CE6018 End Semester

Pravin Ravi [CE22B092]

Question - 65

a. **Bayesian Neural Network using Dataset-05**

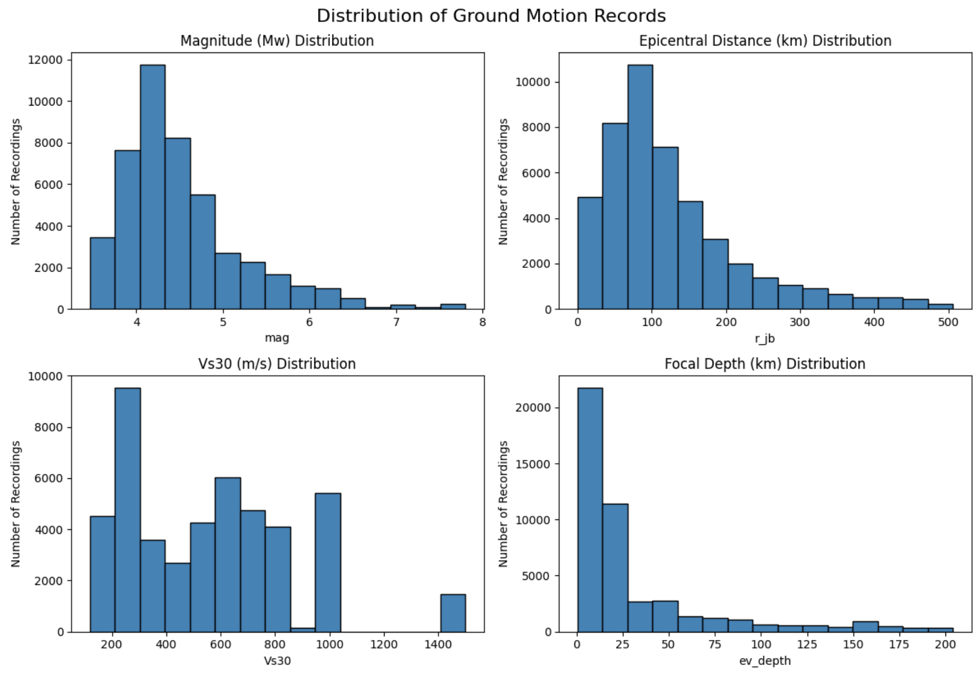
We implement a Bayesian Neural Network (BNN) using torchbnn to predict Spectral Acceleration (SA) values based on seismic parameters. The model accounts for uncertainty by learning distributions over weights, making it suitable for high-risk applications like earthquake engineering.

Unlike other models that deal with deterministic data, this model deals with probabilistic data. For very uncertain data like EQ data, such a model can truly prove to have more accuracy as compared to other models.

However, the disadvantage of this model is that the number of unknowns are more than twice as large as a normal ANN. Therefore the run time also increases significantly.

Data:

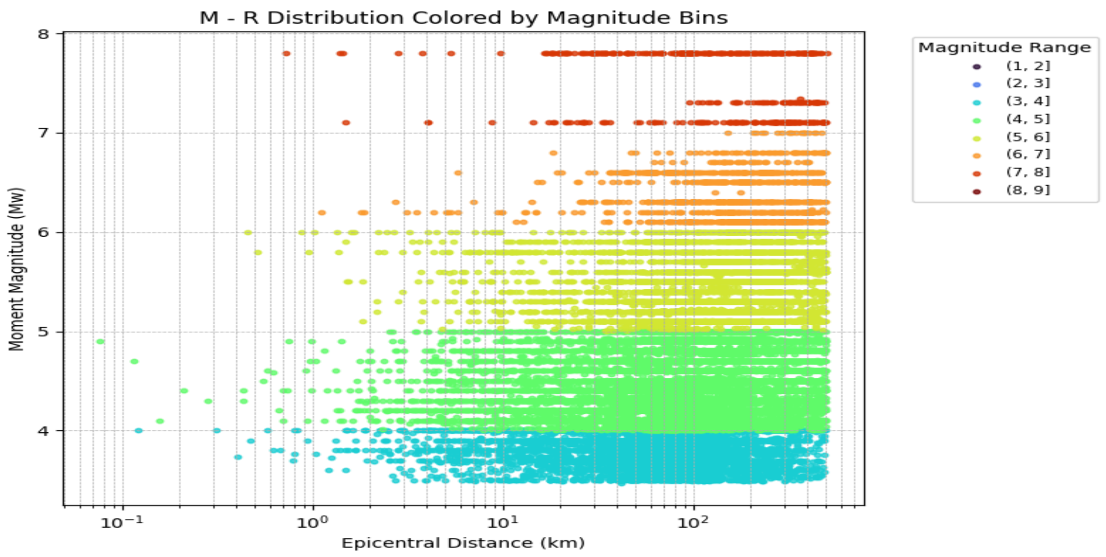
The explicit input variables available are: Magnitude, Rjb, Vs30 and depth.
We can plot the frequencies of each:



As we can see, Magnitude and epicentral distance are more or less normally distributed.

The input parameters we will be using to train the model are:
X = {mag, r_jb, log(r_jb), depth, log(Vs30)}

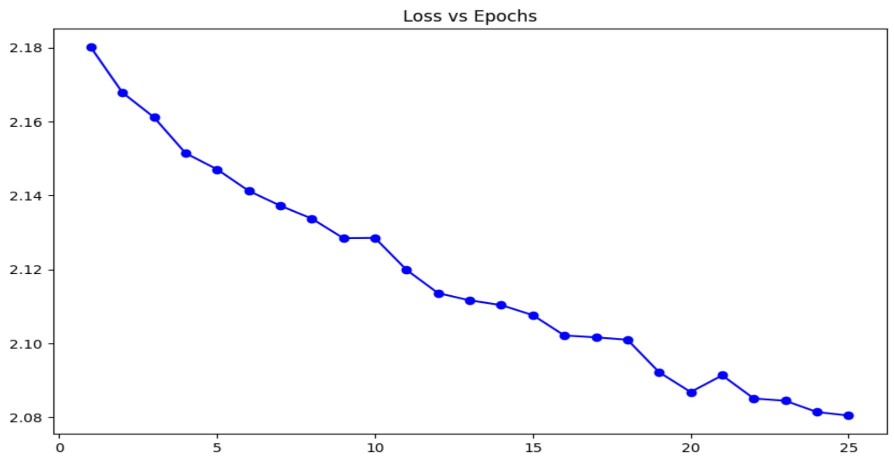
We can check the M-R distribution as well.



The data seems quite uniform for the low magnitude events. However for the high magnitude events, there are only high epicentral distance data. This can have some overfitting effects towards the low and medium magnitude earthquakes (mag < 6).

Training the model:
I have used torchbnn to train the models.
Initially, I had a template code used for another dataset. The model is as follows:
Hidden Layers = 256, 128, 64 neurons

However, after training this was the loss vs epoch curve

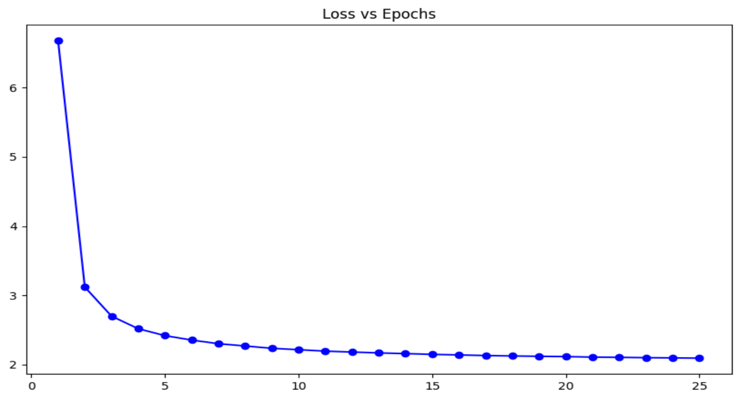


This shows that the model is not stable for the given dataset.
Reason being that hidden layer neurons are much more than input + output layers' neuron count. Also, training time is high.

Then I used a 3 layer BNN with the following hyper parameters:

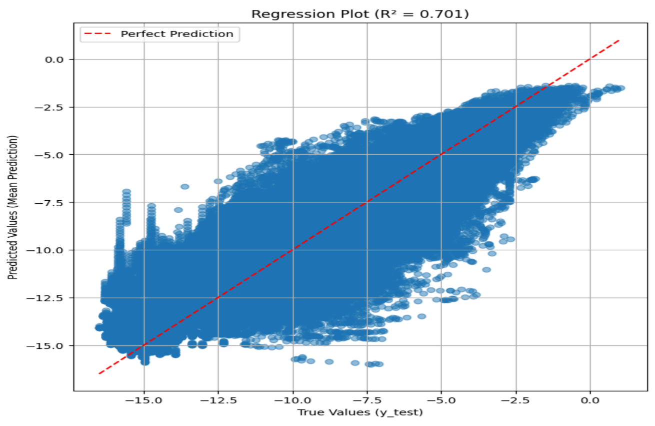
Number of layers = 3
Number of neurons per layer = 5 (input), 30, 50, 70, 100(output)
Learning rate = 0.001
Number of epochs = 25 (due to time constraint of exam)

Here are the results:

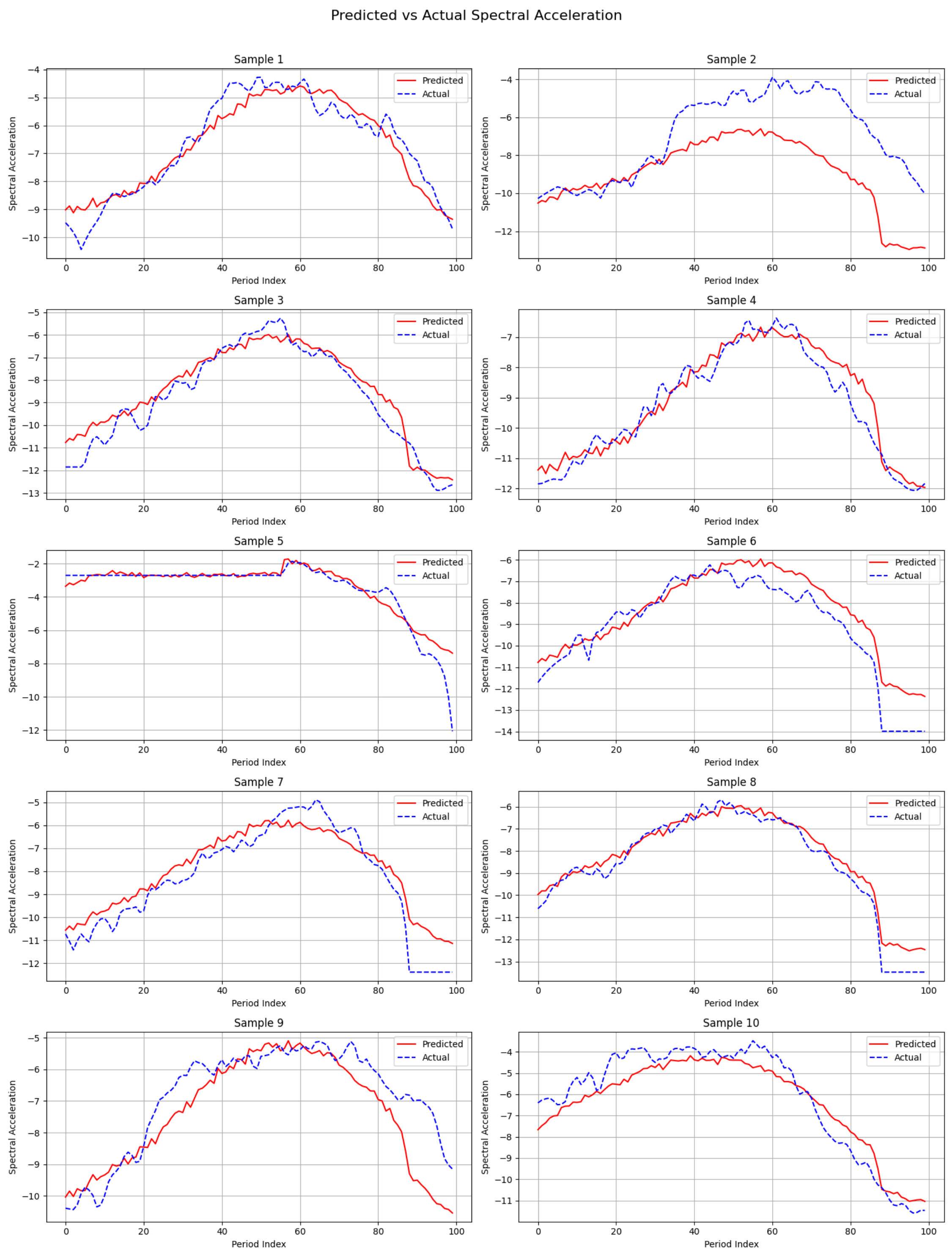


Now, the loss curve is much better than the initial case as it reaches a minimum.
Average loss = 2.0943753085805876
R^2 = 0.701
This shows that the model is a great fit for the data as R^2 is over 70%!

Regression Plots:



When the data is closer to the red dotted line, then the accuracy is greater. This shows that there is some distortion from the original data but still it has a high R^2.



These are some of the predictions that this model gave. There is a very good accuracy as the data predicted is very close to the actual data.

Hyper-parameter tuning:
Due to limited time available to train the models, I reduced the epochs to 10 to test out different models.

Firstly, testing using the number of layers

Number of Layers	Hidden Layer Neurons	R^2 Value
1	50	0.689
2	30, 70	0.691
3	30, 50, 70	0.701

As we can see, increasing the number of layers has improved the performance of the model. This is due to increase in the number of unknowns in the model.
However, the negative is the increase in the run time.

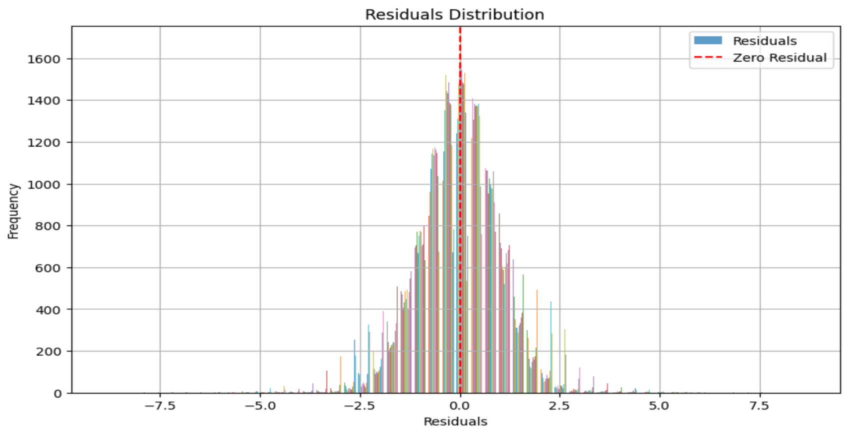
Furthermore, after increasing the number of neurons in the 3 layer network, I compared the results.

New network hidden layer neurons = 40,50,80

As expected, the model performed better over 25 epochs.

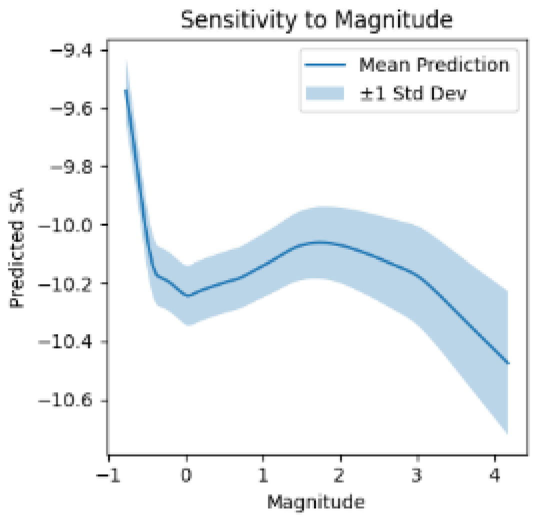
R^2 value = 0.702

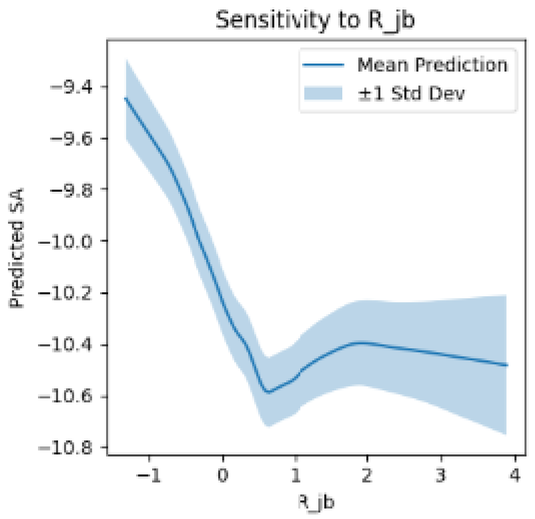
However, there is not a significant difference.



- The residuals distribution is clearly normally distributed showing that the model has well behaved.
- Also, the mean of residuals is zero indicating that the model is unbiased.
- There is no significant positive or negative residual

Checking Sensitivity to Parameters:





Both have sharp slopes indicating dependence of each variables to be high.
Non-linear graph shows complex interdependence between y and the parameters.

Conclusion:
The BNN model accurately predicted SA while quantifying epistemic uncertainty. Residuals and sensitivity plots were consistent with ground motion theory. This approach offers interpretable, uncertainty-aware predictions essential for seismic risk modeling.

b. DBSCAN and GMM using Dataset-06

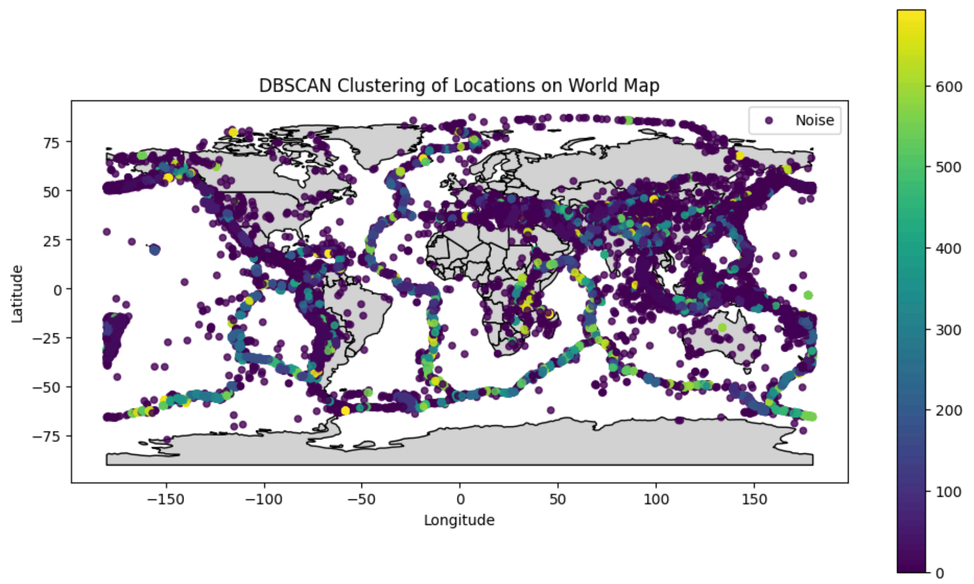
For our code, we are going to use latitude, longitude and depth to demonstrate how each model performs and draw inferences.

DBSCAN:

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is an unsupervised clustering algorithm that groups together points that are closely packed (high density areas) while marking points in low-density regions as outliers.

It requires two main parameters: eps (maximum distance between two points to be considered neighbors) and min_samples (minimum number of points to form a dense region).

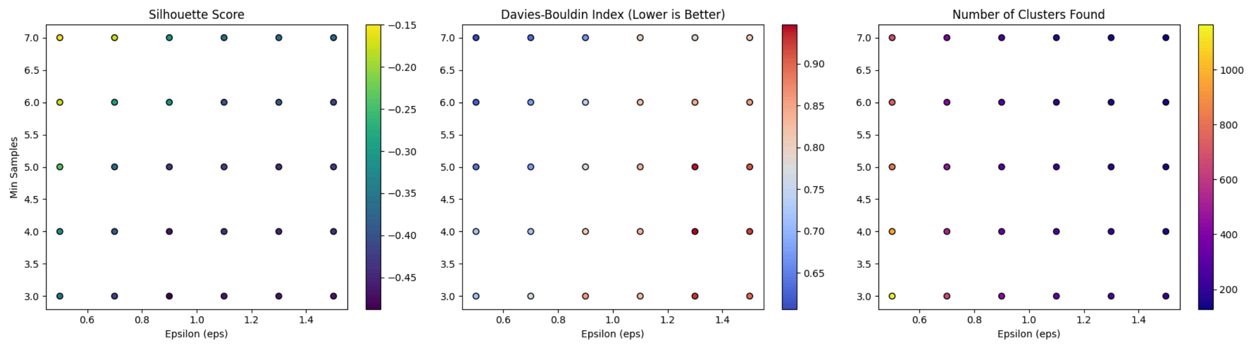
Code used for training the model has been attached in the submission mail. Thos report will present the clustered plots.



This is for eps = 0.5 and min_samples = 7.

In order to check which hyper-parameters perform the best for the given data, I used Silhouette Score, Davies-Bouldin Index, and Calinski-Harabasz Score to evaluate clustering quality. Silhouette measures how well-separated clusters are, DB Index checks how compact they are, and CH Score favors distinct, well-spread clusters.

Results:



Best Hyperparameters Found:

- Epsilon (eps): 0.5
- Min Samples: 7
- Best Silhouette Score: -0.1499

A negative Silhouette Score means points are closer to other clusters than their own. This happens when clusters overlap or when too many points are marked as noise, making the remaining clusters poorly defined.

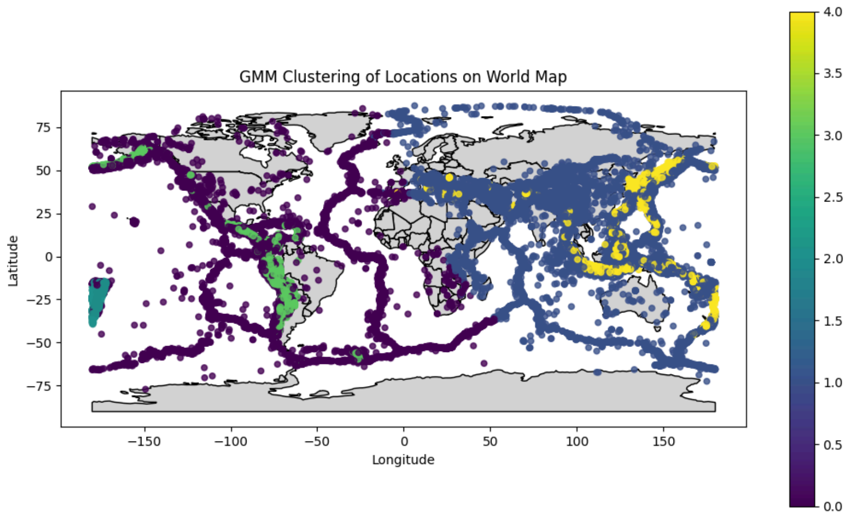
Overall Conclusions

1. Density-based methods work well for spatial data like earthquakes but struggle when density varies too much or data is too sparse.
2. For large datasets (70,000+ points), model-based clustering like Gaussian Mixture Models (GMM) or hierarchical clustering might perform better.
3. When we limit the number of points, we seemingly get better results in these density based functions.

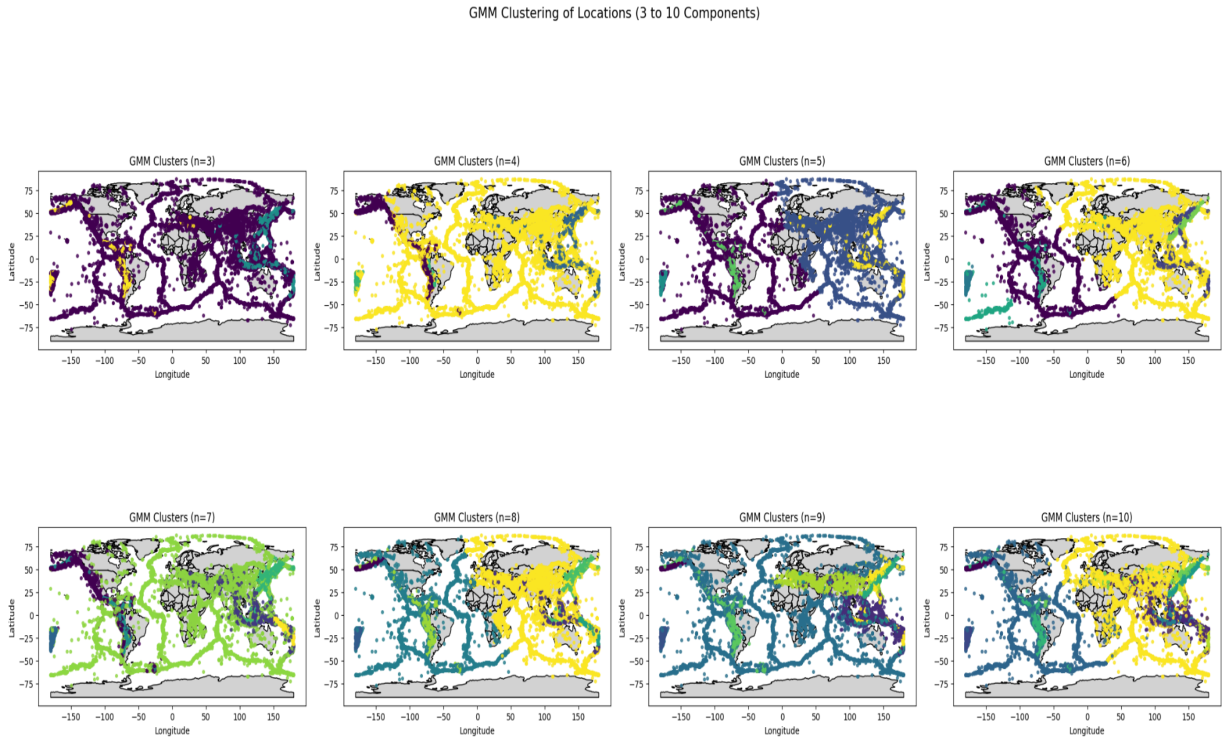
Gaussian Mixture Model:

Gaussian Mixture Models (GMM) are a probabilistic clustering method that assumes data points are generated from a mixture of several Gaussian distributions, each representing a cluster.

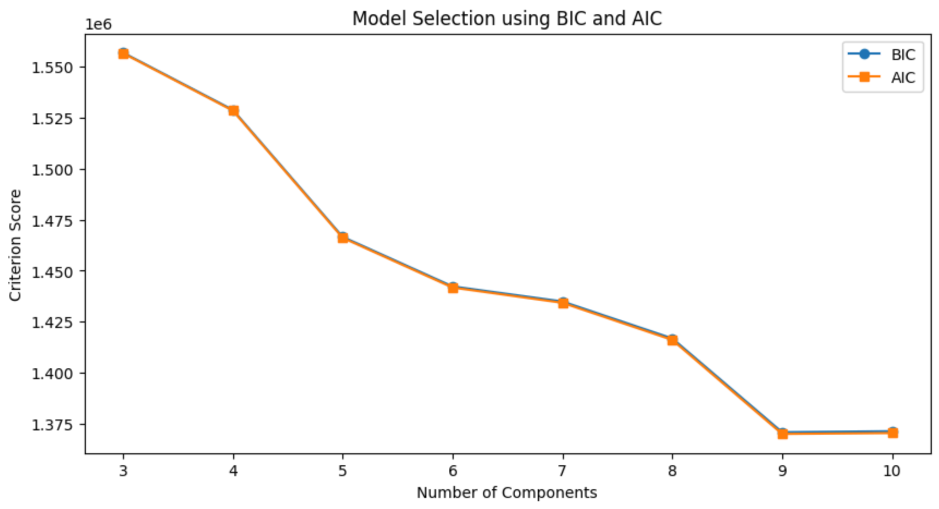
Unlike hard clustering methods like K-Means, GMM assigns probabilities to each point belonging to each cluster, allowing for soft clustering. This makes GMM particularly effective when clusters overlap or have different shapes and sizes.



This is for 5 samples being plotted. As we can observe, the results are far better than what we got from DBSCAN. This is because GMM is better at clustering closely packed and large datasets.



To evaluate the effectiveness of the number of clusters shown, we can use Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC).



There is not clear elbow forming, however, 9 components seems to be quite low showing that it is the best result achieved out of this form of clustering.

Overall Conclusions

GMM performed well for large datasets by modeling elliptical clusters.

Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC) were used to determine the optimal cluster number, minimizing overfitting.

The BIC gradient method provided a refined estimate of the optimal cluster count.