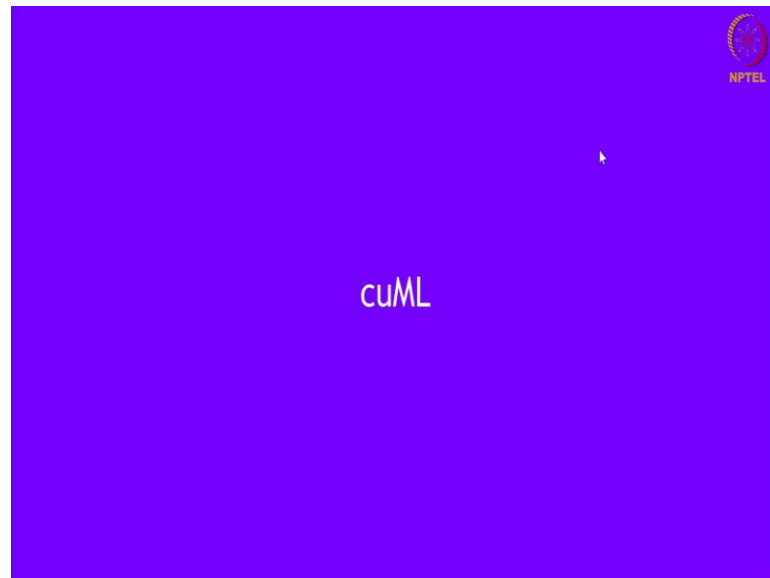


Applied Accelerated Artificial Intelligence
Prof. Bharatkumar Sharma
Department of Computer Science and Engineering
Indian Institute of Technology, Palakkad

Lecture - 48
Accelerated Machine Learning

(Refer Slide Time: 00:18)



Welcome back everyone to the next lecture on the same topic that we have been working on, which is accelerated data analytics. Our previous lectures the last lecture that we did was dedicated towards using the cuDF which is the CUDA related data frame environment to accelerate the first stage of the overall data analytics pipeline.

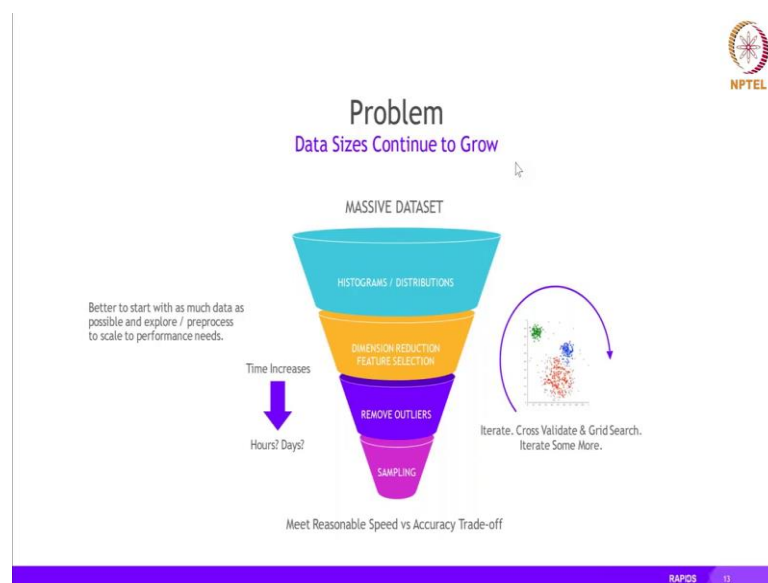
Today we are going to work on additional I would say the stages of the accelerated data analytics pipeline, the second one in this pipeline is acceleration of the machine learning algorithms themselves followed by the last one, which is which is the acceleration of overall task not just on 1 GPU, but across the whole system.

So, let us get started. Hopefully you remember the overall stack that we saw last time and today we are going to concentrate on the second window as you can see here, which is the machine learning part of it. So, we had seen the overall scenarios under which we want acceleration we talked about relationship of the increasing data set size to the amount of computation power which is required.

And if I talk it from a point of view of data science perspective itself, it is generally recommended to start with as much as data as possible and then keep on reducing that data and explore possibilities to pre-process it or scale to meet the performance demands. Because it is a known fact that the datasets keep on increasing, but then we use different techniques to reach a particular level, where we are saying that we are able to meet the accuracy to the speed that we want to deploy it in the real world.

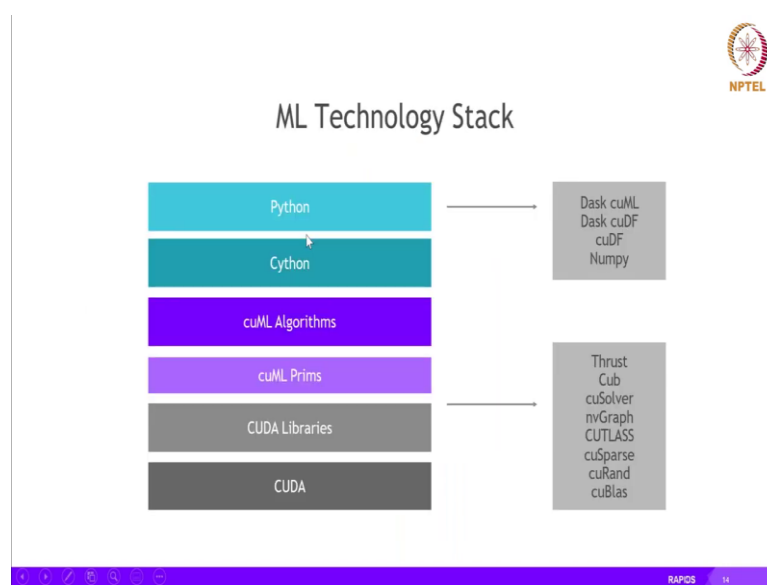
So, we start with the larger data set, we apply certain pre-processing techniques maybe create histograms or distributions around it. We apply techniques which will reduce the overall dimensions, we only select the features which we feel are important which are giving us really good accuracy levels. We try to remove as many outliers as possible, we may in fact remove certain samples and only have at a particular frequency and if you see this is basically the whole objective is to reduce the overall time as much as possible.

(Refer Slide Time: 03:26)



But without having to sacrifice on the part of on the part of the on the part of the I would say. So, to get as much as accuracy as possible, without having to sacrifice on also taking a lot of time or the speed. But the critical thing here to understand is no matter what we say it is an iterative process and what you require is basically something which helps you in this iteration as fast as possible. So, what you are trying to do is you are trying to do basically validation and grid search and do a lot of iteration to go through this overall process.

(Refer Slide Time: 04:00)



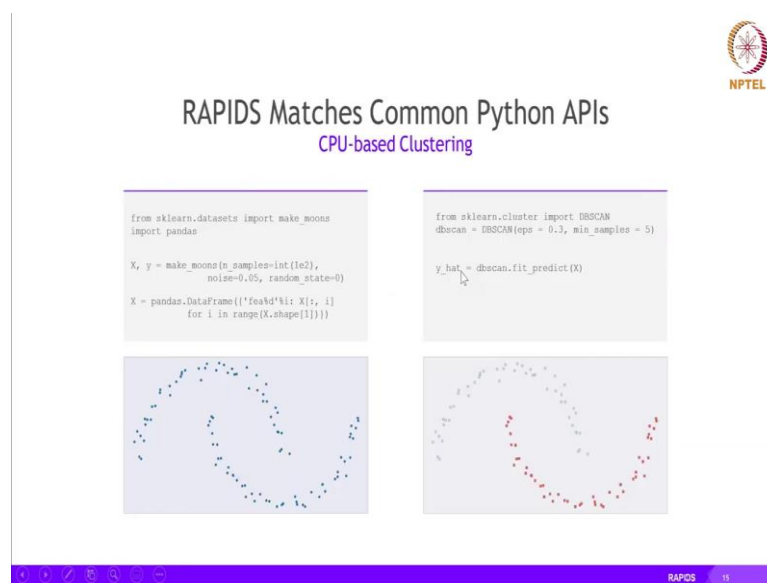
So, to solve one of this problem, we are talking about accelerated ML technology stack, again at the bottommost layer what we have is CUDA which is compute unified device architecture which is the GPU architecture, which exposes parallelism. Over that we have a couple of libraries, as we said last time also like similar to cuDF these libraries are primarily written in C or C++ thrust which is the library for which is a STL equivalent on GPU standard template library running on the GPU.

We have different other libraries like nvgraph for doing graph related problems and cuRand for random number generators and many more cuBlas for basic linear algebra operations and so on and so forth.

All these basic libraries you have higher level libraries which kind of provides a cuML related primitives or algorithms, which are still C, C++ based and then these are exposed to the Python programmer which is the primary language in Python for all of the data scientist in form of wrappers. So, basically, they will internally use Cython to basically call the ML based algorithms which are there.

So, the stack which we showed in cuDF here, it is kind of similar here as well, we are having the bottom most layer or parallel computing above is the C++ libraries, because it is the most prominent one giving the highest amount of performance. And then the Python basically is calling those C++ libraries using the Cython bindings.

(Refer Slide Time: 05:53)

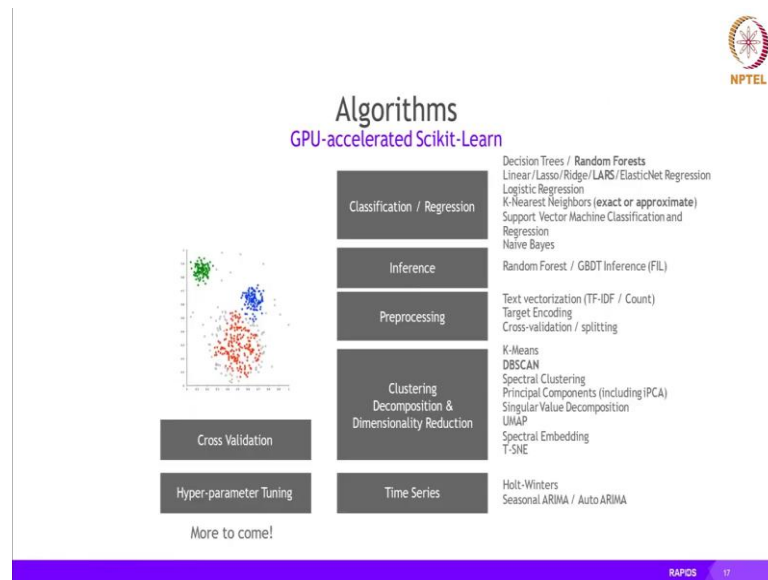


So, let us look at an example of the RAPIDS and in this particular case we are still using C++ equivalent version which is sklearn. You can see here or scikit learn which is the most popular machine learning package and we are using pandas here. So, we are creating this is more of a clustering based thing.

So, you can see here you are calling the, you are creating the pandas and then you call the sklearn clustering algorithm, here we are using DBSCAN and we do the fit predict which kind of highlights the two clusters which exist at this particular data set, that we have. But just to highlight, there is a small change we are going to look at a realistic example in the Jupyter Notebooks, like we showed it to you last time.

But in short what we are trying to say here is that the idea behind RAPIDS is that there is minimal change in the libraries, calls or the source code that you write which are traditionally being written using sequential or a CPU based platform. And you just have to change, import and you have to change very minimal changes you have to do to your code to make use of GPU. So, only here you can see there are only 2-3 changes that you need to do.

(Refer Slide Time: 07:26)

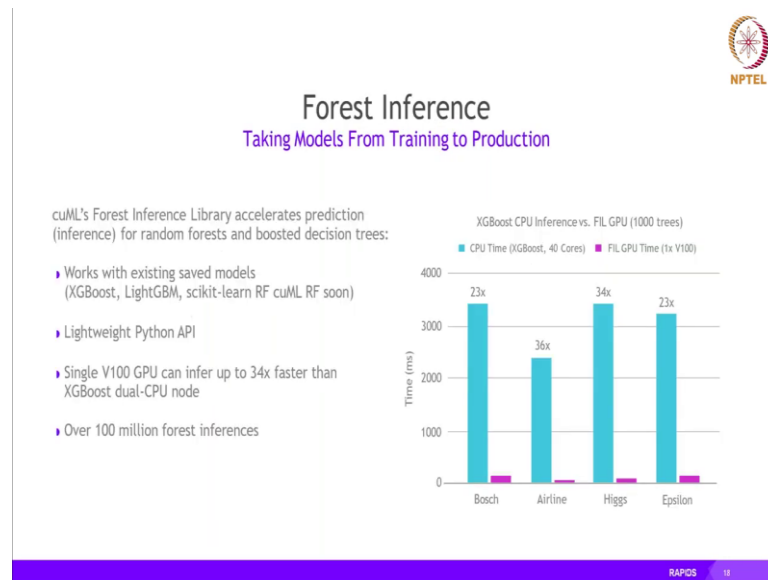


So, over time cuML has evolved, I think it is still having a subset of things which exist in scikit learn, sklearn you have different examples for classification or regression you have APIs for inferencing, pre-processing, also you have the APIs or algorithms particularly in the clustering algorithm or dimensionally reduction or to handle time series data.

And we are also going to look at an example of hyper parameter tuning and cross validation also in the demo, that I am going to show you. So, RAPIDS is the evolving I would say initiative, open source initiative there are many algorithms which are already existing and many keep on getting added over time. So, this might not be the complete list by the time you are actually going to use it or by the time I am showcasing it to you and more algorithms would have been added by this time.

So, it is always recommended to go to the open source project and see which all machine learning algorithms are supported.


(Refer Slide Time: 08:40)



But in to show you certain examples, like we showed it to you last time also this is an example of forest inferencing and you can see here this work is basically an output of a 100 million forest inferencing done for different data sets. And you can see here the time taken by the GPU versus the CPU on how much speedups you can expect while running only the machine learning part, I am not talking about the overall pipeline.

Yesterday we saw the initial part of data frame and how you can do those pre-processing quite efficiently using cuDF, this is accelerating only the cuML or the machine learning algorithm part in general.

(Refer Slide Time: 09:28)



RAPIDS Integrated into Cloud ML Frameworks

Accelerated machine learning models in RAPIDS give you the flexibility to use hyperparameter optimization (HPO) experiments to explore all variants to find the most accurate possible model for your problem.

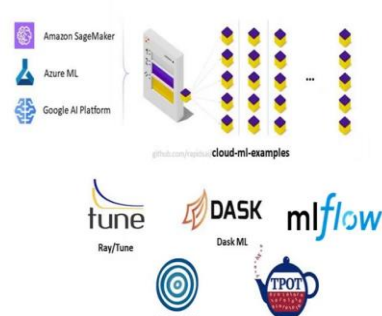
With GPU acceleration, RAPIDS models can train 25x faster than CPU equivalents, enabling more experimentation in less time.

The RAPIDS team works closely with major cloud providers and OSS solution providers to provide code samples to get started with HPO in minutes.

<https://rapids.ai/hpo>

MANY PATHS TO RAPIDS HPO

RAPIDS Integration into Cloud/Distributed Frameworks



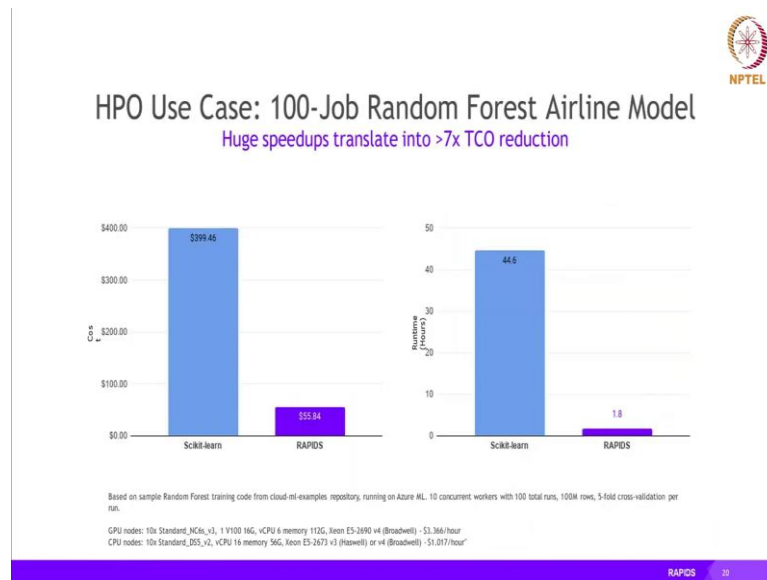
github.com/rapidsai/cloud-ml-examples

RAPIDS 19

The good part about RAPIDS is also that it is integrated with most of the cloud ML frameworks. So, if you are using any of the cloud frameworks you can go to rapids.ai/hpo and you can basically look at where all this acceleration has happened on say Amazon SageMaker, Azure ML or Google AI platform.

And you can go to [github dot io rapids ai cloud ML examples](https://github.com/rapidsai/cloud-ml-examples) to see how to use these in different platforms or primarily here you can see the three key platforms which are there. So, there is an integration of RAPIDS which is already happened in all of these platforms also.

(Refer Slide Time: 10:15)




So, this is another example again this is a 100-job Random Forest Airline Model, you can see here one of the critical thing that I mentioned sometime back also to you is the artificial intelligence is not just about improving accuracy, but showing the return on investment overall.

And this is one of the most critical thing, is to see that if you were to do a particular training for different, here you can see the airline model of 100 job random forest and if you were to do it on a platform which is pure sequential and you were to charge on say the same on a cloud based scenario and then they may be in different cloud models.

Here you are seeing an example of azure ML versus if you are spending the same effort and running the same thing on a GPU based system, what kind of a return on investment that you can do or how much money you can save when you are deploying your model.

And that is one of the most critical thing which is very very much prominent why accelerated computing is also required about having a real world impact.

(Refer Slide Time: 11:32)



Overview of cuML Algorithms

RAPIDS 0.18 - February 2021

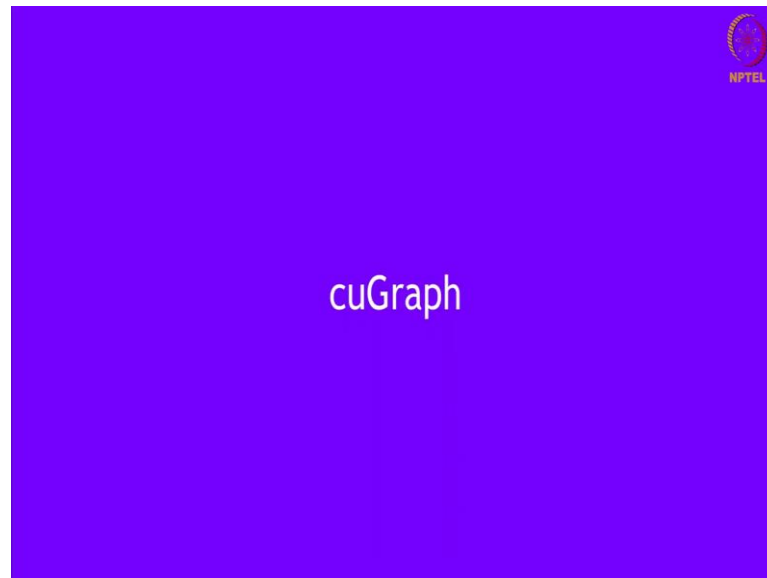
cuML	Single-GPU	Multi-Node-Multi-GPU
Gradient Boosted Decision Trees (GBDT)		
Linear Regression		
Logistic Regression		
Random Forest		
K-Means		
K-NN		
DBSCAN		
UMAP		
Holt-Winters		
ARIMA		
T-SNE		
Principal Components		
Singular Value Decomposition		
SVM		

RAPIDS 31

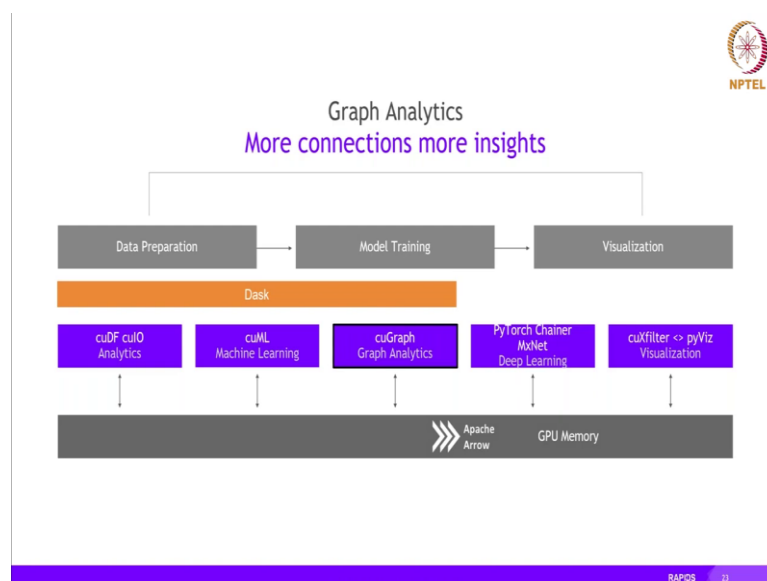
And as I told you from a point of view of ML algorithms, this is just a list of some of those. When I talk about the functionality some of those functionalities are primarily supported on single GPU, which means it will run only on a single GPU, while some of them are supported on multi node multi GPU environment.

So, even if algorithm support is there, it is always good idea to see if the functionality supports single GPU or multi node multi GPU kind of environment also and this list keeps on changing. As I said we are in 2022 and this already this particular support for other cuML based framework and also multi node support might have already been established at this particular point of time.

(Refer Slide Time: 12:24)




(Refer Slide Time: 12:34)



I also wanted to highlight one more part here which is cuGraph. So, if you remember some I we showed you the overall, there are different components inside RAPIDS one of the component there is also in case you are working in graph analytics, in that case there is a support inside RAPIDS for cuGraph library.

(Refer Slide Time: 12:51)



GOALS AND BENEFITS OF CUGRAPH

Focus on Features and User Experience

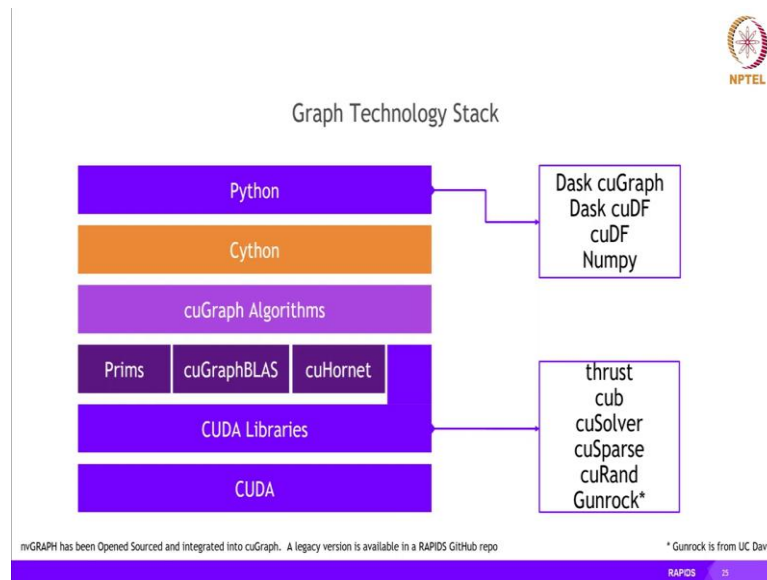
Breakthrough Performance <ul style="list-style-type: none">• Up to 500 million edges on a single 32GB GPU• Multi-GPU support for scaling into the billions of edges	Multiple APIs <ul style="list-style-type: none">• Python: Familiar NetworkX-like API• C/C++: lower-level granular control for application developers
Seamless Integration with cuDF and cuML <ul style="list-style-type: none">• Property Graph support via DataFrames	Growing Functionality <ul style="list-style-type: none">• Extensive collection of algorithm, primitive, and utility functions

RAPIDS 34

And the idea is basically to make use of GPU computing to for solving graph problems and you can see here that you can do 500 million edges calculation on a 1 single GPU of 32 GB and it can support billions of edges in a multi GPU environment as well. And it is seamlessly integrated with cuDF and cuML also.

And there are various algorithms which are happening, but it is equivalent to the network X API, which is one of the most popular ones for graph analytics in the sequential world and again it has the Python equivalent for networkX, but lower level at the lower level it calls the C++ API.

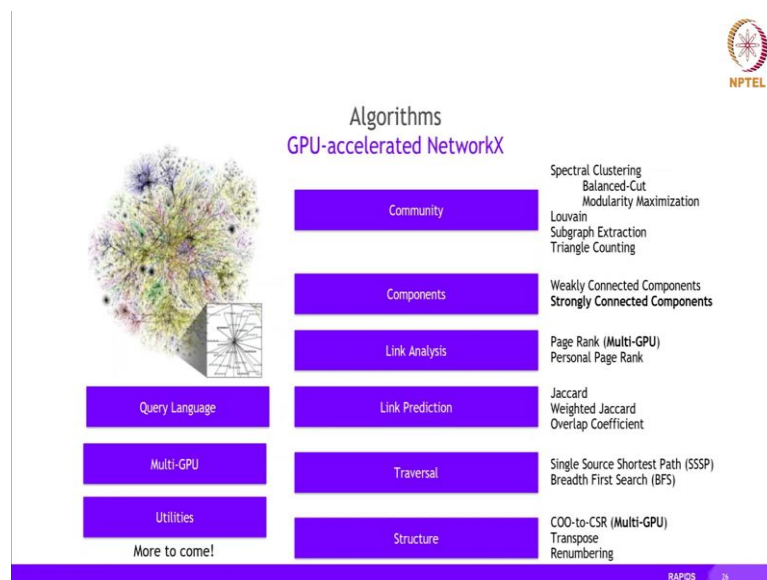
(Refer Slide Time: 13:41)



Same thing like we saw cuML, there is the same stack which exists at the lower level you have CUDA architecture you have the CUDA libraries.

Which are primarily C, C++ libraries and then there are a DASK based or cuDF based extension for the graph, which finally uses Python to call the C, C++, APIs that we have.

(Refer Slide Time: 14:07)



And there are various things, you can do with it, it is again community based and you can create different kinds of components like strongly connected, weakly connected, you can do various kinds of analysis, which if you are working in this field you know like

creating page ranks right or if you would like to do traversals like shortest path breadth first search and various other kinds of things are possible.


For doing graph analytics there are various things which are already there are more which are coming and it will be supported in the future as well.

(Refer Slide Time: 14:44)



But you can see the kind of speedup that we are expecting here, for different kinds of routines which are present in the networkX and you can see here that we are talking about speedup numbers performance speedup numbers in few thousands as compared to few hundreds also. So, we are looking at numbers which can accelerate the graph analytics at you can see here up to 11000 also for certain data sets which are available.

(Refer Slide Time: 15:17)



Road to 1.0
March 2020 - RAPIDS 0.14

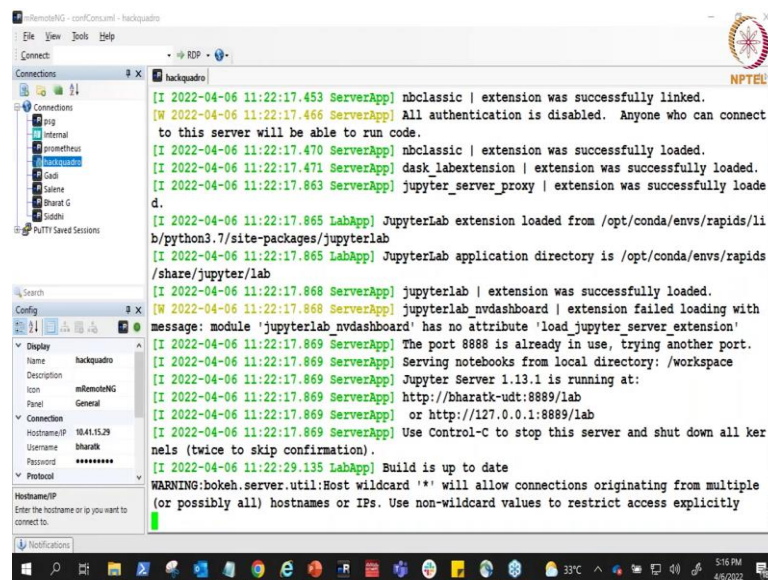
cuGraph	Single-GPU	Multi-GPU	Multi-Node Multi-GPU
Graph and Graphlet Counts			
Page Rank			
Betweenness Centrality			
WCC			
WCC			
Triangle Counting			
Network Compression			
Node Partitioning			
Graph Partitioning			
Connected Components (Weak and Strong)			
Travelling			
Spectral Clustering			
Centrality			
Graphlet			

RAPIDS 18

And again, the same thing there are various kinds of algorithms available, sorry for the font, which may not be visible to you, but it kind of highlights various kinds of algorithms used in graph analytics like page ranking breadth first search and spectral clustering and various other kinds of methods which are there.

And again there are support for some of them on single GPU, some of them are supported on multi GPU within the same node or some of them are also supported on multi GPU multi node. So, with that let me quickly just highlight the part of showing you a demo.

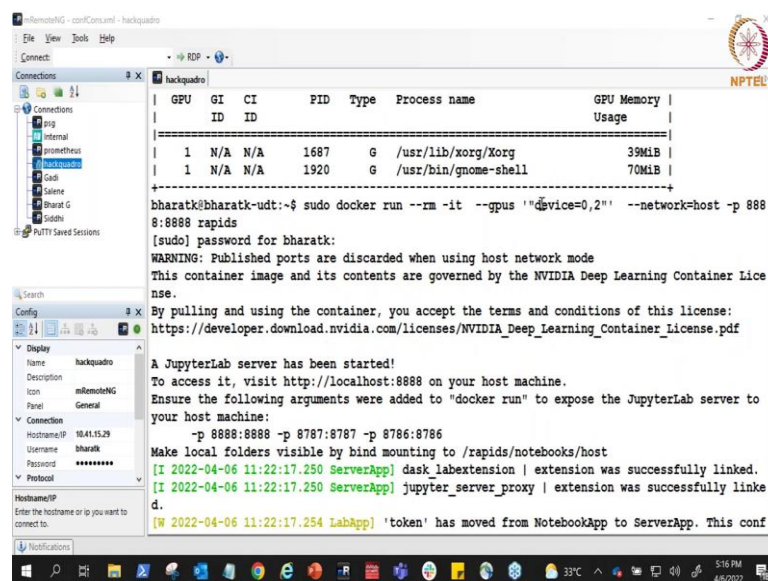
(Refer Slide Time: 16:13)



The screenshot shows the Hackquadro application window. The left sidebar contains a 'Connections' list with 'hackquadro' selected. The main window displays a log of events. The log shows the following messages:

```
[I 2022-04-06 11:22:17.453 ServerApp] nbclassic | extension was successfully linked.
[W 2022-04-06 11:22:17.466 ServerApp] All authentication is disabled. Anyone who can connect
to this server will be able to run code.
[I 2022-04-06 11:22:17.470 ServerApp] nbclassic | extension was successfully loaded.
[I 2022-04-06 11:22:17.471 ServerApp] dask_labextension | extension was successfully loaded.
[I 2022-04-06 11:22:17.863 ServerApp] jupyter_server_proxy | extension was successfully load
e d.
[I 2022-04-06 11:22:17.865 LabApp] JupyterLab extension loaded from /opt/conda/envs/rapids/li
b/python3.7/site-packages/jupyterlab
[I 2022-04-06 11:22:17.865 LabApp] JupyterLab application directory is /opt/conda/envs/rapids
/share/jupyter/lab
[I 2022-04-06 11:22:17.868 ServerApp] jupyterlab | extension was successfully loaded.
[W 2022-04-06 11:22:17.868 ServerApp] jupyterlab_nvdashboard | extension failed loading with
message: module 'jupyterlab_nvdashboard' has no attribute 'load_jupyter_server_extension'
[I 2022-04-06 11:22:17.869 ServerApp] The port 8888 is already in use, trying another port.
[I 2022-04-06 11:22:17.869 ServerApp] Serving notebooks from local directory: /workspace
[I 2022-04-06 11:22:17.869 ServerApp] Jupyter Server 1.13.1 is running at:
[I 2022-04-06 11:22:17.869 ServerApp] http://bharatk-udt:8889/lab
[I 2022-04-06 11:22:17.869 ServerApp] or http://127.0.0.1:8889/lab
[I 2022-04-06 11:22:17.869 ServerApp] Use Control-C to stop this server and shut down all ker
nels (twice to skip confirmation).
[I 2022-04-06 11:22:29.135 LabApp] Build is up to date
WARNING:bokeh.server.util:Host wildcard '*' will allow connections originating from multiple
(or possibly all) hostnames or IPs. Use non-wildcard values to restrict access explicitly
```

(Refer Slide Time: 16:21)



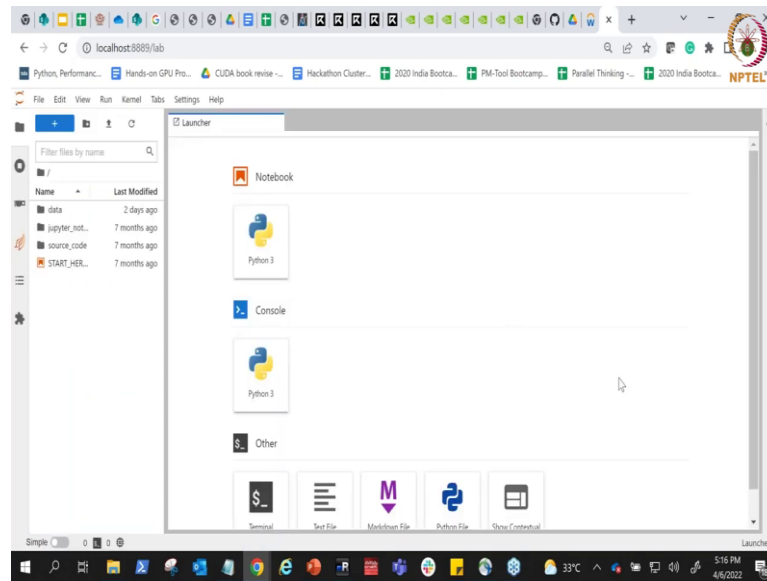
The screenshot shows the Hackquadro application window. The left sidebar contains a 'Connections' list with 'hackquadro' selected. The main window displays a terminal output. The terminal shows the following commands and output:

```
bharatk@bharatk-udt:~$ sudo docker run --rm -it --gpus '"device=0,2"' --network=host -p 888
8:8888 rapids
[sudo] password for bharatk:
WARNING: Published ports are discarded when using host network mode
This container image and its contents are governed by the NVIDIA Deep Learning Container Lice
nse.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.download.nvidia.com/licenses/NVIDIA_Deep_Learning_Container_License.pdf

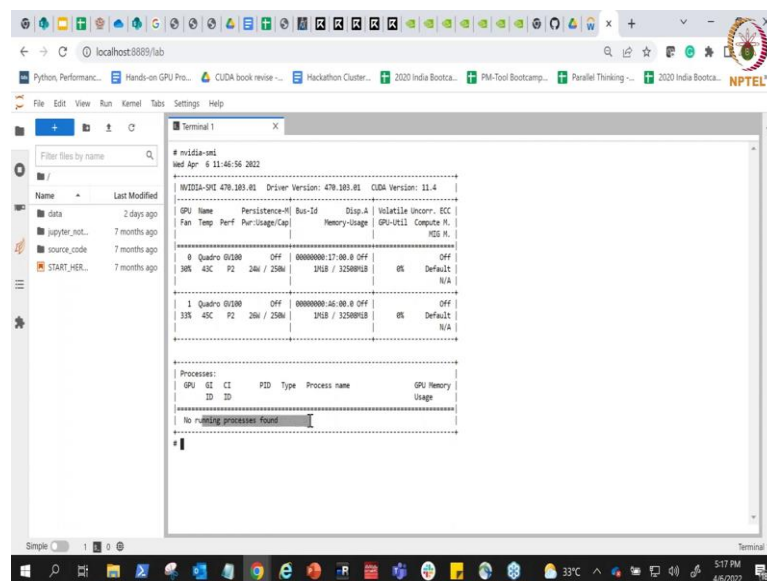
A JupyterLab server has been started!
To access it, visit http://localhost:8888 on your host machine.
Ensure the following arguments were added to "docker run" to expose the JupyterLab server to
your host machine:
-p 8888:8888 -p 8787:8787 -p 8786:8786
Make local folders visible by bind mounting to /rapids/notebooks/host
[I 2022-04-06 11:22:17.250 ServerApp] dask_labextension | extension was successfully linked.
[I 2022-04-06 11:22:17.250 ServerApp] jupyter_server_proxy | extension was successfully linke
d.
[W 2022-04-06 11:22:17.254 LabApp] 'token' has moved from NotebookApp to ServerApp. This conf
```

So, just give me a second and I can bring it back to you. So, like last time, I have already run the container the same container, which I had run last time and it is running inside the docker container and I am exposing it to two GPUs. So, I am going to just go ahead and just refresh the scene, just to make sure that it is still connected.

(Refer Slide Time: 16:33)

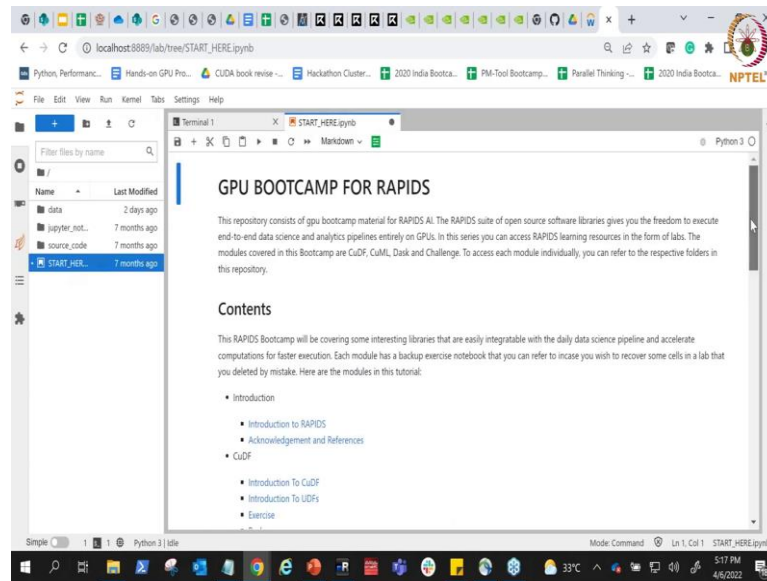


(Refer Slide Time: 16:36)

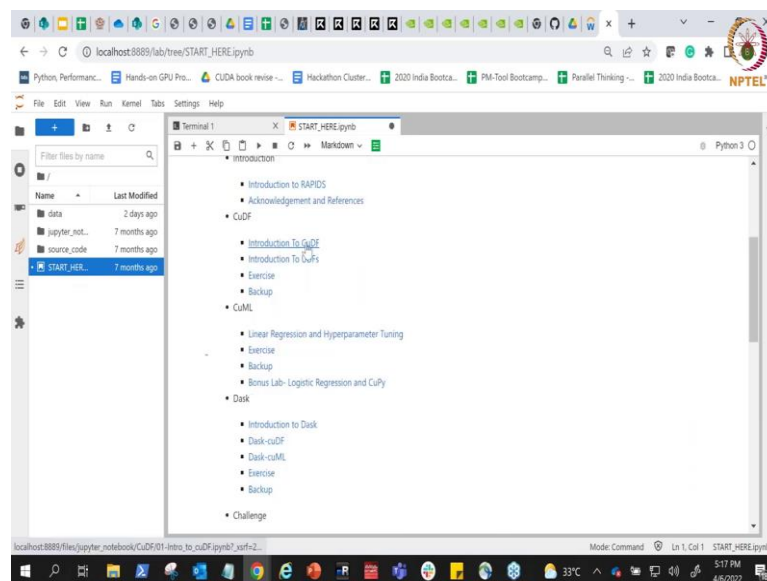


So yes, it is still connected I have my Jupyter Notebook and let me see, how many GPUs I have at my disposal. You can see here currently, I have exposed only two GPUs to it which is of volta architecture Quadro GV 100 and there is currently no process running on it at this time.

(Refer Slide Time: 16:54)

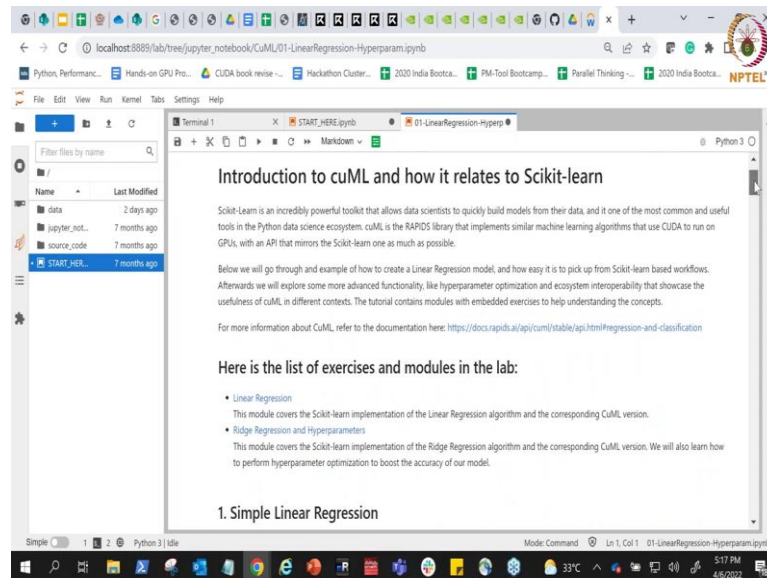


(Refer Slide Time: 16:57)



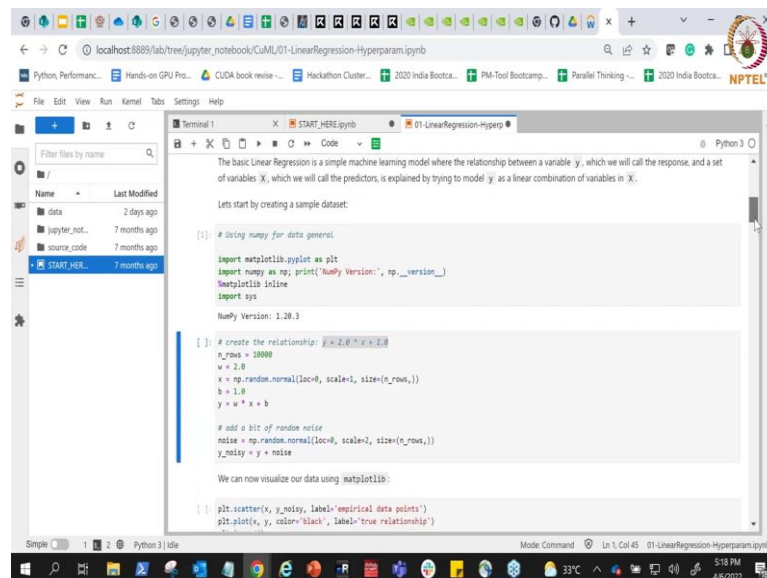
So, yesterday we saw hands on of particularly cuDF, today I am going to show you certain characteristics of how to use cuML.

(Refer Slide Time: 17:08)



So, for cuML as I said it is primarily a replacement or the accelerated version of scikit learn and it is what I am going to show you.

(Refer Slide Time: 17:21)

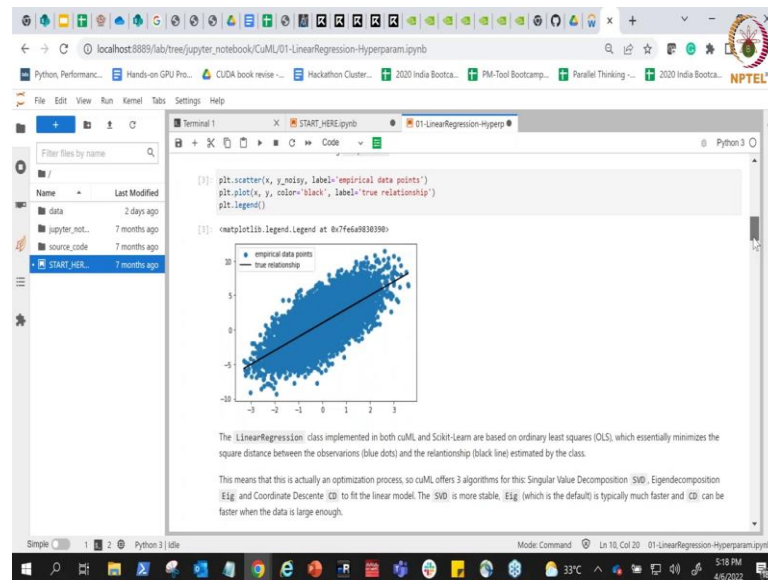


So, the example that I am going to show you first is a simple linear regression example, then we will enhance it and we will end up the cuML documentation with how to do hyper parameter tuning using say a grid search algorithm as well.

So, again here, this is a simple example of CPU where you have imported numpy matplotlib for visualization. And if you can see here what we are doing is we are trying

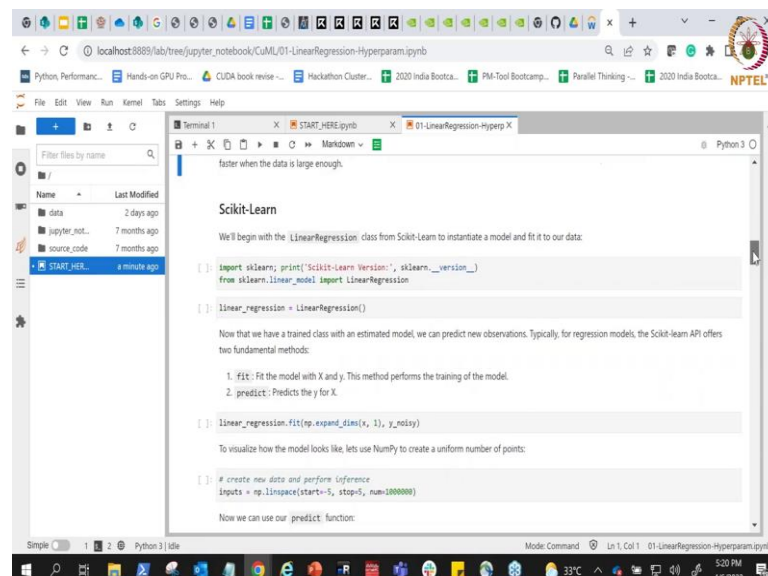
to create a data set which kind of adheres to the linear relationship and. So, we are just creating some random values, but it they are kind of linear in nature, having a linear relationship to each other.

(Refer Slide Time: 18:05)



And let us just plot the values. So, you can see here this is our actual true relationship and we have also added some noise to it also. So, you can see here we have added some noise to it also. So, you will see the empirical data points around it, but this is the true nature of the relationship that is existing, based on which we created the dataset.

(Refer Slide Time: 18:32)

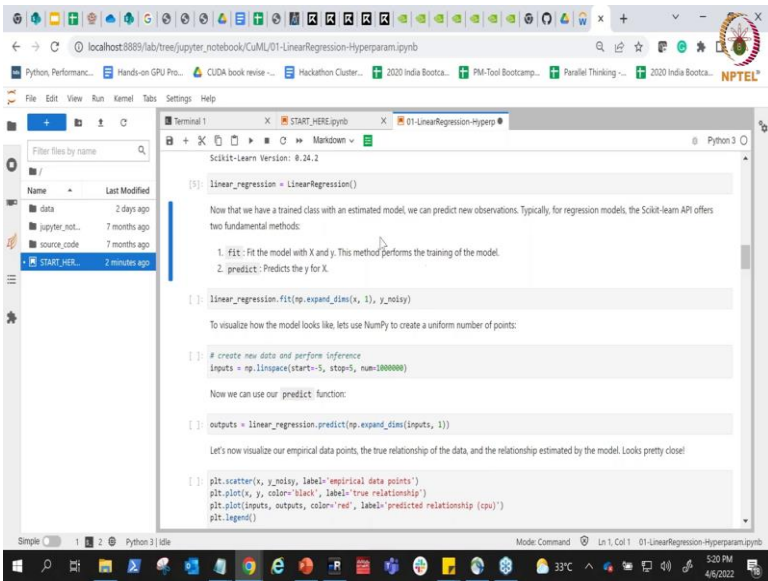


So, what we are looking at is; obviously, we know it is a linear relationship and that is why we are going to use the linear regression which is primarily a class, which is implemented both in the CPU version of it, which is the scikit learn and also the cuML version of it. But what we have to understand is whenever we are using any package behind the scene, every module can offer different kinds of algorithms to solve the same thing, even for linear regression just to give an example.

Like for cuML we basically provide 3 algorithms to find the relationship, it may use singular value decomposition, it may use eigen method or it may use a coordinate decent method to fit the linear model right. And this is very very critical to know because when you start moving your model from say a scikit base to a cuML, the default algorithm which are chosen by the cuML might be different from the sequential version.

And you might get certain different results as compared to what you would see in the sequential algorithm, that you are using. Like for cuML basically we find that SVD is more stable generally found out, but eigen is the one which is the default right. Because it is much more faster as compared to say doing a SVD based thing, right. So, these are just certain examples of what happens behind the scene, by default you are going to use some default versions, but you can override it, to use the same algorithm behind the scene which you have been traditionally using, right.

(Refer Slide Time: 20:24)



The screenshot shows a Jupyter Notebook titled '01-LinearRegression-Hyperparam.ipynb' running in a web browser. The notebook content includes the following code and text:

```
[5]: linear_regression = LinearRegression()

Now that we have a trained class with an estimated model, we can predict new observations. Typically, for regression models, the Scikit-learn API offers two fundamental methods:

1. fit: Fit the model with X and y. This method performs the training of the model.
2. predict: Predicts the y for X.

[1]: linear_regression.fit(np.expand_dims(x, 1), y_noisy)

To visualize how the model looks like, let's use NumPy to create a uniform number of points:

[ ]: # create new data and perform inference
inputs = np.linspace(start=-5, stop=5, num=1000000)

Now we can use our predict function:

[ ]: outputs = linear_regression.predict(np.expand_dims(inputs, 1))

Let's now visualize our empirical data points, the true relationship of the data, and the relationship estimated by the model. Looks pretty close!

[ ]: plt.scatter(x, y_noisy, label='empirical data points')
plt.plot(x, y, color='black', label='true relationship')
plt.plot(inputs, outputs, color='red', label='predicted relationship (cpu)')
plt.legend()
```

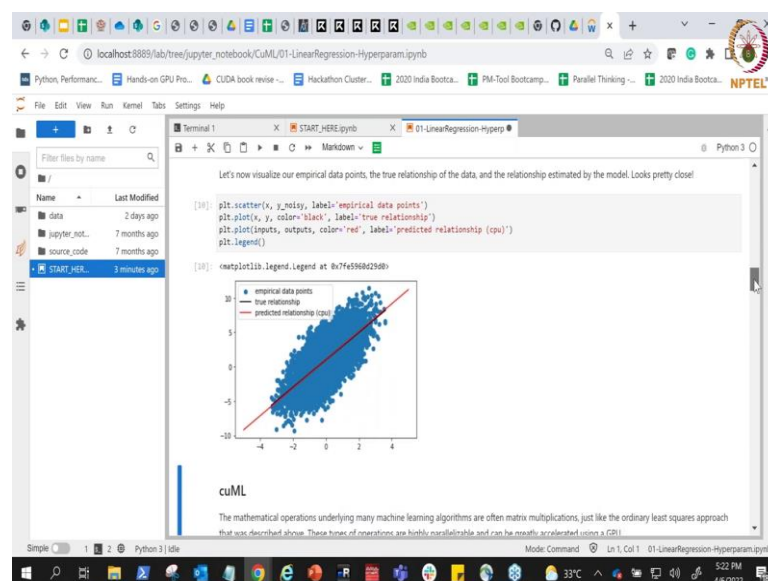
The interface also shows a file explorer on the left with files like 'data', 'jupyter_notebook', 'source_code', and 'START_HERE'. The bottom status bar indicates 'Python 3 | Mode: Command | Ln 1, Col 1 | 01-LinearRegression-Hyperparam.ipynb' and the system clock shows '33°C' and '1:20 PM 4/6/2022'.

So, here as you can see we are importing sklearn dot linear model and importing it as linear regression, we are creating an object of it. And in if you have been working in this field of machine learning, generally we call a fit function, the fit function is the one which is going to basically find out the relationship or the parameters and it is going to do the training part of your cycle of artificial intelligence.

So, you do the fit function first and then you use predict for seeing how well your model have actually learned. So, you can see here we are calling the fit function on the data set that we have and in the end if you want to see. So, what we are doing is we are creating a new data again for inferencing or for testing the relationship.

And so once you have created the new data. So, we are using the linear regression dot predict to see how well our data can predict the newer data set that we are exposing it to. So, we have predicted the value and finally in the end what we will do is to use matplotlib to scatter and see how well our sklearn is able to predict.

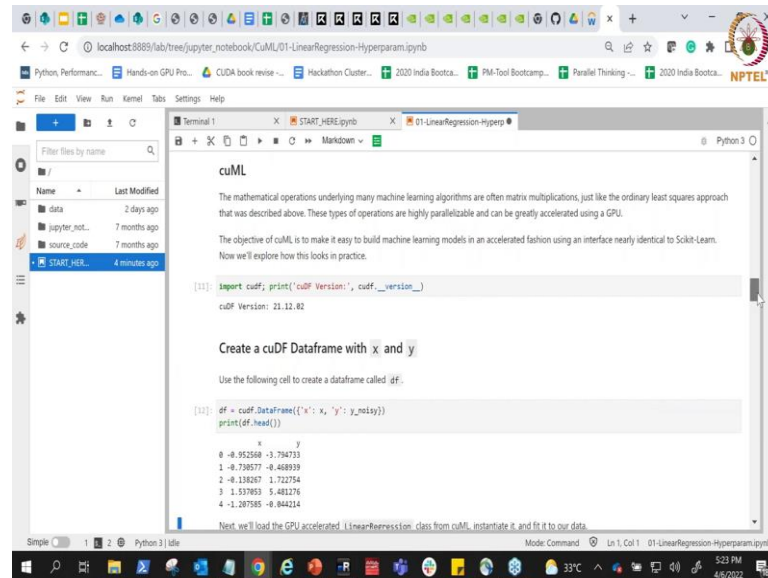
(Refer Slide Time: 21:40)



So, as you can see here there are three things, one is the empirical data points, the second one is the true relationship that we know already exist which we had proposed earlier which is the black line, which is kind of almost completely hidden with the predicted relationship, which is the sklearn based predict value.

So, you can see here the predicted value and the true relationship is kind of almost overlapping with each other, hence we can say that that they are actually doing a good job in terms of predicting the for the test for the inferencing part of it.

(Refer Slide Time: 22:21)

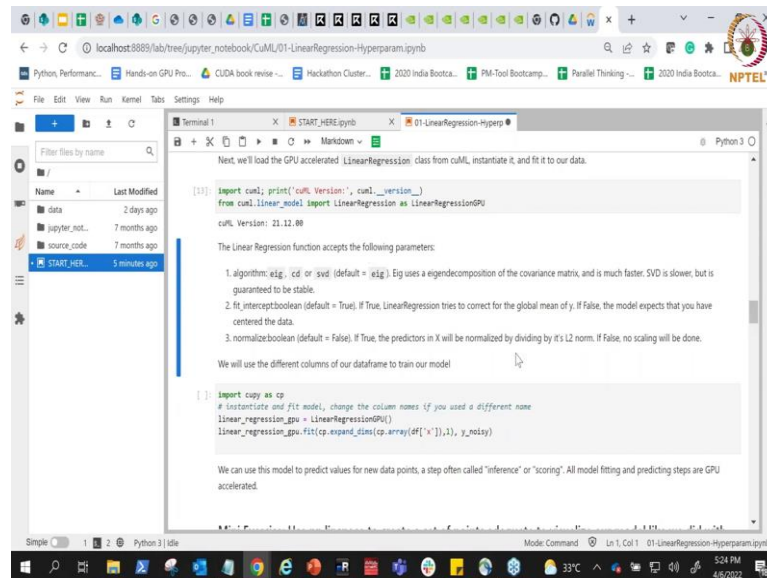


The screenshot shows a Jupyter Notebook interface with a file browser on the left and a code editor on the right. The code editor displays the cuML documentation and the execution of a code cell. The code cell imports cuDF and prints its version, then creates a cuDF DataFrame with columns 'x' and 'y' and prints its head. The output shows the cuDF version as 21.12.02 and the head of the DataFrame as follows:

	x	y
0	-8.952568	-3.794733
1	-8.738577	-8.468939
2	-8.138267	1.722754
3	1.337693	5.482276
4	-1.287585	-8.844214

Now, let us move towards the; so far we have been running it on the CPU. Now let us do the same thing on the GPU, the idea is to have the code which you have seen earlier almost similar as much similar to the sequential world and let us. So, we are importing cuDF that we have seen last time also. So, we are creating a cuDF data frame. Why do we create in cuDF? Again as we have told earlier cuDF will basically allocate the memory on the GPU and you will be basically doing stuff on the GPU.

(Refer Slide Time: 23:01)



The screenshot shows a Jupyter Notebook titled '01-LinearRegression-Hyperparam.ipynb'. The code in the cell is as follows:

```
[13]: import cuml; print('cuml Version:', cuml.__version__)
      from cuml.linear_model import LinearRegression as LinearRegressionGPU

cuml Version: 21.12.00

The Linear Regression function accepts the following parameters:

1. algorithm: {sfg, cd or jvd} (default = sfg). Sfg uses an eigendecomposition of the covariance matrix, and is much faster. SVD is slower, but is
   guaranteed to be stable.
2. fit_intercept: boolean (default = True). If True, LinearRegression tries to correct for the global mean of y. If False, the model expects that you have
   centered the data.
3. normalize: boolean (default = False). If True, the predictors in X will be normalized by dividing by it's L2 norm. If False, no scaling will be done.

We will use the different columns of our dataframe to train our model

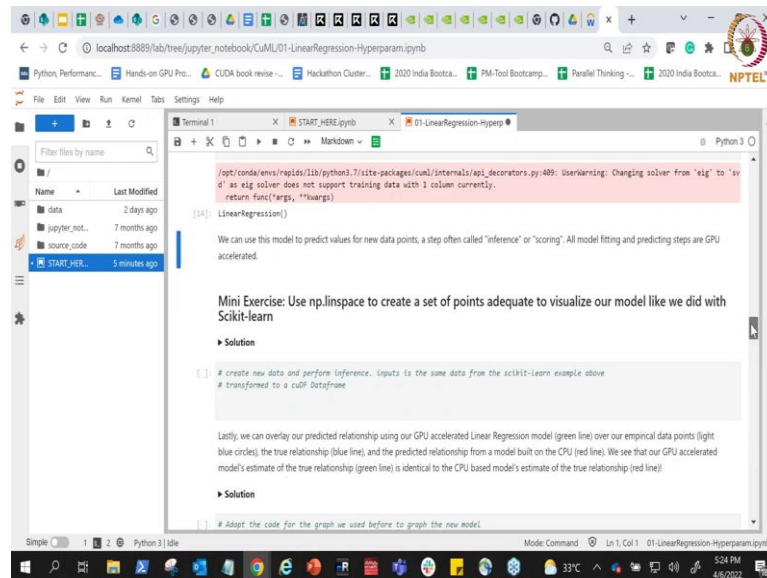
[ ]: import copy as cp
      # instantiate and fit model, change the column names if you used a different name
      linear_regression_gpu = LinearRegressionGPU()
      linear_regression_gpu.fit(cp.expand_dims(cp.array([v])), 1, y_misay)
```

Below the code, there is explanatory text: 'We can use this model to predict values for new data points, a step often called "inference" or "scoring". All model fitting and predicting steps are GPU accelerated.'

So, ideally, we should be passing it a cudf expression and again like sklearn, instead of using sklearn dot linear model we are using cuML dot linear model and we are basically importing it. And as I told you earlier it accepts various parameters, but where you are going to use the default version, but you can override it like the algorithm that you can use is the eigenvalue or the SVD 1, the default one is eigen.

If you want to set certain additional parameter like normal do you want to normalize the data or if you want to do fit intercept and all by default it is true, if you want to if you want to correct it then you can also set it to false. So, there are different parameters by default we are using the default ones, but if you would see that your linear regression of scikit learn may use something else, while ours will use a different version and you might see different accuracy levels.

(Refer Slide Time: 24:02)



```
/opt/conda/envs/rapids/lib/python3.7/site-packages/cuml/interiors/api_decorators.py:489: UserWarning: Changing solver from 'eig' to 'svd' as eig solver does not support training data with 1 column currently.
  return func(*args, **kwargs)

[34]: LinearRegression()

We can use this model to predict values for new data points, a step often called "inference" or "scoring". All model fitting and predicting steps are GPU accelerated.

Mini Exercise: Use np.linspace to create a set of points adequate to visualize our model like we did with Scikit-learn

> Solution

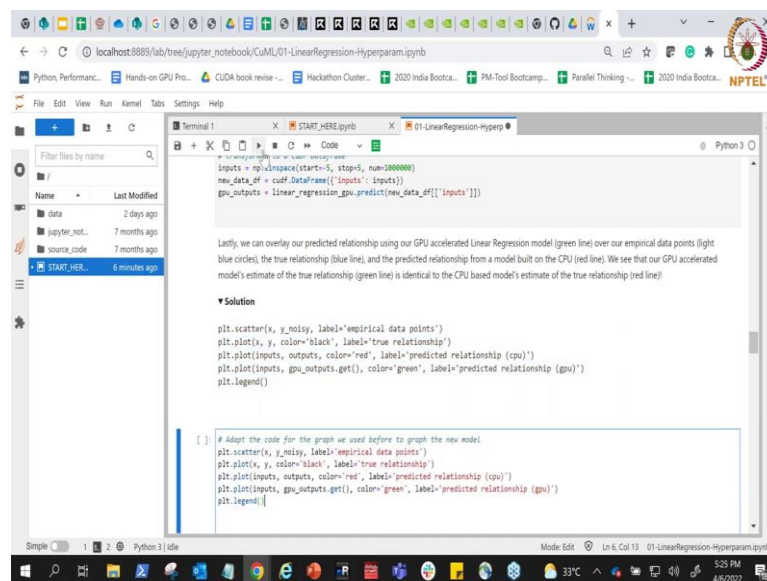
[ ] # create new data and perform inference. Inputs is the same data from the scikit-learn example above
# transformed to a cuDF DataFrame

Lastly, we can overlay our predicted relationship using our GPU accelerated Linear Regression model (green line) over our empirical data points (light blue circles), the true relationship (blue line), and the predicted relationship from a model built on the CPU (red line). We see that our GPU accelerated model's estimate of the true relationship (green line) is identical to the CPU based model's estimate of the true relationship (red line).

> Solution

[ ] # Adopt the code for the graph we used before to graph the new model
```

(Refer Slide Time: 24:17)



```
inputs = np.linspace(start=5, stop=5, num=1000000)
new_data_df = cudf.DataFrame({'inputs': inputs})
gpu_outputs = linear_regression_gpu.predict(new_data_df[['inputs']])

Lastly, we can overlay our predicted relationship using our GPU accelerated Linear Regression model (green line) over our empirical data points (light blue circles), the true relationship (blue line), and the predicted relationship from a model built on the CPU (red line). We see that our GPU accelerated model's estimate of the true relationship (green line) is identical to the CPU based model's estimate of the true relationship (red line).

> Solution

plt.scatter(x, y_noisy, label='empirical data points')
plt.plot(x, y, color='black', label='true relationship')
plt.plot(inputs, outputs, color='red', label='predicted relationship (cpu)')
plt.plot(inputs, gpu_outputs.get(), color='green', label='predicted relationship (gpu)')
plt.legend()

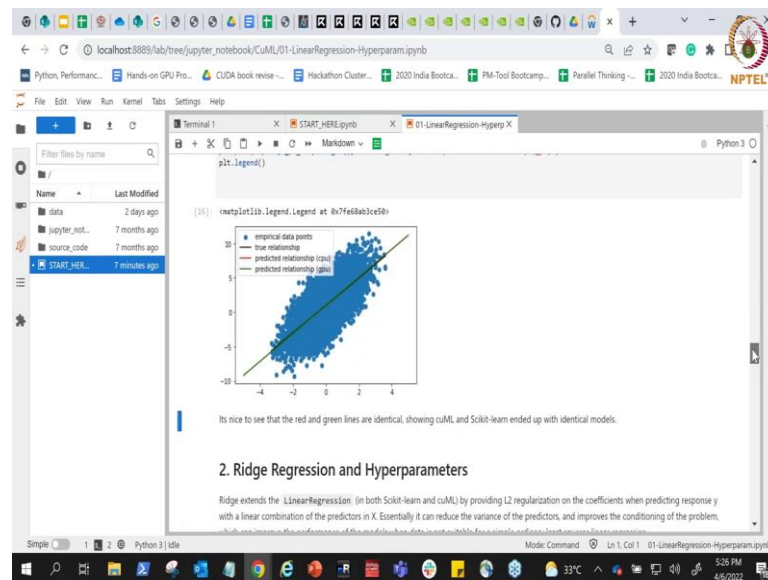
[ ] # Adopt the code for the graph we used before to graph the new model

plt.scatter(x, y_noisy, label='empirical data points')
plt.plot(x, y, color='black', label='true relationship')
plt.plot(inputs, outputs, color='red', label='predicted relationship (cpu)')
plt.plot(inputs, gpu_outputs.get(), color='green', label='predicted relationship (gpu)')
plt.legend()
```

So, just take a note of it while you start putting your data to this right. So, we have kind of trained the model and what we are doing here is we just want to basically set the points and we want to just visualize it. So, I am going to not spend some effort in explaining. So, what we are doing is we are just predicting the values.

So, we are creating the new data set for it to be predicted and then we are doing the prediction here, for the test data set that we created and again we are going to just to move a visualization just to see how good or bad our inferencing is doing.

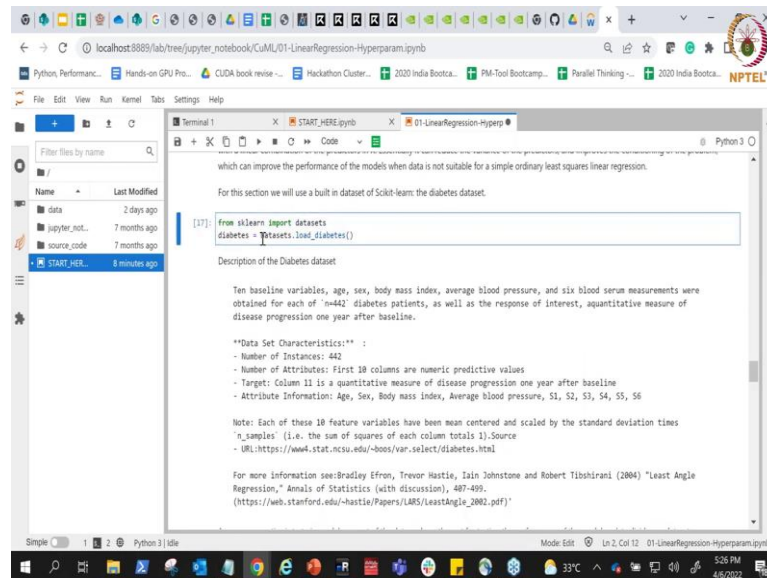
(Refer Slide Time: 24:51)



So, again you can see here. Now, so we have 4 data set points now, the blue dots are basically the empirical data points, the black is the true relationship based on which we know we created the data set on. You can see here that the predicted relationship of the CPU, which is the red and the green one which is the predicted relationship with respect to GPU are almost overlapping with each other. In fact, I do not even see the red bar which is the CPU version it is behind the green bar.

So, in short that the red and green lines are identical, which kind of states at both cuML and the CPU version which is the scikit learn both ended up with the identical models exactly, what you would have expected just that it was running on the GPU and it will be much much more faster.

(Refer Slide Time: 25:50)



The screenshot shows a Jupyter Notebook environment. The left sidebar displays a file explorer with folders like 'data', 'jupyter_notebook', 'source_code', and 'START_HERE'. The main area contains a code cell with the following text:

```
which can improve the performance of the models when data is not suitable for a simple ordinary least squares linear regression.
```

For this section we will use a built in dataset of Scikit-learn: the diabetes dataset.

```
[17]: from sklearn import datasets
      diabetes = datasets.load_diabetes()
```

Description of the Diabetes dataset

Ten baseline variables, age, sex, body mass index, average blood pressure, and six blood serum measurements were obtained for each of ~442 diabetes patients, as well as the response of interest, quantitative measure of disease progression one year after baseline.

Data Set Characteristics :

- Number of Instances: 442
- Number of Attributes: First 10 columns are numeric predictive values
- Target: Column 11 is a quantitative measure of disease progression one year after baseline
- Attribute Information: Age, Sex, Body mass index, Average blood pressure, S1, S2, S3, S4, S5, S6

Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times

- 'n_samples' (i.e. the sum of squares of each column totals 1).Source
- URL: <https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html>

For more information see: Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regression," Annals of Statistics (with discussion), 487-499.
(https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)

So, this is the ridge expression which is a extended version of linear regression and it kind of helps by providing you L2 regularization and it can predict linear combinations quite well, because it basically reduces the variation. And so we are not going to cover the part of what ridge expression is how is it good.

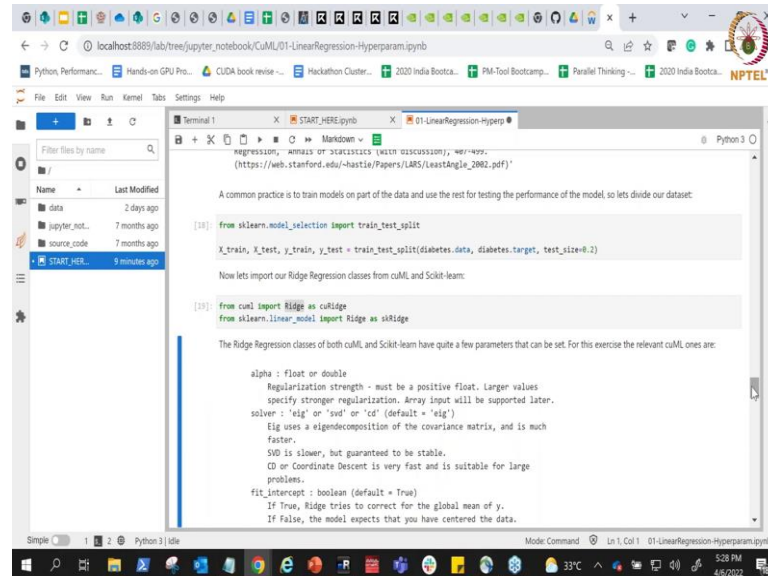
We are not going to cover that part, but the reason why we wanted to show this to you is to tell you that with there are so many parameters which exist and how the relationship of parameters can change the accuracy levels when you move from one framework to the other.

So, here again we are importing sklearn and we are loading a particular data set which is a diabetes dataset, which comes with it. The diabetes data set basically consists of 10 base variables, it consist of age, sex, body mass, index, average values. And it also consists of 6 blood serum measurements which are there and it has basically 442 rows for different diabetes patients. And also, it has the value for the response of interest and what we are trying to do is to see we what we are trying to do is to find a quantitative measure of the disease progression one year after the baseline.

So, that is what we want to predict for this particular dataset. So, again it has 442 different patient it has 10 attributes the 10 columns are basically the numeric values of predicted values and also you have the 11th column which is the quantitative measure of

the disease progression one year after the baseline. So, this is the data set and you can find more details of the data set if you require here in this particular link.

(Refer Slide Time: 27:53)



```
regression, wmass or statistics (with discussion); wsr=www:
(https://web.stanford.edu/~hastie/Papers/LARS/leastAngle_2002.pdf)'

A common practice is to train models on part of the data and use the rest for testing the performance of the model, so lets divide our dataset:

[10]: from sklearn.model_selection import train_test_split
      X_train, X_test, y_train, y_test = train_test_split(diabetes.data, diabetes.target, test_size=0.2)

Now lets import our Ridge Regression classes from cuML and Scikit-learn:

[11]: from cuML import Ridge as cuRidge
      from sklearn.linear_model import Ridge as skRidge

The Ridge Regression classes of both cuML and Scikit-learn have quite a few parameters that can be set. For this exercise the relevant cuML ones are:

alpha : float or double
    Regularization strength - must be a positive float. Larger values
    specify stronger regularization. Array input will be supported later.
solver : 'eig' or 'svd' or 'cd' (default = 'eig')
    Eig uses a eigendecomposition of the covariance matrix, and is much
    faster.
    SVD is slower, but guaranteed to be stable.
    CD or Coordinate Descent is very fast and is suitable for large
    problems.
fit_intercept : boolean (default = True)
    If True, Ridge tries to correct for the global mean of y.
    If False, the model expects that you have centered the data.
```

But let us let us get the data set and then we split the data set into training and the testing part as well. So, we are just doing a train test split.

And as you can see here, instead of just importing a normal linear regression we are importing the ridge algorithm here. And the ridge as I said kind of does a lot of thing; it has various classes inside, like the alpha value can be the float or double.

It kind of defines the regularization strength and you can define it as a positive value, the larger the value specifies the stronger regularization right; you can define the solver to be either eigen SVD or CD like the linear regression. And you can also define other things like whether the predicted values are normalized or not normalized by default it is kind of false.

(Refer Slide Time: 28:52)

```
processes:
fit_intercept : boolean (default = True)
    If True, Ridge tries to correct for the global mean of y.
    If False, the model expects that you have centered the data.
normalize : boolean (default = False)
    If True, the predictors in X will be normalized by dividing by its L2
    norm.
    If False, no scaling will be done.

Most of the parameters are the same for Scikit-learn, except for solver, where the options are:

solver : 'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga'

It is important to see that even though both libraries are performing a Ridge Regression, underneath there are different solvers being used (except for
'svd'). This can lead to slightly different results (and final performance) where both results are technically correct even though they differ.

For the exercise feel free to use 'solver=auto' for Scikit-learn, otherwise we recommend using 'cholesky'.

Exercise: Create the Ridge Regression objects for both cuML and Scikit-learn with matching parameters
and fit X_train and y_train

► Solution for Scikit-learn
► Solution for cuML
```

(Refer Slide Time: 28:55)

```
Exercise: Create the Ridge Regression objects for both cuML and Scikit-learn with matching parameters
and fit X_train and y_train

▼ Solution for Scikit-learn

alpha = np.array([1.0])
fit_intercept = True
normalize = False

ridge = skRidge(alpha=alpha, fit_intercept=fit_intercept, normalize=normalize, solver='cholesky')
ridge.fit(X_train, y_train)

► Solution for cuML

[ ]: alpha = np.array([1.0])
    fit_intercept = True
    normalize = False

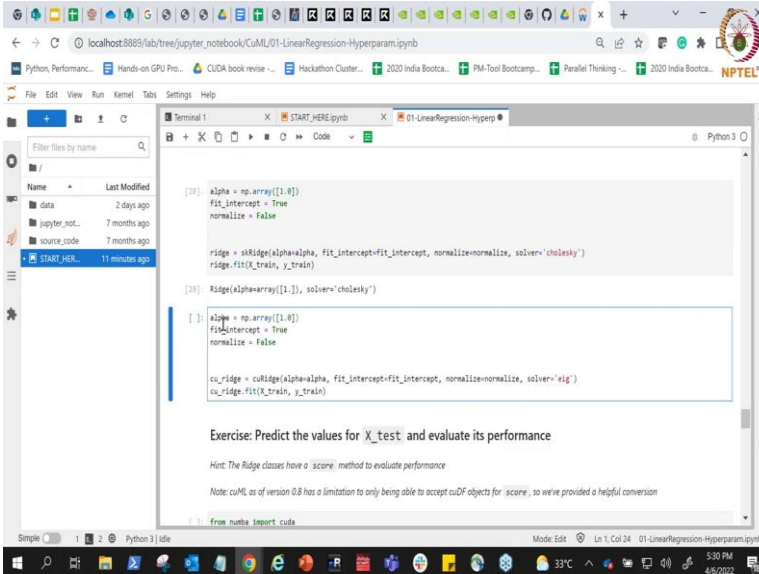
    ridge = skRidge(alpha=alpha, fit_intercept=fit_intercept, normalize=normalize, solver='cholesky')
    ridge.fit(X_train, y_train)

[ ]:
```

So, so, once we have defined this, let us try to create the thing and just try to make it work on the CPU as well as on the GPU. So, what I am doing is I am just copying pasting the solution for scikit learn again. So, you can see here I am calling an API called as skRidge and I am giving it certain parameters which are by default or I am over overwriting it. So, you can see here, I am giving it a particular alpha value which I have taken, which is 1.0.

And I am also saying the solver that I am going to use is cholesky, instead of the default one which is used and I can just do that expression and the same thing I am going to do for a cuML, but I am going to use different parameters here in the cuML space.

(Refer Slide Time: 29:45)



The screenshot shows a Jupyter Notebook titled '01-LinearRegression-Hyperparam.ipynb' running on a local host. The notebook contains two code cells. The first cell (index 20) imports `sklearn` and `sklearn.linear_model`, and defines `alpha`, `fit_intercept`, and `normalize`. It then creates a `Ridge` object with `alpha=alpha`, `fit_intercept=fit_intercept`, `normalize=normalize`, and `solver='cholesky'`. The second cell (index 21) imports `cuML` and `cuML.linear_model`, and defines `alpha`, `fit_intercept`, and `normalize`. It then creates a `cu_ridge` object with `alpha=alpha`, `fit_intercept=fit_intercept`, `normalize=normalize`, and `solver='eig'`. Below the code cells, there is an exercise prompt: 'Exercise: Predict the values for `X_test` and evaluate its performance'. A hint is provided: 'Hint: The Ridge classes have a `score` method to evaluate performance'. A note is also present: 'Note: cuML as of version 0.8 has a limitation to only being able to accept cuDF objects for `score`, so we've provided a helpful conversion'.

```
[20]: alpha = np.array([1.0])
fit_intercept = True
normalize = False

ridge = sklearn.linear_model.Ridge(alpha=alpha, fit_intercept=fit_intercept, normalize=normalize, solver='cholesky')
ridge.fit(X_train, y_train)

[21]: Ridge(alpha=array([1.]), solver='cholesky')

[ ]: alpha = np.array([1.0])
fit_intercept = True
normalize = False

cu_ridge = cuML.linear_model.cu_ridge(alpha=alpha, fit_intercept=fit_intercept, normalize=normalize, solver='eig')
cu_ridge.fit(X_train, y_train)

Exercise: Predict the values for X_test and evaluate its performance

Hint: The Ridge classes have a score method to evaluate performance

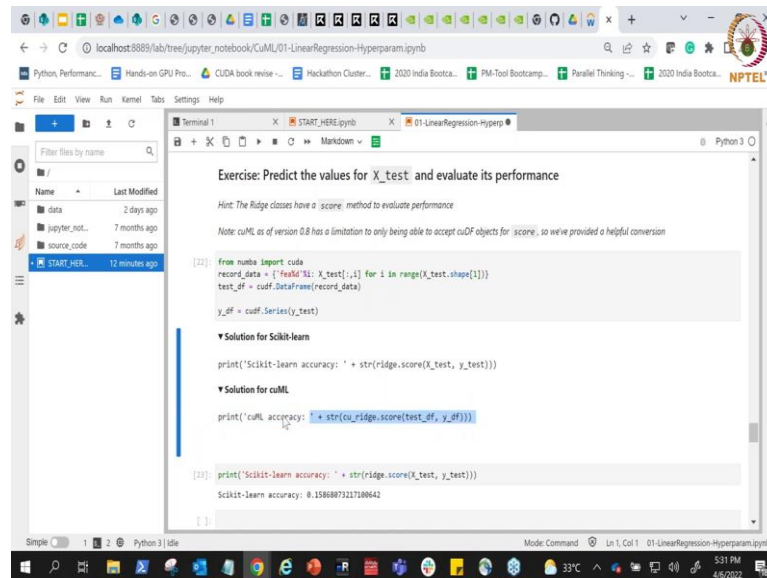
Note: cuML as of version 0.8 has a limitation to only being able to accept cuDF objects for score, so we've provided a helpful conversion

from math import e
```

So, in the cuML my alpha value is the same, but I am saying that I am going to use a different solver, which is eigen instead of using a solver which was cholesky here. So, let me just run this particular part and then what we are going to do is we are going to just, we are going to measure the accuracy levels at which we are able to predict the values.

And for which basically there is a class which is called as score, which is used to measure the performance of the ridge expression.

(Refer Slide Time: 30:25)



```
Exercise: Predict the values for  $X_{test}$  and evaluate its performance

Hint: The Ridge classes have a score method to evaluate performance

Note: cuML as of version 0.8 has a limitation to only being able to accept cuDF objects for score, so we've provided a helpful conversion

[22]: from numpy import cuda
record_data = {'features': X_test[1:] for i in range(X_test.shape[1])}
test_df = cudf.DataFrame(record_data)
y_df = cudf.Series(y_test)

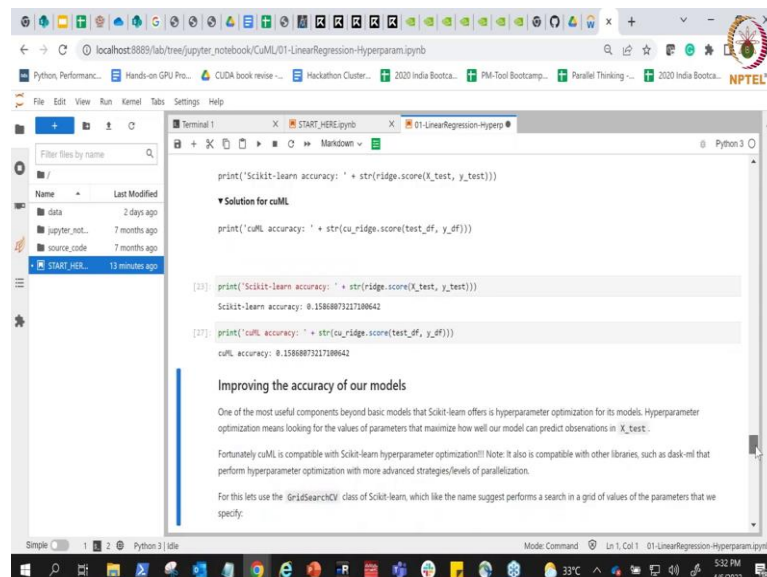
▼ Solution for Scikit-learn
print('Scikit-learn accuracy: ' + str(ridge.score(X_test, y_test)))

▼ Solution for cuML
print('cuML accuracy: ' + str(cu_ridge.score(test_df, y_df)))

[23]: print('Scikit-learn accuracy: ' + str(ridge.score(X_test, y_test)))
Scikit-learn accuracy: 0.1586807321708642
```

So, let me just try to do the same thing for both sklearn as well as to do to calculate the score basically kind of saying how well our model does on the test data set. So, in this particular case, you can see the scikit learn accuracy the score is 0.1586 let us do the same thing for cuML as well and see how well it performs; I am sorry on this part.

(Refer Slide Time: 31:04)



```
print('Scikit-learn accuracy: ' + str(ridge.score(X_test, y_test)))

▼ Solution for cuML
print('cuML accuracy: ' + str(cu_ridge.score(test_df, y_df)))

[23]: print('Scikit-learn accuracy: ' + str(ridge.score(X_test, y_test)))
Scikit-learn accuracy: 0.1586807321708642

[27]: print('cuML accuracy: ' + str(cu_ridge.score(test_df, y_df)))
cuML accuracy: 0.1586807321708642

Improving the accuracy of our models

One of the most useful components beyond basic models that Scikit-learn offers is hyperparameter optimization for its models. Hyperparameter optimization means looking for the values of parameters that maximize how well our model can predict observations in  $X_{test}$ .

Fortunately cuML is compatible with Scikit-learn hyperparameter optimization!!! Note: It also is compatible with other libraries, such as dask-ml that perform hyperparameter optimization with more advanced strategies/levels of parallelization.

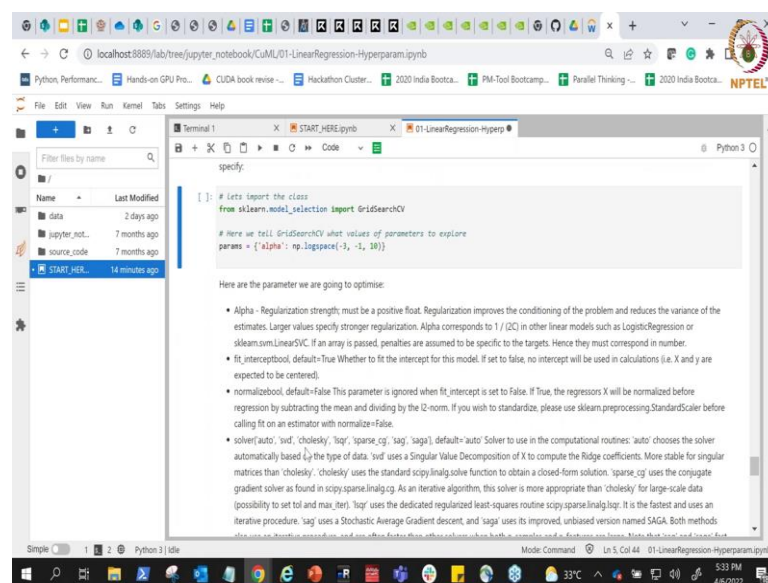
For this lets use the GridSearchCV class of Scikit-learn, which like the name suggest performs a search in a grid of values of the parameters that we specify:
```

So, you can see here it is almost the same, you can see here it gives 0.15868073 to a large extent both of them are giving the same results. But if I change the value for something else maybe instead of using eigen, if I say use SVD we might start getting

different results here. So, let me again run this ridge expression, I have run it for specifically for the cuML part of it.

And let me just again create this and you can see here it has slightly changed at the few decimal places later on also. So, but just to show that there are different kinds of hyper parameters which exist, but the good part is that you can actually select the hyper parameters or you can do a search for the hyper parameters and improve the accuracy of your models.

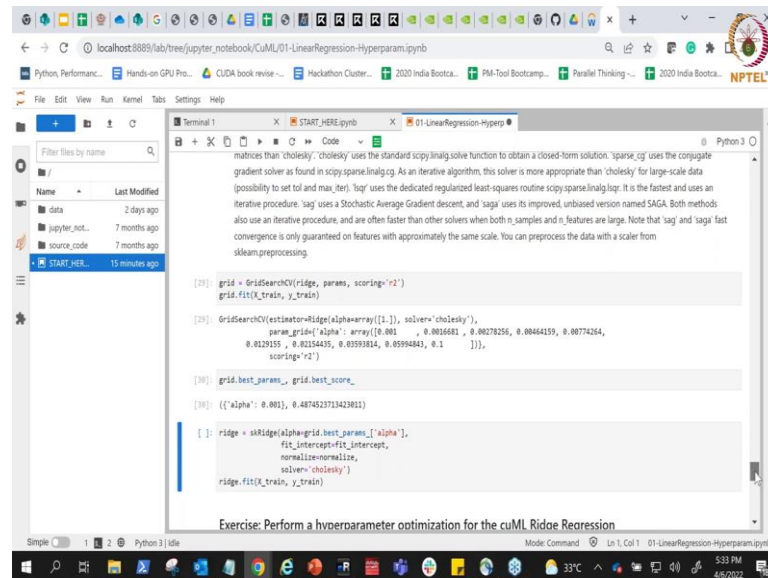
(Refer Slide Time: 32:10)



And for this particular case, we are using a method which exists in scikit learn which is the GridsearchCV it is present in the scikit learn as the name suggests it will do a great search for values. For the parameters that we specify, like in this particular case I am saying that I would like to search within a parameter space for only alpha.

So, I am saying that I want to do a great search which alpha values is more better for me and there are different methods you can also add other parameters also like you can choose whether you want to try out different versions of the solver or you want to just try out alpha.

(Refer Slide Time: 33:01)



```
matrices than 'cholesky'. 'cholesky' uses the standard scipy.linalg.solve function to obtain a closed-form solution. 'sparse_cg' uses the conjugate gradient solver as found in scipy.sparse.linalg.cg. As an iterative algorithm, this solver is more appropriate than 'cholesky' for large-scale data (possibility to set tol and max_iter). 'lsqr' uses the dedicated regularized least-squares routine scipy.sparse.linalg.lsqr. It is the fastest and uses an iterative procedure. 'sag' uses a Stochastic Average Gradient descent, and 'saga' uses its improved, unbiased version named SAGA. Both methods also use an iterative procedure, and are often faster than other solvers when both n_samples and n_features are large. Note that 'sag' and 'saga' fast convergence is only guaranteed on features with approximately the same scale. You can preprocess the data with a scaler from sklearn.preprocessing.
```

```
[29]: grid = GridSearchCV(ridge, params, scoring='r2')
      grid.fit(X_train, y_train)

[30]: GridSearchCV(estimator=Ridge(alpha=array([1.]), solvers='cholesky'),
      param_grid={'alpha': array([0.001, 0.003681, 0.00278256, 0.00464159, 0.00774264,
      0.0212935, 0.02154435, 0.0593814, 0.0594843, 0.1
      ])}),
      scoring='r2')

[31]: grid.best_params_, grid.best_score_

[32]: (['alpha': 0.001], 0.4874523713423811)

[ ]: ridge = skRidge(alpha=grid.best_params_['alpha'],
      fit_intercept=fit_intercept,
      normalize=normalize,
      solvers='cholesky')
      ridge.fit(X_train, y_train)
```

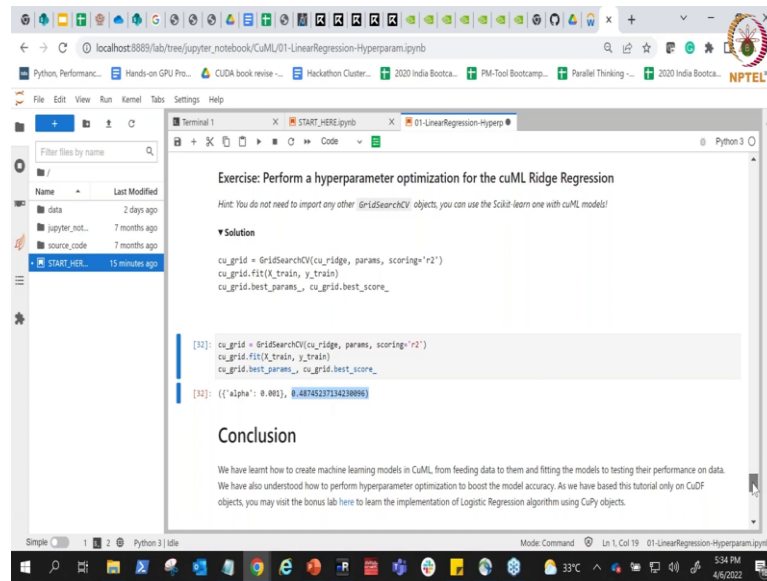
Exercise: Perform a hyperparameter optimization for the cuML Ridge Regression

So, you can add as many parameters as you need, that you would like your grid search to basically figure out to and spit out what is the best method for, what is the best value which it is going to give out.

So, you can see here that I am trying to do the gridsearchCV and it is going to split out what is the best score for which particular alpha value. So, you here you can see here that it did a search for a different alpha values and it figured out that 0.001 is the one which is having a much more better score as compared to what we saw earlier.

If you remember the score there was 0.1 something, here the score is much higher which is 0.48 and then we can do the final fit to see that in fact, for all the test data, if you are getting the best results or not.

(Refer Slide Time: 33:48)



The screenshot shows a Jupyter Notebook titled "01-LinearRegression-Hyperparam.ipynb". The notebook content includes:

- Exercise:** Perform a hyperparameter optimization for the cuML Ridge Regression.
- Hint:** You do not need to import any other `GridSearchCV` objects, you can use the Scikit-learn one with cuML models!
- Solution:**

```
cu_grid = GridSearchCV(cu_ridge, params, scoring='r2')
cu_grid.fit(x_train, y_train)
cu_grid.best_params_, cu_grid.best_score_
```
- Conclusion:** We have learnt how to create machine learning models in CuML from feeding data to them and fitting the models to testing their performance on data. We have also understood how to perform hyperparameter optimization to boost the model accuracy. As we have based this tutorial only on CuDF objects, you may visit the bonus lab [here](#) to learn the implementation of Logistic Regression algorithm using CuPy objects.

So, this is the same thing that you can do, once you are done with the grid search then if you want you can provide the same to cuML as well, do the grid search for cuML and find out if it is indeed giving the. So, it can see here the same grid search 0.01 is the best alpha value according to it and it is providing your score of 0.18.

So, like in Deep Learning even for Machine Learning you can use some of this hyper parameter tuning methods like grid search and try to find out which hyper parameter works the best for the problem statement that you have in mind. So, this was a very quick demo for you to understand the cuML and how cuML is very similar to if you were running on a on a sequential environment using scikit learn, you just have to import the right package.

And you have to take care of additional things like if you are using the right hyper parameters or not, because if you because there will be certain changes in accuracy which can happen.