1. **Calculation of topological index**

Firstly, construct the adjacency matrix, solve the eigenvalue problem of the adjacency matrix, calculate the eigenvalues and corresponding eigenvectors of the adjacency matrix. The spectral moment is a matrix composed of eigenvalues, and then sum the eigenvalues to obtain the topological index.

Code implementation: **tuopu.py**

The final generated table should include columns such as SMILES, Adjacency Matrix, Kappa 1-Kappa Inf, etc. The Kappa 1-Kappa Inf terms for reactants and products will be used as descriptors for model training in the future. Tables 1 and 2 correspond to the topological indices of reactants and products obtained after running the code. In the subsequent data tables (data, data-all) used for model training, replace the column names with k1-kI和k\_1-k\_I。

1. Morgan Fingerprint extraction

Use RDkit library to process SMILES information of reactant products (reactant rsmi, product psmi) and extract Morgan fingerprints of different dimensions. By modifying the nBits parameter in the code, fingerprints of any dimension can be extracted.

Code implementation: **Morganfingerprint.py**

1. Model training

Using k1 kInf, k in the data table (5040)\_ 1-k\_ Inf, Hrxn, reactants and product fingerprints with fingerprint dimensions of 700 (Reactant\_700, Product\_700) are used as descriptors as inputs, and activation energy is used as the target value to train the machine learning model. Run the corresponding code below to obtain the R2, MAE, RMSE values of the model and the training effect diagram. The import data path needs to be changed according to the storage location of the data table.

1. gbr.py
2. SVR.py
3. BR.py
4. KNN.py
5. RF.py
6. ann.py

The table shows the results of running the corresponding code

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ANN | SVR | GBR | RF | KNN | BR |
| R2 | 0.954 | 0.620 | 0.635 | 0.512 | 0.288 | 0.481 |
| MAE | 0.088 | 0.446 | 0.440 | 0.522 | 0.639 | 0.544 |
| RMSE | 0.194 | 0.596 | 0.584 | 0.675 | 0.816 | 0.696 |

Data\_ All is a data table with a total of 11926 reactions for all CCSD data. The descriptor extraction method is the same as above, and corresponding training can be performed by replacing the table name.

1. Neural networks with different fingerprint dimensions and hidden layers

Train the neural network based on fingerprints extracted from different dimensions by Morganfingerprint.py. We need to modify the Reactants in the following two lines of code\_ 700 and Product\_ 700 will be replaced with fingerprints of different dimensions. The fingerprint data of different dimensions has been stored in the table and can be directly replaced.

# Processing Morgan Fingerprint for SMILES columns

smiles\_fingerprints = data['Reactant\_700'].apply(lambda x: [int(c) for c in x])

# Processing Morgan Fingerprint for psmi columns

psmi\_fingerprints = data['Product\_700'].apply(lambda x: [int(c) for c in x])

In addition, it is necessary to use different numbers of hidden layers and modify the code corresponding to the number of layers when defining the neural network.

# 2 hidden layers

model = Sequential()

model.add(Dense(500, activation='elu', kernel\_initializer='he\_normal', input\_dim=scaled\_features.shape[1]))

model.add(Dropout(0.05))

model.add(Dense(250, activation='elu', kernel\_initializer='he\_normal'))

model.add(Dropout(0.05))

model.add(Dense(1))

# 3 hidden layers

model = Sequential()

model.add(Dense(500, activation='elu', kernel\_initializer='he\_normal', input\_dim=scaled\_features.shape[1]))

model.add(Dropout(0.05))

model.add(Dense(250, activation='elu', kernel\_initializer='he\_normal'))

model.add(Dropout(0.05))

model.add(Dense(125, activation='elu', kernel\_initializer='he\_normal'))

model.add(Dropout(0.05))

model.add(Dense(1))

# 4 hidden layers

model = Sequential()

model.add(Dense(500, activation='elu', kernel\_initializer='he\_normal', input\_dim=scaled\_features.shape[1]))

model.add(Dropout(0.05))

model.add(Dense(250, activation='elu', kernel\_initializer='he\_normal'))

model.add(Dropout(0.05))

model.add(Dense(125, activation='elu', kernel\_initializer='he\_normal'))

model.add(Dropout(0.05))

model.add(Dense(62, activation='elu', kernel\_initializer='he\_normal'))

model.add(Dropout(0.05))

model.add(Dense(1))

# 5 hidden layers

model = Sequential()

model.add(Dense(500, activation='elu', kernel\_initializer='he\_normal', input\_dim=scaled\_features.shape[1]))

model.add(Dropout(0.05))

model.add(Dense(250, activation='elu', kernel\_initializer='he\_normal'))

model.add(Dropout(0.05))

model.add(Dense(125, activation='elu', kernel\_initializer='he\_normal'))

model.add(Dropout(0.05))

model.add(Dense(62, activation='elu', kernel\_initializer='he\_normal'))

model.add(Dropout(0.05))

model.add(Dense(31, activation='elu', kernel\_initializer='he\_normal'))

model.add(Dropout(0.05))

model.add(Dense(1))

# 6 hidden layers

model = Sequential()

model.add(Dense(500, activation='elu', kernel\_initializer='he\_normal', input\_dim=scaled\_features.shape[1]))

model.add(Dropout(0.05))

model.add(Dense(250, activation='elu', kernel\_initializer='he\_normal'))

model.add(Dropout(0.05))

model.add(Dense(125, activation='elu', kernel\_initializer='he\_normal'))

model.add(Dropout(0.05))

model.add(Dense(62, activation='elu', kernel\_initializer='he\_normal'))

model.add(Dropout(0.05))

model.add(Dense(31, activation='elu', kernel\_initializer='he\_normal'))

model.add(Dropout(0.05))

model.add(Dense(15, activation='elu', kernel\_initializer='he\_normal'))

model.add(Dropout(0.05))

model.add(Dense(1))

The optimal result is a neural network with 4 hidden layers and a fingerprint dimension of 300. The code corresponds to annv p.py. In addition, a curve graph of the loss of the neural network with the optimal result as the number of iterations increases was plotted, with code Loss1.py. Additionally, a line graph of the performance of the model with hidden layers at a fingerprint dimension of 300 was plotted, **layer1.py**。

5、Using the GBR model to obtain feature importance

Running code **feature2. py** can draw a ranking chart of feature importance

By modifying the top\_ The value of n=60 can determine the number of features displayed in the graph, modify top\_ N\_ Inset=20 can change the number of features displayed in the nested axis.