Machine Learning Engineer Nanodegree

Capstone Project

Predicting Molecular Properties - A Kaggle Competition

Submitted By :
Aditi Basu

UDACITY

Contents

1 Project Overview

Nuclear Magnetic Resonance (NMR), a closely related technology to MRI, is typically used by researchers around the world to further their understanding of the structure and dynamics of molecules, across areas like environmental science, pharmaceutical science, and materials science.

One of the properties measured is the scalar coupling constant. These are effectively the magnetic interactions between a pair of atoms. The strength of this magnetic interaction depends on intervening electrons and chemical bonds that make up a molecule's three-dimensional structure.

There is currently some literature on this topic. The journal paper, Determination of scalar coupling constants by inverse Fourier transformation of in-phase multiplets, discusses a method for deriving scalar coupling constants for a particular coupling type (3JHN). Other papers discuss using the scalar coupling constant for various applications; this suggests the importance of the availability of this property and hence the need to innovate solutions to deriving this property using cost-effective methods like data science and machine learning.

2 Problem Statement

The objective of this project is to develop an algorithm that can predict the magnetic interaction between two atoms in a molecule (i.e. the scalar coupling constant).

Using state-of-the-art methods from quantum mechanics, it is possible to accurately calculate scalar coupling constants given only a 3D molecular structure as input. However, these quantum mechanics calculations are extremely expensive (days or weeks per molecule), and therefore have limited applicability in day-to-day workflows.

A fast and reliable method to predict these interactions will allow medicinal chemists to gain structural insights faster and cheaper, enabling scientists to understand how the 3D chemical structure of a molecule affects its properties and behavior.

Ultimately, such tools will enable researchers to make progress in a range of important problems, like designing molecules to carry out specific cellular tasks, or designing better drug molecules to fight disease.

3 Metrics

Kaggle has specified the evaluation metric for this competition to be the Log of the Mean Absolute Error (MAE), calculated for each scalar coupling type, and then averaged across types, so that a 1% decrease in MAE for one type provides the same improvement in score as a 1% decrease for another type.

$$score = \frac{1}{T}\sum_{t=1}^{T} log(\frac{1}{n_t}\sum_{i=1}^{n_t}|y_i - \hat{y_i}|)$$

The MAE metric penalizes huge errors but is not too sensitive to outliers. Considering the datasets have a few outliers present, this is a suitable evaluation metric. Taking the log of the MAE allows for predictions that are highly deviant to not alter the overall score, as the gradient of the log of the MAE is very small for large positive values of MAE.

4 Data Exploration

The datasets used in this project are provided courtesy of Kaggle. The following data/files are provided:

- train.csv - the training set, where the first column (molecule_name) is the name of the molecule where the coupling constant originates (the corresponding XYZ file is located at ./structures/.xyz), the second (atom_index_0) and third column (atom_index_1) is the atom indices of the atom-pair creating the coupling and the fourth column (scalar_coupling_constant) is the scalar coupling constant that we want to be able to predict
- test.csv - the test set; same info as train, without the target variable
- structures.zip - folder containing molecular structure (xyz) files, where the first line is the number of atoms in the molecule, followed by a blank line, and then a line for every atom, where the first column contains the atomic element (H for hydrogen, C for carbon etc.) and the remaining columns contain the X, Y and Z Cartesian coordinates (a standard format for chemists and molecular visualization programs)
- structures.csv - this file contains the same information as the individual xyz structure files, but in a single file

The following data is provided for the molecules in train.csv only.

- dipole_moments.csv - contains the molecular electric dipole moments. These are three dimensional vectors that indicate the charge distribution in the molecule. The first column (molecule_name) are the names of the molecule, the second to fourth column are the X, Y and Z components respectively of the dipole moment.
- magnetic_shielding_tensors.csv - contains the magnetic shielding tensors for all atoms in the molecules. The first column (molecule_name) contains the molecule name, the second column (atom_index) contains the index of the atom in the molecule, the third to eleventh columns contain the XX, YX, ZX, XY, YY, ZY, XZ, YZ and ZZ elements of the tensor/matrix respectively.

- Mulliken_charges.csv - contains the Mulliken charges for all atoms in the molecules. The first column (molecule_name) contains the name of the molecule, the second column (atom_index) contains the index of the atom in the molecule, the third column (Mulliken_charge) contains the Mulliken charge of the atom.
- potential_energy.csv - contains the potential energy of the molecules. The first column (molecule_name) contains the name of the molecule, the second column (potential_energy) contains the potential energy of the molecule.
- scalar_coupling_contributions.csv - The scalar coupling constants in train.csv (or corresponding files) are a sum of four terms. scalar_coupling_contributions.csv contain all these terms. The first column (molecule_name) are the name of the molecule, the second (atom_index_0) and third column (atom_index_1) are the atom indices of the atom-pair, the fourth column indicates the type of coupling, the fifth column (fc) is the Fermi Contact contribution, the sixth column (sd) is the Spin-dipolar contribution, the seventh column (pso) is the Paramagnetic spin-orbit contribution and the eighth column (dso) is the Diamagnetic spin-orbit contribution.

This section will be used for exploring the input space (or feature distribution). Correlations between features will be explored in Section 5.

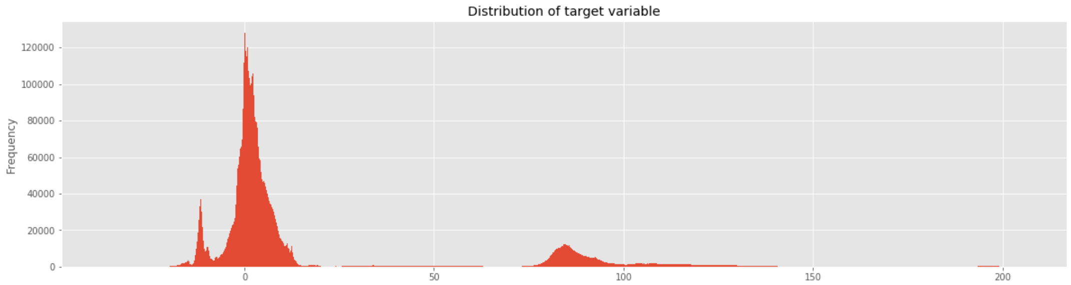


Figure 1: Distribution of Target Variable
This figure depicts the distribution of the target variable in the training set. It seems to be a bi-modal distribution, having two peaks near -20, zero, and a smaller peak at 80.

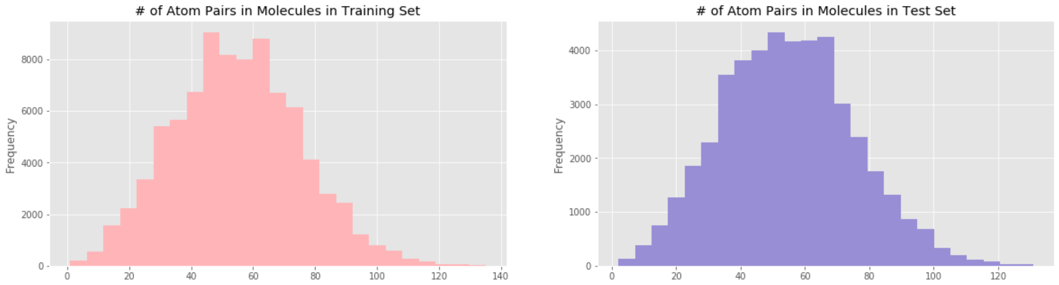


Figure 2: Distribution of Number of Atom Pairs in Molecules in Training and Test Sets
This figure uses a histogram plot to provide insight into how many atomic pairs are present in molecules. The most common number of atomic pairs is 60 for both training and test sets. Both observe a normal distribution around 60.

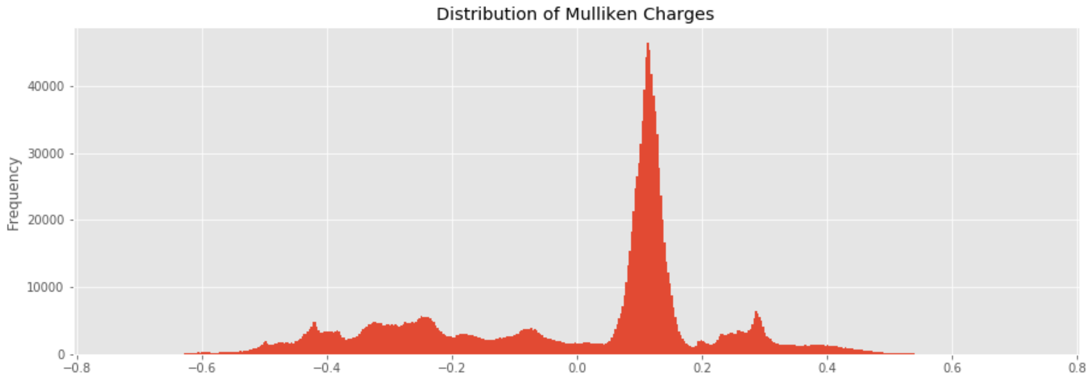


Figure 3: Distribution of Mulliken Charges
This figure illustrates the distribution of Mulliken charges using a histogram plot. There is a sharp peak present at about 0.1.

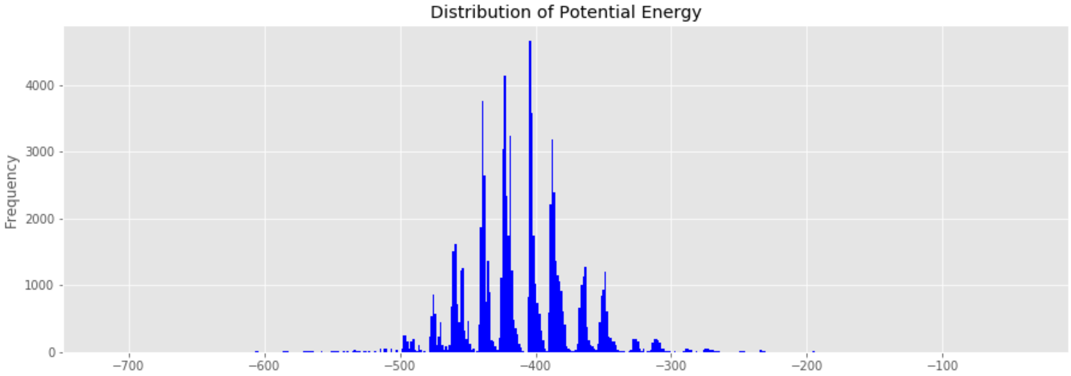


Figure 4: Distribution of Potential Energy
This figure depicts the distribution of potential energy using a histogram plot. The frequency seems to peak at just under -400. The selected number of bins (500) allows for visibility of the sharp spikes that occur every 20 units between -500 and -300.

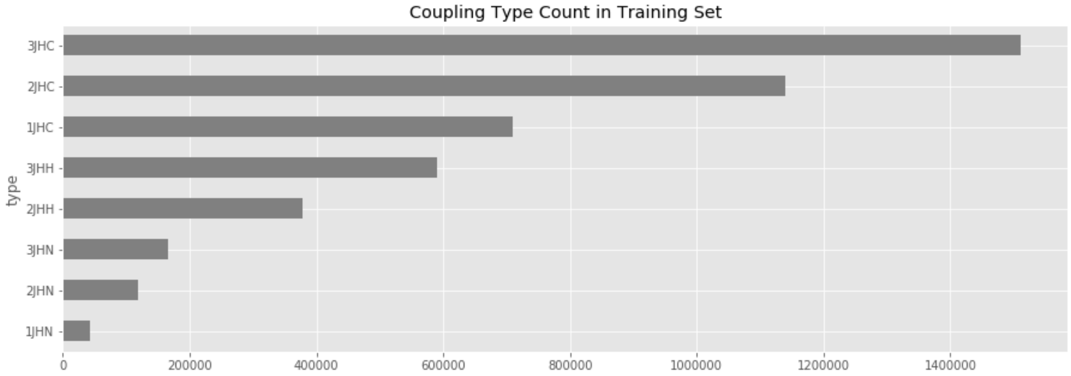


Figure 5: Coupling Type Count in Training Set
This figure shows the number the coupling types present in the training set. The most common coupling is of type 3JHC, and the least common is type 1JHN.

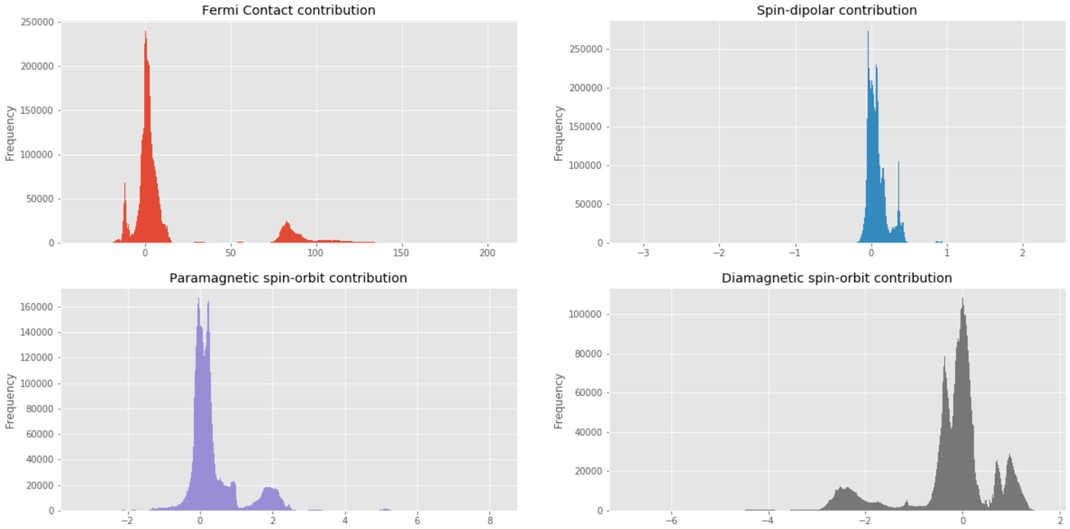
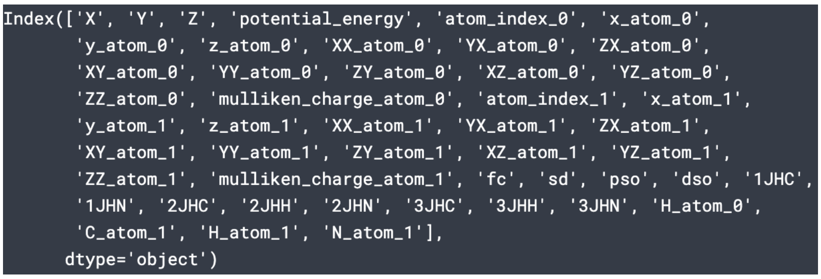


Figure 6: Distribution of Scalar Coupling Contributions
This figure plots the distribution of the individual scalar coupling contributions - fc, sd, pso and dso. All four contributions seem to follow a bi-modal distribution, except the dso contribution which has a small extra peak.

5 Exploratory Visualization

To get an idea of what the data looks like and to characterize it, a bunch of visualizations were performed.

Figure 7 shows the columns present in the training set:



```
Index(['X', 'Y', 'Z', 'potential_energy', 'atom_index_0', 'x_atom_0',
       'y_atom_0', 'z_atom_0', 'XX_atom_0', 'YX_atom_0', 'ZX_atom_0',
       'XY_atom_0', 'YY_atom_0', 'ZY_atom_0', 'XZ_atom_0', 'YZ_atom_0',
       'ZZ_atom_0', 'mulliken_charge_atom_0', 'atom_index_1', 'x_atom_1',
       'y_atom_1', 'z_atom_1', 'XX_atom_1', 'YX_atom_1', 'ZX_atom_1',
       'XY_atom_1', 'YY_atom_1', 'ZY_atom_1', 'XZ_atom_1', 'YZ_atom_1',
       'ZZ_atom_1', 'mulliken_charge_atom_1', 'fc', 'sd', 'pso', 'dso', '1JHC',
       '1JHN', '2JHC', '2JHH', '2JHN', '3JHC', '3JHH', '3JHN', 'H_atom_0',
       'C_atom_1', 'H_atom_1', 'N_atom_1'],
      dtype='object')
```

Figure 7: Features (independent variables) in Training Set

5.1 Analyzing the Molecular Structure Data

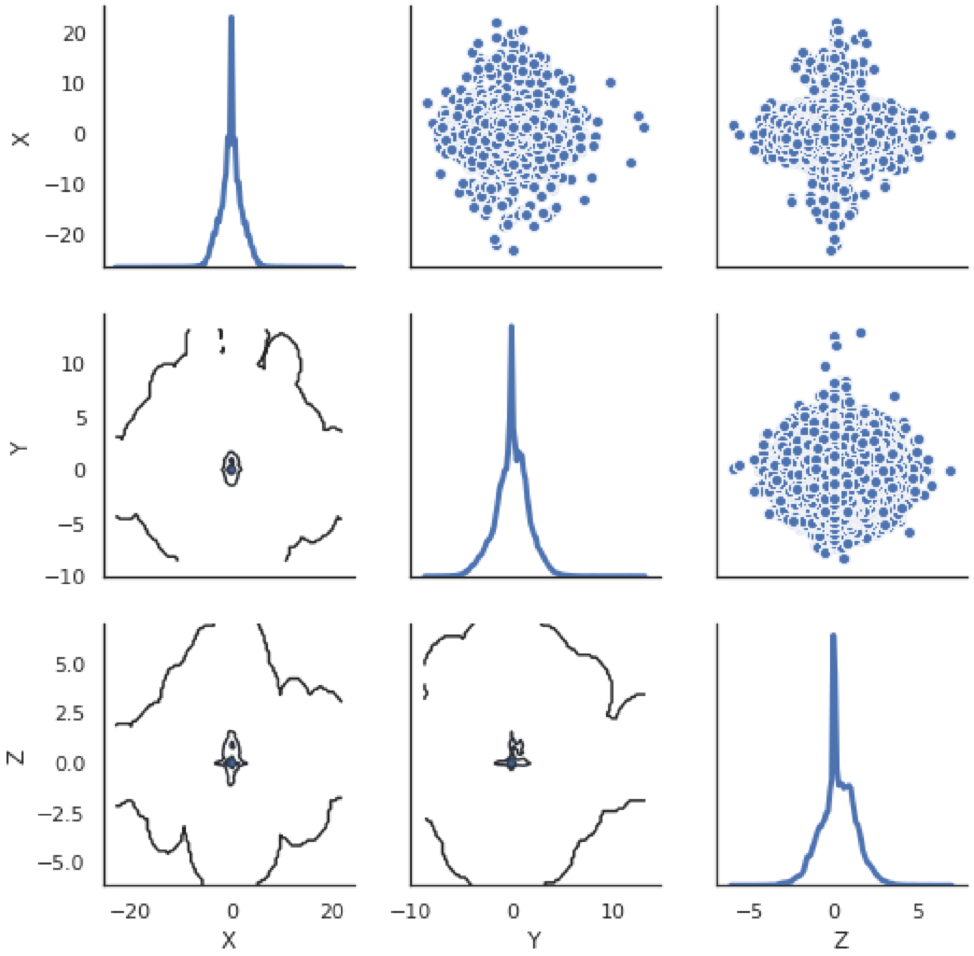


Figure 8: Pair Grid Representation of Molecular Structure of Every Atom in Each Coupling The figure above shows a pair-grid representation of the X, Y, Z Cartesian coordinates of each of the atoms in each molecule. The plots along the diagonal (i.e. when each variable is plotted against itself) and below the diagonal are in KDE form, while the plots above the diagonal are in scatter form. This visualization gives us insight into the relative space that the atoms in each molecule occupy.

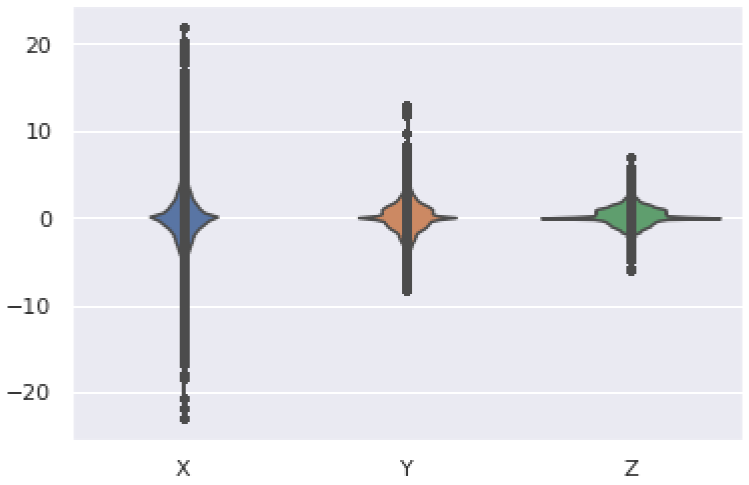


Figure 9: Violin Plot of the X, Y, Z Cartesian Coordinates of all Atoms
The above visualization shows a violin plot of each cartesian coordinate, X, Y, Z and provides insight into their spread.

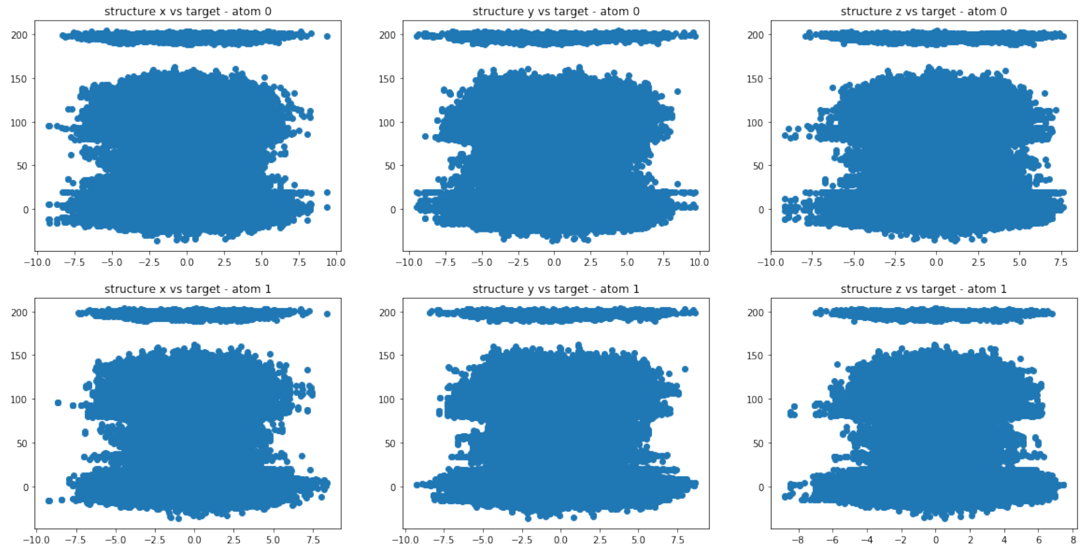


Figure 10: Molecular Structure vs Target Variable for Each Atomic Pair
The figure above is a set of plots that contains a Cartesian coordinate plotted against the target or the scalar coupling constant. The upper row consists of atom_0 coordinates and the lower row consists of atom_1 coordinates. This visualization was performed to gain insight into the correlations between the molecular structure and the target variable.

5.2 Scalar Coupling Constant Contributions

This section looks at the four features that, when summed up, equal the scalar coupling constant. These features are specifically Fermi Contant (fc), Spin-dipolar (sd), Paramagnetic spin-orbit (pso) and Diamagnetic spin-orbit (dso) and are contained in the scalar_coupling_contributions.csv dataset.

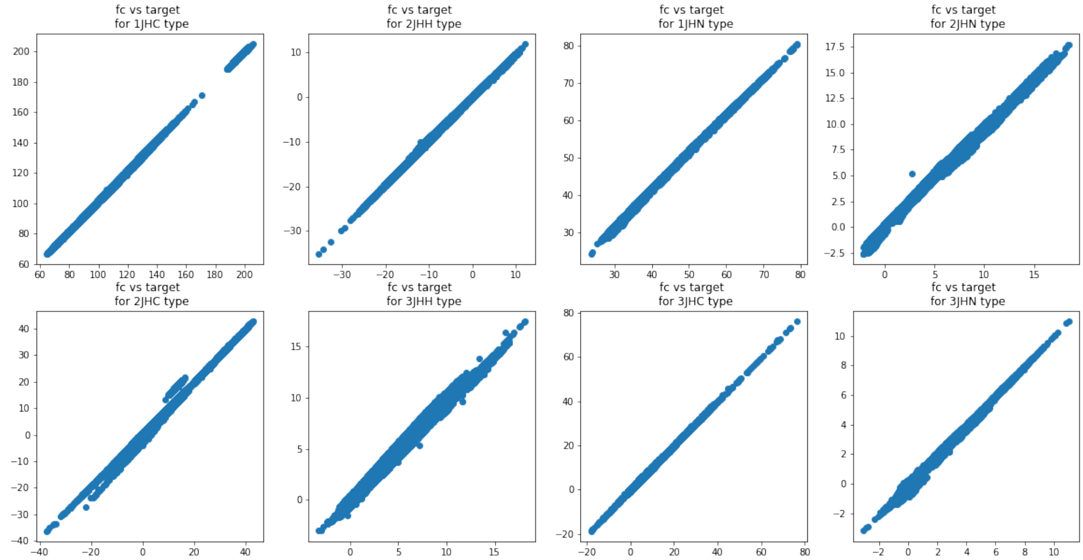


Figure 11: Fermi Constant (fc) vs Target Variable Categorized by Coupling Type

The figure above shows a series of plots depicting the correlation between fc and the target variable for each coupling type. As can be seen, there is a strong linear correlation between fc and the target variable for all coupling types.

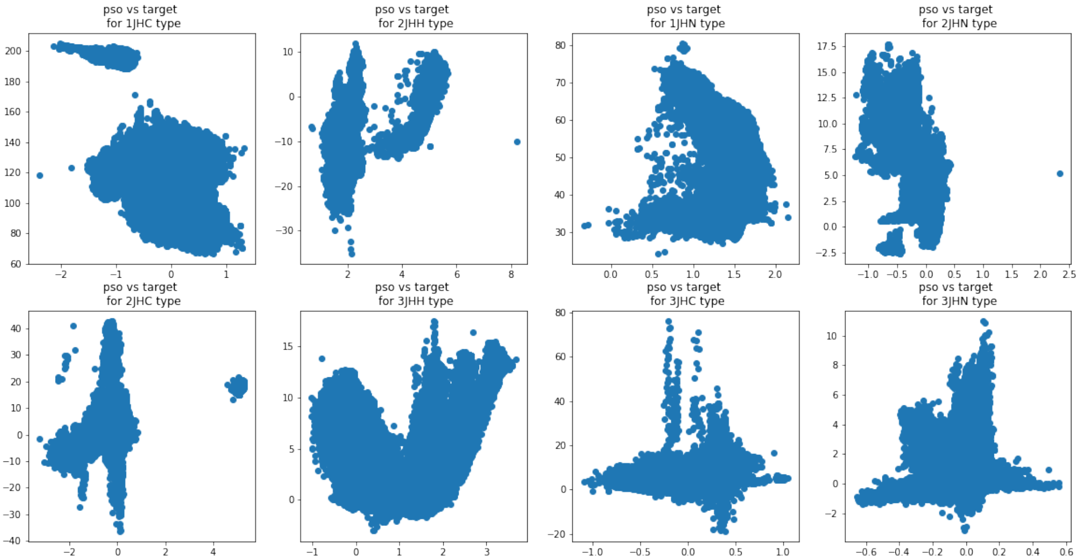


Figure 12: Paramagnetic Spin-Orbit (pso) vs Target Variable Categorized by Coupling Type

The figure above shows a series of plots depicting the correlation between pso and the target variable for each coupling type. From the scatter plot, there is no obvious correlation between pso and the target variable.

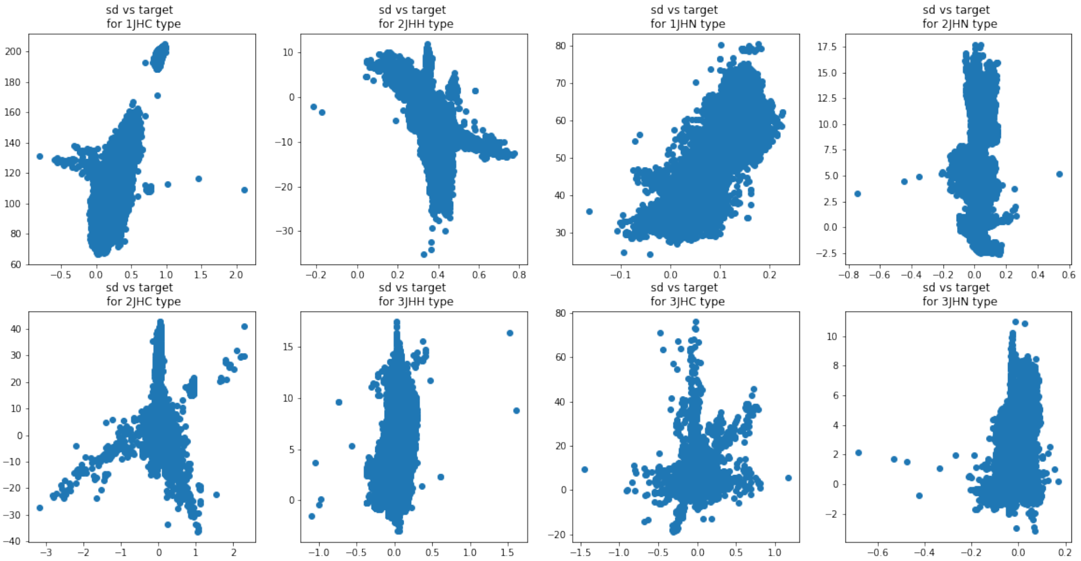


Figure 13: Spin-Dipolar (sd) vs Target Variable Categorized by Coupling Type

The figure above shows a series of plots depicting the correlation between pso and the target variable for each coupling type. From the scatter plot, there is no obvious correlation between sd and the target variable.

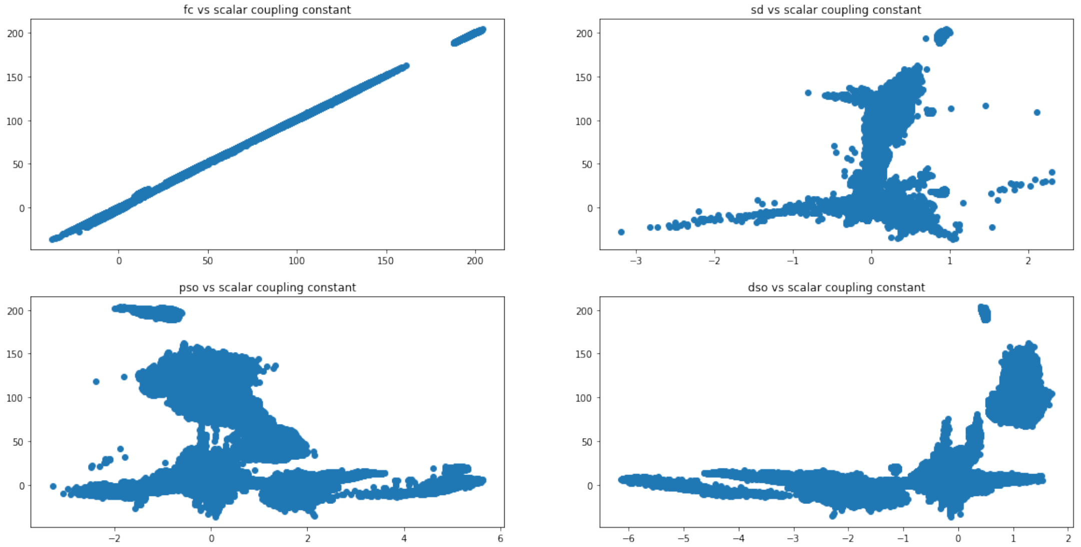


Figure 14: Scalar Coupling Constant Contributions

The figure above shows the correlation of each of the four contributions with the target variable; however, this time the plots are not categorized by the coupling type.

5.3 Correlation between Scalar Coupling Constant and Other Properties

This section explores the correlation between the target variable and other features not discussed in previous sections.

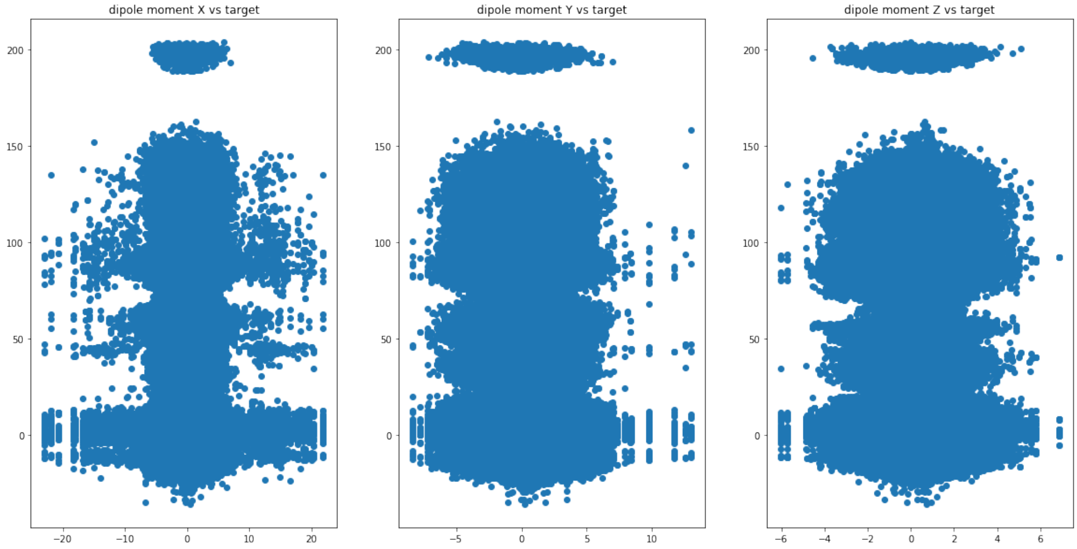


Figure 15: Dipole Moment vs Target Variable

The series of plots above show the correlation between the dipole moments in each direction and the target variable. From the plots, there doesn't seem to be any obvious correlation between the dipole moments and the target variable.

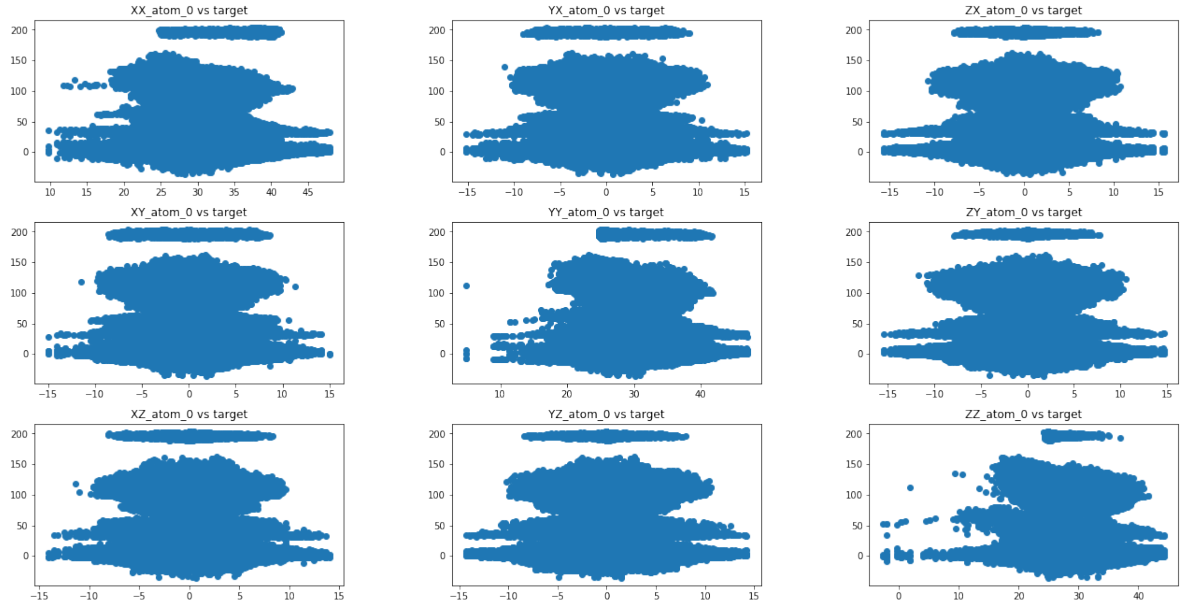


Figure 16: Magnetic Shielding Tensors vs Target Variable for Atom 0

The figure above shows the correlation between each element of the magnetic shielding tensors and the target variable for the first atom in the coupling (i.e. atom_0). There does not seem to be an obvious pattern in the spread of the magnetic shielding tensors over the target variable.

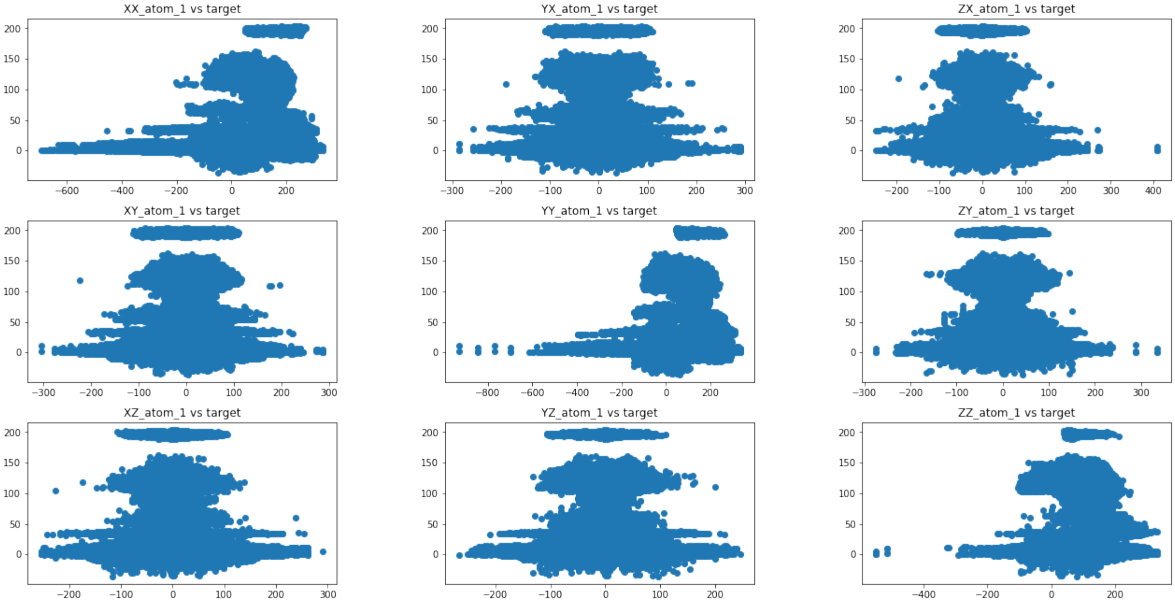


Figure 17: Magnetic Shielding Tensors vs Target Variable for Atom 1

The figure above shows the correlation between each element of the magnetic shielding tensors and the target variable for the first atom in the coupling (i.e. atom_1). In general, there does not seem to be an obvious pattern in the spread of the magnetic shielding tensors over the target variable. However, for the XX, YY, and ZZ elements of the shielding tensors (along the diagonal of the figure above), there seems to be a few outliers and the distribution is skewed to the right. For the other elements (outside the diagonal), the distribution is more normal and centered around zero.

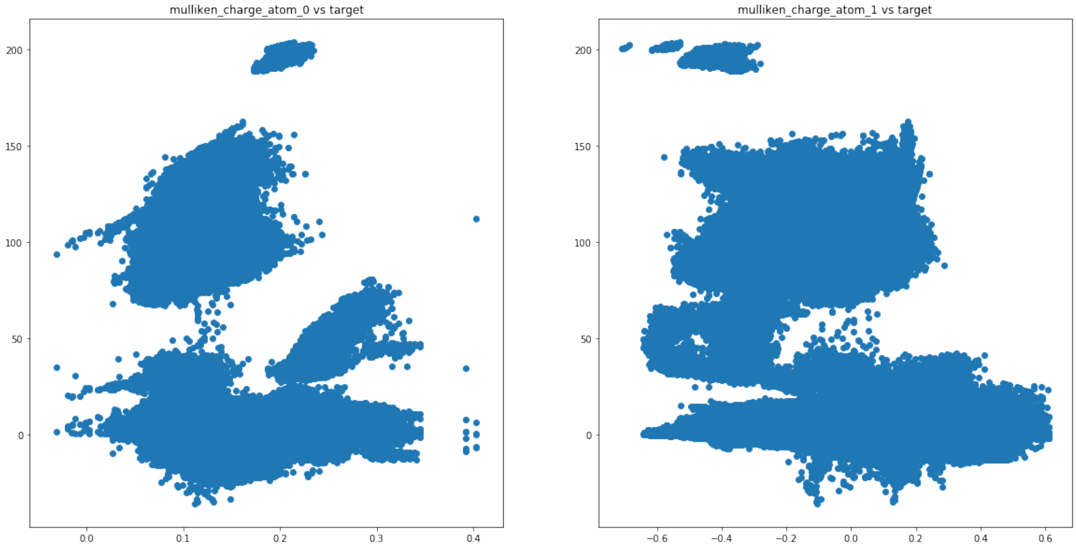


Figure 18: Mulliken Charge vs Target Variable
The figure above shows the correlation between the Mulliken charge and the target variable.

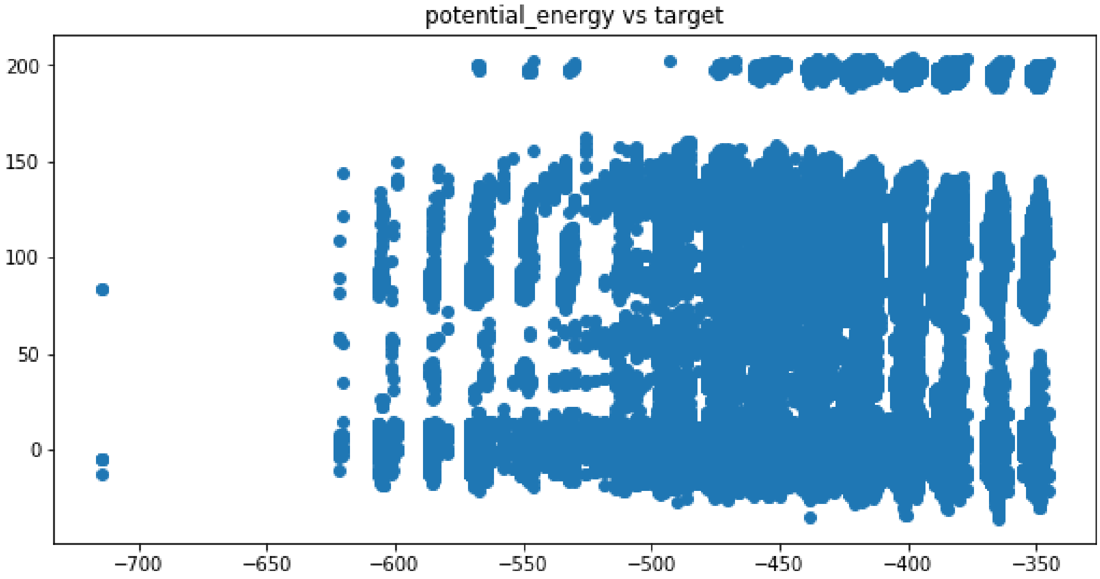


Figure 19: Potential Energy vs Target Variable
The figure above shows the correlation between the potential energy of the coupling and the target variable.

6 Algorithms & Techniques

For this project, a few learning models were tested. These learning models are described in the next few sections.

6.1 Random Forest

The Random Forest algorithm is a 'universal' and rather popular algorithm for building predictive models. It is classified as an ensemble method and can be used for both classification and regression problems. Ensemble methods make use of multiple weak learning models and produce better predictive results. As such, random forests use multiple uncorrelated decision trees to arrive at the best possible result. To understand random forests, their building block - the decision tree, needs to be understood.

Decision trees utilize a top-down approach in which the root node creates binary splits until a certain criteria is met. The splitting of nodes provides a predicted value based on the interior nodes leading to the leaf (or terminal) nodes. In the context of this project, which is a regression problem, the leaf nodes will produce a prediction of the target variable which is a floating number. Decision trees tend to overfit on training data, and hence lead to poor prediction performance on unseen data.

A random forest fits a number of decision tree classifiers/regressors on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control overfitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if specified in the code.

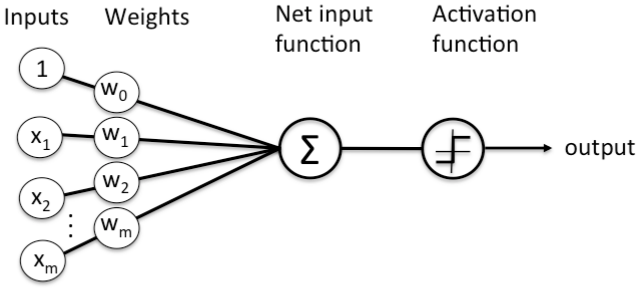
The following are common parameters tuned to optimize a random forest model:

- The number of trees in the forest
- The maximum depth of the tree
- The minimum number of samples required to split an internal node
- The minimum number of samples required to be at a leaf node
- Whether bootstrap samples are used when building trees

6.2 Neural Network

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text, or time series, must be translated. Neural networks can be used to solve problems in both classification and regression domains.

A neural network consists of layers, which consist of nodes. A node is a place where computation occurs. A node combines input from the data with a set of coefficients, or weights, that either amplify or dampen that input, thereby assigning significance to inputs with regards to the task the algorithm is trying to learn. These input-weight products are summed and then the sum is passed through a node's so-called activation function, to determine whether and to what extent that signal should progress further through the network to affect the ultimate outcome. If the signal passes through, the neuron has been 'activated'. The figure below is an example of what a node looks like:



A typical training procedure for a neural network is as follows:

- Define the neural network that has some learnable parameters (or weights)
- Iterate over a dataset of inputs
- Process input through the network
- Compute the loss (how far is the output from being correct)
- Propagate gradients back into the network's parameters
- Update the weights of the network, typically using a simple update rule:

$$weight = weight - learning\_rate * gradient$$

The following are common parameters tuned to optimize a neural network:

- Number of samples per gradient sample
- Number of epochs to train the model. An epoch is an iteration over the entire data provided.
- Fraction of the training data to be used as validation data
- Method used to optimize weights

7 Benchmark

The benchmark model was characterized to be a model with minimal modifications to its default parameters and to use a dataset that contains the original features only. Kaggle has provided various datasets for molecules in the train.csv file. However, molecular structure data (as found in the structures.csv file) is the only data that has been provided for molecules in both the train.csv and test.csv files. As such, the benchmark model will train only on the molecular structure data for molecules in a subset of the training set and predict the scalar coupling constant (target variable) for the validation set (0.2% of the original training set) to determine the score of the model using R$\hat{2}$ scoring.

The benchmark model used was a Random Forest and was able to get a R$\hat{2}$ score of 0.930 using the following parameters:

- n_estimators = 500
- training_size = 50,000
- random_state = 0

The same model was also run using the evaluation metric specified by Kaggle - the Log of the Mean Absolute Error (MAE). Using the above parameters, the MAE was calculated to be 3.327.

8 Data Preprocessing

Kaggle has provided a number of different datasets to aid us in predicting the scalar coupling constant. Data preprocessing included checking for missing values and outliers in each dataset. After analysis, it turned out that there were no missing values and that datasets either followed a close-to-normal or bi-modal distribution.

Kaggle has provided the datasets in separate files. As such, in order to get all the data into a learning model or algorithm, a single variable (or dataframe) needs to be created that includes all the features from the various datasets. Once all the features were combined, categorical features, such as the 'type', were one-hot-encoded to produce one column for each 'type'.

Hereafter, new features were either engineered from existing features or a subset of the required features were extracted to feed into the learning model at hand.

9 Implementation

9.1 Random Forest using Molecular Structure Data

The datasets provided are such that there is a lot more information available for the training set (train.csv) than for the test set (test.csv). This means that the training set contains more independent variables than the test set.

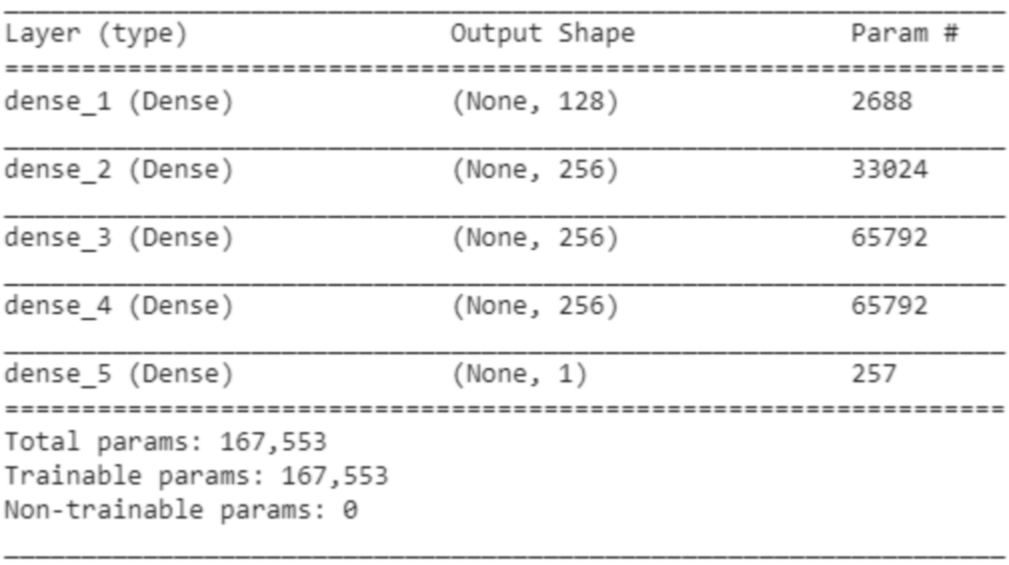
As such, the first model that was attempted (aka the benchmark model) is a Random Forest trained on solely the molecular structure data (provided in the structures.csv file). It was worth exploring this model because molecular structure data was the only dataset that was provided for both the training and test sets. This model and use of this simple dataset will help determine whether the molecular structure data is enough to make accurate predictions.

9.2 Simple Neural Network using Molecular Structure Data

Moving beyond the benchmark model discussed in the previous section, a simple neural network model was chosen and trained solely on the molecular structure data. Four iterations of this model were performed as outlined in the next few sections.

Iteration 1

The first iteration of the neural network contained 3 hidden layers and was trained over 8 epochs. The following figure shows the architecture of the first iteration of the neural network.



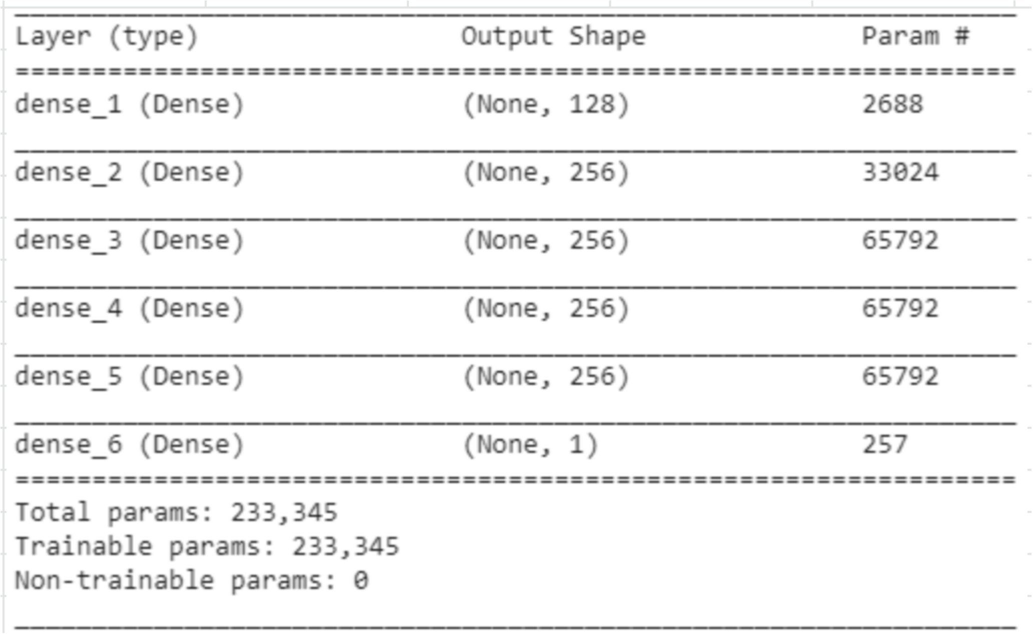
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               2688
_________________________________________________________________
dense_2 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_3 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_4 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 257
=================================================================
Total params: 167,553
Trainable params: 167,553
Non-trainable params: 0
_________________________________________________________________
```

Figure 20: Neural Network with 3 Hidden Layers

This architecture was able to obtain a score (evaluated using the Log of Mean Absolute Error) of 3.049, which is an improvement from the benchmark model.

Iteration 2

The second iteration of the neural network contained 4 hidden layers and was trained over 8 epochs.



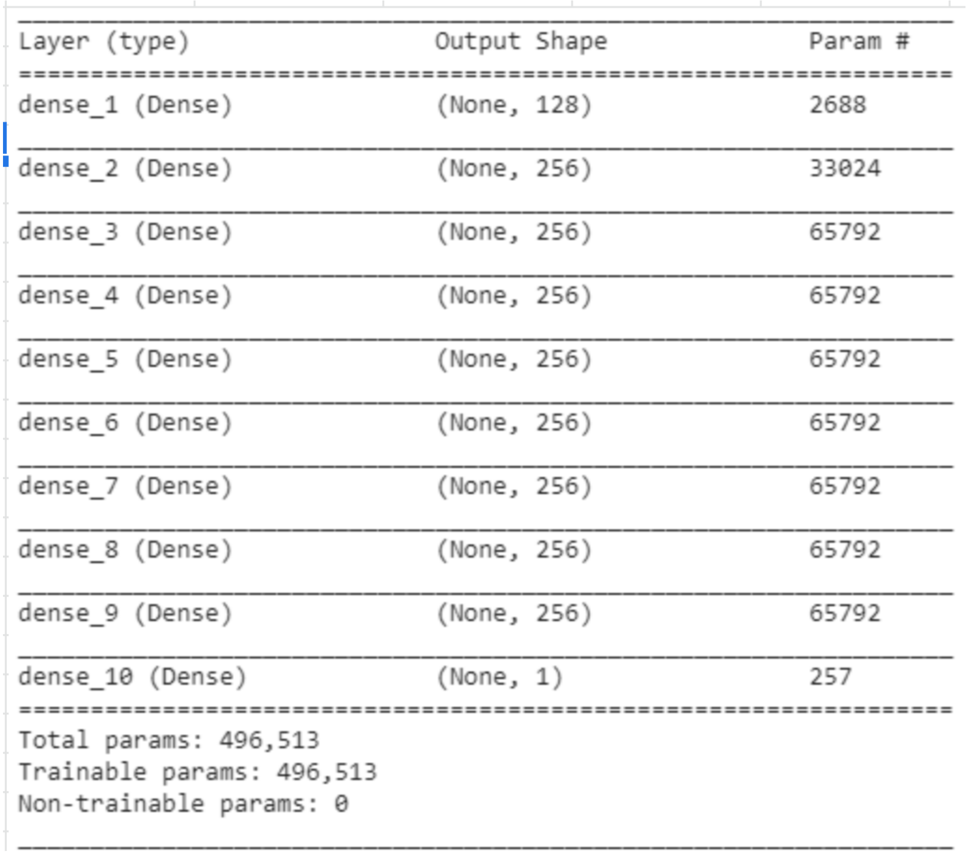
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               2688
_________________________________________________________________
dense_2 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_3 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_4 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_5 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 257
=================================================================
Total params: 233,345
Trainable params: 233,345
Non-trainable params: 0
_________________________________________________________________
```

Figure 21: Neural Network with 4 Hidden Layers

This architecture was able to obtain a score (evaluated using the Log of Mean Absolute Error) of 2.791, which is an improvement from Iteration 1.

Iteration 3

The third iteration of the neural network contained 8 hidden layers and was trained over 8 epochs.



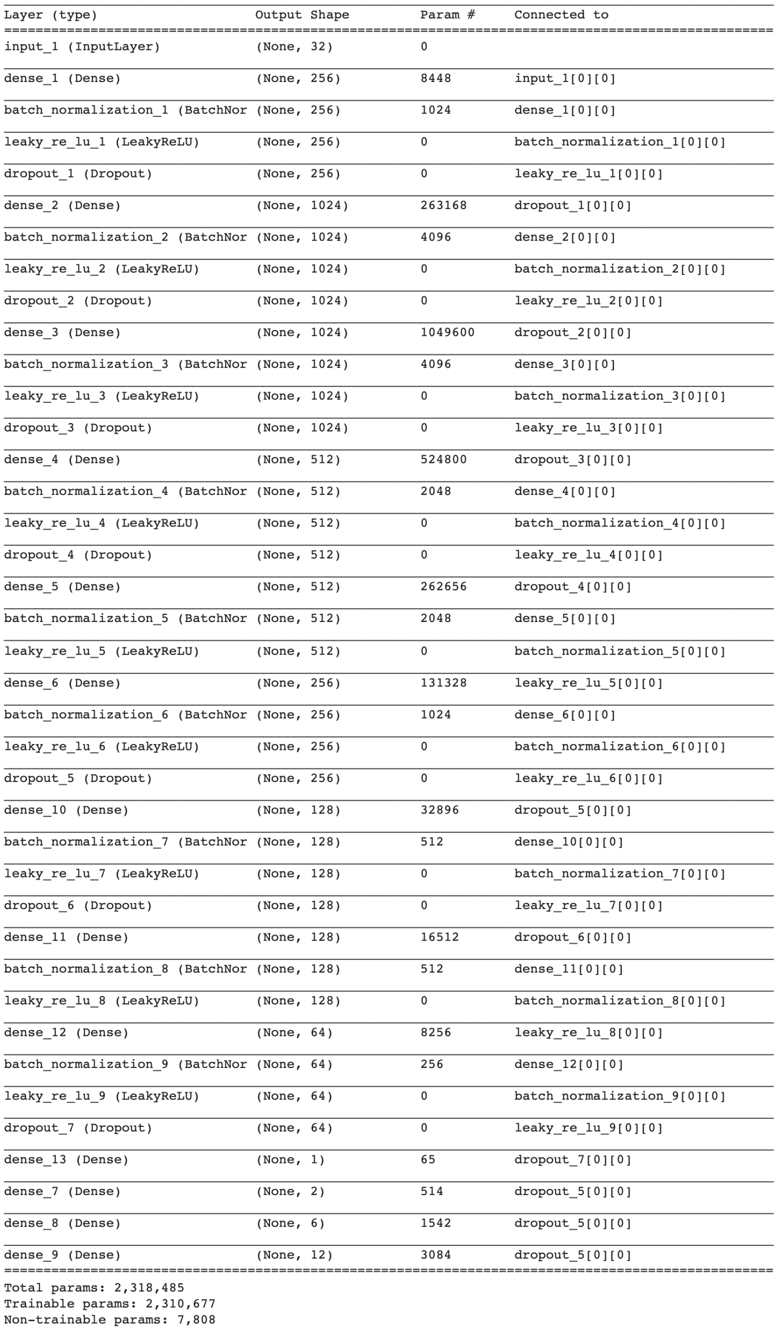
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               2688
_________________________________________________________________
dense_2 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_3 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_4 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_5 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_6 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_7 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_8 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_9 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 257
=================================================================
Total params: 496,513
Trainable params: 496,513
Non-trainable params: 0
_________________________________________________________________
```

Figure 22: Neural Network with 8 Hidden Layers

This architecture was able to obtain a score (evaluated using the Log of Mean Absolute Error) of 3.065, which is lower than the architecture discussed in Iteration 2. This could signify the possibility of over-fitting.

9.3 Neural Network using Engineered Features

The next model used is also a neural network, but with the following architecture:



```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 32)           0
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 256)          8448        input_1[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 256)          1024        dense_1[0][0]
__________________________________________________________________________________________________
leaky_re_lu_1 (LeakyReLU)       (None, 256)          0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 256)          0           leaky_re_lu_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 1024)         263168      dropout_1[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 1024)         4096        dense_2[0][0]
__________________________________________________________________________________________________
leaky_re_lu_2 (LeakyReLU)       (None, 1024)         0           batch_normalization_2[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 1024)         0           leaky_re_lu_2[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 1024)         1049600     dropout_2[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 1024)         4096        dense_3[0][0]
__________________________________________________________________________________________________
leaky_re_lu_3 (LeakyReLU)       (None, 1024)         0           batch_normalization_3[0][0]
__________________________________________________________________________________________________
dropout_3 (Dropout)             (None, 1024)         0           leaky_re_lu_3[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 512)          524800      dropout_3[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 512)          2048        dense_4[0][0]
__________________________________________________________________________________________________
leaky_re_lu_4 (LeakyReLU)       (None, 512)          0           batch_normalization_4[0][0]
__________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 512)          0           leaky_re_lu_4[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 512)          262656      dropout_4[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 512)          2048        dense_5[0][0]
__________________________________________________________________________________________________
leaky_re_lu_5 (LeakyReLU)       (None, 512)          0           batch_normalization_5[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 256)          131328      leaky_re_lu_5[0][0]
__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 256)          1024        dense_6[0][0]
__________________________________________________________________________________________________
leaky_re_lu_6 (LeakyReLU)       (None, 256)          0           batch_normalization_6[0][0]
__________________________________________________________________________________________________
dropout_5 (Dropout)             (None, 256)          0           leaky_re_lu_6[0][0]
__________________________________________________________________________________________________
dense_10 (Dense)                (None, 128)          32896       dropout_5[0][0]
__________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 128)          512         dense_10[0][0]
__________________________________________________________________________________________________
leaky_re_lu_7 (LeakyReLU)       (None, 128)          0           batch_normalization_7[0][0]
__________________________________________________________________________________________________
dropout_6 (Dropout)             (None, 128)          0           leaky_re_lu_7[0][0]
__________________________________________________________________________________________________
dense_11 (Dense)                (None, 128)          16512       dropout_6[0][0]
__________________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 128)          512         dense_11[0][0]
__________________________________________________________________________________________________
leaky_re_lu_8 (LeakyReLU)       (None, 128)          0           batch_normalization_8[0][0]
__________________________________________________________________________________________________
dense_12 (Dense)                (None, 64)           8256        leaky_re_lu_8[0][0]
__________________________________________________________________________________________________
batch_normalization_9 (BatchNor (None, 64)           256         dense_12[0][0]
__________________________________________________________________________________________________
leaky_re_lu_9 (LeakyReLU)       (None, 64)           0           batch_normalization_9[0][0]
__________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 64)           0           leaky_re_lu_9[0][0]
__________________________________________________________________________________________________
dense_13 (Dense)                (None, 1)            65          dropout_7[0][0]
__________________________________________________________________________________________________
dense_7 (Dense)                 (None, 2)            514         dropout_5[0][0]
__________________________________________________________________________________________________
dense_8 (Dense)                 (None, 6)            1542        dropout_5[0][0]
__________________________________________________________________________________________________
dense_9 (Dense)                 (None, 12)           3084        dropout_5[0][0]
==================================================================================================
Total params: 2,318,485
Trainable params: 2,310,677
Non-trainable params: 7,808
```

Figure 23: Neural Network Model with Batch Normalization and Dropouts

The following new features were engineered for this model:

1. the 'x' component of the distance between each pair of atoms in each molecule
2. the 'y' component of the distance between each pair of atoms in each molecule
3. the 'z' component of the distance between each pair of atoms in each molecule
4. distance between each pair of atoms in each molecule
5. the mean 'x' coordinate of each molecule (calculated from the individual 'x' coordinates of each atom in the molecule)
6. the mean 'y' coordinate of each molecule (calculated from the individual 'y' coordinates of each atom in the molecule)
7. the mean 'z' coordinate of each molecule (calculated from the individual 'z' coordinates of each atom in the molecule)
8. number of atoms in each molecule
9. nearest distance between a pair of atoms present in each molecule
10. furthest distance between a pair of atoms present in each molecule
11. distance of each atom from the center of the molecule
12. distance of each atom from the smallest coordinate in every direction of the molecule
13. distance of each atom from the largest coordinate in every direction of the molecule
14. ratio of 'x' component of distance of each atom from the center of the molecule to the distance itself
15. ratio of 'y' component of distance of each atom from the center of the molecule to the distance itself
16. ratio of 'z' component of distance of each atom from the center of the molecule to the distance itself
17. ratio of 'x' component of distance of each atom from the smallest coordinate in every direction of the molecule to the distance itself
18. ratio of 'y' component of distance of each atom from the smallest coordinate in every direction of the molecule to the distance itself
19. ratio of 'z' component of distance of each atom from the smallest coordinate in every direction of the molecule to the distance itself
20. ratio of 'x' component of distance of each atom from the largest coordinate in every direction of the molecule to the distance itself
21. ratio of 'y' component of distance of each atom from the largest coordinate in every direction of the molecule to the distance itself

22. ratio of 'z' component of distance of each atom from the largest coordinate in every direction of the molecule to the distance itself
23. ratio of 'x' coordinate of distance between a pair of atoms to the distance itself
24. ratio of 'y' coordinate of distance between a pair of atoms to the distance itself
25. ratio of 'z' coordinate of distance between a pair of atoms to the distance itself
26. cosine similarity between the resultant vectors in 17-19 for each atomic pair
27. cosine similarity between the resultant vectors in 20-22 for each atomic pair
28. cosine similarity between the resultant vectors in 14-16 for each atomic pair
29. cosine similarity between the resultant vectors in 23-25 for each atomic pair
30. cosine similarity between the resultant vector in 17-19 for one atom and 23-25
31. cosine similarity between the resultant vector in 20-22 for one atom and 23-25

This set of features along with existing features were fed into the previously specified neural network. This model was able to obtain a log(MAE) score of 1.99.

9.4 Complications

During the course of this project, a few challenges were encountered. The first challenge was writing the predictions into a properly formatted CSV file, as specified by Kaggle. Formatting included getting the header of the CSV file and the columns of data to be in the right format, and removing any unnecessary white spaces.

One of the things that I would like to have tried in this project is to use the RDKit package. RDKit is an open-source cheminformatics software used for visualization of molecular structures and for extracting important physical features of molecules. However, because of OS and disk storage limitations, the package could not be installed and used.

In general, long training and execution times of the learning models were a hindrance to rapidly improving upon existing parameters, especially when multiple epochs were required for training.

10 Refinement

The first solution (aka the benchmark model) adopted for this project was a simple Random Forest using only existing molecular structure data from the structures.csv file and no new engineered features. This model yielded a log(MAE) score of 3.327.

The final solution to this project consisted of a neural network with batch normalization and dropouts, and the corresponding training data consisted of existing features along with newly engineered features. This model yielded a log(MAE) score of 1.99, a significant improvement from the benchmark model.

11 Model Evaluation & Validation

The final model, a neural network, consists of a total of 14 layers, or 13 hidden layers. Batch normalization is used for all layers except the input layer and the last 4 layers. This is a technique for training neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

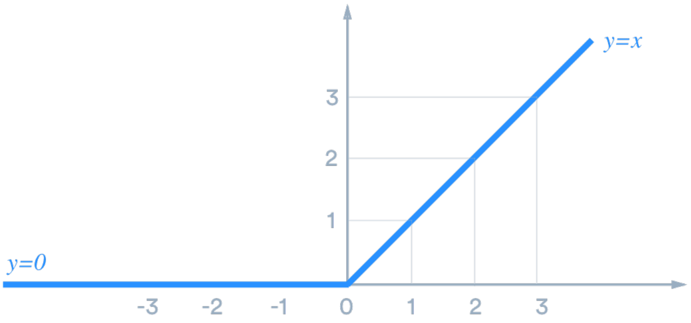


Figure 24: ReLU Activation Function

ReLU, short for Rectified Linear Unit, is an activation function that outputs the same value for values greater than zero, and outputs zero for values less than zero. This is graphically presented in the figure above.

Leaky ReLU is a variant of ReLU that allows a small positive gradient when the unit is not active, as shown by the equation below. In this equation, the gradient of 0.01 can substituted for any other small number. Leaky ReLU the activation function used for all units in 9 out of the 13 hidden layers.

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{otherwise.} \end{cases}$$

Dropout is another technique used in this neural network. It helps to reduce over-fitting and improve the generalization of neural networks. By randomly dropping out nodes during training, a single model can be used to simulate having a large number of different network architectures. In this network, nodes are set to drop out at a probability of 0.2.

By utilizing techniques like batch normalization and dropout, the model tends to be a lot more robust than the neural network architecture discussed in Section 9.2. The architecture and performance of this model can definitely be improved, and will be discussed in a later section of this report.

12 Justification

Compared to the benchmark model which yielded a score of 3.327, the final model resulted in a much improved score of 1.99. However, considering that the best score (using log(MAE)) can be as low as -20.72 if all predictions are perfect, there is still a lot of room for improvement (see Improvement section below).

13 Free-form Visualization

All the data provided for this project is in tabular or numeric form. Since we're dealing with molecular structure, it's worth getting a visual idea of the structure. This section illustrates the structure of some of the molecules in the training set.

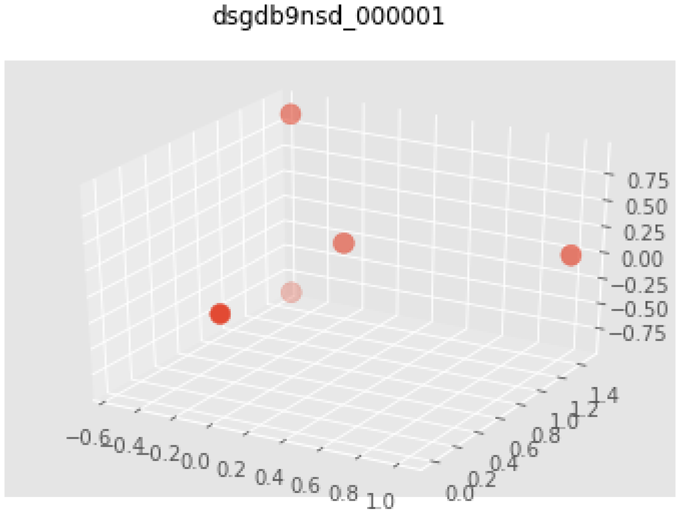


Figure 25: Structure of Molecule dsgdb9nsd_000001

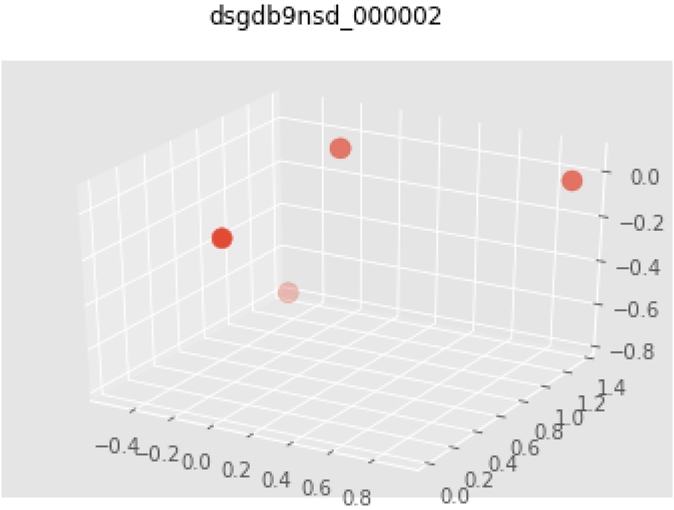


Figure 26: Structure of Molecule dsgdb9nsd_000002

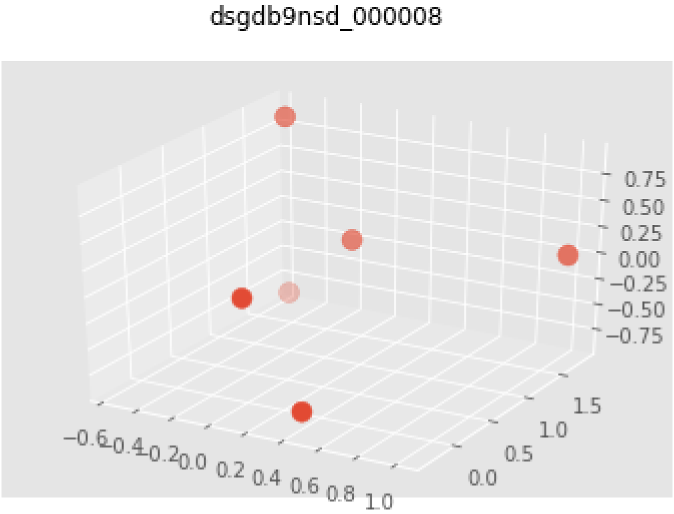


Figure 27: Structure of Molecule dsgdb9nsd_000008

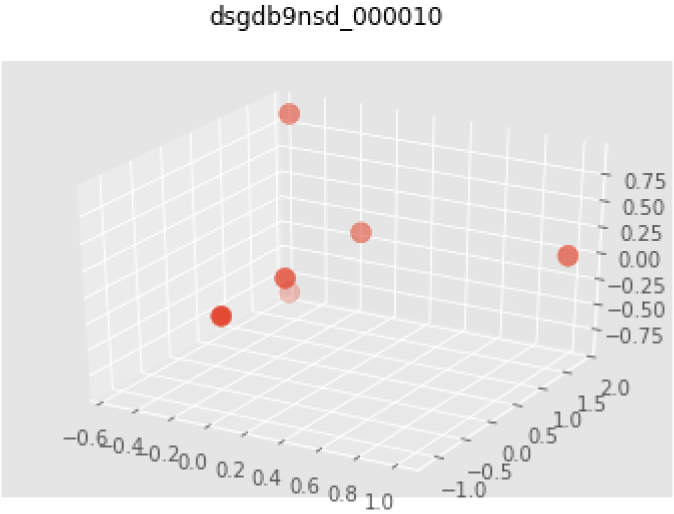


Figure 28: Structure of Molecule dsgdb9nsd_000010

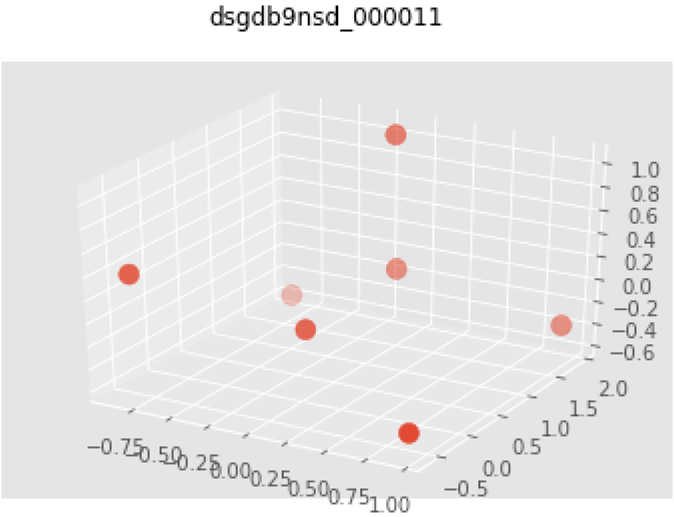


Figure 29: Structure of Molecule dsgdb9nsd_000011

14 Reflection

The process used for this project can be summarized in the following steps:

1. Understand the problem statement - this also involves getting acquainted with domain-specific knowledge.
2. Understand and analyze the datasets provided - this involves understanding the formats of the files in which the datasets are provided, the way the data has been presented, and identifying any missing values from the data.
3. Visualize the data - perform visualizations on subsets of data that might provide more insight into the problem.
4. Review Kaggle kernels to get ideas on solving the problem at hand.
5. Develop solutions to the problem starting with the simplest one
    - Random Forest using only the provided molecular structure data in structures.csv (benchmark solution)
    - Simple Neural Network using only the provided molecular structure data in structures.csv
    - A more sophisticated neural network using newly engineered features
6. Fine-tune models
7. Compare final solution to benchmark solution

The hardest part of the project was getting started. This was due to two reasons - 1) lack of domain knowledge, and 2) not knowing how to start brainstorming solutions to the problem. However, upon reviewing a few Kaggle kernels, a few ideas were spurred and getting started was made easier. Kaggle kernels also provide interesting perspective on the data and the problem at hand. They are an easy way to work with models one has personally not worked with before.

15 Improvement

Even though the final solution was a significant improvement from the benchmark solution in terms of the evaluation metric, log (MAE), there is a lot of room for improvement. However, due to time constraints, the solution could not be further developed. The following are ways the solution can be improved upon:

- Using the RDKit package (mentioned in Section 9.4) to engineer new features that the layman, who does not have strong domain knowledge, can use in their learning models.

- Apart from structures.csv, all the other datasets provided cater to the molecules in the training set (train.csv) but not the test set (test.csv). For this reason, it can be worth doing the following:
  - Combine all data provided into one single dataset
  - Split the dataset into a training set and a validation set
  - Choose a learning model and train it using the training set
  - Use trained model to predict features, present in training set (and validation set) but not the test set, for the validation set and determine accuracy.
  - Once accuracy is good enough, these features can be predicted for the molecules in the test set
  - Choose a learning model to predict the target variable (scalar coupling constant) for the molecules in the test set

Bibliography

[1] CHAMPS (CHemistry And Mathematics in Phase Space). (2019, May 31). Predicting Molecular Properties
Retrieved from *https://www.kaggle.com/c/champs-scalar-coupling/overview*

[2] Gaetano T. Montelione, S. Donald Emerson, Barbara A. Lyons. (1992, April). A general approach for determining scalar coupling constants in polypeptides and proteins.
Retrieved from *https://onlinelibrary.wiley.com/doi/abs/10.1002/bip.360320406*

[3] T Szyperski, P Güntert, G Otting, K Wüthrich. (1992, October 1). Determination of scalar coupling constants by inverse Fourier transformation of in-phase multiplets.
Retrieved from *https://www.sciencedirect.com/science/article/pii/002223649290209P*

[4] Microsoft Azure Machine Learning: Algorithm Cheat Sheet. (2015).
Retrieved from *https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-cheat-sheet*

[5] Best Practices for Feature Engineering. (2017, July 26).
Retrieved from *https://elitedatascience.com/feature-engineering-best-practices*

[6] Random Forest Classifier.
Retrieved from *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html/*

[7] The Sequential model API.
Retrieved from *https://keras.io/models/sequential/*

[8] A Beginner's Guide to Neural Networks and Deep Learning.
Retrieved from *https://skymind.ai/wiki/neural-networkdefine*

[9] Jason Brownlee. (2019, January 16). Accelerate the Training of Deep Neural Networks with Batch Normalization.
Retrieved from *https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/*

[10] Danqing Liu. (2017, November 30). A Practical Guide to ReLU
Retrieved from *https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7*

[11] Rectifier (neural networks)
Retrieved from *https://en.wikipedia.org/wiki/Rectifier_(neural_networks)Leaky_ReLUs*

[12] Jason Brownlee. (2018, December 3). A Gentle Introduction to Dropout for Regularizing Deep Neural Networks
Retrieved from *https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/*