DEEP-LEARNING MODEL FOR LUNAR EXPLORATION

Presenters: Amen, Kalpit, Rinoj
Advisors: Dr Xin Liang, Dr Daoru Han

Outline

Introduction to Lunar Regolith Beneficiation (Amen)

Data Processing and Feature Engineering for Yield prediction. Model development and analysis using Regression methods, and Deep Learning. (Kalpit)

Decision Tree, and SVM modelling for Yield prediction. Final results comparison and discussion. (Rinoj)

Lunar Regolith Beneficiation (Introduction)

My research project is focused on the Kinetic Modeling of the Electrostatic Sieve for Lunar Regolith Beneficiation

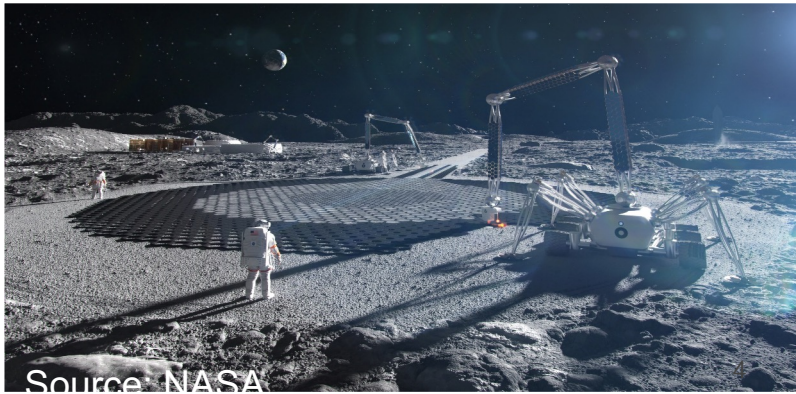
One of the goals of NASA for lunar exploration is *in-situ* resource utilization (ISRU) which means utilizing the available resources on the Moon instead of transporting them from Earth

Astronauts would need to source oxygen, water and other life sustaining resources for survival

Lunar Regolith Beneficiation

Materials such as aluminum and titanium would be needed for lunar surface construction

To do that, they would need *in-situ* techniques to reduce necessary equipment from Earth



Lunar Regolith Beneficiation (Electrostatic Sieve)

The goal is to investigate the electrostatic sieve method of regolith beneficiation

Regolith beneficiation involves separation of bulk lunar regolith (soil) to get the portion that contains the specific minerals we desire

Then we can use ISRU methods to get bulk necessary materials (aluminum or titanium)

Lunar Regolith Beneficiation

There are several methods of beneficiation

In this case, the goal is to develop a simple method that is portable, consumes little power

The electrostatic sieve uses the balance between the weight of the lunar soil particles and electrostatic force to separate the particles by size

Methods

Use of Accept-Reject method to sample lunar regolith particle radii

Use of electrostatic force from four-phase electric field

Use of Immersed Finite Element (IFE) Poisson solver

Allows for the collection of a select range of particle radii

Electrostatic Sieve

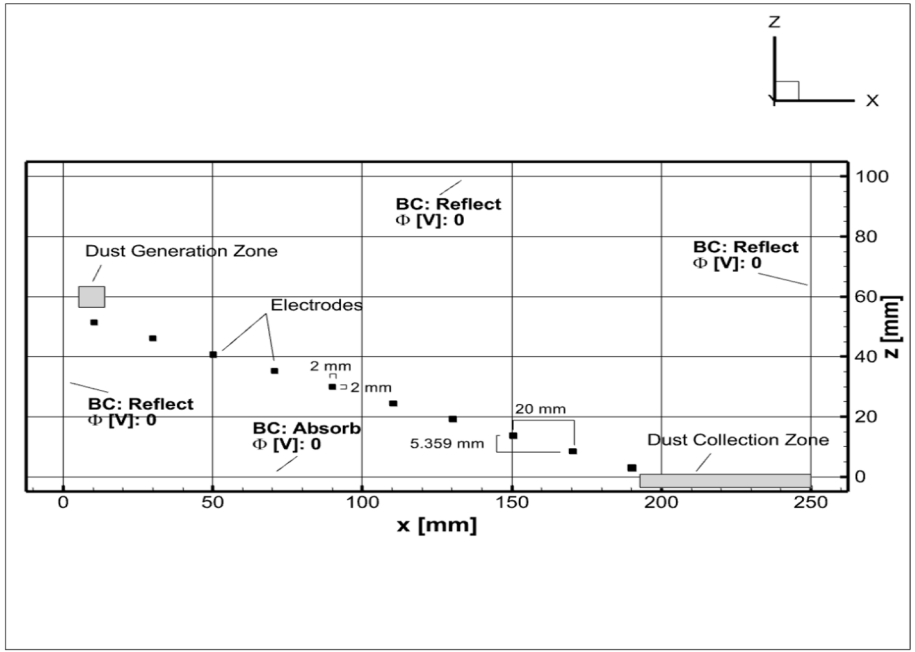
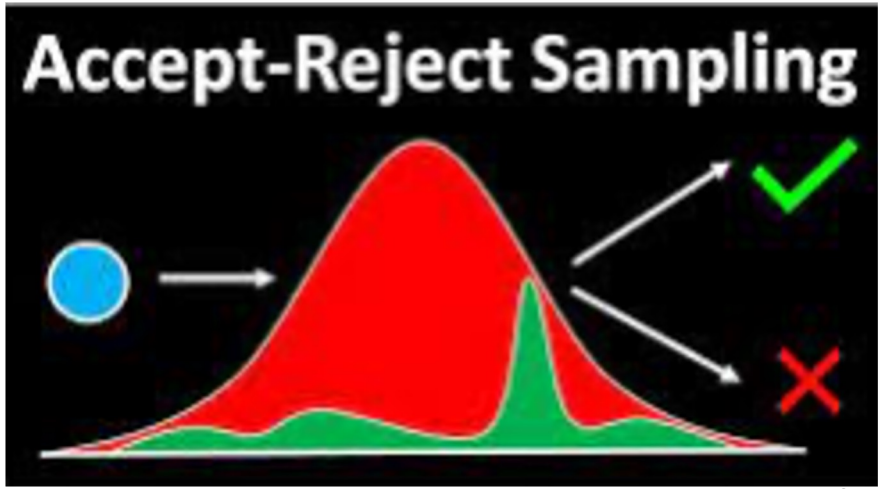


Fig. 2. Setup and boundary conditions of the simulation domain

Sampling of Lunar Regolith Particle Radii

Accept-Reject Method: This involves generating a random number, from a distribution function that is not easily sampled using an easily sampled distribution function



Source: Ritvikmath

Sampling of Lunar Regolith Particle Radii

The lunar regolith particle radii follow the Logarithmic-Normal Distribution according to Park et al. (2008), and it would be difficult to use the inverse of its cumulative distribution function (CDF) to generate samples

So, the Accept-Reject method is used to generate the particle radii using the uniform distribution to generate samples

Four-Phase Electric Field

The electric field was a four-phase electrostatic travelling wave introduced in (Kawamoto and Adachi (2014))

The square wave was created using an array of electrodes with potential of a certain magnitude (± 500, 1000, 2000V, etc.)

The phases oscillated with pattern: +--+, ++--, -++-, --++

The phase-shifting of the wave transported the particles

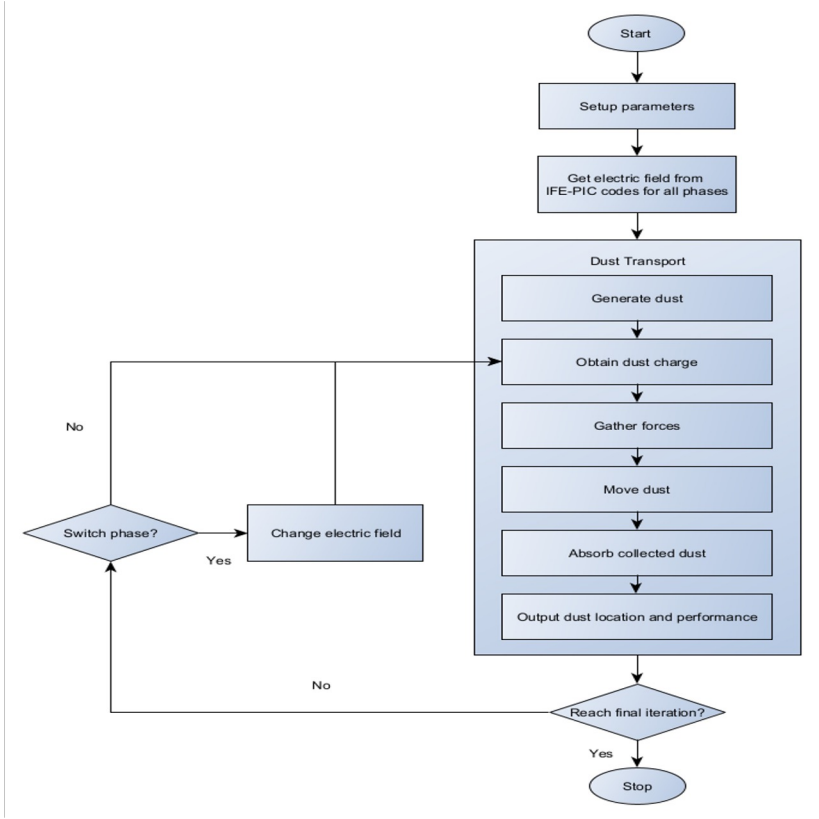
Use of IFE Poisson Solver

An IFE-Poisson Solver was used to solve the electric potential and electric field solving for Electrostatic Potential

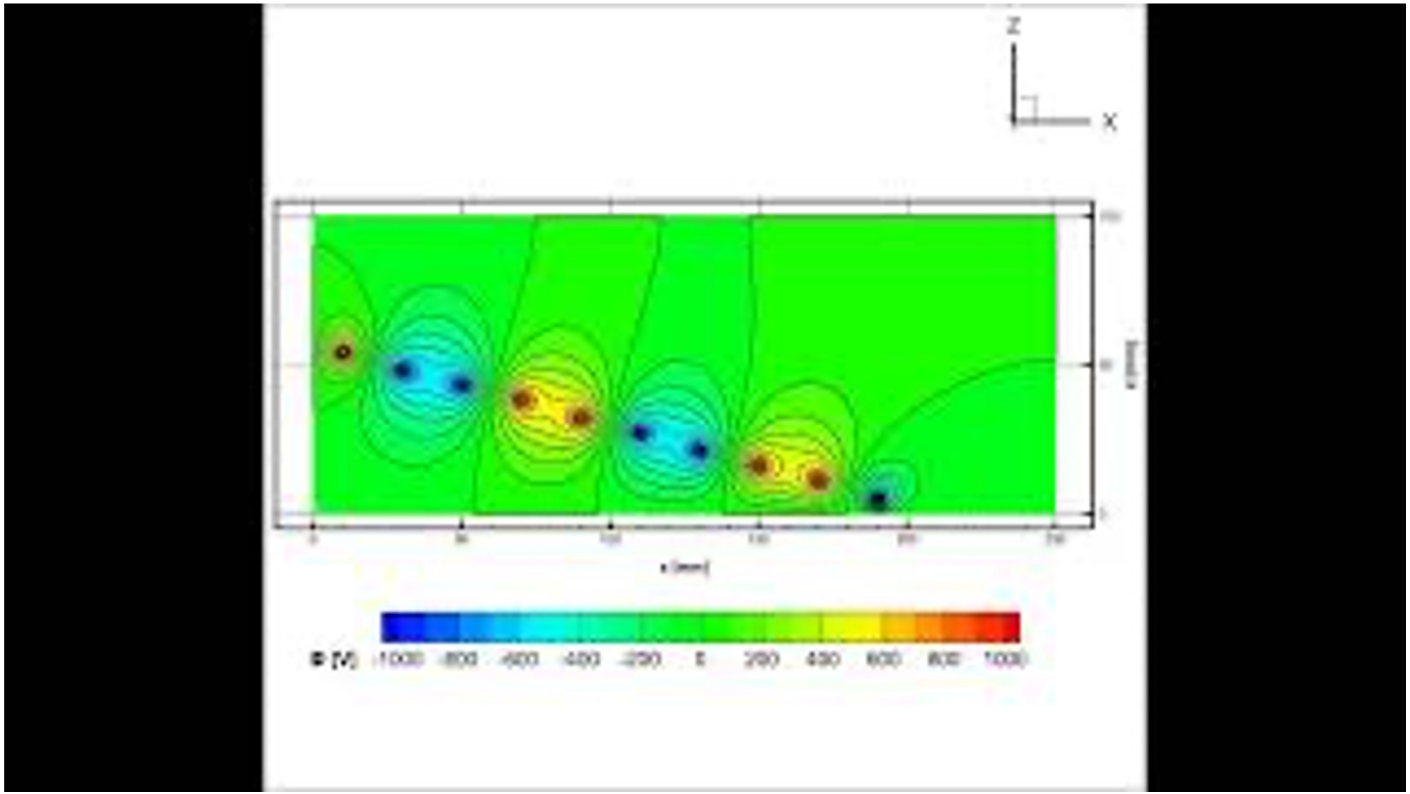
In this case, the Laplace's equation for electrostatic potential was solved since the dust particles were injected into the domain after the electric potential and field in the domain were calculated

It was assumed that there was no charge enclosed in the computation domain prior to injection of particles

Flowchart of the Process

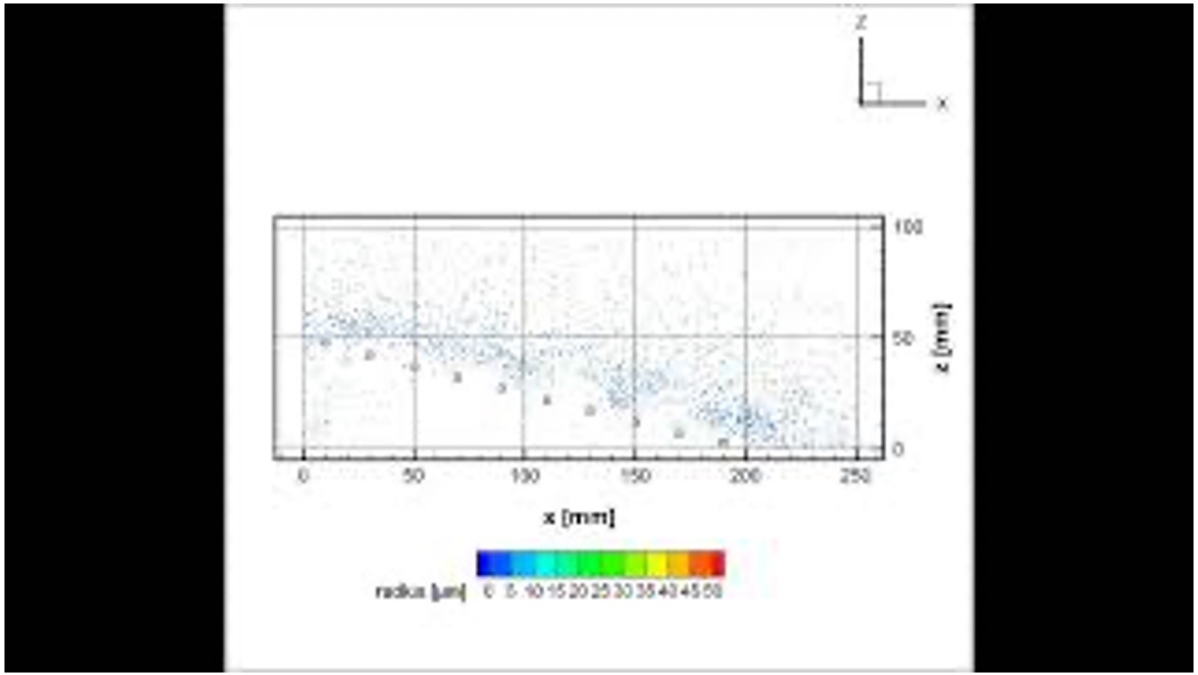


Four Phase Electrostatic Traveling Wave



Animation of Electrostatic Potential Contour

Charged Dust Motion



Animation of Charged Dust

Yield of the sieve

The yield of the electrostatic sieve is the ratio of the number of particles smaller than a certain radius (10 collected from the experiment to the total number of particles smaller than a certain radius injected into the domain

Data Generation for Model Testing and Training

The input parameters for the electrostatic sieve are the angle of inclination of the electrostatic sieve, the voltage magnitude, the voltage frequency, number of electrodes, electrode dimension, distance between electrodes, number of phases, the direction of inclination of the sieve, the mass density of the lunar soil particles, number density of the particles, and so on

The output parameter is the yield of the electrostatic sieve

Deep Learning Workshop Project

The plan for the workshop was to determine which combination of input parameters will give a certain yield for a certain lunar regolith radii range.

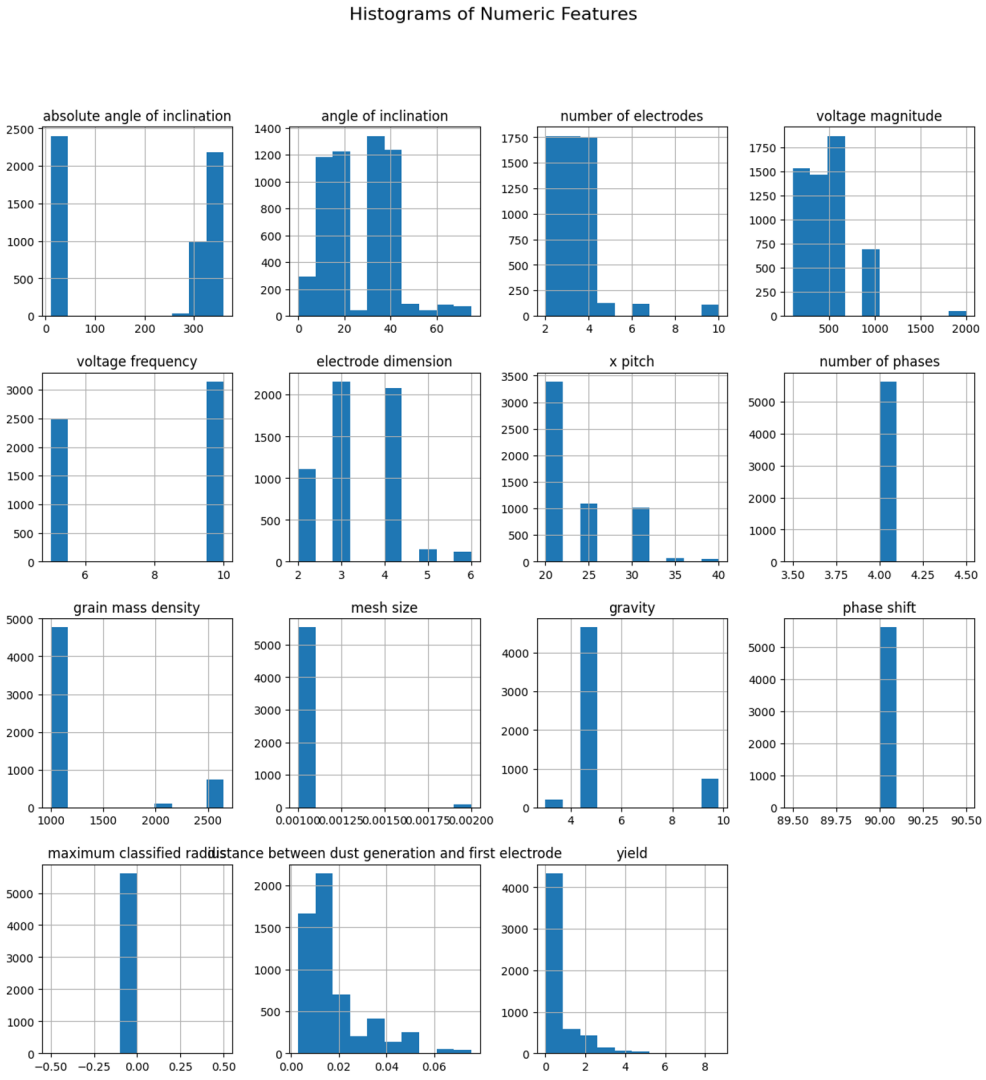
Also, we wanted to minimize the amount of time that it takes to generate the yield for one set of input parameters

Modelling the yield with traditional ML methods.

[47]:

	absolute angle of inclination	angle of inclination	number of electrodes	voltage magnitude	voltage frequency	electrode dimension	x pitch	number of phases	grain mass density	mesh size	gravity	phase shift	maximum classified radius	distance between dust generation and first electrode	yield
0	355.0	5.0	3.0	1000.0	10.0	2.0	20.0	4.0	2650.0	0.001	9.81	90.0	0.00001	0.007250	0.126490
1	355.0	5.0	4.0	1000.0	10.0	2.0	20.0	4.0	2650.0	0.001	9.81	90.0	0.00001	0.006375	0.092699
2	355.0	5.0	5.0	1000.0	10.0	2.0	20.0	4.0	2650.0	0.001	9.81	90.0	0.00001	0.005500	0.082674
3	355.0	5.0	6.0	1000.0	10.0	2.0	20.0	4.0	2650.0	0.001	9.81	90.0	0.00001	0.009626	0.175985
4	355.0	5.0	2.0	1000.0	10.0	3.0	20.0	4.0	2650.0	0.001	9.81	90.0	0.00001	0.007625	0.356726
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5615	360.0	0.0	10.0	2000.0	10.0	3.0	25.0	4.0	2650.0	0.001	9.81	90.0	0.00001	0.003500	5.148757
5616	345.0	15.0	10.0	2000.0	10.0	2.0	20.0	4.0	2650.0	0.001	9.81	90.0	0.00001	0.009885	5.838564
5617	345.0	15.0	10.0	2000.0	10.0	3.0	25.0	4.0	2650.0	0.001	9.81	90.0	0.00001	0.008356	8.673150
5618	330.0	30.0	10.0	2000.0	10.0	2.0	20.0	4.0	2650.0	0.001	9.81	90.0	0.00001	0.012038	4.840568
5619	330.0	30.0	10.0	2000.0	10.0	3.0	25.0	4.0	2650.0	0.001	9.81	90.0	0.00001	0.013548	4.931840

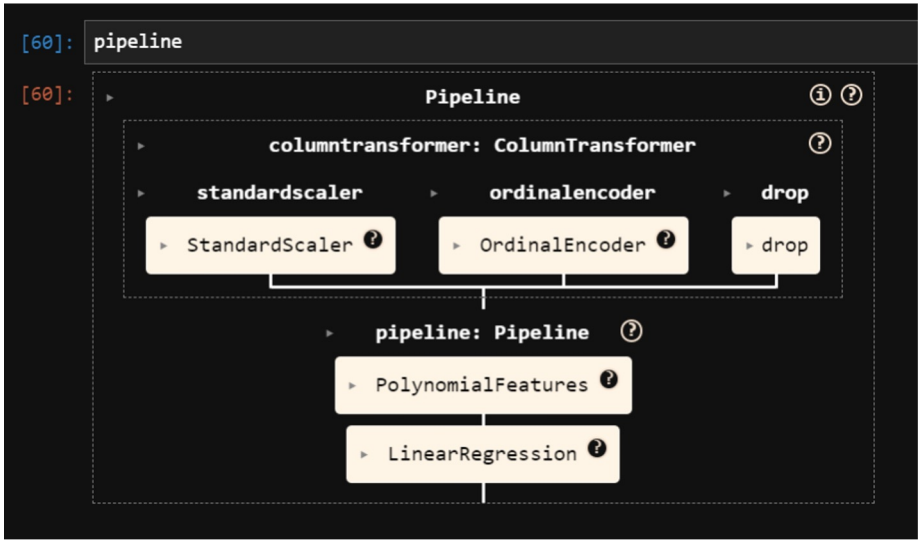
5620 rows × 15 columns


Histograms of Numeric Features
absolute angle of inclination
angle of inclination
number of electrodes
voltage magnitude
voltage frequency
electrode dimension
x pitch
number of phases
grain mass density
mesh size
gravity
phase shift
maximum classified radius
distance between dust generation and first electrode
yield

Data Pipeline Architecture

```
[53]: from sklearn.model_selection import train_test_split
      X = data.drop(target, axis=1)  # Features (independent variables)
      y = data[target]               # Target (dependent variable)

      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```



Loss Function and Model Selection

[56]:
```python
# Define RMSE calculation
def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))
```

[61]:
```python
# Models to be evaluated
models = {
    'dummy': DummyRegressor(strategy='mean'),
    'linear_regression': LinearRegression(),
    'ridge': Ridge(),
    'lasso': Lasso(),
    'polynomial2': make_pipeline(PolynomialFeatures(degree=2), LinearRegression()),
    'polynomial3': make_pipeline(PolynomialFeatures(degree=3), LinearRegression()),
    'polynomial4': make_pipeline(PolynomialFeatures(degree=4), LinearRegression()),
    'polynomial5': make_pipeline(PolynomialFeatures(degree=5), LinearRegression())
}

param_grid = {
    'ridge': {'ridge__alpha': [0.1, 1.0, 10.0, 100.0, 1000.0]},
    'lasso': {'lasso__alpha': [0.001, 0.01, 0.1, 1]}
}
```

Model Fitting and Evaluation

```
[63]: # Display the results
      pd.DataFrame(results_dict).T
```

[63]:

	fit_time	score_time	train_score	test_score	best_params
dummy	0.005596	0.002616	1.019214	1.018845	NaN
linear_regression	0.006031	0.002821	0.744382	0.746974	NaN
ridge	0.005771	0.002681	0.744382	0.746974	{'ridge__alpha': 0.1}
lasso	0.011862	0.003635	0.744399	0.746939	{'lasso__alpha': 0.001}
polynomial2	0.0161	0.004262	0.621655	0.628805	NaN
polynomial3	0.061839	0.005674	0.54648	0.568472	NaN
polynomial4	0.309767	0.016454	0.460928	0.511432	NaN
polynomial5	2.562855	0.02345	0.400722	2626877.69755	NaN

Simple NN with 2 Hidden Layers (Size: 32)

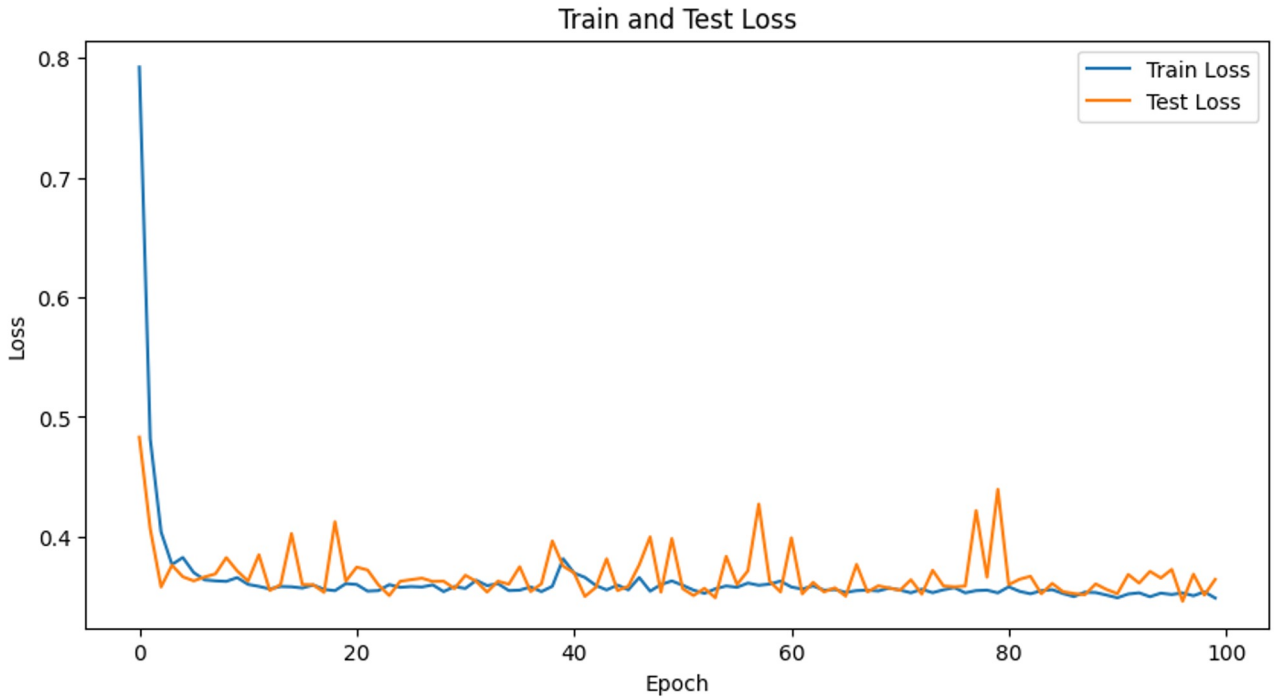
[5]:
```
dataset = CustomDataset(data)
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = torch.utils.data.random_split(dataset, [train_size, val_size])

train_loader = DataLoader(train_dataset, batch_size=1, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=1, shuffle=False)
```

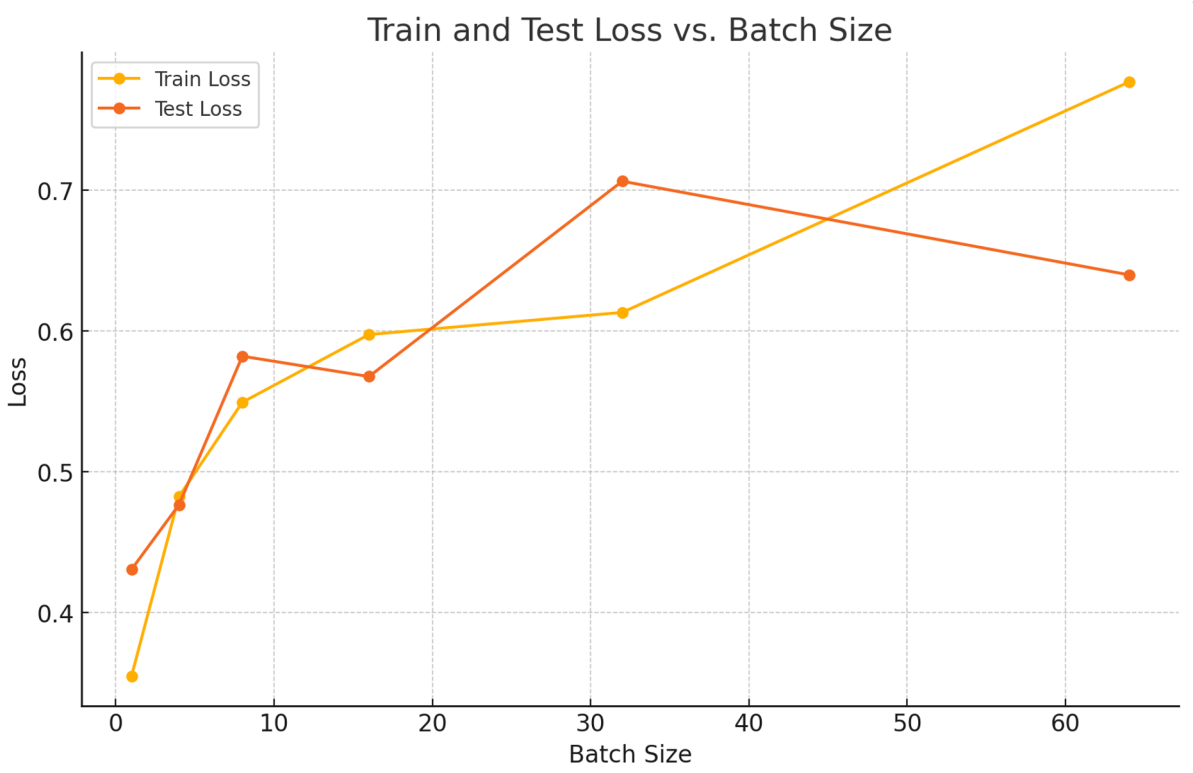
[6]:
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(len(numeric_features), 32)
        self.fc2 = nn.Linear(32, 32)
        self.fc3 = nn.Linear(32, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

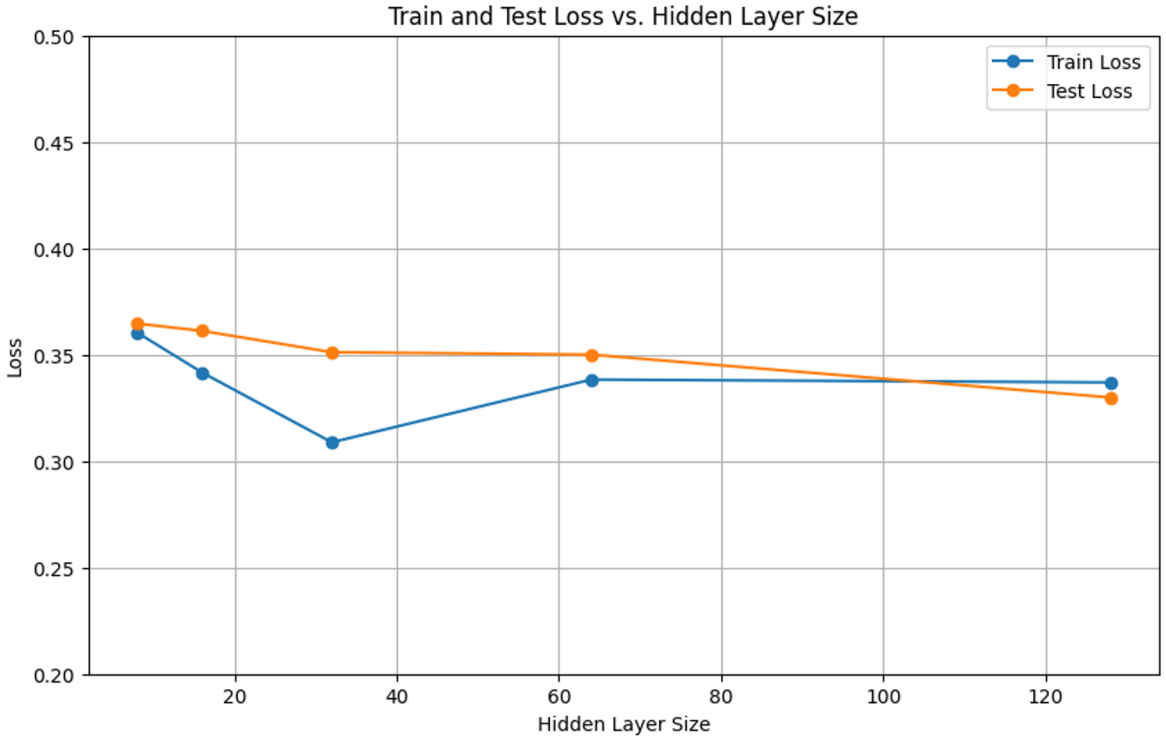
Simple NN with 2 Hidden Layers (Size: 32)



Changing Batch Size of Simple NN



Changing Hidden Layer Size



Support Vector Regressor(SVR)

```
param_grid = {
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'C': [0.1, 1, 10],
    'gamma': ['scale', 'auto'],
    'degree': [2, 3]  # Only used for 'poly' kernel
}
```

{'fit_time': 0.4842268466949463,
'score_time': 0.05727086067199707,
'train_score': 0.5873619619062651,
'test_score': 0.6153240077128813,
'best_params': {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}}

Decision Tree Regression

```
param_grid = {
    'max_depth': [3, 5, 7, 10,12],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4, 6],
    'max_features': ['auto', 'sqrt', 'log2']
}
```

{'fit_time': 0.002956104278564453,
'score_time': 0.0005843639373779297,
'train_score': 0.42444051869029326,
'test_score': 0.48282082762321965,
'best_params': {'max_depth': 12, 'max_features': 'sqrt',
'min_samples_leaf': 2, 'min_samples_split': 10}}

Performance analysis and comparison

Model	Mean fit time (sec)	Mean score time (sec)	Mean_train_score (rmse)	Mean_test_score (rmse)	rsme_test
Linear Regression	0.005	0.002	0.744	0.747	
Polynomial Regression	0.309	0.017	0.460	0.511	
Decision tree	0.0029	0.00058	0.4244	0.4828	0.48072
SVR	0.4842	0.05727	0.5873	0.6153	0.5354
Deep Neural Nets(GPU)	278.386	0.036	0.308	0.351	

Limitation, future work, and conclusion

There were issues with generating large amounts of data as the code for generating data was not parallelized and each data point took about 5 minutes

Also, the code was not designed to generate a large amount of data points so modification took time which contributed to the data imbalance

A significant imbalance in the target variable can create a misleading impression of a neural network's performance. (This is the reason for good performance of Decision Tree)

Without a sufficiently representative dataset, a complex problem may appear artificially simple when approached with deep learning.

The observed trend of improved model performance with increasing dataset size suggests that complex models may be better suited for handling larger, more comprehensive datasets.

As the problem is based on Physics, it would be interesting to explore the dataset with Physics Informed Neural Networks (PINNs)

References

Berkhoff, A., Ingram, E., Rezaei, F., Smith, J., Bayless, D., Schonberg, W., & Han, D. (2022). Kinetic Modeling of Electrostatic Transport of Lunar Regolith Particles, Under Review

Kawamoto, H. and Adachi, M. (2014). Electrostatic particle-size classification of lunar regolith
for in-situ resource utilization. Retrieved October 16, 2023, from https://www.researchgate.net/publication/289219757_Electrostatic_particle-size_classification_of_lunar_regolith_for_in-situ_resource_utilization

References

Park, J., Liu Y., Kihm K. D., and Taylor, L. A. (2008). Characterization of lunar dust for toxicological studies i: Particle size distribution.” *Journal of Aerospace Engineering*, 21(4), 266-271.