Dimensionality Reduction for Studying Diffuse Circumgalactic Medium

Basic Information

Name: Rituraj Dutta

University: Gauhati University Institute of Science and Technology, Guwahati

Email: riturajdutta400@gmail.com

LinkedIn: https://www.linkedin.com/in/rituraj-dutta-979976198

GitHub: https://github.com/Rituraj-commits

Slack: Rituraj

Country of Residence: India

Timezone : IST (GMT + 05:30)

Primary Language: English

I am currently a second-year engineering student pursuing my Btech in Information Technology from Gauhati University Institute of Science and Technology. My exams will be over by late April leaving me enough time to extensively work on the project. If I am shortlisted for the project I will be able to put in 30-35 hrs per week for the project depending on the workload.

Problem Description

The Circumgalactic Medium (CGM) is loosely defined as the region spanning a few hundred kiloparsecs from a galaxy that contains an extended reservoir of diffuse gas. Studying the CGM allows us to understand how star formation takes place in galaxies. The CGM is the source of the galaxy's star-forming fuel. We analyze these gases in the CGM through the quasar absorption spectrum.

Light from the quasars passes through the CGM and certain frequencies of the electromagnetic radiation get absorbed resulting in a plunge (dark lines in case of visible spectra) in the spectrum. The spectrum thus obtained is a quasar absorption spectrum. Using the data from the quasar absorption spectrum we can determine the star-forming gas composition, temperature and density.

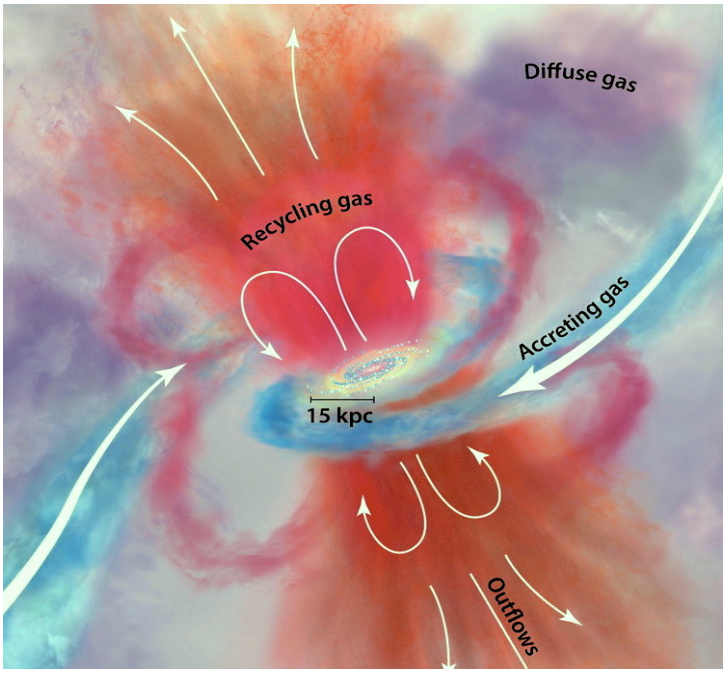
But these datasets contain hundreds of features which makes them computationally expensive and even sometimes prone to overfitting. Furthermore, it becomes much more complex to visualize the data as dimensions increase. So, this project focuses on implementing Dimensionality Reduction to reduce the features as much as possible without losing significant performance of the model.

Motivation

The universe is expanding at an enormous rate of 82.4 km/s. The universe is a mystery which if studied properly can give rise to numerous facts and help us explore more. Though we have only explored 4 per cent of the visible Universe with the age of computing we can explore more than before. As computers are becoming more robust and powerful we are slowly unveiling the deepest and darkest corners of the universe. Moreover, with the advancement of Algorithms, we can solve millions of calculations in few hours. Machine learning analyzes tons of data and helps machines to get acquainted with diverse scenarios that they can face while exploring the universe. But to make things simpler and less compute-intensive we as programmers must make the information less crowded while still preserving the data. Humans are prone to error so Machine Learning can help us quite significantly in obtaining what we desire. This will help us in analyzing billions of data without much use of resources.

Background Research in CGM

Several works related to Circumgalactic Medium (CGM) have been reported. Among several works that I have studied, the one that has helped me a lot to expand my knowledge in Circumgalatic Medium (CGM) is available at https://arxiv.org/abs/1709.09180. As clearly stated the CGM has played an important role in our understanding of galaxy evolution by observing the diffused and complex materials found. After analyzing the quasar absorption spectra with multiple absorption redshifts it pointed out that it was caused by gas in the extended halos of normal galaxies. The absorption lines represent that of Lymanα, C IV and other metal lines which are richly structured in densities, temperature and ionisation. Studying the CGM poses the answers to many unanswered questions such as “Why do dark matter halos give rise to galaxies with drastically different star formation and chemical histories ?”, “Why do such a small fraction of cosmic baryons and metals reside in the galaxies ?”. In abstract terms, the CGM is the galaxy’s fuel tank, waste dump and recycling centre all at the same time.



In the above picture, we can see what a CGM looks like theoretically. The galaxy’s core and blue gaseous disk are fed by the accretion disk from the Intergalactic Medium (IGM). The pink and orange gases are the outflows while some of the ejected gases are being reused. The gas in purple is the diffuse gas which is likely caused by the mixing of all other gases.

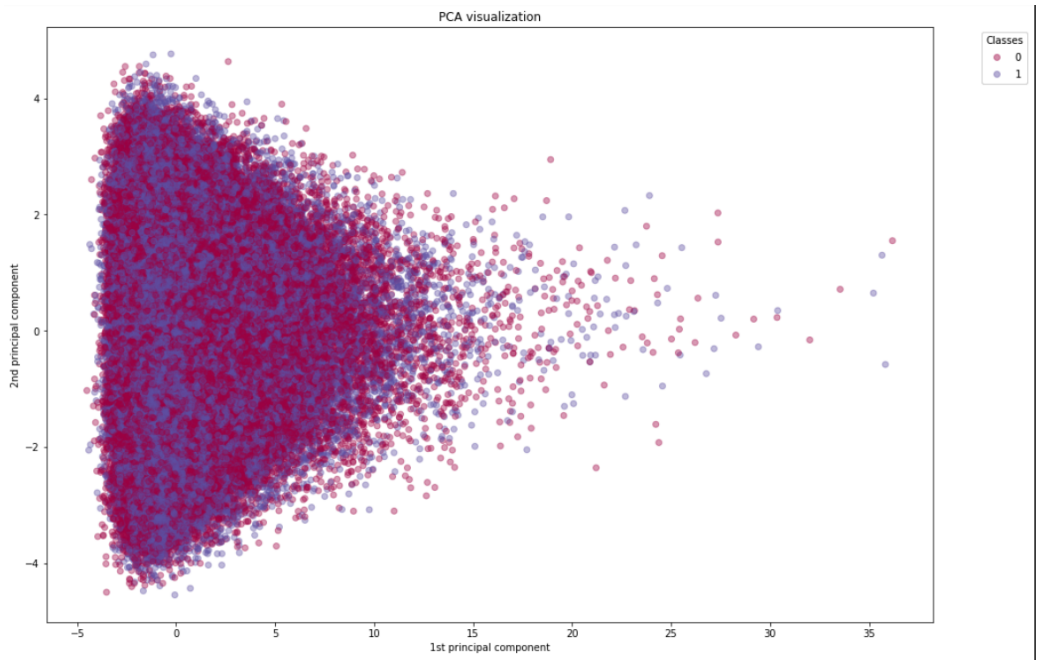
All these things are studied using physical tools and simulations as they provide a favourable environment where physical properties, histories and the future of all gases are all known and can be manipulated according to needs. We can use simulations from a broad range of techniques and groups to look for insights into how the CGM participates in galaxy evolution.

Implementation Plan

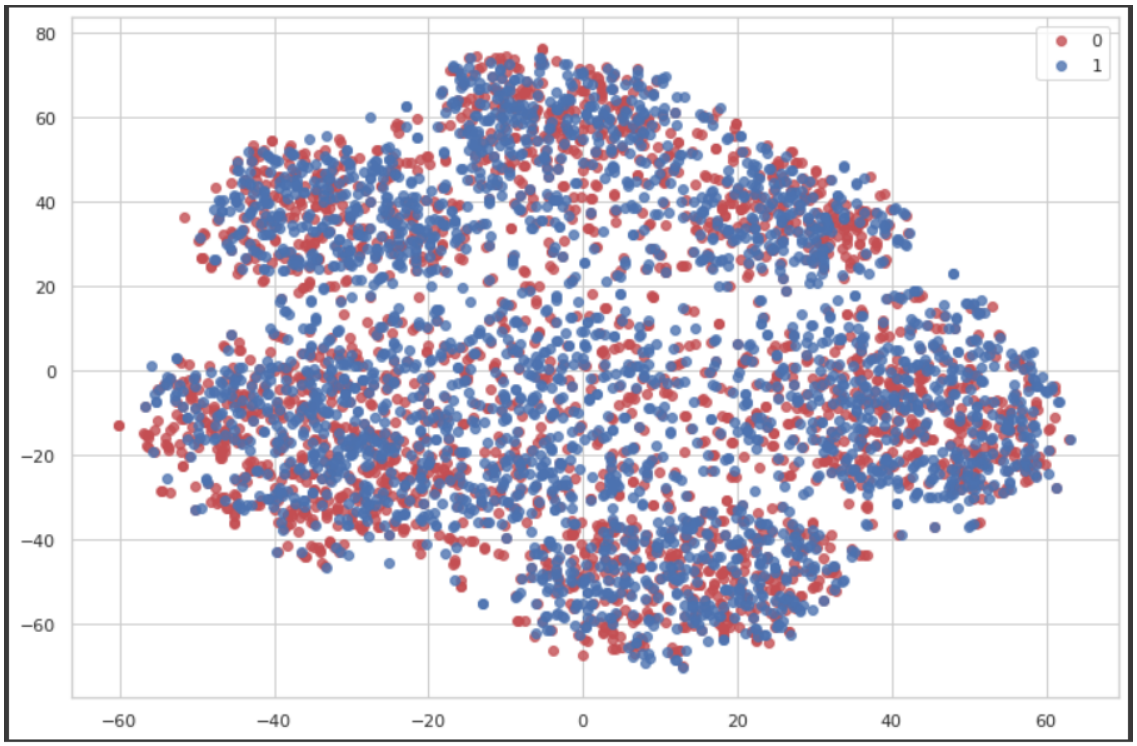
The project consists of the following plan:

1. Visualizing the dataset

   At first, it's important to visualize the dataset to get a rough idea about how the data is distributed in the space. This makes the data more natural for the human mind to comprehend and thus makes it easier to identify patterns, randomness and outliers within large datasets. There are various techniques to visualize a dataset either in 2D or 3D space which includes using Principal Component Analysis (PCA) to decompose the dataset into the required number of components (say 2) and then analyze the data with the 2 principal components. For example, as we have used PCA to decompose the dataset for the Higgs Boson sample.



Another way to visualize the data more distinctly and properly is using t-distributed stochastic neighbour embedding (t-SNE). It is computationally expensive to plot data using t-SNE for high features so it is suggested to reduce the features first using PCA then use t-SNE to visualize the data.



Here, we used t-SNE to plot the first 5000 data of the Higgs Boson dataset. As you can see, it is far more superior than PCA because PCA is a linear dimensionality reduction technique whereas t-SNE is specifically designed to handle non-linear data and it does so by preserving the neighbourhood of every point while embedding to lower dimension.

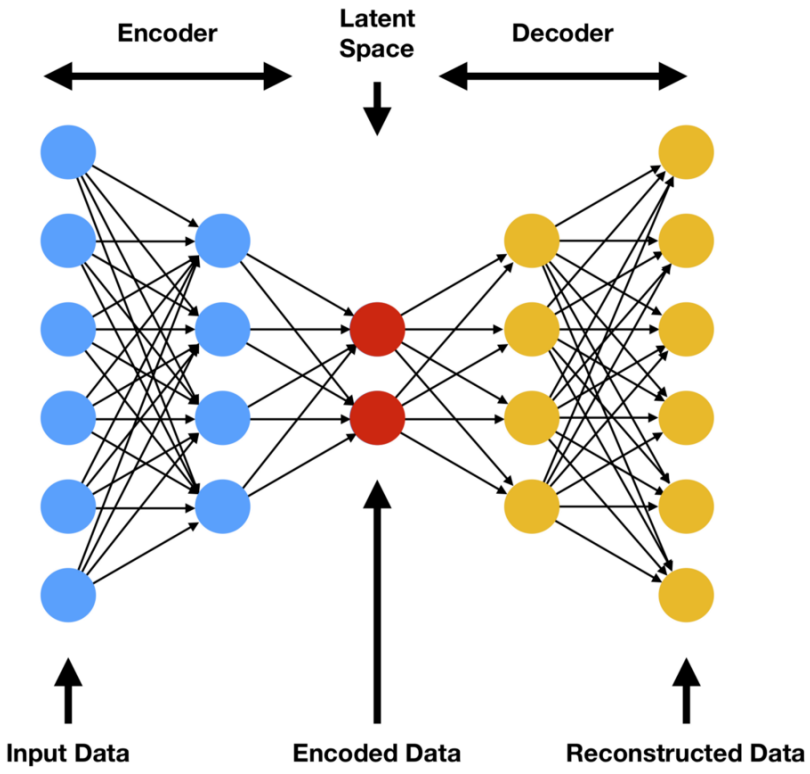
Uniform Manifold Approximation and Projection (UMAP) can also be used in-place of t-SNE as it is less computationally expensive and yields comparable visualization performance with t-SNE.

2. Feature Scaling the Data

Feature scaling/ normalisation is important to reduce skewness towards a particular feature while training the model. Scaling also makes the training faster and prevents the model from getting stuck in local optima. However, some models such as Logistic regression and Decision Trees do not require scaling the data but in the case of neural networks, normalisation is required. In the case of neural networks scaling allows the gradient descent to converge much faster than without it. One more reason is that in the activation function scaling would help not to saturate too fast.

3. Building the Neural Network

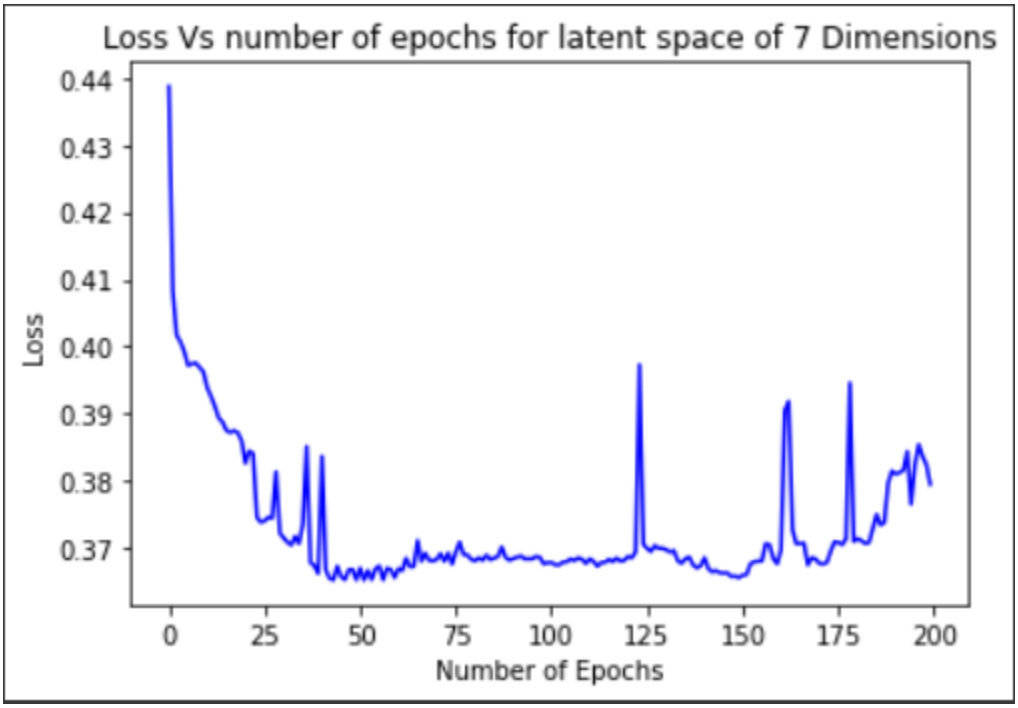
The final step includes building the neural network for the preprocessed data. Using Principal Component Analysis (PCA) allows us to significantly reduce the number of features but with the cost of reducing much accuracy of the classifier as PCA is essentially a linear transformation and based on the principle that higher variance features have more importance thus removing the lower variance features. However, Autoencoders are capable of handling complex non-linear functions. A single-layer autoencoder with a linear activation function is pretty similar to PCA. An autoencoder is a deep neural network that learns to compress data from the input layer into a compressed representation and then uncompress the data into something that matches the raw data. In abstract terms, an autoencoder learns the inputs given to it.



Implementing an autoencoder in Keras looks something like this

```
# Building the Input Layer
input_layer = Input(shape =(X.shape[1], ))

# Building the Encoder network
encoded = Dense(25, activation ='tanh',
                activity_regularizer = regularizers.l1(10e-5))(input_layer)
encoded = Dense(22, activation ='tanh',
                activity_regularizer = regularizers.l1(10e-5))(encoded)
encoded = Dense(19, activation ='tanh',
                activity_regularizer = regularizers.l1(10e-5))(encoded)
encoded = Dense(16, activation ='tanh',
                activity_regularizer = regularizers.l1(10e-5))(encoded)
encoded = Dense(13, activation ='tanh',
                activity_regularizer = regularizers.l1(10e-5))(encoded)
encoded = Dense(10, activation ='tanh',
                activity_regularizer = regularizers.l1(10e-5))(encoded)
encoded = Dense(7, activation ='tanh',
                activity_regularizer = regularizers.l1(10e-5))(encoded)

encoded = Dense(5, activation ='relu')(encoded)    ## latent space

# Building the Decoder network
decoded = Dense(7, activation ='relu')(encoded)
decoded = Dense(10, activation ='relu')(decoded)
decoded = Dense(13, activation ='relu')(decoded)
decoded = Dense(16, activation ='relu')(decoded)
decoded = Dense(19, activation ='relu')(decoded)
decoded = Dense(22, activation ='relu')(decoded)
decoded = Dense(25, activation ='relu')(decoded)

# Building the Output Layer
output_layer = Dense(X.shape[1], activation ='relu')(decoded)
```

An autoencoder consists of three parts: the encoding layer, the latent space/bottleneck layer and the decoding layer. Here, we have used 7 layers for encoding and 7 layers for decoding. Since here our input data is real numbers we have used a non-linear activation function for the encoding layer (tanh) and for the decoding layer we have used the relu function.
Making the neural network more deep makes it susceptible to overtraining if the training data is small. So to avoid overfitting we need to provide it with lots of data. Regularizers are one more important thing to be used while avoiding overfitting. Without the use of regularizers the loss function looks something like this:



However, training the model endlessly without any restriction on the trainable parameters might lead to overfitting. In regularization, we restrict the size of the parameters the model is learning.

Regularization in terms of the loss function

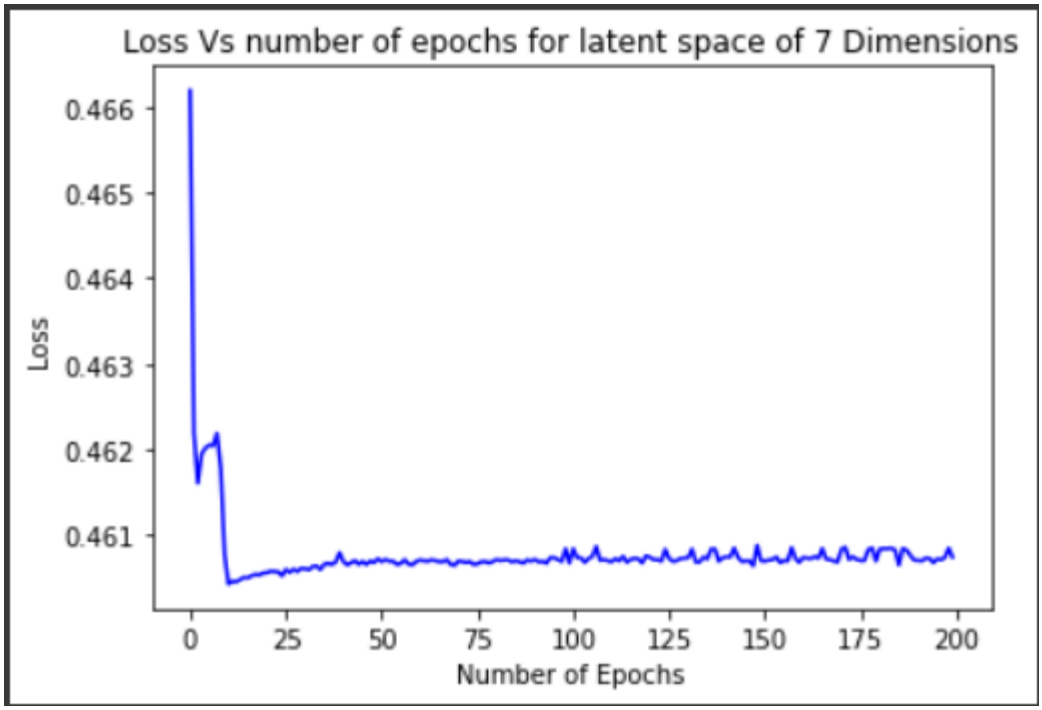
1. L2 :
   Reduces the trained parameters to small values.

$$CostFunction = \sum_{i=1}^{n}(x^{(i)} - output^{(i)})^2 + \lambda \sum_{i=1}^{n} \beta_i^2$$

2. L1 :
   Knocks a few trained parameters to zero.

$$CostFunction = \sum_{i=1}^{n}(x^{(i)} - output^{(i)})^2 + \lambda \sum_{i=1}^{n} |\beta_i|$$

After we successfully apply the L1 regularizer in the neural network the trained model looks like this:



4. Training the neural network

After building the neural network we need to train it on the data.

```
autoencoder = Model(input_layer, output_layer)



autoencoder.compile(optimizer = "adam", loss ="mse",metrics=['accuracy'])

# Early stopping to avoid overfitting

callback = tf.keras.callbacks.EarlyStopping(patience=5,restore_best_weights=True)

# Training the Auto-encoder network
autoencoder.fit(X_scaled_0, X_scaled_0,
                batch_size = 32, epochs = 50,
                shuffle = True, validation_split = 0.20,callbacks=[callback])
```

Here, we have used Keras to train the neural network. The optimizer can be used depending on the nature of the problem. Adam is a type of replacement optimization algorithm for stochastic gradient descent for training deep neural networks. It combines the best properties of

AdaGrad and RMSProp algorithms to provide an algorithm that can handle sparse gradients on noisy problems. However, we are free to test with other optimization algorithms also. For the loss function, Mean Squared Error (MSE) is used when for regression analysis whereas binary cross-entropy is used for classification problems. Batch size and epochs have to be tested for different values to obtain the level of accuracy however too many epochs can cause overtraining and the model not learning at all. Thus, the use of early stopping is highly recommended to avoid overtraining.
One more thing worth to be mentioned is to tune the value of the learning rate, by default in Adam it is set to 0.001. We can use exponential decay to tune the learning rate while training the model to achieve the desired accuracy. It can be implemented using TensorFlow:

```
tf.compat.v1.train.exponential_decay(
    learning_rate, global_step, decay_steps, decay_rate, staircase=False,
name=None
)
```

The function returns the decayed learning rate which can be computed as:

```
decayed_learning_rate = learning_rate *
                        decay_rate ^ (global_step / decay_steps)
```

For example decay every 100000 steps with a base of 0.96:

```
global_step = tf.Variable(0, trainable=False)
starter_learning_rate = 0.1
learning_rate =
tf.compat.v1.train.exponential_decay(starter_learning_rate,
global_step,
                                           100000, 0.96, staircase=True)
# Passing global_step to minimize() will increment it at each step.
learning_step = (
    tf.compat.v1.train.GradientDescentOptimizer(learning_rate)
    .minimize(...my loss..., global_step=global_step)
```

Batch normalization will also be used to improve the deep neural networks performance by significantly reducing the number of training epochs to train the network. It would also allow fixing the problem of vanishing gradients. As an example, in the above plot, after we use the L1 regularizer, the loss doesn't reduce because of vanishing gradients. Using batch normalization we would have obtained a smooth curve.

Timeline

This week-by-week guideline provides an abstract of how the project will be done.

Phase 0: [Pre GSOC Period]

Before May 17:

- To familiarize myself with Circumgalactic Medium (CGM), star formations, Quasars and their absorption spectra, galaxy evolution and cosmology.
- To study some more papers and watch explanatory videos related to CGM to get more comfortable with complex problems.
- Gain in-depth knowledge on t-SNE and UMAP and visualizing data in 3D space.

Phase 1: [Community Bonding Period]

May 17 - June 7:

- To develop good bonding with all the mentors and discuss my final goals about the project.
- During this period I will be in constant touch with my mentors. I will remain active on the IRC channel and mailing lists to discuss and finalize the proposed goals and modifications(if any).

Phase 2: [Coding Period 1]

June 7 - July 12:

- To analyze the dataset using t-SNE and UMAP either in 2D or 3D space and rendering using GPU for performance and speed up.
- To find patterns/distributions among the data and to draw a conclusion about the dataset.
- To apply feature scaling/ normalisation in the dataset.
- To apply PCA to obtain a rough estimate about the variance of the features.

Phase 3: [GSOC Phase 1 Evaluations]

July 12 - July 16:

- This period will be used to write a detailed report about the work done in coding period 1. All the work done will be uploaded and documents will be created/uploaded in a GitHub repo.

Deliverables

- To apply one or more dimensionality reduction techniques to visualize the dataset either by using PCA along with t-SNE or UMAP or both.
- Full documentation of the dataset after observing patterns and distributions.
- Apply Feature Scaling/ normalisation.

Phase 4: [Coding Period 2]

July 16 - August 2:

- Build the neural network (autoencoder) using TensorFlow's Keras library.
- Train the neural network using Keras.
- Tune the neural network by implementing early stopping, exponential decay of learning rate, experimenting on different batch sizes and optimizers and implementing batch normalization for obtaining the desired level of accuracy.

August 2 - August 16:

- A buffer of two weeks has been kept for any unpredictable delay or adding additional code improvements.

Deliverables

- Implementing the neural network for Dimensionality Reduction using Keras.
- Experimenting with different values of the learning rate, batch size and using early stopping to counteract overtraining.
- Plotting the graphs for each experimented value and observing similarity and differences among them.
- Documentation stating results obtained by experimenting with different values and optimizers for the neural network.

Phase 5: [Final Evaluations]

August 16 - August 23:

- All final documentation and codes will be uploaded to the GitHub repo and all the deliverables promised will be provided by this stage.

Personal Information

Personal Details

I'm Rituraj Dutta, an undergraduate pursuing my Bachelors at Guwahati University Institute of Science and Technology (GUIST), Guwahati, India. I started programming back when I was 16 till then I have always been fascinated by how programming can change the world and how it can change the way you find solutions for every problem. I believe programming is an art and you need the patience to master it. I have been doing programming for 3 years now and still, I'm not done yet and want to learn more. I have also been interested in Physics for quite a long time. I have been studying books related to Astrophysics and Nuclear Physics back when I was 15. Seriously, I used to understand complex particle physics phenomenons such as Compton Scattering, Inelastic Scattering, Elastic Scattering when our teachers were just starting to explain nuclear fission and fusion. I never thought about how we can harness the power of programming to solve complex problems in Physics but now since a year or so after I started learning Machine Learning I came to realize how Machine Learning can make things easier for us. I think I'm the right fit for this project since I'm interested in the theoretical part as well as the implementation part and I'm always ready to learn more.

I do most of my coding in Windows but when it comes to dealing with complex tasks I shift to my Ubuntu 20.04 LTS. I've also won two of my inter-college hackathons in which the first one we designed a Smart Irrigation Monitoring System and in the second one we gave a solution to classify brain signals generated via EEG for stress management. I also have a little exposure to competitive programming. My codechef handle: **ri_2_raj**

Background Work

I have previously worked on a case study to classify breast cancer whether the tumour is malignant or benign. I have used Support Vector Machine (SVM) along with a Radial Basis function (RBF) kernel as a classifier to classify the tumour. My model achieved 87.23 % accuracy with k-fold cross Validation and after using Hyperparameter tuning using Randomized Search my model achieved 91.46 % accuracy. Currently, I've started my research work on Few Shot Learning which is a type of Machine learning problem where the training dataset contains limited amounts of information. Currently, I'm going through Prototypical Neural Networks for Few Shot learning by improving it to work more precisely in the case of Zero-Shot learning. I also have working knowledge in Reinforcement Learning (Multi-Arm Bandit Problem), Associate Rule Learning (Apriori), Deep Learning specifically in Artificial Neural Networks and Convolutional Neural Networks and also about Gradient Boosting Machines(GBM) which is according to me best for large datasets. I am familiar with most Machine Learning Models for Supervised and Unsupervised learning.

Post GSoC Plans

I would love to work on this project or any similar project (if any) after GSOC as new things always fascinate me. I have always been an eager student in Computer Science and Physics and always wanted to learn out of the box and if given a chance I'll give my best to solve the project(s) and provide the desired outcome. I'm always open to learning and sharing new things in this field. I would be eager to study and work more on CGM as this is arguably the last major components of galaxies to be added and our final understanding of how galaxies are formed but it will be also much more challenging to work on as it much fainter and smaller as compared to the IGM. I am hoping to take up this problem soon.