CH5650 - Molecular Data Science and Informatics End Term Examination

Report - M Bharathi Mozhian - CH18B044

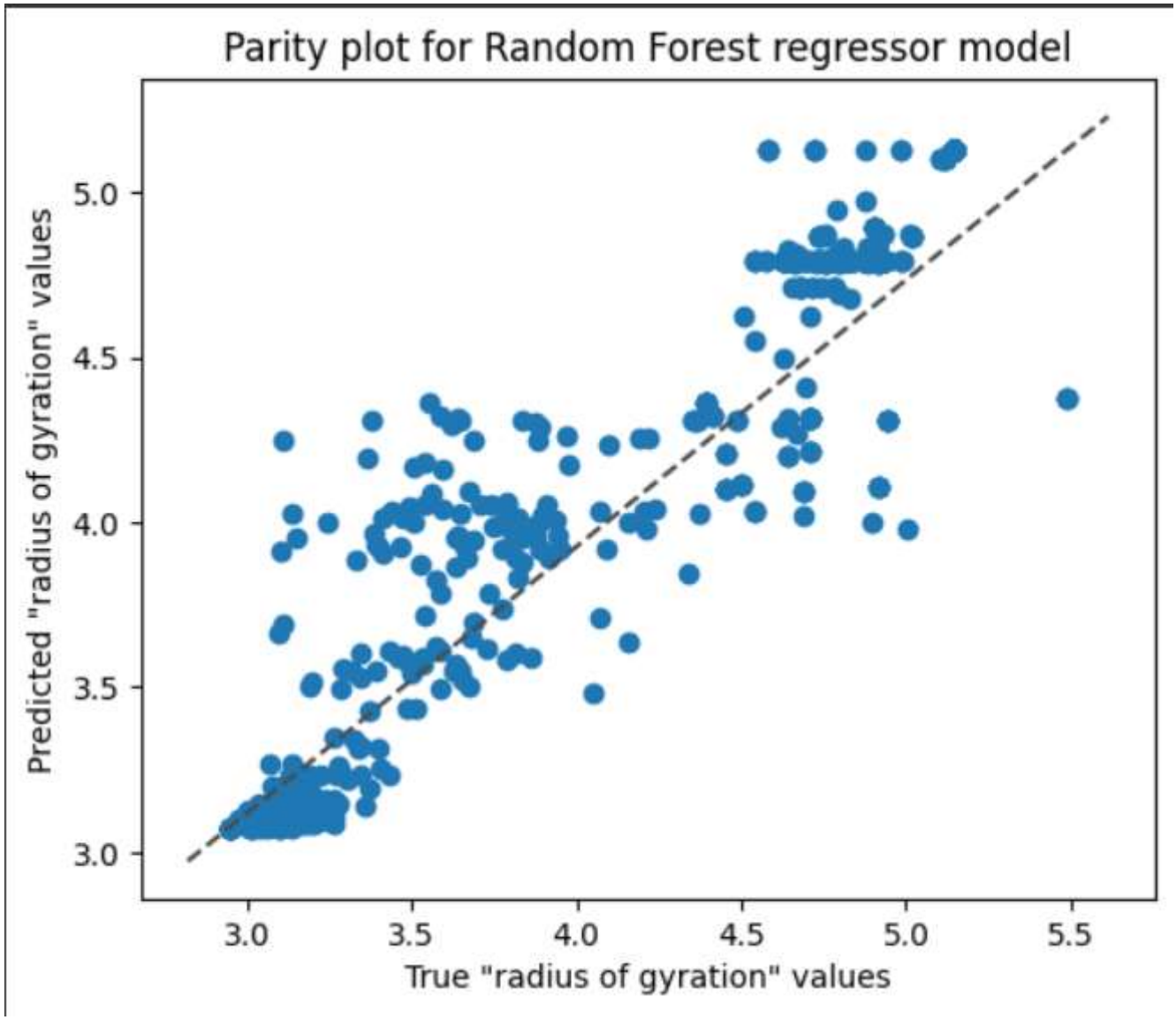
Question 1:

The sequences are read and stored as a numpy array. To extract features for a machine learning model from sequences of several molecules made of 2 moieties, we can use the concept of n-grams. N-grams are contiguous sequences of n items from a given sample of text or speech. In our case, the n-grams will be contiguous sequences of n characters from the sequences of the molecules. By extracting n-grams from the sequences, we can create numerical features that can be used as input to the machine learning model.
We use a sklearn 's random forest regressor andxgboost's XGBoost regressor model to train radius of gyration prediction on the train data with 'mean squared error' as the loss function.

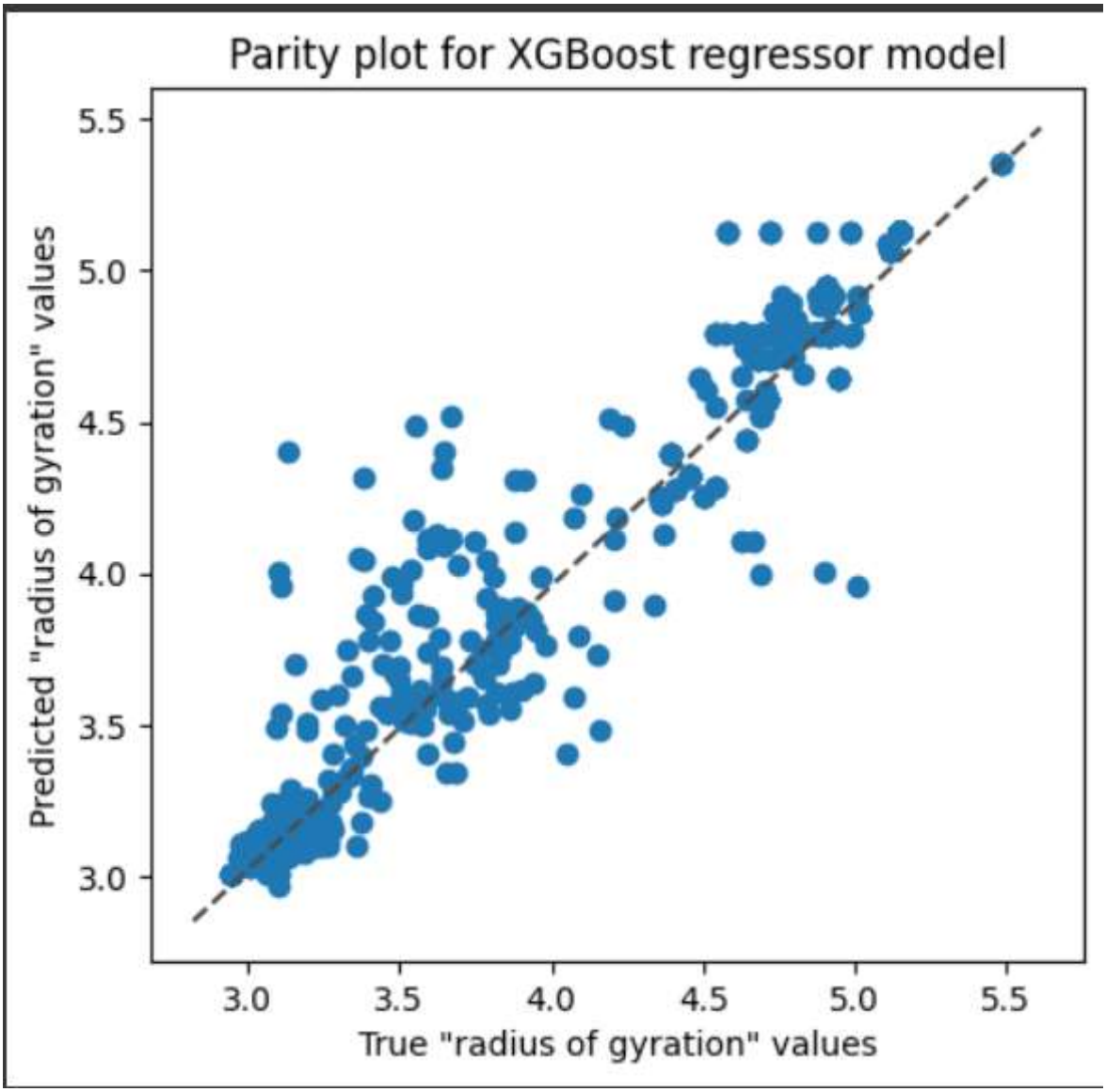
- Test mean squared error - random forest regressor: 0.03259
- Test MSE for predicting "radius of gyration" by XGBoost model: 0.03259

Parity plots on test data:

1. For random forest regressor



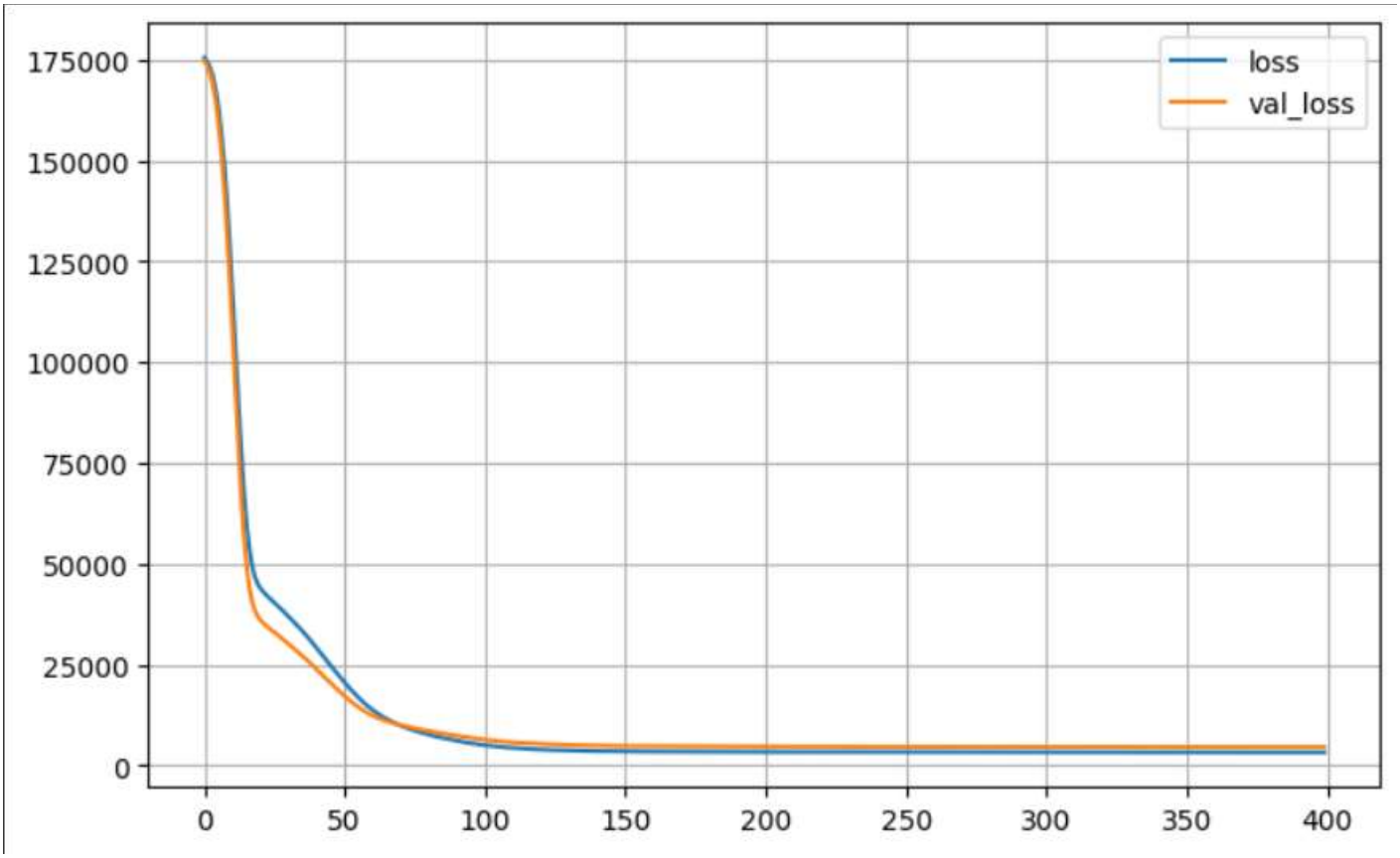
2. For xgboost regressor



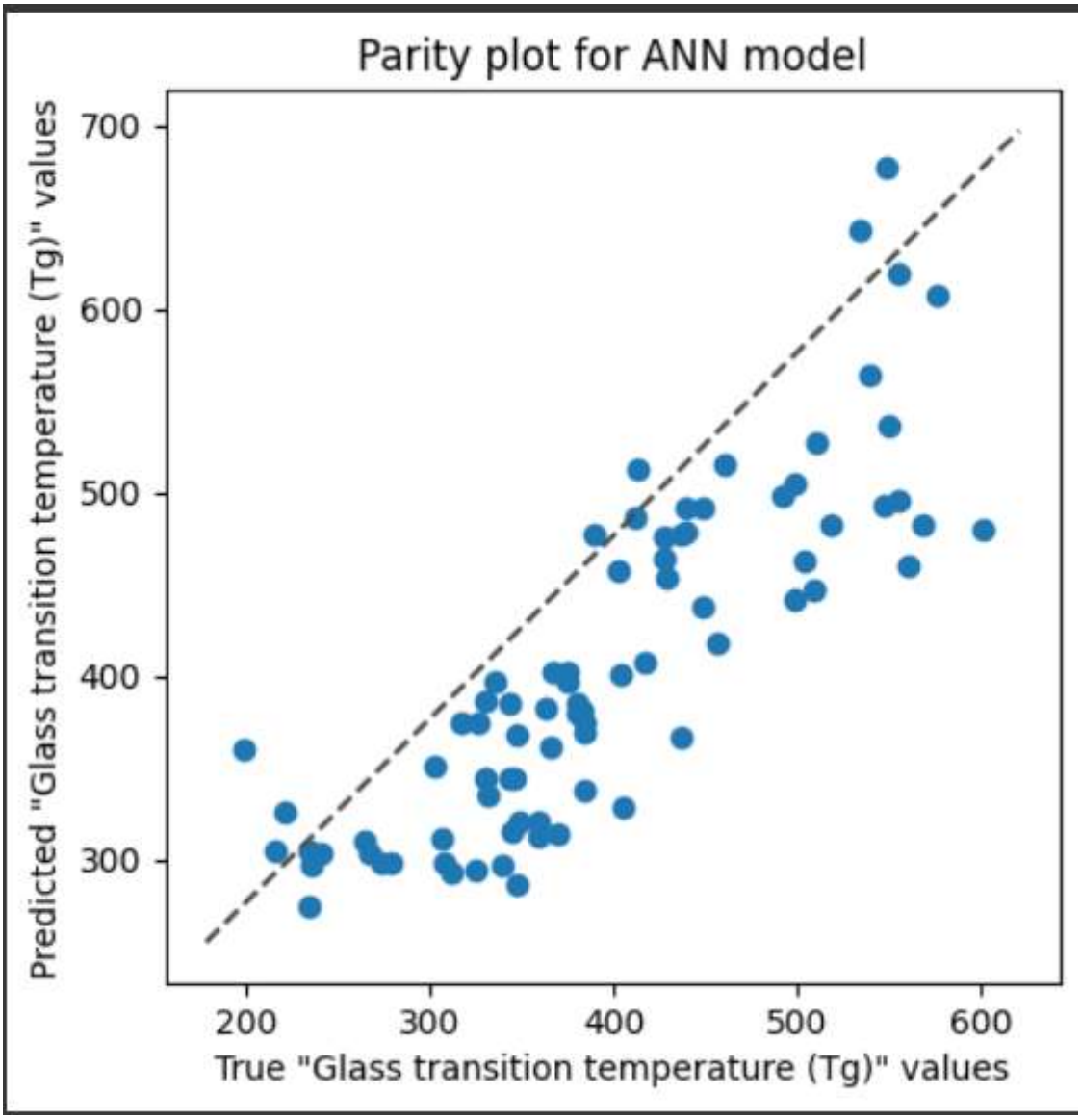
Question 2:

First, we generate 208 physicochemical descriptors from SMILES of each molecule. We notice that some SMILES are invalid and are unable to generate the descriptors (gives None values). Those instances of SMILES and its descriptors are removed from the dataframe. Later, we perform Pearson's correlation to find the amount of correlation between each of the descriptor and the output variable - 'Glasss transition temperature'. We use a cutoff of 0.65 (pearson' s correlation) to shortlist the descriptors which contribute significantly to the Tg values. We then obtain 11 descriptors and take it for training the model. Since, we had discussed in class, that when using physicochemical descriptors, ANNs generally work well, we define a keras feed forward model with 40 neurons in first layer, 10 neurons in the second layer and 1 neuron in the output layer. The dimension of the input size is 11. The loss function used is mean squared error, and the optimzer used is Adam with learning rate = 0.01. The ANN model resulted in: Mean squared error in test data = 2991.498046875 and gave Actual values: [436.5 267. 347.5 331.5 273.5]; Predicted values: [476.46927 303.88007 286.014 335.16125 297.6724] Standard deviation between actual Tg and predicted Tg in test data = 133.73276615147788 Mean difference between predicted and actual Tg values in test data = 43.87295417785644

The training and validation loss vs number of iterations:



Parity plot on test data:



Question 3:

To extract structural fingerprints from SMILES (Simplified Molecular Input Line Entry System) of a molecule, you can use the RDKit library in Python. RDKit is an open-source software development kit for cheminformatics and provides a wide range of functions for working with molecular data.

In the below code, we first define a SMILES string for the molecule of interest. We then use the Chem.MolFromSmiles() function to convert the SMILES string to a molecule object. Next, we generate a Morgan fingerprint using the AllChem.GetMorganFingerprintAsBitVect() function, which takes the molecule object and two parameters: radius, which defines the radius of the fingerprint and nBits, which defines the length of the fingerprint.

Finally, we convert the fingerprint to a numpy array using the ToBitString() function and print the resulting array.

First, the dataset is loaded containing the SMILES and the dependent variables namely, 'Hg', 'Enthalpy', and 'Cp'.

SMILES of the molecules are converted into morgan fingerprints. The resulting dataframe containing the morgan fingerprints as the features and the dependent variables. The data is split into training and testing sets using train_test_split() function from scikit-learn.

Next, we define a convolutional neural network (CNN) model using Keras. The model has two convolutional layers each followed by max pooling layers and dropout layers, then two dense layers with a dropout layer in between. The last layer is a single output layer as we want to predict a single numerical value (each output variable is predicted individually).

We compile the model using mean squared error (MSE) loss and the Adam optimizer with a learning rate of 0.01. We then train the model using the fit() function with a batch size of 32 and 50 epochs, and use 10% of the training data for validation. Finally, we evaluate the model on the test set using the evaluate() function and print the test loss.

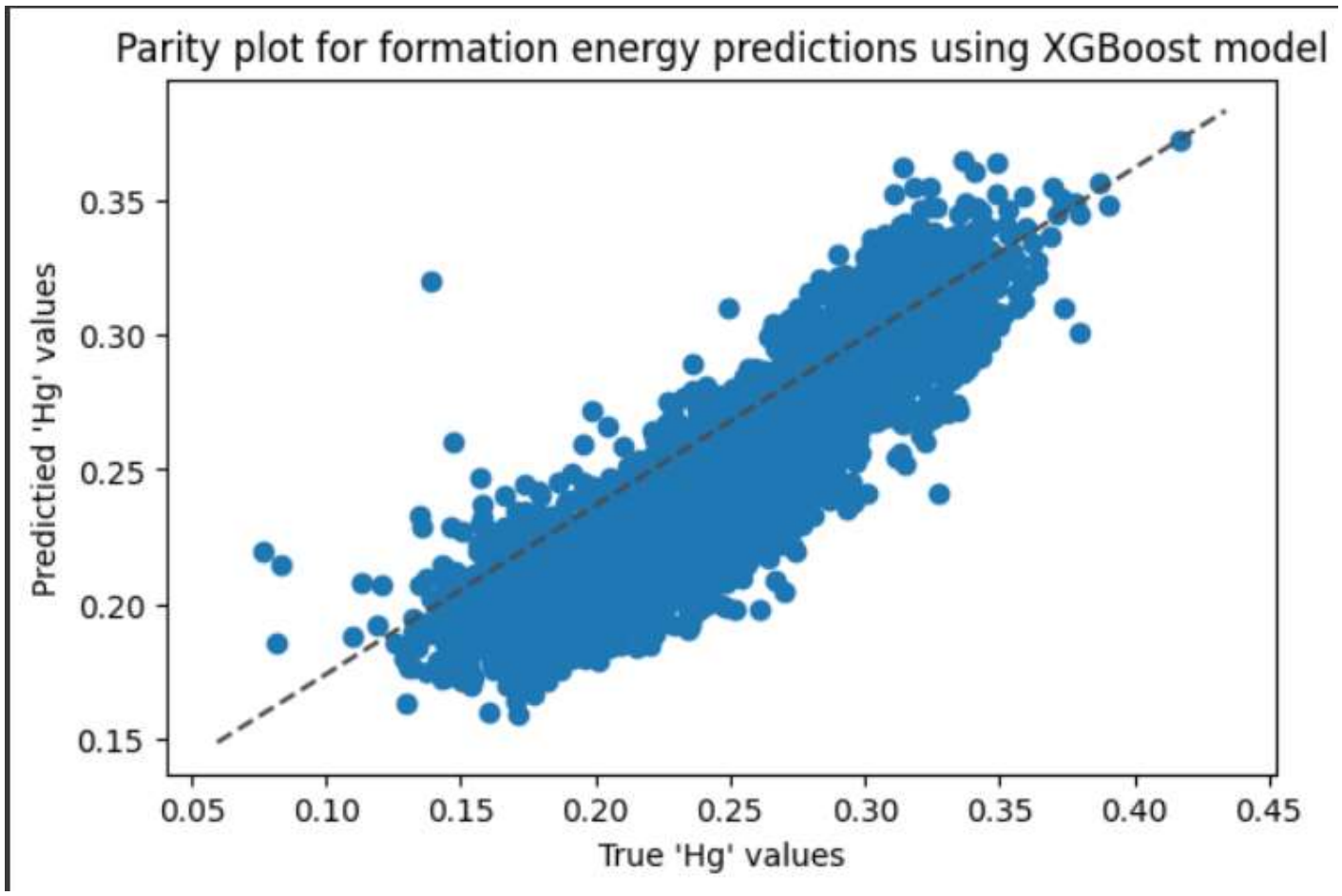
The CNN model works well for predicting Formation energy.
Test error for formation energy = 6.11899 However, the test error for - Enthalpy = 9594.9052 - Specific heat = 942.971 However we find that the CNN model didn't work well for predicting enthalpy and specific heat. So, we try XGBoost algorithm to model the prediction of them.

Results upon using XGBoost for 'Enthalpy' and 'Cp': Test error for enthalpy = 0.65629 Standard deviation between predicted and actual enthalpy values = 24.8543 Mean difference between predicted & actual enthalpy values = 17.411 Actual values: [-383.764391 -458.952075 -456.560753 -420.53807 -394.815076]; Predicted values: [-373.50858 -433.95438 -429.9226 -411.8949 -397.7279]

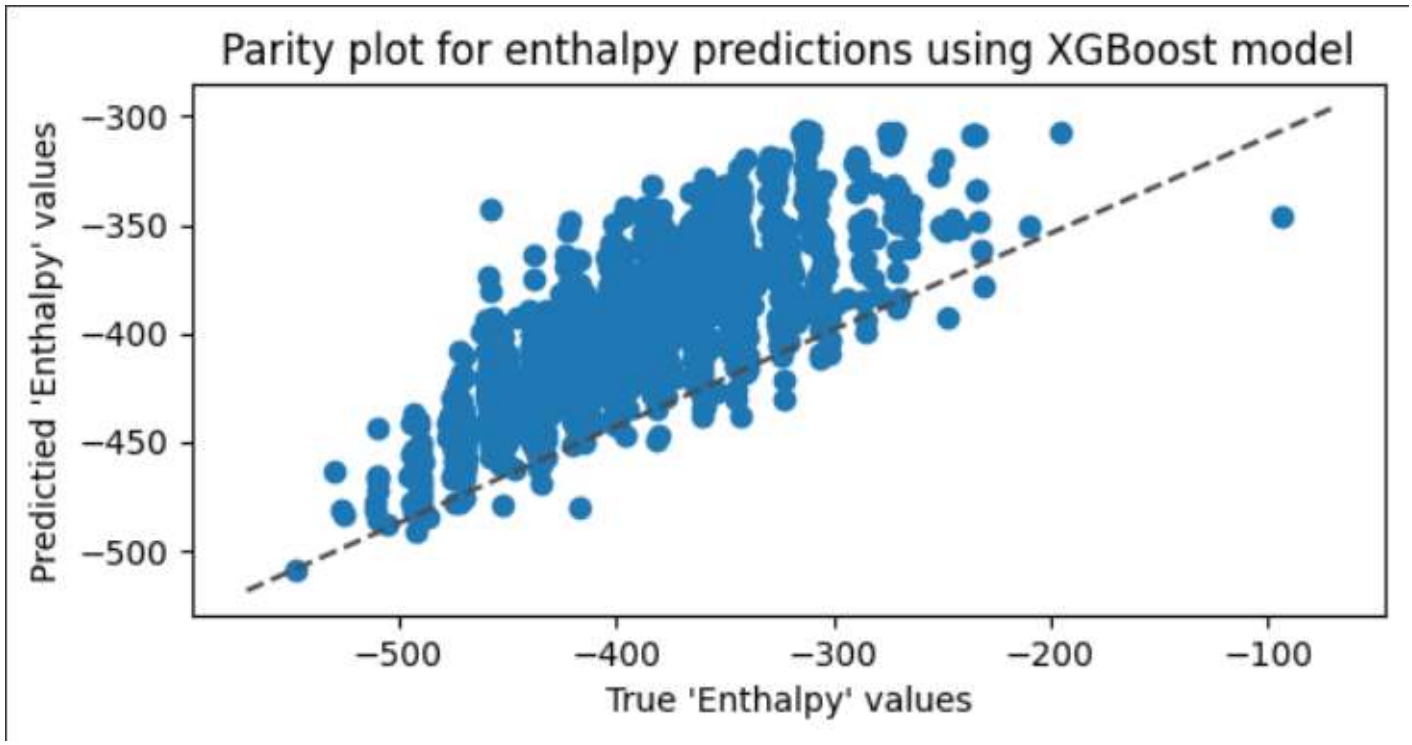
Test error for specific heat = 0.59170 Standard deviation between predicted and actual Cp values = 2.258 Mean difference between predicted and actual CP values = 1.676 Actual values: [29.11 29.568 26.777 29.111 25.203]; Predicted values: [28.860216 29.114943 27.103693 27.297962 25.886719]

The results convey that the XGBoost model for predicting enthalpy and specific heat give decentlfairly accurate predictions compared to the CNN model. Hence, we finalize the CNN model for predicting formation energy values & the XGBoost model for predicting enthalpy and specific heat.

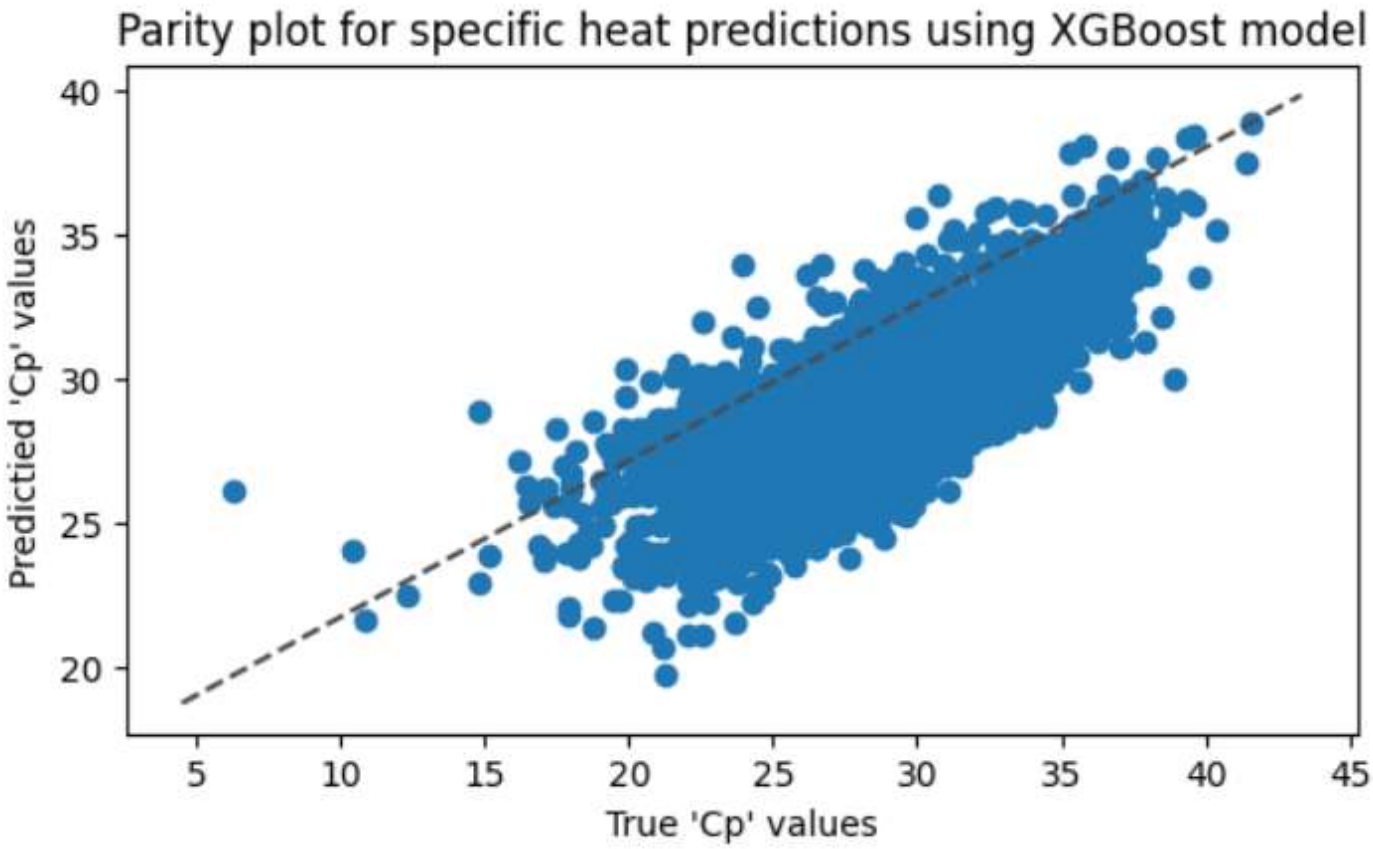
Parity plot on test data - Formation energy predictions



Parity plot on test data - Enthalpy predictions



Parity plot on test data - Specific heat predictions


Parity plot for specific heat predictions using XGBoost model
Predictied 'Cp' values
True 'Cp' values
40
35
30
25
20
5
10
15
20
25
30
35
40
45