  plt.scatter(train_test_2D[np.where(dbscan_mnist.labels_ == label), 0],
              train_test_2D[np.where(dbscan_mnist.labels_ == label), 1], c=colors[ind])
```



RAPIDS DBSCAN Speed



DLI Accelerated Data Science Teaching Kit

Thank You