Documentation

Step 1: Understand the dataset

The dataset is a subset of GDB-13 (a database of nearly 1 billion stable and synthetically accessible organic molecules) composed of all molecules of up to 23 atoms (including 7 heavy atoms C, N, O, and S), totaling 7165 molecules. The Coulomb matrix representation of these molecules and their atomization energies computed similarly to the FHI-AIMS implementation of the Perdew-Burke-Ernzerhof hybrid functional (PBE0) is provided. This dataset features various molecular structures such as double and triple bonds, cycles, carboxy, cyanide, amide, alcohol, and epoxy. The Coulomb matrix is defined as:

$$C_{ii} = \frac{1}{2} Z_i^{2.4}$$

$$C_{ij} = \frac{Z_i Z_j}{|R_i - R_j|}$$

Figure 1: Coulomb matrix

1.Input (X)

- Coulomb matrices representing the molecular structures, shape 7165 x 23 x 23
- Each Coulomb matrix is a 23 x 23 array representing a molecular structure
- There are a total of 7165 matrices in the data set

2.Output (T)

- Shape: 7165
- Each value in T represents the atomization energy of a corresponding molecule

3.Cross-validation splits (P)

- Shape: 5 x 1433
- Each row in the array represents a specific split
- 5 rows: 5 different cross-validation splits, each row contains 1433 indices

- The P array is to partition the dataset into training and validation sets according to predefined splits

4.Atomic charges (Z)

- Shape: 7165 x 23
- Each row corresponds to one molecule
- Each entry in a row represents the atomic charge of an atom in the molecule

5.Cartesian Coordinates (R)

- Shape: 7165 x 23 x 3
- Each row corresponds to one molecule
- Each subarray within a row contains the 3D coordinates for each atom

Step 2: Data preprocessing

Load data

```
# Load data
if not os.path.exists('qm7.mat'):
    os.system('wget http://www.quantum-machine.org/data/qm7.mat')
dataset = scipy.io.loadmat('qm7.mat')
```

Extract training data

The P two-dimensional array contains the cross-validation split data for the training and testing set. Four rows in the array will be used for training and the remaining row will be used for testing which make 75% of the data for training and 25% of the data for testing.

```
# Test split for cross-validation (between 0 and 5)
split = int(sys.argv[1])

# Extract training data
# Train indices 75%, test 25%
train_indices = list(range(0, split)) + list(range(split + 1, 5))

# Convert 2D array to 1D array
P = dataset['P'][train_indices].flatten()

# Input: select only those rows (molecules) that correspond to the training data
X = dataset['X'][P]

# T contains the corresponding output targets (atomization energies) for the training data
T = dataset['T'][0, P]
```

```python
# Flatten X
X_flat = X.reshape(X.shape[0], -1) # convert 3D array to 2D array
```

Standardize the data

Using the StandardScaler() class from the scikit-learn library. StandardScaler is a class in scikit-learn that standardizes features by removing the mean and scaling to unit variance. Standardization is important because many machine learning algorithms perform better or converge faster when the input features are scaled. The standard score of a sample x is calculated as: `z = (x - u) / s` where u is the mean of the training samples or zero if with_mean=False, and s is the standard deviation of the training samples or one if with_std=False.

```python
# import statement
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_flat = scaler.fit_transform(X_flat)
```

Step 3: Implement Machine Learning Model

Define and train the model

This step aims to create a set of diverse models to be trained and evaluated on the dataset. Different models are used to compare their performance and select the best one. This approach used merely the traditional Machine Learning method including Linear Regression, Support Vector Regression, Gaussian Process, and Multilayer Perceptron. These four models are provided by the scikit-learn library.

```python
# import statement
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.neural_network import MLPRegressor
```

- Linear regression models the relationship between the dependent variable (target) and one or more independent variables (features) by fitting a linear equation to the observed data.
- SVR is a type of Support Vector Machine (SVM) used for regression tasks. It tries to fit the best line within a threshold value (epsilon) that maximizes the margin between data points.
- Gaussian Process Regression is a non-parametric, Bayesian approach to regression that provides a distribution over functions. It is particularly useful for problems where the data is noisy or the underlying function is complex.
- MLP is a type of feedforward artificial neural network. It consists of multiple layers of neurons, including an input layer, one or more hidden

layers, and an output layer. Each neuron in a layer is connected to every neuron in the subsequent layer.

```
# Define and train models
models = {
    "Linear Regression": LinearRegression(),
    "Support Vector Regression": SVR(),
    "Gaussian Process": GaussianProcessRegressor(),
    "Multilayer Perceptron": MLPRegressor(hidden_layer_sizes=(400, 100), max_iter=1000)
}
```

Step 4: Model training

The loop iterates over the dictionary of models, trains each model, makes predictions, and evaluates the performance using Mean Absolute Error (MAE).

```
for name, model in models.items():
    # Train the model
    model.fit(X_flat, T) # X_flat: input, T: output
    # Predict
    predictions = model.predict(X_flat)
    # Calculate Mean Absolute Error
    mae = mean_absolute_error(T, predictions)
    results[name] = mae
    print(f'{name} Mean Absolute Error: {mae}')

# Save models
for name, model in models.items():
    with open(f'{name.replace(" ", "_")}_model.pkl', 'wb') as f:
        pickle.dump(model, f)
```

Step 5: Model evaluation

MSE & MSE

In the train.py file the four models are evaluated using mean absolute error and mean squared error. Mean absolute error (MAE) is a simple, popular and powerful metric to evaluate the accuracy of regression models. It measures the average absolute difference between the predicted values and the actual target values. The smaller MAE value, the better the model's prediction. Mean square error (MSE) is the average of the squared differences between predicted and actual values.

Model Comparison Based on Mean Absolute Error

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

Figure 2: MAE

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

MSE	= mean squared error
n	= number of data points
Y_i	= observed values
$\hat{Y}_i$	= predicted values

Figure 3: MSE

Model	Training MAE (kcal/mol)	Test MAE (kcal/mol)
Linear Regression	19.46	144.07
Support Vector Regression	72.99	73.34
Gaussian Process	0.00	1537.59
Multilayer Perceptron	8.90	18.71

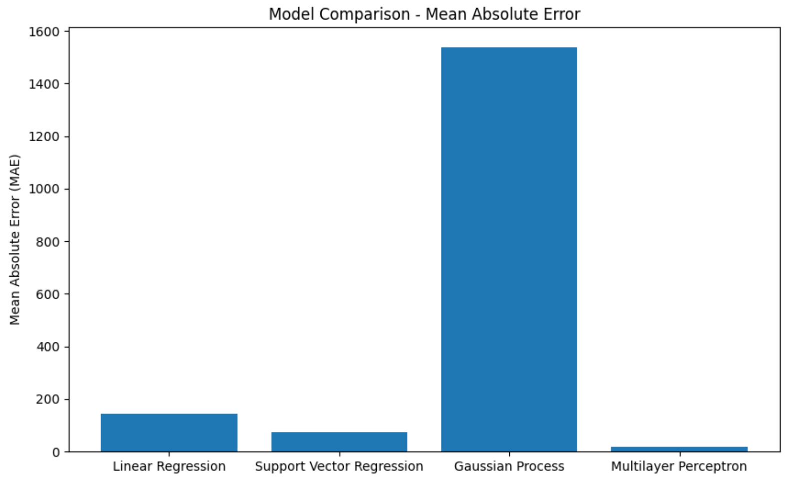

Figure 4: Model comparison mae

Model Comparison Based on Root Mean Squared Error

Model	Training RMSE (kcal/mol)	Test RMSE (kcal/mol)
Linear Regression	25.47	3505.74
Support Vector Regression	122.92	124.01
Gaussian Process	0.00	1554.03
Multilayer Perceptron	14.48	26.38

Step 6: Visualization and analysis

For each model, the predicted atomization energies are plotted against the actual atomization energies:

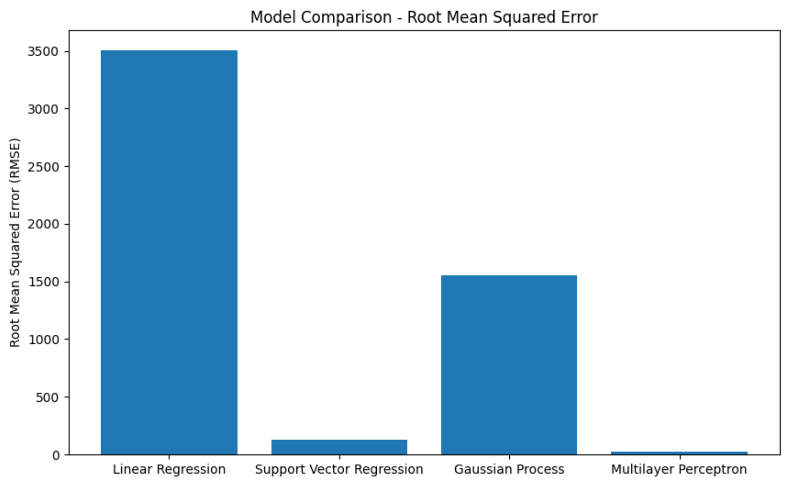

Figure 5: Model comparison rmse

- Scatter Plot: Each point represents a molecule. The x-axis shows the actual atomization energies and the y-axis shows the predicted energies.
- Red Line: The red dashed line represents the ideal case where predicted values perfectly match actual values.

Linear regression

- Training MAE: 19.46 kcal/mol
- Test MAE: 144.07 kcal/mol

The Linear Regression model shows a significant increase in error from training to test data. This indicates overfitting, where the model performs well on the training data but poorly on unseen test data. The model may not generalize well due to a lack of complexity to capture the underlying patterns in the data.

Gaussian Process

- Training MAE: 0.00 kcal/mol
- Test MAE: 1537.59 kcal/mol

The Gaussian Process model has a training MAE of 0.00, which indicates severe overfitting. A training error of 0.00 means that the model has memorized the training data perfectly. The extremely high test MAE shows that the model performs very poorly on unseen data which means the model is overfitting.

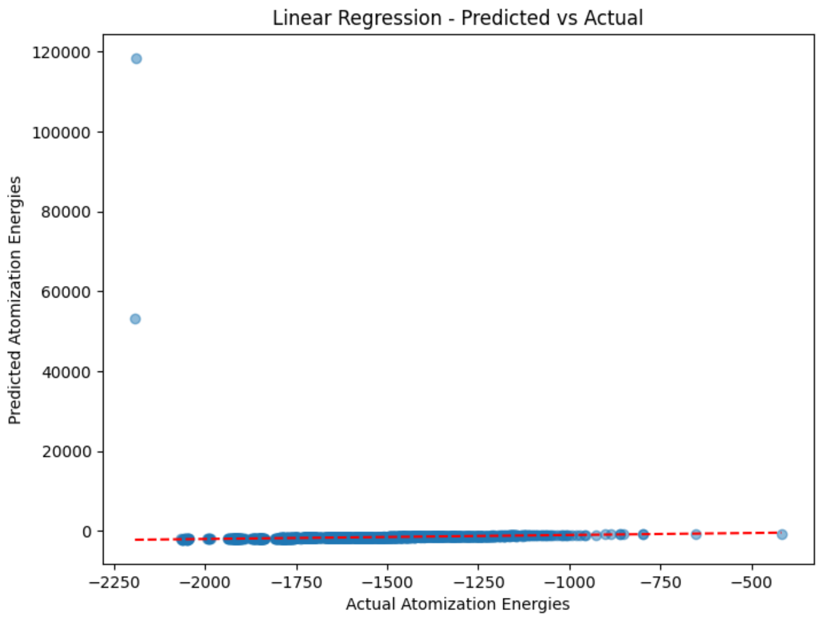

Figure 6: Linear Regression predicted vs actual

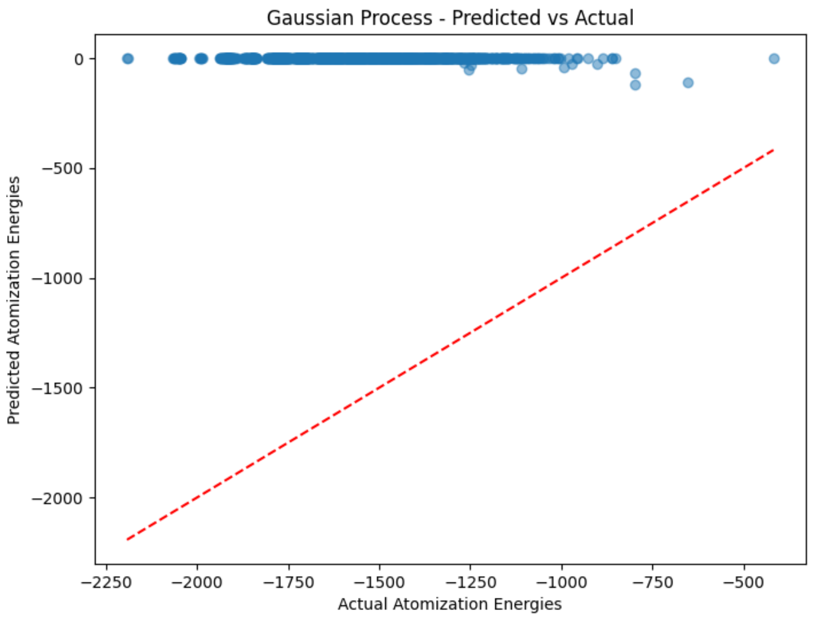

Figure 7: Gaussian Process predicted vs actual

Multilayer perceptron

- Training MAE: 8.90 kcal/mol
- Test MAE: 18.71 kcal/mol

The Multilayer Perceptron (MLP) model has a relatively low training and test MAE, and the difference between the two is modest. This indicates good generalization and that the model is likely capturing the underlying patterns in the data effectively. The MLP has the best-performing model among the four models in this application.

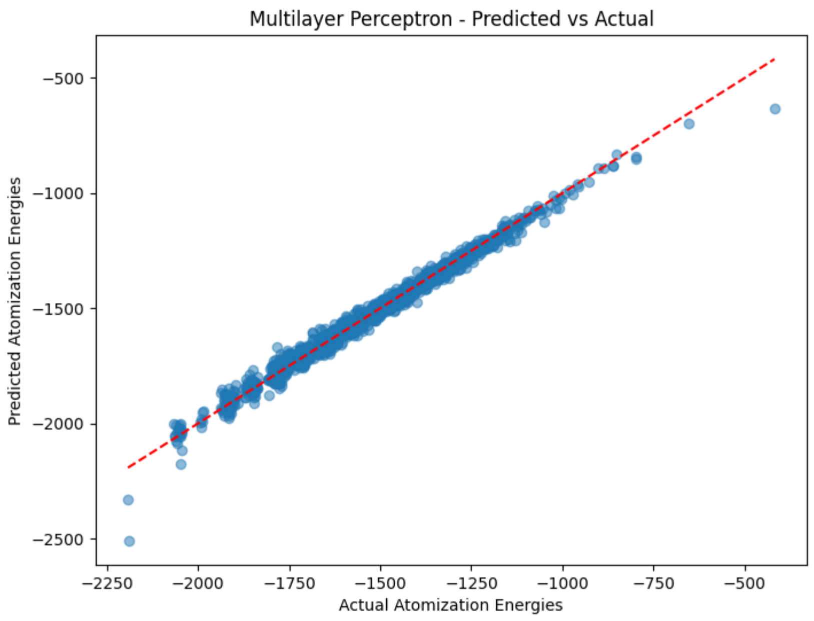

Figure 8: Multilayer Perceptron predicted vs actual

Support Vector Regression

- Training MAE: 72.99 kcal/mol
- Test MAE: 73.34 kcal/mol

The SVR model has very similar errors in both training and test data. This suggests that the model is not overfitting and generalizes well. The relatively high MAE indicates that the model might not be capturing all the underlying patterns, but it is consistent.

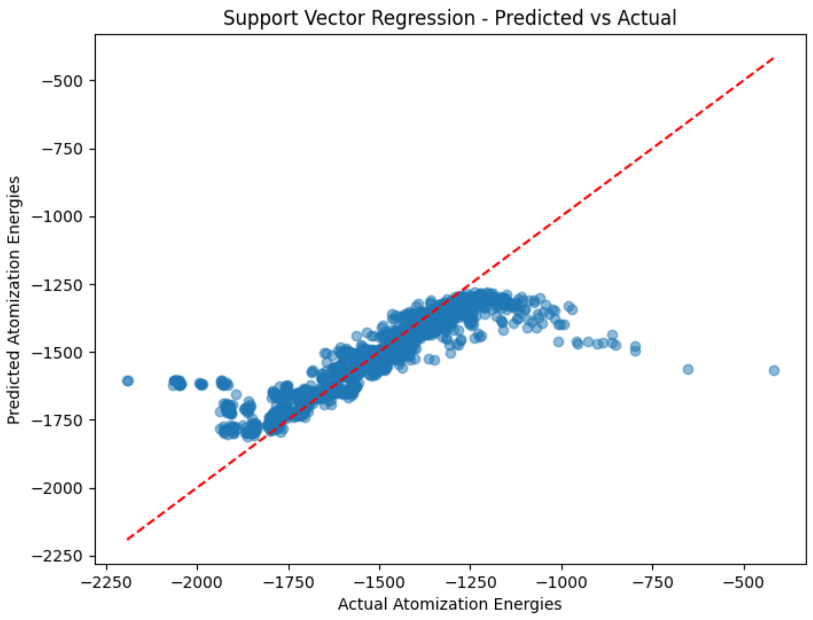

Figure 9: Support Vector Regression predicted vs actual

Advantages and Disadvantages of the author's work

The author's work uses neural network to train the model. These are some advantages and disadvantages of using neural networks for this dataset:

Advantages

Neural networks can capture complex, non-linear relationships between input features and target variables. This makes them suitable for tasks where simple linear models may fail to provide accurate predictions. It can automatically learn and extract relevant features from raw input data. This reduces the need for manual feature engineering, allowing the model to identify important patterns and interactions. Neural networks scale well with large datasets. With sufficient computational resources (such as GPUs), they can handle large volumes of data and complex models, making them suitable for big data applications. It is highly flexible and can be tailored to a wide range of tasks by adjusting its architecture such as the number of layers, number of neurons per layer, or activation functions. This allows for customization based on the specific needs of the problem. Finally, neural networks have been shown to achieve state-of-the-art performance on various tasks, particularly in fields such as image recognition, natural language processing, and speech recognition.

Disadvantages

Training neural networks (NN), especially deep networks, can be computationally intensive and time-consuming. They require significant computational resources, such as powerful GPUs, and can take a long time to train. Specifically, the author's work using a neural network can take up to two days for training. Neural networks have many hyperparameters such as learning rate, batch size, number of layers, and number of neurons per layer that need to be carefully tuned to achieve optimal performance. This tuning process can be complex and time-consuming. Neural networks are prone to overfitting, especially if the model is too complex for the amount of training data available. Regularization techniques and careful model validation are necessary to mitigate this risk. NN are often considered "black boxes" because their decision-making process is not easily interpretable. Understanding why a neural network makes a certain prediction can be challenging, which may be a drawback in applications where interpretability is crucial.

References

1.QM7 Dataset

2.StandardScaler

3.Understanding Mean Absolute Error (MAE) in Regression: A Practical Guide

Github repository

Github repository of this code and documentary: Github repository