Example Zoo

Organization: mlpack

—

Name: Roshan Nrusing Swain

University: National Institute of Technology, Agartala

Email: swainroshan001@gmail.com

Github: https://github.com/swaingotnochill

Timezone: IST (+5:30 UTC)

My Contribution

I have started to use mlpack for various machine learning implementations. Here is one issue which I fixed and is being reviewed:

- https://github.com/mlpack/mlpack/pull/2901

Project Abstract

I wish to work on the project Example Zoo(mlpack idea page). I liked the idea of having some code that newcomers can simply go through and understand the potential usage through various real-world domains. This will allow us to demonstrate the potential of our library at the same time having some ready-made code that users can copy and use in their use-cases.

For the project, I would like to work on corresponding datasets:

1. California Housing Price Data
2. Anime Face Data
3. MNIST
4. Human Celeb Faces

Project Deliverables:

Things I plan on doing :

1. Showcasing the use of API in examples.
2. Using Jupyter notebook(Binder) for the examples.
3. Using visualization software like Seaborn, Matplotlib-CPP for plotting graphs.
4. Improving Documentation and writing a blog on examples.

Expected Impact of this project:

1. Users can directly go through code samples and understand the flow and usage of various mlpack methods and models easily.
2. Users can directly copy-paste some code through a blog or sample code for their Users will no longer feel overwhelmed with the new library and easily run the code using online Jupyter notebooks like a binder.
3. Increasing the user interaction as users will repetitively reference the examples.

Implementation Details:

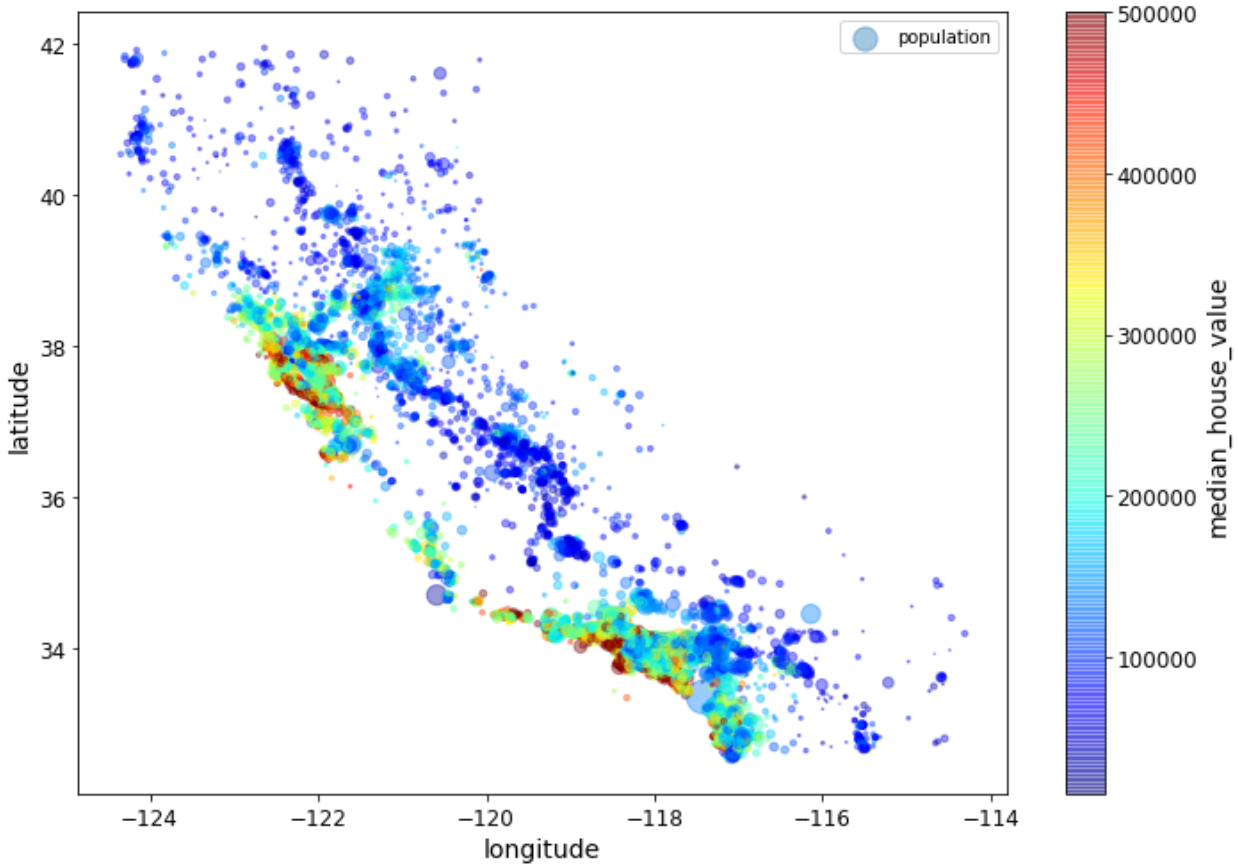
1. Linear Regression for California House Price Data

I plan to work on a linear regression model for predicting California Housing Prices. The detailed implementation is:

- Setting up the dependencies: Including mlpack dependencies and matplotlib-CPP for plotting.
- Downloading the data: The data can be downloaded for free from Kaggle. The links to the dataset are here.
- Loading the data: Using the data:: Load() method, I will read the data.
- Plotting the data: We will use matplotlib-CPP which is a wrapper on matplotlib and works well with C++. For example, which uses python binding of mlpack, I will use matplotlib and seaborn for plotting data.

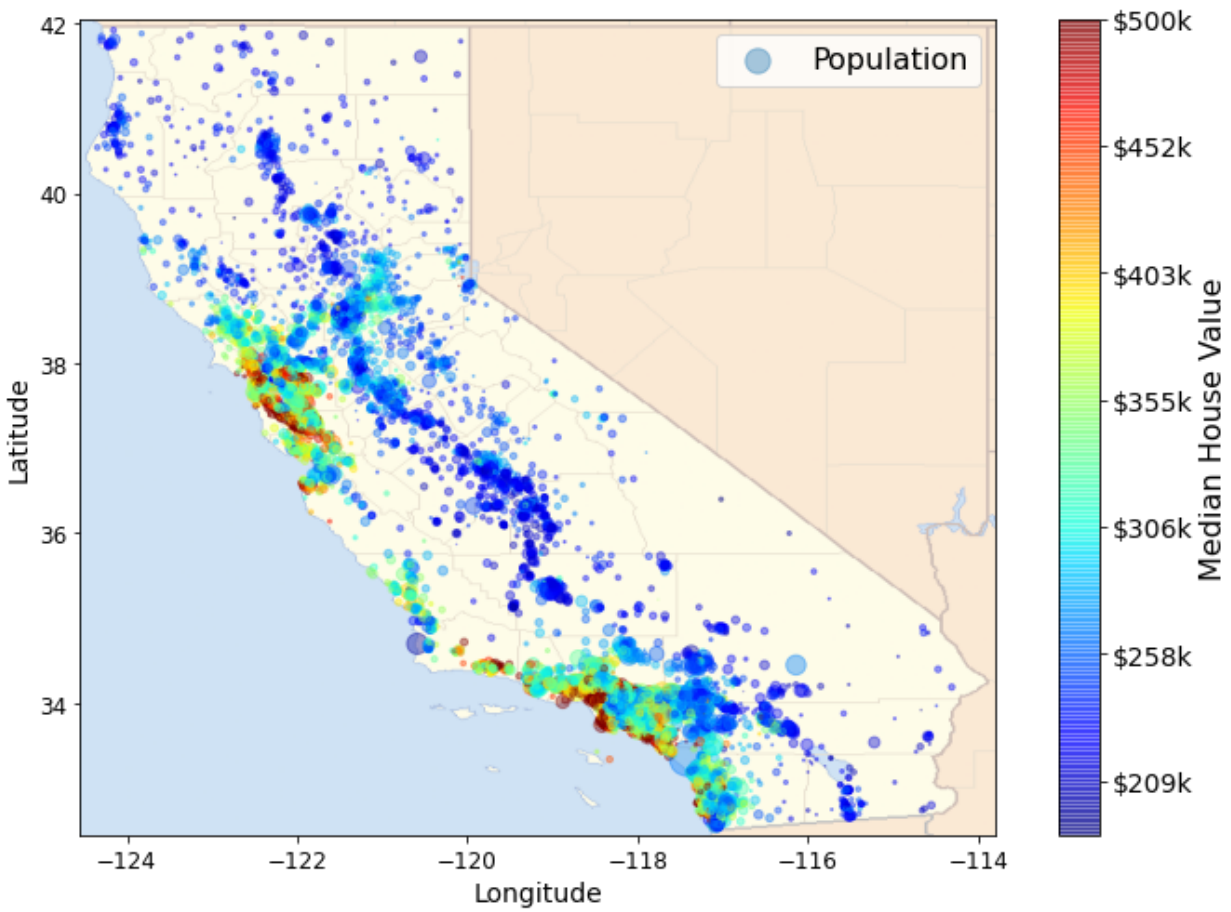
Below is a data plotted using matplotlib for California housing data,

```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
             s=housing["population"]/100, label="population", figsize=(10,7),
             c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
             sharex=False)
plt.legend()
```



One more interesting visualization will be to map this visualization on California Map and see the result. The following code will implement the idea:

```
housing.plot(kind="scatter", x="longitude", y="latitude", figsize=(10,7),
                  s=housing['population']/100, label="Population",
                  c="median_house_value", cmap=plt.get_cmap("jet"),
                  colorbar=False, alpha=0.4)
plt.imshow(california_img, extent=[-124.55, -113.80, 32.45, 42.05],
alpha=0.5,cmap=plt.get_cmap("jet"))
```



- Preprocessing the data: Making some preprocessing on data like normalization for better performance. The below code can be used for preprocessing:

```
// Scale all data into the range (0, 1) for increased numerical
stability.
data::MinMaxScaler scaleX;
// Scaler for predictions.
data::MinMaxScaler scaleY;

// Fit scaler only on training data.
scaleX.Fit(trainX);
scaleX.Transform(trainX, trainX);
scaleX.Transform(validX, validX);

// Scale training predictions.
scaleY.Fit(trainY);
scaleY.Transform(trainY, trainY);
```

```
scaleY.Transform(validY, validY);
```

- Splitting the data : The data need to be split into inputs and labels which can be done by using the below code:

```
arma::Row<size_t> labels =
    arma::conv_to<arma::Row<size_t>>::from(input.row(input.n_rows - 1));
input.shed_row(input.n_rows - 1);
```

- Creating and Training the model: The next step will be to define the model. Mlpack has ready-to-use models for our use. It can be implemented using below code:

```
LinearRegression<> lr(input, labels, 0.0 /* no regularization */);
lr.Train(inputs, labels);
```

- Predict the data: Final step will be predicting the data. This can be done using Predict function in linear regression. It can be implemented as follows:

```
lr.Predict(inputs, predictions);
cout<<predictions;
```

Comparison

Another add-on to the project will be training a Least Angle Regression (LARS) model and using the predictions obtained from them and comparing it with the performance of Linear Regression.

Least Angle Regression, LAR or LARS for short, is an alternative approach to solving the optimization problem of fitting the penalized model. Technically, LARS is a forward stepwise version of feature selection for regression that can be adapted for the Lasso model.

Unlike the Lasso, it does not require a hyperparameter that controls the weighting of the penalty in the loss function. Instead, the weighting is discovered automatically by LARS.

The following code can be used to implement it:

```
LARS<> lars(input, labels);
```

We will then calculate the Residuals for both the model which is ⟶

Residual = | Y_pred - Y_original | and then plot it using matplotlib-CPP.

The other comparison can be as simple as comparing the Mean Square Error for both models.

This is the direction the project will go. Besides that, it will also include feature selection and more plotting in the direction of a proper end-to-end machine learning project with include Preprocessing, Exploratory data analysis & Visualization, and Machine learning Modeling. The project will be implemented in C++, Python, and R(mlpack python and R bindings). There are a lot of tools available for Visualization for Python and R use-cases like matplotlib and ggplot2 respectively.

2. Generative Adversarial Network for generating synthetic images

For the second implementation of mlpack to generate MNIST handwritten digits, Anime Faces and Human Faces.

Abstract

Deep neural networks are used mainly for supervised learning: Classification or Regression. Generative Adversarial Networks or GANs, however, use neural networks for a very different purpose: Generative modeling.

Generative modeling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way

that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset

To get a sense of the power of generative models, just visit the link. Every time you reload the page, a new image of a person's face is generated on the fly.

Approach

There are two neural networks: a Generator and a Discriminator. The generator generates a "fake" sample given a random vector/matrix, and the discriminator attempts to detect whether a given sample is "real" (picked from the training data) or "fake" (generated by the generator). Training happens in tandem: we train the discriminator for a few epochs, then train the generator for a few epochs, and repeat. This way both the generator and the discriminator get better at doing their jobs.

GANs, however, can be notoriously difficult to train and are extremely sensitive to hyperparameters, activation functions, and regularization. In this project, we will first train on the MNIST dataset, then slowly increase the difficulty to Anime Faces and Human Celeb Faces.

Implementation:

The idea is to work around GANs in 3 levels:

Level 1: Generating Handwritten Digits

This should be fairly simple considering the complexity of images. Fine Tuning, Testing, and Final training can be done in less time.

Level 2: Generating Anime Faces

The fine-tuning and testing process will take a little longer as there will be extra dimensions to the images as they are RGB images, along with shades of hair and expression.

Level 3: Generating Human Faces

This will be quite challenging to train and fine-tune as there are a lot of extra dimensions and realism compared to the anime faces. I will have to spend experimenting with different networks after a few iterations.

Data :

The vector dimension will depend on the type of images, for instance, MNIST images are grayscale, hence they are single channelimages, while Anime and Human Celeb Faces are colored images, hence they are three channel images. We will download and load the data first and then work on the model. Tare coloured, hence 3 channeled. The imahe difference with first use case is that we will be using a deep neural network model.

Discriminator:

The discriminator takes an image as input, and tries to classify it as "real" or "generated". In this sense, it's like any other neural network. We'll use a convolutional neural network (CNN) which outputs a single number output for every image. We'll use a stride of 2 to progressively reduce the size of the output feature map.

```
Convolution(3, 64, kernel_size=4, stride=2, padding=1),
 BatchNorm(64),
 LeakyReLU(0.2),
 # out: 64 x 32 x 32

 Convolution(64, 128, kernel_size=4, stride=2, padding=1),
 BatchNorm(128),
 LeakyReLU(0.2),
 # out: 128 x 16 x 16

 Convolution(128, 256, kernel_size=4, stride=2, padding=1),
 BatchNorm(256),
```

```
    LeakyReLU(0.2),
    # out: 256 x 8 x 8

    Convolution(256, 512, kernel_size=4, stride=2, padding=1),
    BatchNorm(512),
    LeakyReLU(0.2),
    # out: 512 x 4 x 4

    Convolution(512, 1, kernel_size=4, stride=1, padding=0),
    # out: 1 x 1 x 1

   Flatten(),
   Sigmoid()
```

Generator:

The input to the generator is typically a vector or a matrix of random numbers (referred to as a latent tensor) which is used as a seed for generating an image. The generator will convert a latent tensor of shape (128, 1, 1) into an image tensor of shape 3 x 28 x 28. To achieve this, we'll use the TransposeConvulation from mlpack.

```
# in: latent_size x 1 x 1

    TransposeConvulation(latent_size, 512, kernel_size=4, stride=1,
padding=0),
    BatchNorm(512),
    ReLU(True),
    # out: 512 x 4 x 4

    TransposeConvulation(512, 256, kernel_size=4, stride=2, padding=1),
    BatchNorm(256),
    ReLU(True),
    # out: 256 x 8 x 8

    TransposeConvulation(256, 128, kernel_size=4, stride=2, padding=1),
    BatchNorm(128),
    ReLU(True),
    # out: 128 x 16 x 16

    TransposeConvulation(128, 64, kernel_size=4, stride=2, padding=1),
    BatchNorm64),
    ReLU(True),
```

```
# out: 64 x 32 x 32

TransposeConvulation(64, 3, kernel_size=4, stride=2, padding=1),
Tanh()
# out: 3 x 64 x 64
```

Training :

Discriminator Training:

Since the discriminator is a binary classification model, we can use the Sigmoid Cross-Entropy loss function in mlpack to quantify how well it can differentiate between real and generated images.

Generator Training:

Since the outputs of the generator are images, it's not obvious how we can train the generator. This is where we employ a rather elegant trick, which is to use the discriminator as a part of the loss function. Here's how it works:

- We generate a batch of images using the generator, pass them into the discriminator.
- We calculate the loss by setting the target labels to 1 i.e. real. We do this because the generator's objective is to "fool" the discriminator.
- We use the loss to perform gradient descent i.e. change the weights of the generator, so it gets better at generating real-like images to "fool" the discriminator.

Optimizers

I am planning on using 2 Optimizers for GANs. They are Adam and RMSprop. We can test on a small subset and see which performs better, and then choose the best one.

Adam is an optimizer that computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.

RMSprop is an optimizer that utilizes the magnitude of recent gradients to normalize the gradients.

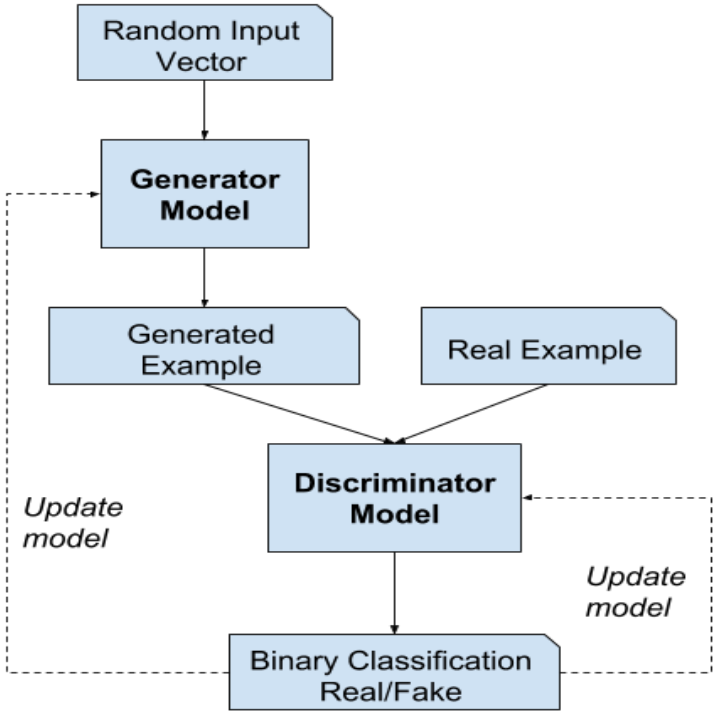
Both of them perform fairly well for GANs and mlpack has the support of both these optimizers.

Below is an example of Adam implementation:

```
 ens::Adam optimizer(
      STEP_SIZE,  // Step size of the optimizer.
      BATCH_SIZE, // Batch size. Number of data points that
are used in each
                  // iteration.
      0.9,        // Exponential decay rate for the first
moment estimates.
      0.999, // Exponential decay rate for the weighted
infinity norm estimates.
      1e-8,  // Value used to initialise the mean squared
gradient parameter.
      MAX_ITERATIONS, // Max number of iterations.
      1e-8,           // Tolerance.
      true);
```

Similarly, RMSprop can be used with its own arguments.

Below is the full training loop:



The above network will generate handwritten digits from random noises. This will demonstrate the Generative Adversarial Networks use case using mlpack which hasn't been done yet. Also, after successfully generating the handwritten digits, we can stage the process into level 2 which is: Generating Anime Face.

The idea is to fine-tune the network and use some tests to check the evaluation of the GAN network.

The 3rd level of the project is to generate Human Celeb Faces. I am planning on training the human celeb faces on the same network and see how it performs, and then fine-tune and test it. Considering the GSoC timeline, I would fine-tune and test Human Celeb Faces only if there is enough time after Blog, Documentation, and Testing of other parts of the project, as I do not wish for this part to have any impacts on the rest of the project.

If we don't have sufficient time, I will fine-tune it after GSoC.

Testing

Manual GAN Generator Evaluation

This involves using the generator model to create a batch of synthetic images, then evaluating the quality and diversity of the images about the target domain. For this, we will use a subset of the dataset and see if the model improves over time. If not, we will modify the model

concerning the images generated and then continue training on the same subset. If the model improves over time, we will then switch to a larger dataset, and continue the same process. Finally, we will use the original dataset for final training.

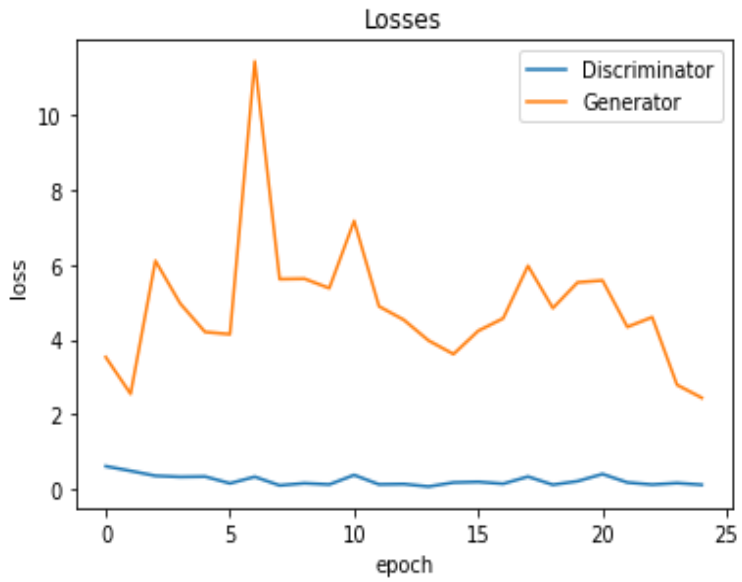
This is the simplest and first step in the evaluation of the GAN network.

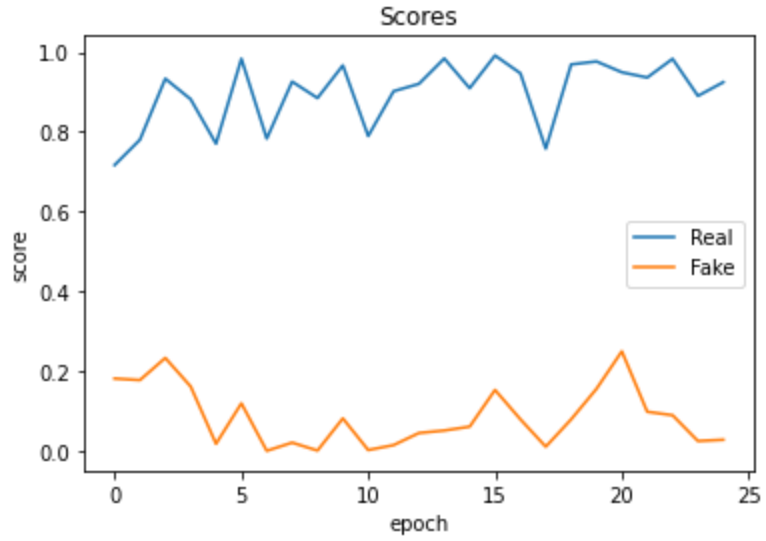
Quantitative and Qualitative Testing

Once our confidence in developing GAN models improves, both the **Inception Score** and the **Frechet Inception Distance** can be used to quantitatively summarize the quality of generated images. There is no single best and agreed upon measure, although, these two measures come close.

The **Nearest neighbor method** can be used to qualitatively summarize generated images.

We can also plot the quantitative loss and qualitative score using matplotlib-CPP. Below are graphs showing them:





Also, at every 5/10 epochs I will save a snapshot of batch of images then create a gif using those images. For this I will be using Magick++ library from imagemagick.org. The following code produces a gif out of images from each batch.

```
Image img1( "100x100", "white" );
img1.pixelColor( 49, 49, "red" );
frames.push_back(img1);
Image img2( "100x100", "red" );
img2.pixelColor( 49, 49, "white" );
frames.push_back(img2);
img1.animationDelay(2000);
img2.animationDelay(2000);*/
Magick::writeImages(frames.begin(), frames.end(), "f:\\2.gif");
```

3. Blog and Documentation

After completion of each example, a concise and well-documented blog for the users to go through the examples along with code snippets will be prepared with the above examples. I will explain all the GAN implementations in a simple way along with all graphs and GIFs, that it will be comfortable for anyone to follow them and implement their skillset on their own.

Timeline:

Community Bonding Period (May 17 - June 7)	• Knowing the community and mentors. • Discussion of the project with mentors. • Go through the documentation and codebase for the methods.
Phase 1 (June 7 - July 16)	
Week 1 (June 7 - June 14)	• Working on the implementation of Linear Regression and LARS in Cpp for California Housing Dataset.
Week 2 (June 14 - June 21)	• Working on Implementation of the same topic in Python and R.
Week 3 (June 21- June 28)	• Discussing the implementation of GAN. • Designing Discriminator and Generator for MNIST.
Week 4 (June 28 -July 5)	• Training and Fine Tuning for MNIST. • Testing and improving the network.
Week 5 (July 5- July 12)	• Working on implementing GANs for Anime Faces by first working on a subset and then modifying the network accordingly.
Phase 1 Evaluation (July 12 - July 16)	Buffer Period
Week 6 (July 17 - July 24)	• Fine Tuning and Testing the GAN for Anime Faces.

	• Working on stabilizing the Discriminator.
Week 7 (July 24 - July 31)	• Working on implementing GAN for Human Celeb Faces. • Training and Testing the GAN (this might take longer than other GANs as it is more complex).
Week 8 (August 1 - August 8)	• Improving and modifying the GAN based on Test results. • Fine-tuning and a final training process.
Week 9 (August 8 - August 16)	• Working on Documentation on worked examples. • Writing blog posts on all examples showcasing my work. • A final blog post summarising my experience with mlpack in GSoC'21.

About Me:

My name is Roshan Swain, an undergraduate student at the National Institute of Technology, Agartala seeking majors in Electrical Engineering. Currently, I am in my 6th semester. I have a great passion for Machine Learning and Quantum Computing. I have prior experience in Machine Learning as an Intern in a startup.

I have an avid interest in Computation and Mathematics. My journey started with an ML Bootcamp and Deep Learning course, and I was so fascinated by Deep Learning, that I decided to work hard and grow my career in this field. I also tried to improve my computation skills and development skills from the Coursera course and personal projects.

I have also participated in different Hackathons, Datathons, and open-source projects. Last year in 2020, I volunteered in DigitalOcean HacktoberFest as a participant and a project maintainer.

This year is my first time participating in Google Summer of Code, as this is an immense opportunity to learn and contribute towards open source this summer. I decided to choose

mlpack because of my love for Machine Learning and also to improve my existing skillset. It provides me the perfect opportunity to work in my domain of choice.

Here is a link to my CV.

Technical Knowledge for this Project:

I have been using mlpack for some implementation and thus familiar with its usage. I have also been reading the documentation for the classes and methods in mlpack and thus can hack around my way during the project. Other than that, I also have experience working with the datasets mentioned above. I have worked on various Machine Learning and Deep Learning projects (Github). Besides that, I also have working experience as a Machine Learning Intern in a startup.

Any other commitments during the main GSoC period?

I am not working on any other projects during my summertime and I will be comfortably able to give 40+ hours per week to this project during GSoC.

Exams or Classes that overlap with this period?

I don't have any exams during this period.

Plan to apply for or have any other jobs or Internships during this period?

No.

Any other short-term commitments during this period?

No.

After GSoC

I would still want to be a part of mlpack and keep working at it even after the GSoC period since I feel my skills would be well invested here.

Why mlpack?

I believe mlpack to be an amazing library for machine learning for c++ users. The fact that it provides different language bindings makes it more versatile and attractive for other programmers. Working for mlpack will improve my machine learning, c++, and mathematical skills which is very essential for someone in my domain. I also wish to be a part of such a community that helps people by providing them excellent open-source software. Learning from senior members and other programmers is what I wish for. I would be glad to be a part of mlpack and contribute my level-best.

Conclusion:

Thank you so much for lending your precious time to this proposal. I hope my proposal proves to be of some value to mlpack. I am looking forward to working at this prestigious organization.