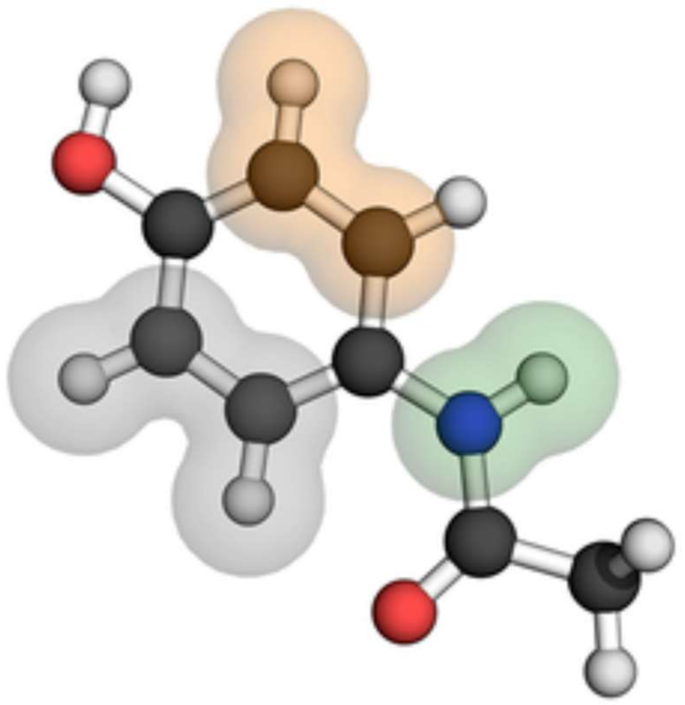


Predicting scalar coupling constant

Nori Kaneshige

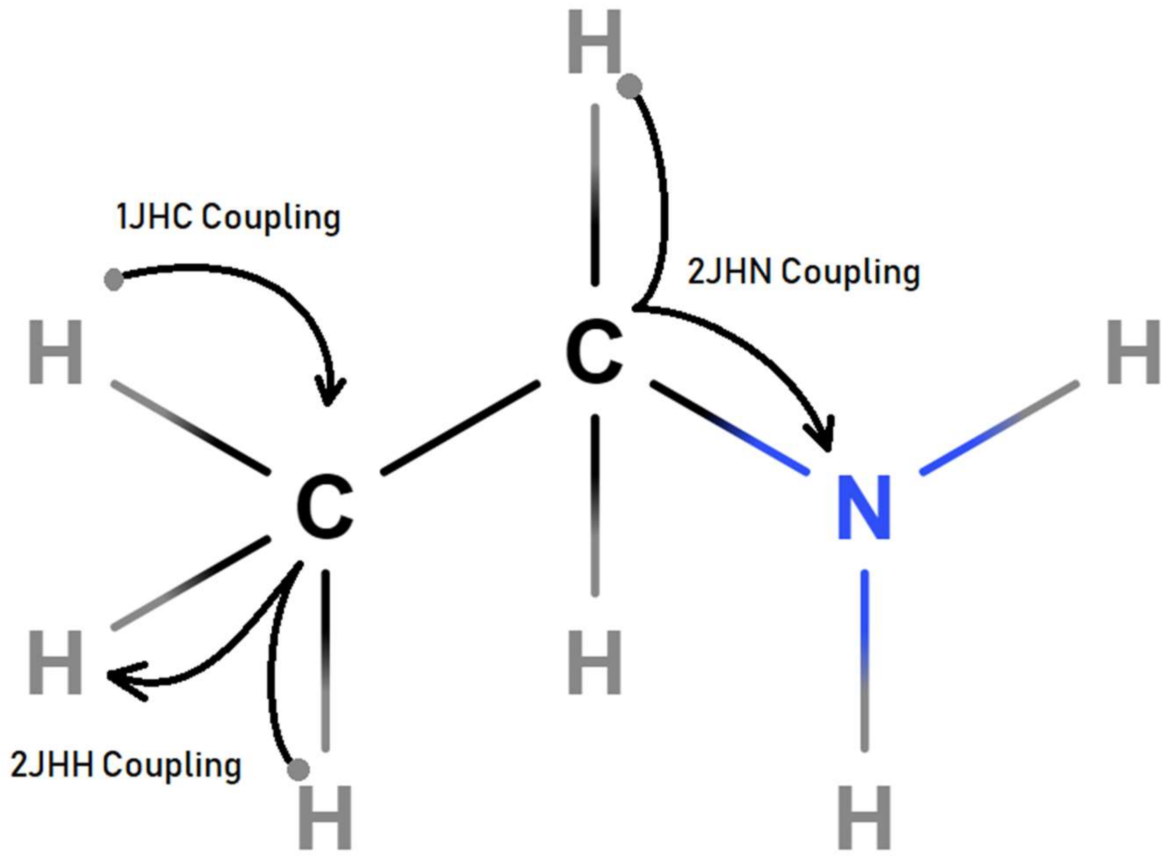
From Kaggle competition

https://www.kaggle.com/c/champs-scalar-coupling

Motivations

- Current quantum mechanics takes days or weeks for J coupling calculation
- We want to gain structure insights quicker and cheaper
- To get a prize from Kaggle host?

Scalar coupling constant



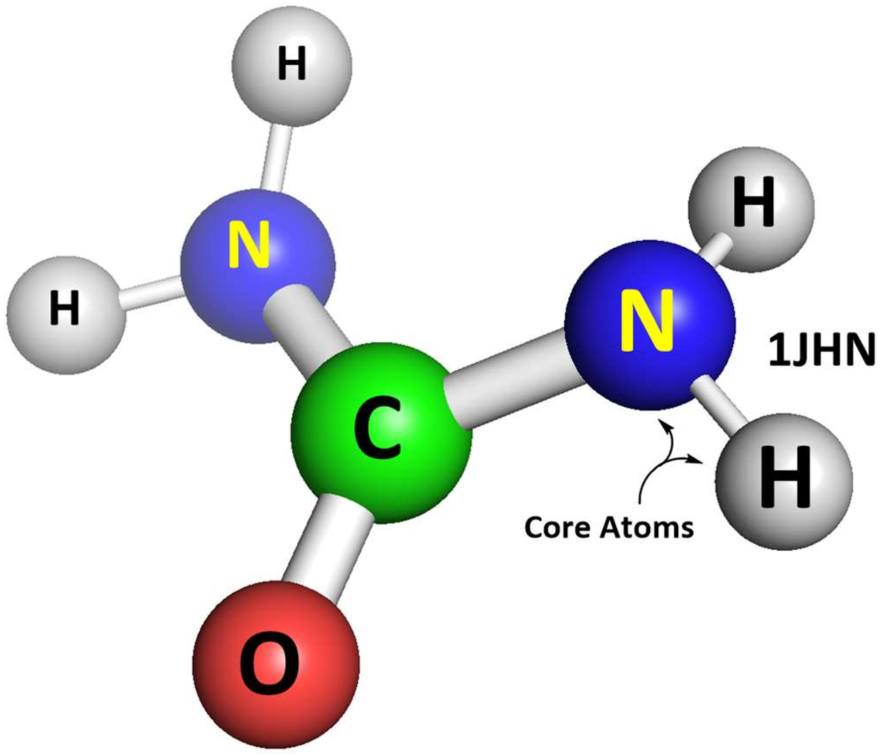
The magnitude of J

- Proximity
- Bond type
- Bond type

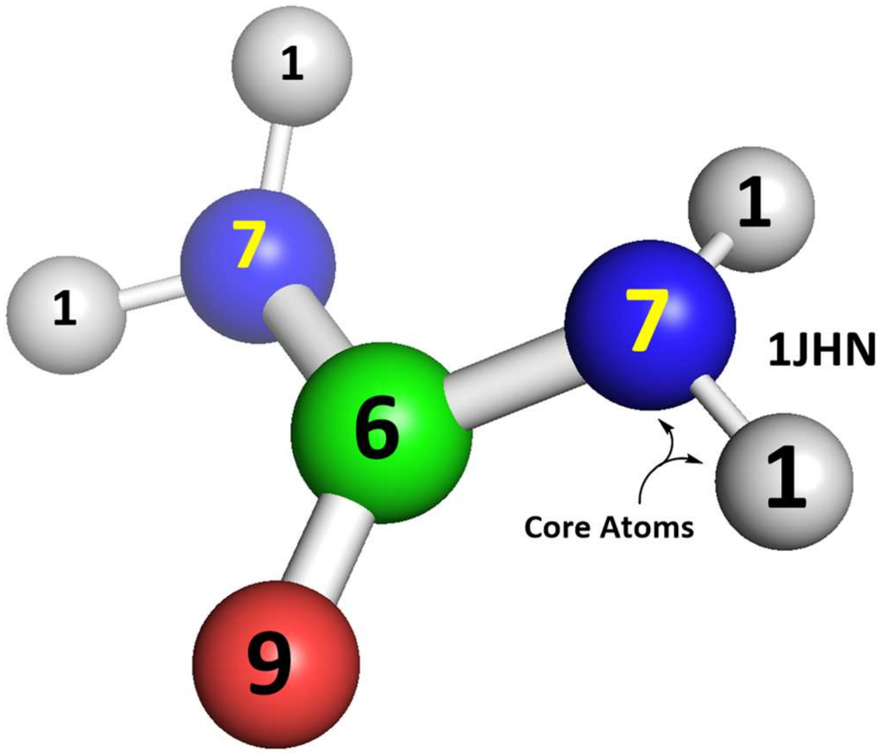
Data sets from Kaggle

- dipole_moments.csv
- magnetic_shielding_tensors.csv
- mulliken_charges.csv
- potential_energy.csv
- **structures.csv (3D coordinates of total 130,789 small molecules)**
- **train.csv**
- **test.csv**

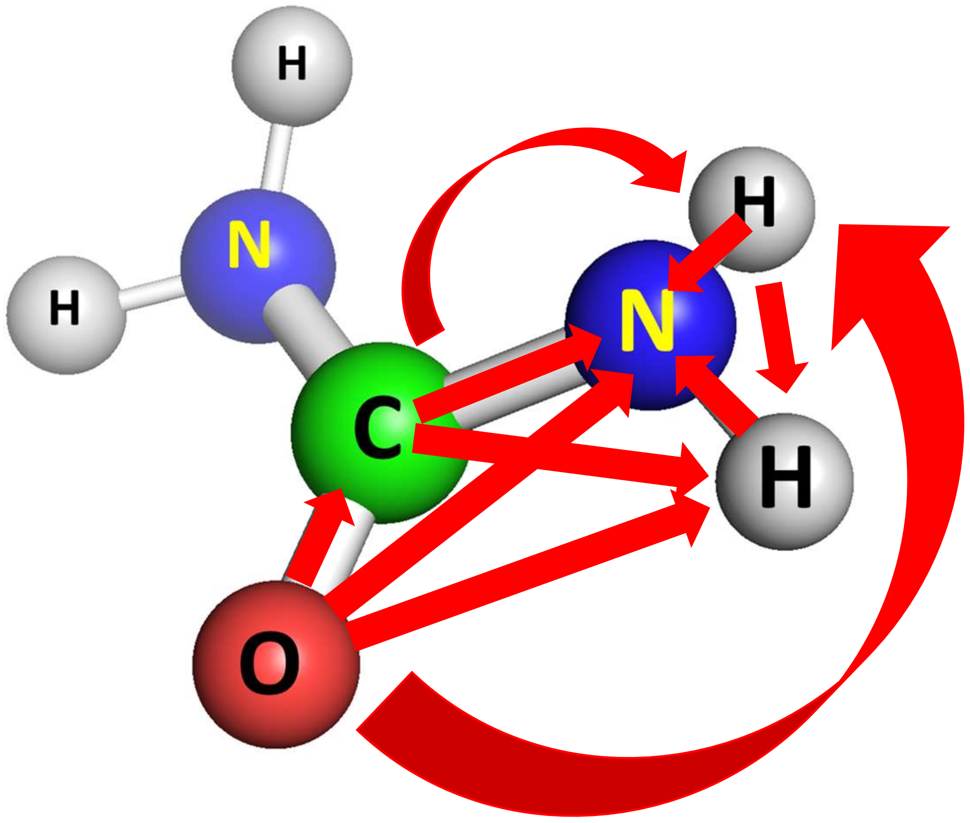
Approach to predict scalar coupling
Surrounding atoms as feature set

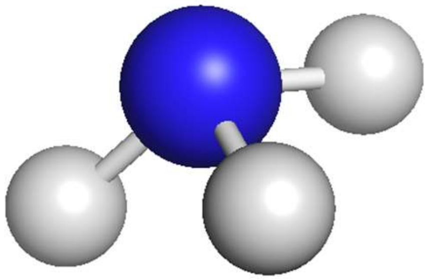


Approach to predict scalar coupling
Surrounding atoms as feature set



Approach to predict scalar coupling
Distances among atoms as feature set





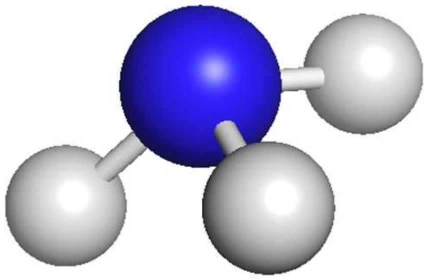
Data Before Process

train.csv:

```
    molecule_index  atom_index_0  atom_index_1  type  scalar_coupling_constant
id
10  2               1             0             1JHN   32.688900
11  2               1             2             2JHH -11.186600
12  2               1             3             2JHH -11.175700
13  2               2             0             1JHN   32.689098
14  2               2             3             2JHH -11.175800
15  2               3             0             1JHN   32.690498
```

structure.csv:

```
   molecule_index  atom_index atom         x         y         z
5  2               0             N  -0.040426  1.024108  0.062564
6  2               1             H   0.017257  0.012545 -0.027377
7  2               2             H   0.915789  1.358745 -0.028758
8  2               3             H  -0.520278  1.343532 -0.775543
```



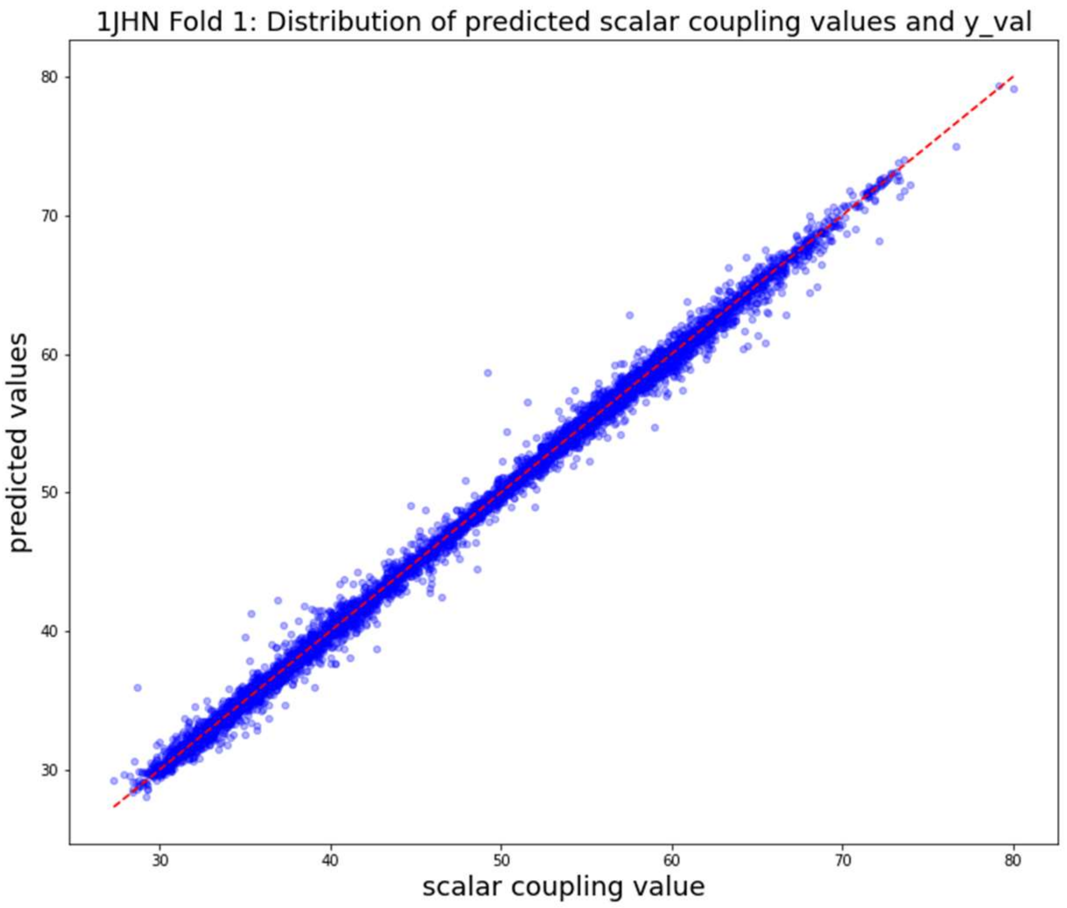
Data After Process

```
X_train:
[[1.        1.        0.        0.        0.        1.01719   1.6185228
  1.0171872 1.6187098 1.0172079 1.6187056 0.        0.        0.
  0.        0.        0.        0.        0.        0.        0.
  0.        0.       ]
 [1.        1.        0.        0.        0.        1.0171872 1.6185228
  1.01719   1.6187056 1.0172079 1.6187098 0.        0.        0.
  0.        0.        0.        0.        0.        0.        0.
  0.        0.       ]
 [1.        1.        0.        0.        0.        1.0172079 1.6187056
  1.0171872 1.6187098 1.01719   1.6185228 0.        0.        0.
  0.        0.        0.        0.        0.        0.        0.
  0.        0.       ]]


y_train:
[32.6889 32.6891 32.6905]
```

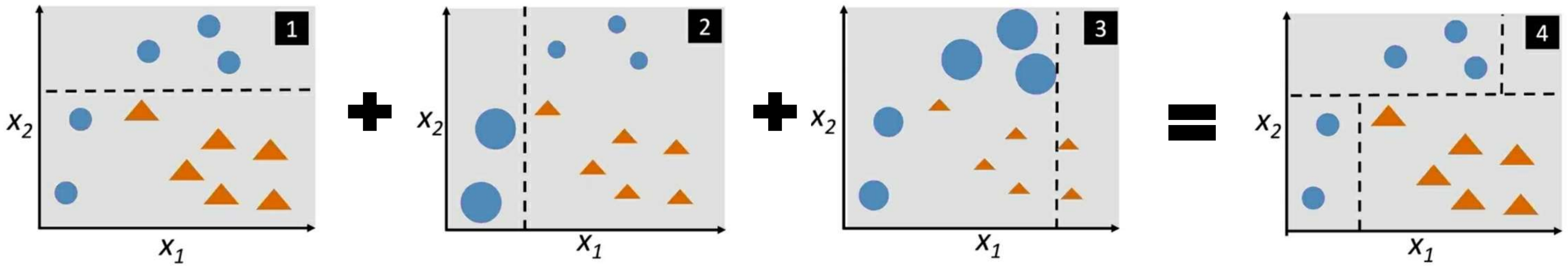
Experiment Learning Algorithms
With only 1JHN coupling type

Algorithm	Mean Absolute Error (log)
Lasso	1.338
Ridge	0.506
ElasticNet	1.898
KMeans	3.852
DBSCAN	4.378
MLPRegressor	-0.795
KerasRegressor	0.323
DecisionTreeRegressor	0.022
RandomForestRegressor	-0.761
GradientBoostingRegressor	-0.795
XGBRegressor	-0.969
LGBMRegressor	**-0.960**

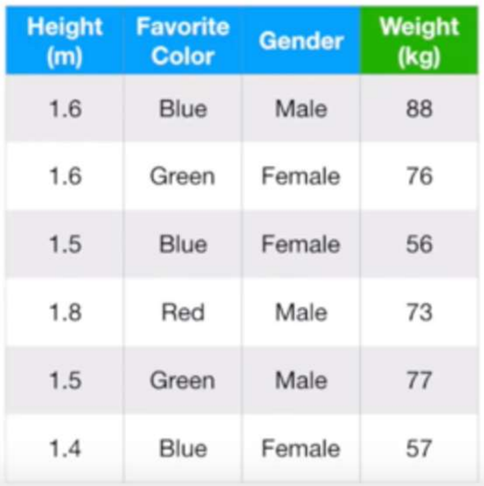


AdaBoost and Gradient Boosting

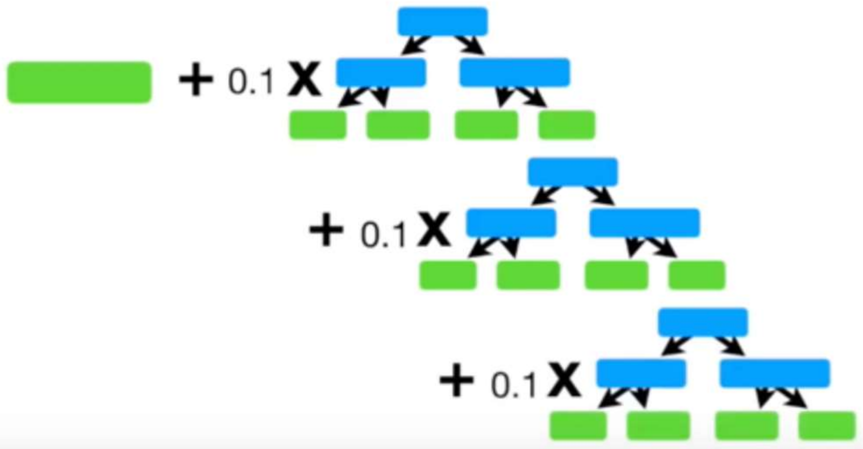
- **AdaBoost**



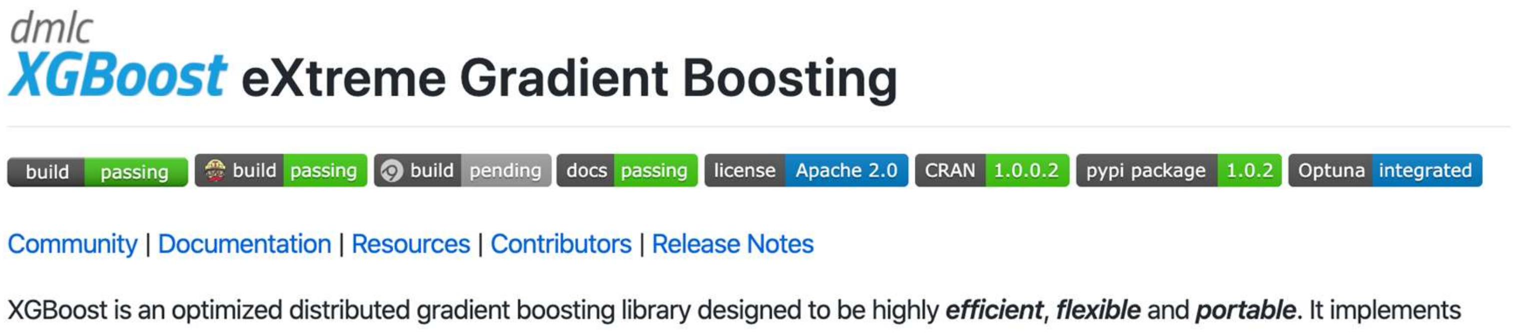
- **Gradient Boosting**



Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57



The most winning algorithm on Kaggle

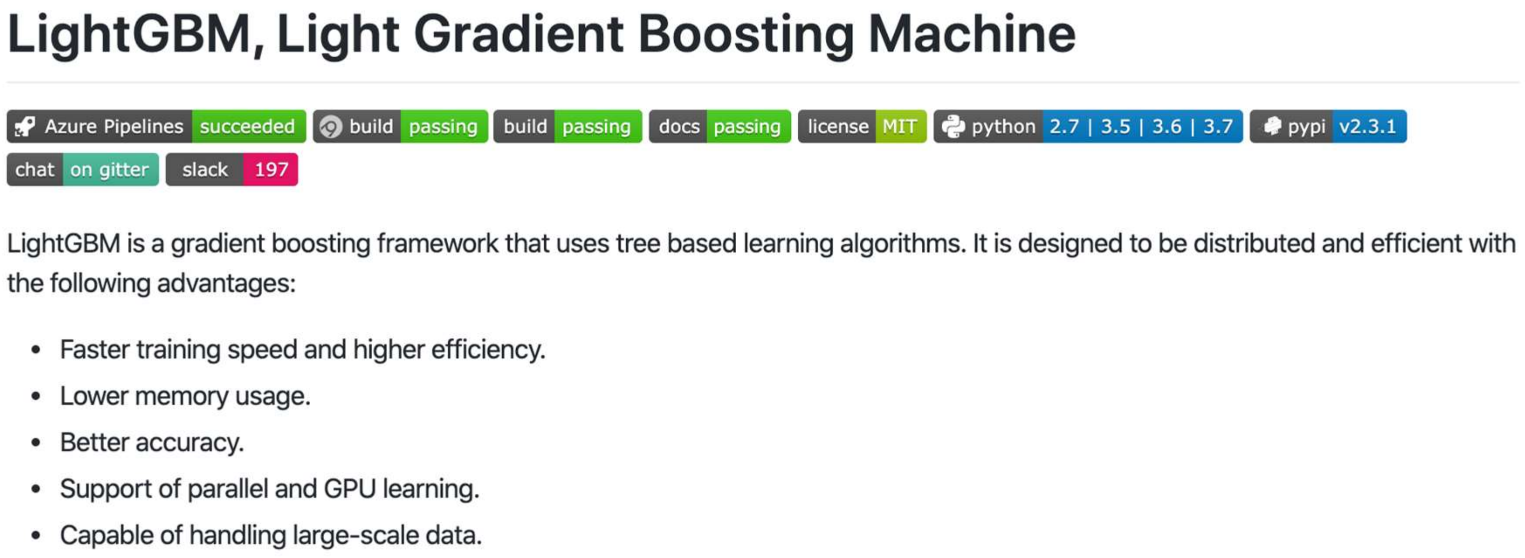


dmlc **XGBoost** eXtreme Gradient Boosting

build passing | build passing | build pending | docs passing | license Apache 2.0 | CRAN 1.0.0.2 | pypi package 1.0.2 | Optuna integrated

Community | Documentation | Resources | Contributors | Release Notes

XGBoost is an optimized distributed gradient boosting library designed to be highly ***efficient***, ***flexible*** and ***portable***. It implements



LightGBM, Light Gradient Boosting Machine

Azure Pipelines succeeded | build passing | build passing | docs passing | license MIT | python 2.7 | 3.5 | 3.6 | 3.7 | pypi v2.3.1 | chat on gitter | slack 197

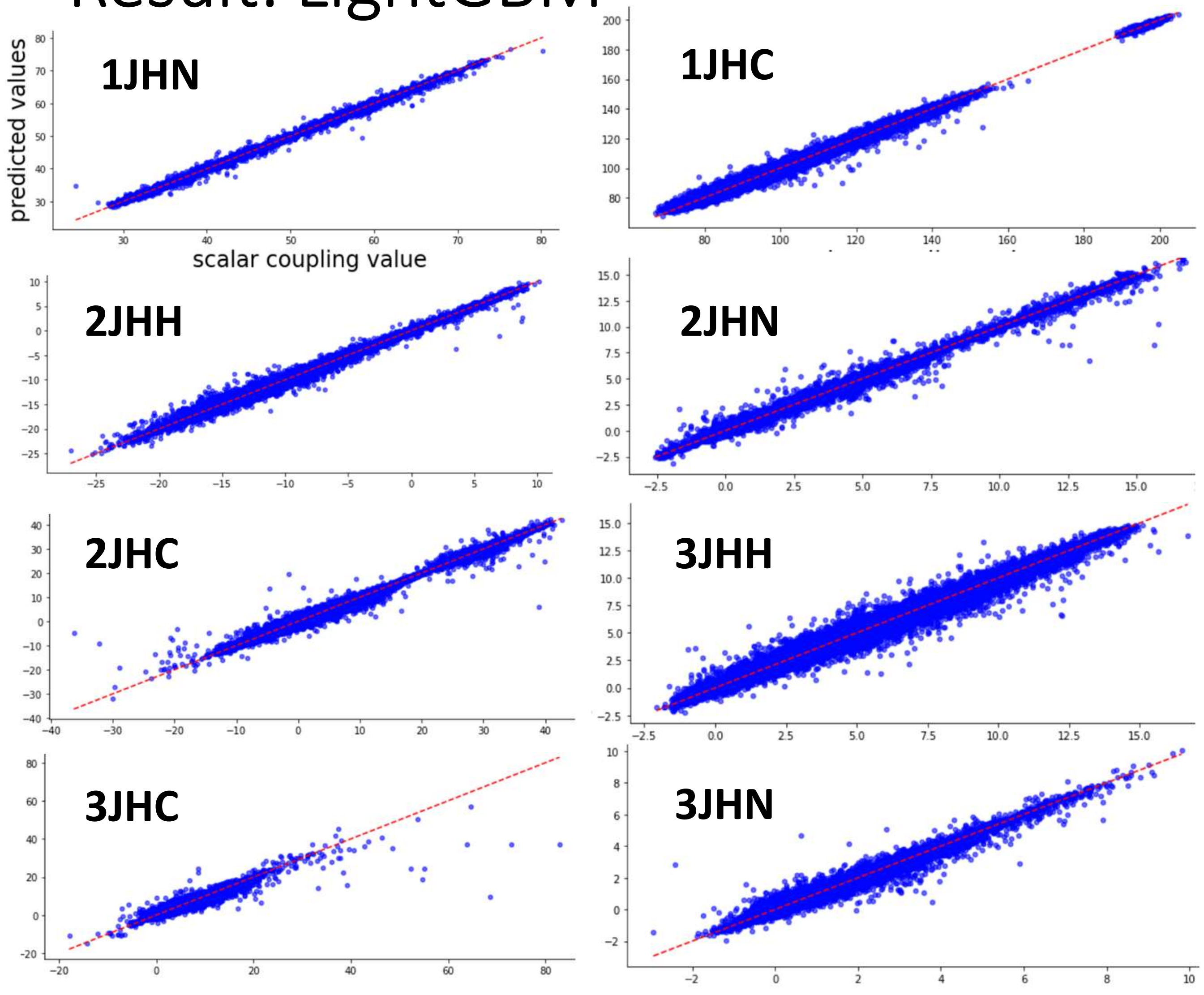
LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency.
- Lower memory usage.
- Better accuracy.
- Support of parallel and GPU learning.
- Capable of handling large-scale data.

Result: LightGBM

#Train_Set	#CV_split	#Feature
4,659,076	3	23

Parameters	settings
objective	'regression'
metric	'mae'
boosting_type	'gbdt'
learning_rate	**0.2**
num_leaves	128
max_depth	**9**
subsample_freq	1
subsample	0.9
reg_alpha	0.1
reg_lambda	0.3
colsample_bytree	1.0
n_estimaters	**1500**



Result: LightGBM

Coupling_Type	CV_Train_MAE	CV_Validation_MAE	Target_value_Mean	Target_value_STD
1JHN	0.119	0.385	47.528	10.925
1JHC	0.514	0.778	94.962	18.287
2JHH	0.092	0.180	-10.271	3.993
2JHN	0.056	0.147	3.109	3.667
2JHC	0.233	0.308	-0.276	10.550
3JHH	0.113	0.176	4.772	3.712
3JHC	0.275	0.340	3.687	3.076
3JHN	0.042	0.113	0.994	1.319

MAE: Mean absolute error
STD: Standard deviation

Potential Improvement

- Tune LGB hyperparameters
- Tune number of atoms for each type (Include more surrounding atoms)
- Try to add other features
- Try to add categorical features for atom types (one-hot-endocing etc)
- Try other tree libraries
- Consider angles among bonds