Computational Biophysics

Algorithms to Applications

Term Project

Determining Partial Charges of a Molecule

Group Members

Kanchi Mohan Krishna - 20CS10030
Gangaram Arvind Sudewad - 20CS30017
Lav Jharwal - 20CS30031

Motivation

In many molecular modeling applications, including molecular docking, molecular dynamics simulations, free energy calculations, and others, partial atomic charges are typically utilized to calculate the electrostatic component of energy.
High-level computations in quantum mechanics might be the most precise method to calculate the partial charges of small molecules, but they require an excessive amount of time to process a lot of molecules.

Machine learning (ML) techniques have been widely employed over the past two decades to predict a wide range of molecular features without the necessity to fully comprehend the underlying chemistry or physics of these properties.
Using training on pre-gathered data on high-level QM charges, ML techniques have recently demonstrated their potential to provide accurate prediction for incomplete charges.

Result

We used the data from the ZINC dataset and ChEMBL dataset with epsilon values 4 and 78 respectively from both the datasets.

In the context of training molecules, epsilon (ε) is a measure of the diversity of the molecules in the set. It represents the average pairwise similarity between all pairs of molecules in the set. A higher epsilon value indicates a more diverse set of molecules, while a lower value indicates that the molecules are more similar to each other.

We proposed a molecule descriptor named Atom-Path-Descriptor (APD) and developed a set of APD-based machine learning (ML) models to predict the partial charges for small molecules with high accuracy. In the APD algorithm, the 3D structures of molecules were assigned with atom centers and atom-pair path-based atom layers to characterize the local chemical environments of atoms.

Then, based on the APDs, two representative ML algorithms, i.e. random forest (RF) and Neural Network, were employed to develop the regression models for partial charge assignment.

Definition of APD

The principle of APD is inspired by the usefulness of atom pairs in characterizing atomic environments.

The APD based on the atom-pair structure learns the idea of combining atom-pair path and circular fingerprints.The main idea of circular fingerprints is to describe molecular structure with the circular atom neighborhoods in the same molecule. It centers on the target atom and defines layers by the atom-pair path, and then gives a maximum-length atom-pair path with the length of the path representing the number of interatomic separations. The neighbors in this maximum path length can be divided into multiple path layers with each layer smaller than the maximum path length. It needs to point out that the atom-pair path is the shortest bond–bond path between atoms 1 and 2, and the path length must be shorter than the maximum path length set by users. APD gets the atom information on each path layer until the path length equals the maximum path length.

Generation of APD

The generation of APD includes four basic steps: (i) feature extraction and calculation: getting atom features from the molecule; (ii) iteratively updating: getting features of the center atom in each layer using the method in stage 1 until the maximum path length is reached; (iii) deleting duplicate features: confirming each feature of an atom is unique; (iv) keeping feature order invariant: whatever order of the inputs are, the same atom should give the same output features.

The detailed processes are summarized as follows.

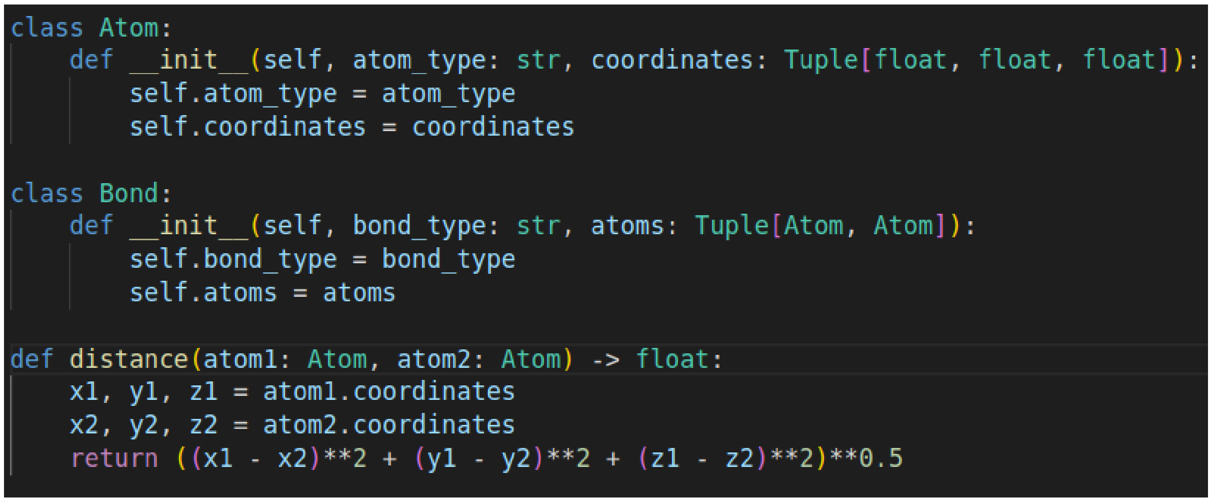
Feature extraction and calculation : The APDs of an atom consists of a set of descriptors for many atom layers based on the path length, in which each layer except layer 1 contains only two types of features: the atom type, such as C, and the distance between this atom (i.e. termed as B) and the center atom A. The distance between the center atom A and atom B, which is connected by the atom-pair path.

Iteratively updating each atom layer : In this stage, the process of stage 1 will be iterated for many times until the *L* equals the maximum length *P*.

Keeping feature order invariant : In the step of feature extraction and calculation, the features are usually extracted from a sdf or mol2 file, where the feature order depends on the atom order.

We have tried creating the APD's without using any libraries initially but to decrease the complexity of the solution and increase the Accuracy of the models we decided to use the RDKit library to create the APD.

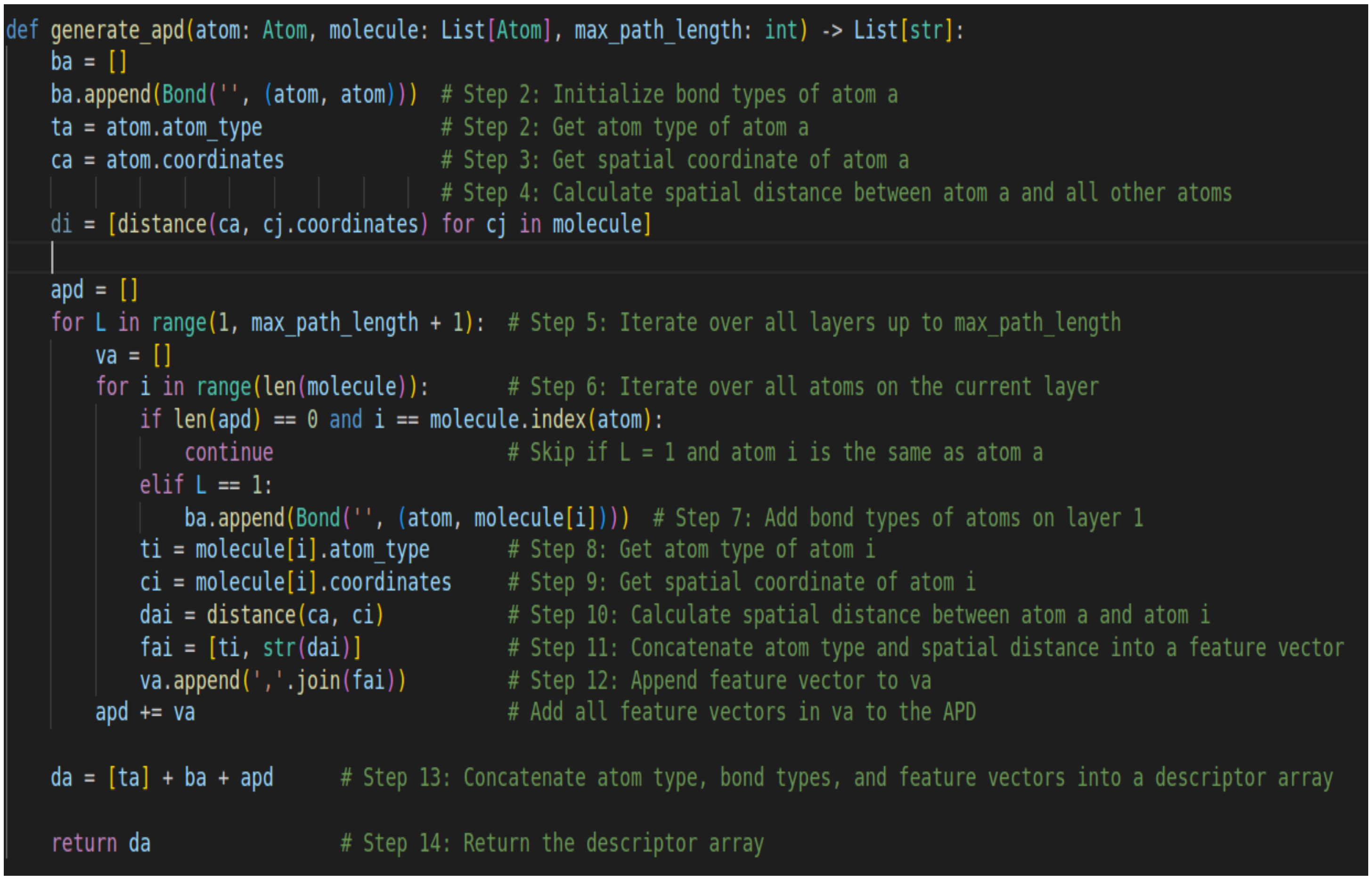
But here is the Algorithm part we tried to implement:



```python
class Atom:
    def __init__(self, atom_type: str, coordinates: Tuple[float, float, float]):
        self.atom_type = atom_type
        self.coordinates = coordinates

class Bond:
    def __init__(self, bond_type: str, atoms: Tuple[Atom, Atom]):
        self.bond_type = bond_type
        self.atoms = atoms

def distance(atom1: Atom, atom2: Atom) -> float:
    x1, y1, z1 = atom1.coordinates
    x2, y2, z2 = atom2.coordinates
    return ((x1 - x2)**2 + (y1 - y2)**2 + (z1 - z2)**2)**0.5
```



```python
def generate_apd(atom: Atom, molecule: List[Atom], max_path_length: int) -> List[str]:
    ba = []
    ba.append(Bond('', (atom, atom)))  # Step 2: Initialize bond types of atom a
    ta = atom.atom_type                # Step 2: Get atom type of atom a
    ca = atom.coordinates              # Step 3: Get spatial coordinate of atom a
                                       # Step 4: Calculate spatial distance between atom a and all other atoms
    di = [distance(ca, cj.coordinates) for cj in molecule]

    apd = []
    for L in range(1, max_path_length + 1):  # Step 5: Iterate over all layers up to max_path_length
        va = []
        for i in range(len(molecule)):       # Step 6: Iterate over all atoms on the current layer
            if len(apd) == 0 and i == molecule.index(atom):
                continue                     # Skip if L = 1 and atom i is the same as atom a
            elif L == 1:
                ba.append(Bond('', (atom, molecule[i])))  # Step 7: Add bond types of atoms on layer 1
            ti = molecule[i].atom_type       # Step 8: Get atom type of atom i
            ci = molecule[i].coordinates     # Step 9: Get spatial coordinate of atom i
            dai = distance(ca, ci)           # Step 10: Calculate spatial distance between atom a and atom i
            fai = [ti, str(dai)]             # Step 11: Concatenate atom type and spatial distance into a feature vector
            va.append(','.join(fai))         # Step 12: Append feature vector to va
        apd += va                            # Add all feature vectors in va to the APD

    da = [ta] + ba + apd      # Step 13: Concatenate atom type, bond types, and feature vectors into a descriptor array

    return da                 # Step 14: Return the descriptor array
```

Database :

For training and testing the models , We have used the data of the molecules present in the research collection of ETH Zurich. Where the dataset is divided into 4 modules based on the diversity of molecules present in the dataset and the source of the dataset (ZINC database and ChEMBL database).

The dataset contained more than 3 Lakh molecules, to reduce the running time of the program and also increase the randomness of the molecules present we have taken approximately 2500 molecules from the 4 modules of the database which were chosen randomly. The datasets contained the files in the **sdf format** and the file contains the charges of the atoms along with the coordinates of atoms in the 3D format as per the requirement of the question.

The file contains the coordinates of H-atoms too and the solvent atoms aren't present in the file too, so we didn't have to add H - atoms specifically or remove the solvent molecules unlike in the case of PDB files.

Then we divided the datasets in the ratio 80:20 for training and testing respectively.

Refer : https://www.research-collection.ethz.ch/handle/20.500.11850/230799

APD- Random Forest Model :

Before building the ML models, the APDs were first calculated for each molecule, and then an independent RF regression model was constructed for each atom type using the scikit-learn with the purpose of minimizing the average RMSE of the predicted partial charges and increasing R2-score of the models that are built.

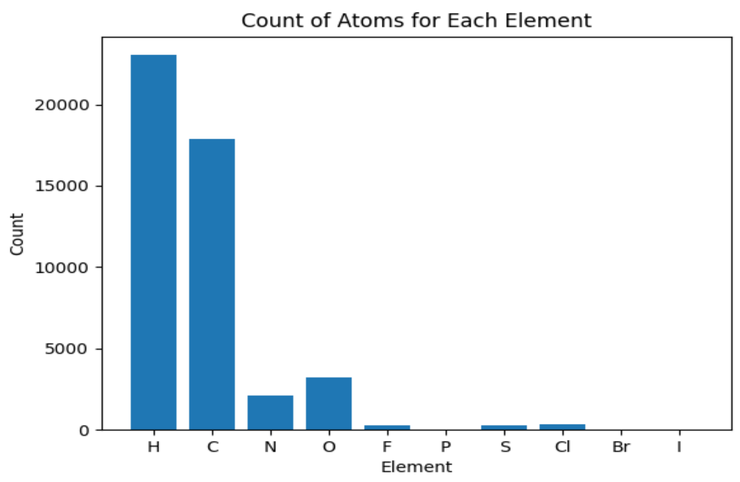
The ***coefficient of determination*, or R$_2$**, is a measure that provides information about the goodness of fit of a model. In the context of regression it is a statistical measure of how well the regression line approximates the actual data. It is therefore important when a statistical model is used either to predict future outcomes or in the testing of hypotheses.

R$_2$ = $1-(\,sum\ squared\ regression\ (SSR)\ /\ total\ sum\ of\ squares\ (SST)\,)$

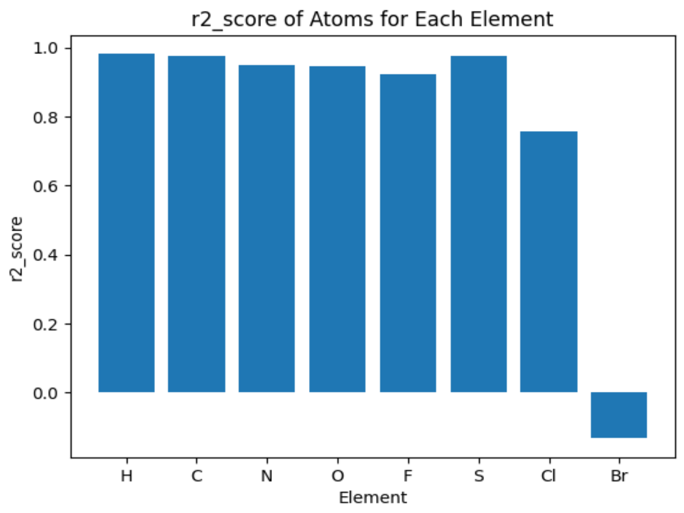
To prove the effectiveness of APD, only the molecular descriptors used were replaced by the APDs for RF model construction, and the hyperparameters were varied to attain maximum accuracy.

The variations according to the hyperparameters are shown below:

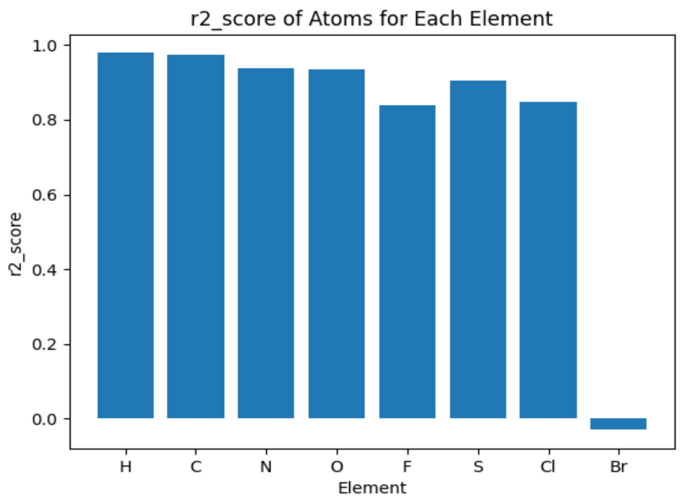
Case I : n_estimators = 100, max_depth = 6 ,min_samples_leaf = 6
Case II : n_estimators = 200 , max_depth = 8, min_samples_leaf = 8
Case III : n_estimators = 50 , max_depth = 4 , min_samples_leaf = 4



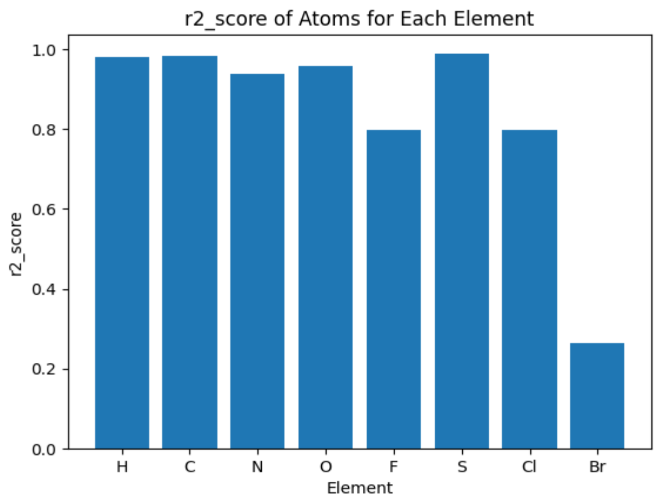
Distribution of Atoms



Case I



Case II



Case III

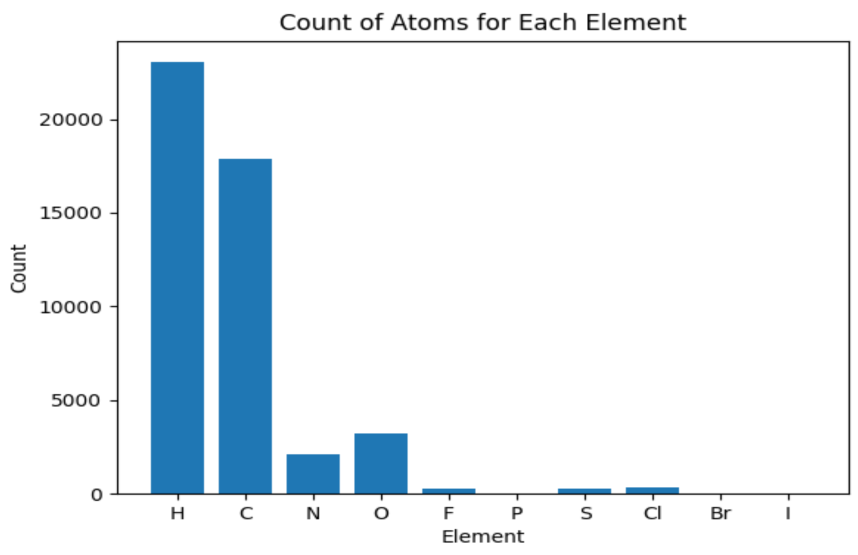
APD- Neural Network Model :

The input features can be the APDs calculated for each molecule, and the output can be the predicted partial charges for each atom type.

To construct a neural network model, the following steps can be taken:

1. Data preparation: The data should be split into training, validation, and testing sets. The APDs should be normalized or standardized to ensure that they are on a similar scale, and missing values should be imputed or removed.
2. Model architecture: The neural network architecture should be defined, including the number of layers, the number of neurons per layer, and the activation functions used. The architecture can be determined using techniques such as grid search or random search.
3. Hyperparameter tuning: Hyperparameters such as learning rate, batch size, and number of epochs should be tuned to optimize the model's performance on the validation set. Techniques such as cross-validation and early stopping can be used to prevent overfitting.
4. Model training: The model can be trained on the training set using an optimization algorithm such as stochastic gradient descent or Adam. The loss function used should be a suitable regression loss such as mean squared error.
5. Model evaluation: The trained model can be evaluated on the testing set using metrics such as mean squared error, root mean squared error, and R-squared. The performance of the neural network model can be compared to that of the RF model to determine which method performs better.

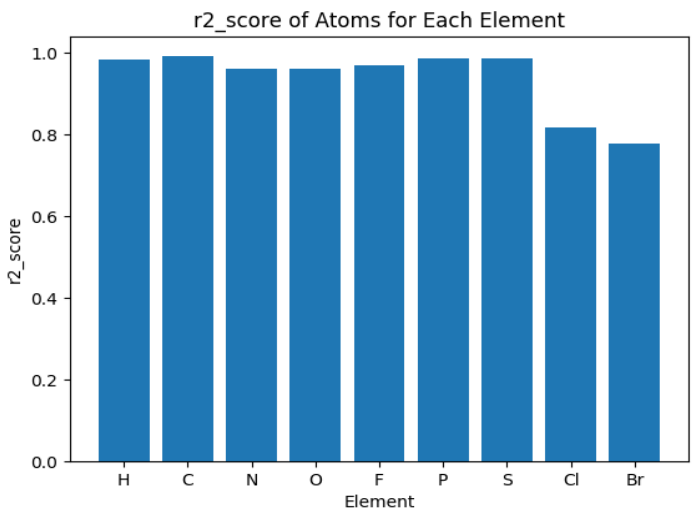
For the database the Count of Atoms for each element for the selected database is displayed below :



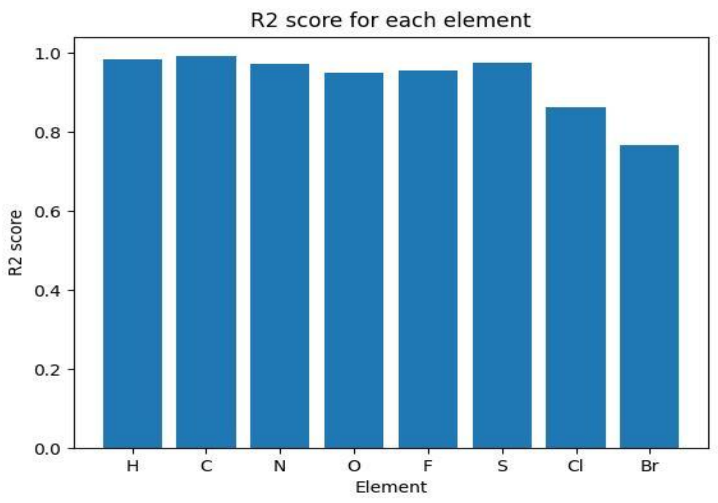
The Neural Network contains 3 layers in which 1 layer is a hidden layer.
The output of the 3rd layer is the predicted partial charge of the atom of the particular model.

The variations according to the hyperparameters are shown below:

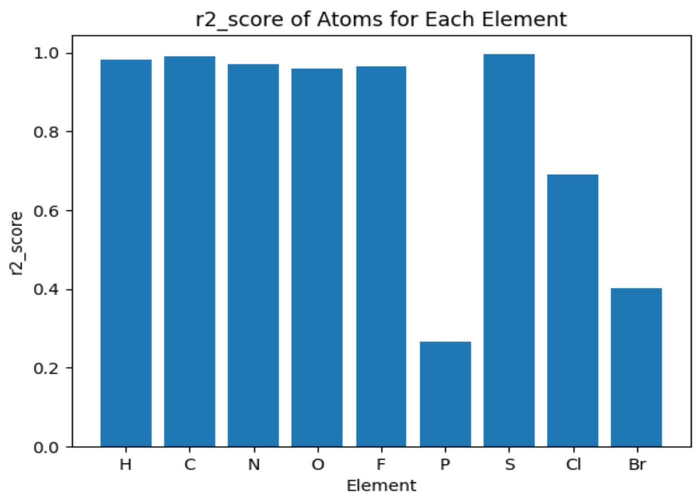
Case - I : Learning rate : 0.001 , Number of epochs = 100 , Batch size = 64 , no.of layers = 64
Case - II : Learning rate : 0.001 , Number of epochs = 100 , Batch size = 32 , no.of layers = 64
Case - III : Learning rate : 0.002 , Number of epochs = 100 , Batch size = 64 , no.of layers = 64
Case - IV : Learning rate : 0.002 , Number of epochs = 100 , Batch size = 32 , no.of layers = 64
Case - V : Learning rate : 0.0005,Number of epochs = 100 , Batch size = 64 , no.of layers = 64
Case - VI : Learning rate : 0.001 , Number of epochs = 100 , Batch size = 64 , no.of layers = 80



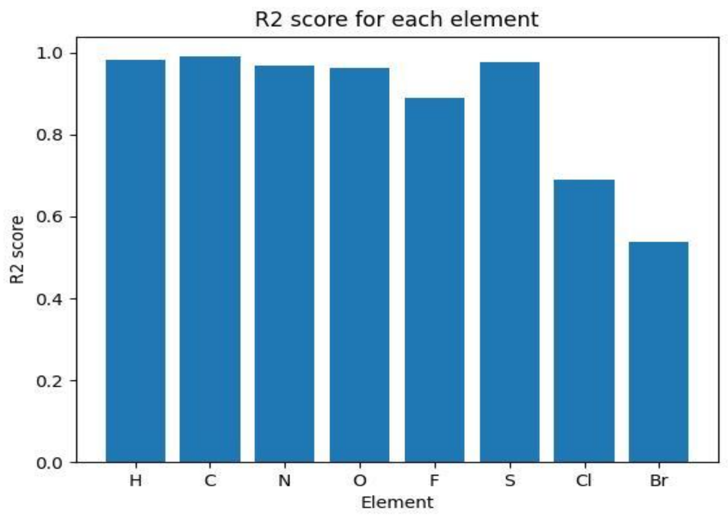
Case - I



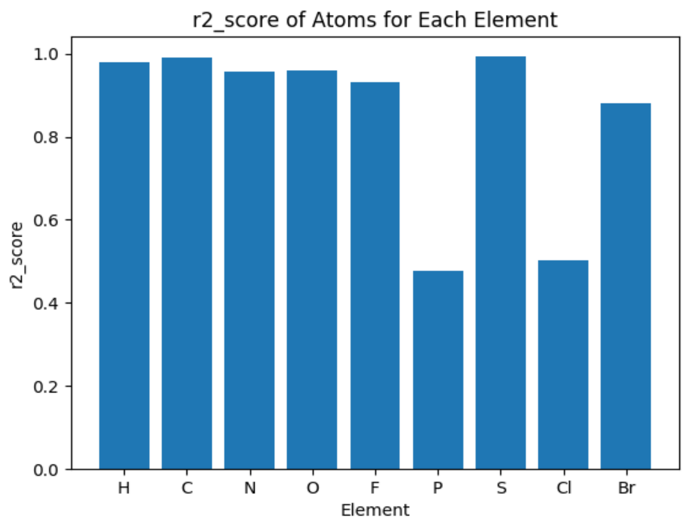
Case-II



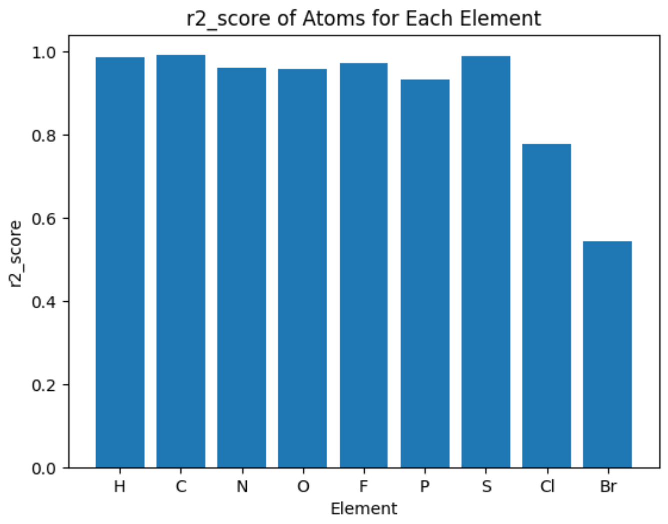
Case - III



Case - IV



Case - V



Case - VI

Conclusion :

APDs were first calculated for each molecule and the molecular descriptors obtained from APD were used in prediction of the partial charges. We created 2 Models using Random Forest and Neural Network methods to predict partial charges. Both the models were trained on the dataset as mentioned above. We have better predictions through the neural network model as compared to the random forest method. Improvements can be made on both the models by increasing the size of the dataset to train the models better and both the models predict the partial charges using different methodologies. So one better way to predict the charges is to use both the models and predict the charges using Ensemble Learning Technique.

To predict the charges using the Random Forest model and Neural Network model we have created a program which takes a sdf file as an input and predicts the charges. We also calculated RMSE using the `calculate_rmse()` function which takes predicted charges and the true charges as input.

We got **better accuracy** for every test set for the Random Forest where the number of trees = **100**, maximum depth = **6**, minimum number of samples to split = **6** and minimum number of samples in leaves = **6**.

We got **better accuracy** for every test set for the neural network in which the hyperparameters were : Learning rate : **0.001** , Number of epochs = **100** , Batch size = **64** and number of nodes as **80** in the first and hidden layers