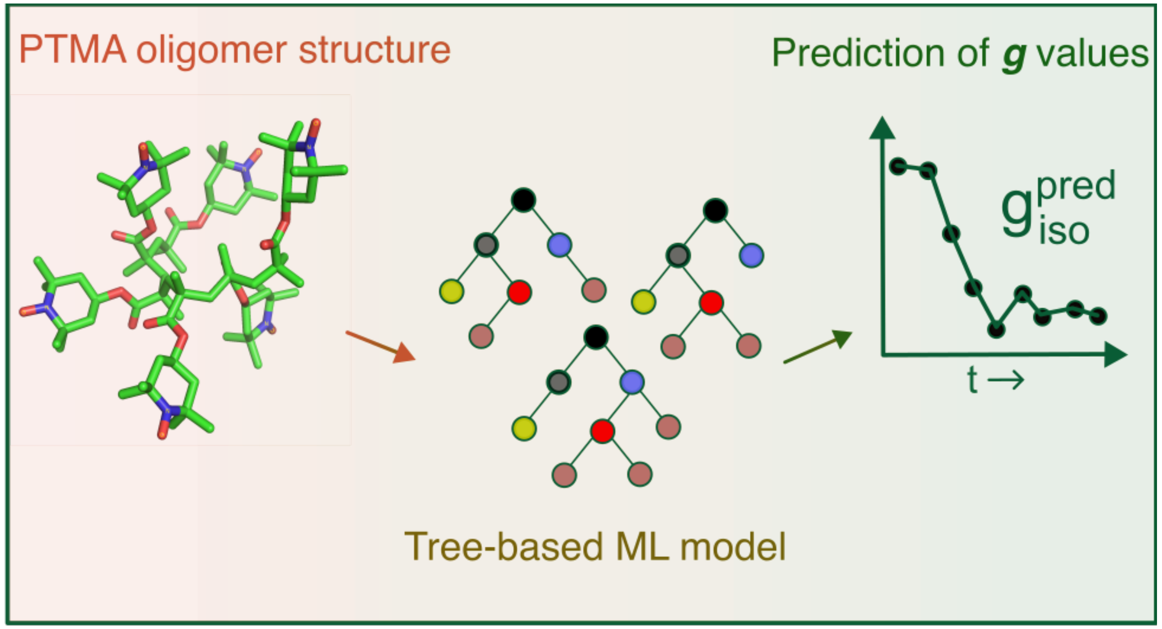
Machine learning isotropic g values of radical polymers

Davis Thomas Daniel[1,2], Souvik Mitra[3], Diddo Diddens[3,4], Rüdiger A. Eichel[1,4] and Josef Granwehr[1,2]

1. Institute of Energy and Climate Research (IEK-9), Forschungszentrum Jülich GmbH, Jülich, 52425, Germany
2. Institute of Technical and Macromolecular Chemistry, RWTH Aachen University, Aachen, 52056, Germany
3. Institute of Physical Chemistry, University of Münster, 48149, Münster, Germany
4. Institute of Physical Chemistry, RWTH Aachen University, Aachen 52056, Germany
5. Helmholtz-Institute Münster (IEK-12), Forschungszentrum Jülich GmbH, Jülich, Münster, 48149, Germany, 52425, Germany



Email : d.daniel@fz-juelich.de

Python libraries :

- numpy : https://numpy.org/install/
- sklearn : https://scikit-learn.org/stable/install.html (v. 1.22 was used in this work.)
- dscribe : https://singroup.github.io/dscribe/latest/install.html (v. 1.2.2 was used in this work.)
- ase : https://wiki.fysik.dtu.dk/ase/install.html (v. 3.22.1 was used in this work.)

1. Import required modules

In [24]:
```
# please install numpy and ase.

import numpy as np
import pickle

# custom prediction functions for directly predicting from xyz files. Also imports ase, so ase should be installed.
from Scripts.prediction_functions import *


## only needed if model is retrained :

#from sklearn.ensemble import ExtraTreesRegressor
#from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

2. Load Feature vectors

(a) Transform XYZ coordinates into feature vectors

Feature vectors are generated from atomic coordinates by transforming them using a molecular descriptor.

Exemplary code to achieve this for MBTR and DAD is given below :

```python
from ase.io import read
from dscribe.descriptors import MBTR
from Scripts.prediction_functions import generate_DAD



xyz = 'structure.xyz' # Path to xyz file
structure = read(xyz) #  Make an atoms object using ASE

ptma_chemical_symbols = set()  # Initiate a set
ptma_chemical_symbols.update(structure.get_chemical_symbols()) # Get chemical symbols in PTMA


# define the MBTR configuration
# MBTR object used in the manuscript is shown below.

mbtr = MBTR(
    species=ptma_chemical_symbols,
    k1 = {'geometry': {'function': 'atomic_number'},
          'grid': {'min': 0, 'max': 10, 'sigma': 0.01, 'n': 10}},
    k2 = {'geometry': {'function': 'inverse_distance'},
          'grid': {'min': 0.04, 'max': 2, 'sigma': 0.1, 'n': 50},
          'weighting': {'function': 'exp', 'r_cut': 10, 'threshold': 0.001}},
    k3 = {'geometry': {'function': 'angle'},
          'grid': {'min': 0, 'max': 180, 'sigma': 4, 'n': 180},
          'weighting': {'function': 'exp', 'r_cut': 5, 'threshold': 0.01}},
    periodic=False,
    flatten=True)

mbtr_output = mbtr.create(structure)

# for DAD

dad_output = generate_DAD(xyz)
```

(b) Loading pre-transformed Training dataset

Two arrays which make up the $\mathbf{TR}$ dataset are loaded in the next step.

- X_train : Contains 150 PTMA structures used for training. (See Dataset generation in main text.)
- Y_train : Contains corresponding g_{iso} values.

In [25]:
```
X_train = np.load('Datasets/TR/train_structure_data_MBTR.npy')  # For DAD use : np.load('Datasets/TR/train_structure_data_DAD.npy')
Y_train = np.load('Datasets/TR/train_giso_DFT_data.npy')
```

3. Model

(a) Hyperparameters Optimisation

Hyperparameters were optmised using a gridsearch. Exemplary code is shown below.

```
from sklearn.model_selection import GridSearchCV

# define the grid
param_grid = {
    "n_estimators": list(range(10,110)),
    "max_features": [None, "sqrt", "log2"],
    "min_samples_split": [2,4, 6],
    "min_samples_leaf": [2, 3, 4],
    "max_depth": [None]+list(range(10,40)),
    "max_leaf_nodes": [None]+list(range(2,21)),
    "bootstrap" : [True,False]
}


# run the gridsearch
gcv = GridSearchCV(estimator=ExtraTreesRegressor(random_state=1,n_jobs=-1),
                   cv=5, param_grid=param_grid,
                   verbose=1,n_jobs=-1,
                   scoring=['r2','neg_mean_absolute_error','neg_root_mean_squared_error'],
                   refit='neg_root_mean_squared_error',
                   error_score='raise')
```

(b) Training

Exemplary code for training an ExtraTreesRegressor using the optimised hyperparameters.

Notes :

- n_jobs is set to -1. This means all processors on the system will be used.
- verbose is set to 1. Training progress will be printed out.

Note : If hyperparameter needs to be changed or a new model needs to be trained, this can be done as shown below.

```
hyperparameters = {'bootstrap': False,
                   'ccp_alpha': 0.0,
                   'criterion': 'squared_error',
                   'max_depth': 10,
                   'max_features': None,
                   'max_leaf_nodes': None,
                   'max_samples': None,
                   'min_impurity_decrease': 0.0,
                   'min_samples_leaf': 2,
                   'min_samples_split': 2,
                   'min_weight_fraction_leaf': 0.0,
                   'n_estimators': 100,
                   'n_jobs': -1,
                   'oob_score': False,
                   'random_state': 1,
                   'verbose': 1,
                   'warm_start': False}

model = ExtraTreesRegressor(**hyperparameters).fit(X_train,Y_train)
```

(c) Cross-validation

For cross-validation using the training (**TR**) data set :

```
from sklearn.model_selection import cross_val_score

n_scoresr2 = cross_val_score(model, X_train, Y_train, scoring='r2', cv=5, n_jobs=-1)
n_scores_MAE = cross_val_score(model, X_train, Y_train, scoring='neg_mean_absolute_error', cv=5, n_jobs=-1)
n_scores_RMSE = cross_val_score(model, X_train, Y_train, scoring='neg_root_mean_squared_error', cv=5, n_jobs=-1)


print("%0.3f R2 with a standard deviation of %0.3f" % (n_scoresr2.mean(), n_scoresr2.std()))
print("%0.6f MAE with a standard deviation of %0.6f" % (n_scores_MAE.mean()*-1, n_scores_MAE.std()))
print("%0.6f RMSE with a standard deviation of %0.6f" % (n_scores_RMSE.mean()*-1, n_scores_RMSE.std()))
```

(d) Load the pre-trained model

The pre-trained model is saved as a python pickle object. To load :

```
In [27]: model_MBTR = pickle.load(open('Models/MBTR_ERT_model.pkl','rb'))

         model_DAD = pickle.load(open('Models/DAD_ERT_model.pkl','rb'))

         # For SOAP use :
         #pickle.load(open('Models/SOAP_ERT_MODEL.pkl','rb'))
```

Model parameters should match the hyperparameters shown above.

```
In [28]: model_MBTR.get_params()
```

```
Out[28]: {'bootstrap': False,
          'ccp_alpha': 0.0,
          'criterion': 'squared_error',
          'max_depth': 10,
          'max_features': None,
          'max_leaf_nodes': None,
          'max_samples': None,
          'min_impurity_decrease': 0.0,
          'min_samples_leaf': 2,
          'min_samples_split': 2,
          'min_weight_fraction_leaf': 0.0,
          'n_estimators': 100,
          'n_jobs': -1,
          'oob_score': False,
          'random_state': 1,
          'verbose': 1,
          'warm_start': False}
```

4. Predictions

Exemplary prediction code for MBTR is shown for TE-1 dataset

TE-1 structures are randomly selected from WSD. (See manuscript for more details)

(a) TE-1 dataset ERT-MBTR

```
In [29]: X_test_te1 = np.load('Datasets/TE-1/MBTR_TE1_structure_data.npy')  ## structure data stored in MBTR representation
         Y_test_te1 = np.load('Datasets/TE-1/TE1_giso_DFT_data.npy')    ## g_iso values calculated using DFT
```

```
In [30]: predictions_te1 = model_MBTR.predict(X_test_te1) ## g_iso values predicted
```

```
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.0s finished
```

In [31]:
```python
#```python
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
print('*** TE-1 error metrics ***\n')
print('R2 : {:.3f}'.format(r2_score(Y_test_te1,predictions_te1)))
print('MAE : {:.6f}'.format(mean_absolute_error(Y_test_te1,predictions_te1)))
print('RMSE : {:.6f}'.format(mean_squared_error(Y_test_te1,predictions_te1,squared=False)))

print('\n*******')
#```
```

```
*** TE-1 error metrics ***

R2 : 0.989
MAE : 0.000097
RMSE : 0.000127

*******
```

(b) TE-1 dataset ERT-DAD

In [32]:
```python
X_test_te1 = np.load('Datasets/TE-1/DAD_TE1_structure_data.npy') ## structure data stored in MBTR representation
Y_test_te1 = np.load('Datasets/TE-1/TE1_giso_DFT_data.npy')   ## g_iso values calculated using DFT
```

In [33]:
```python
predictions_te1 = model_DAD.predict(X_test_te1) ## g_iso values predicted
```

```
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.0s finished
```

In [34]:
```python
#```python
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
print('*** TE-1 error metrics ***\n')
print('R2 : {:.3f}'.format(r2_score(Y_test_te1,predictions_te1)))
print('MAE : {:.6f}'.format(mean_absolute_error(Y_test_te1,predictions_te1)))
print('RMSE : {:.6f}'.format(mean_squared_error(Y_test_te1,predictions_te1,squared=False)))

print('\n*******')
#```
```

```
*** TE-1 error metrics ***

R2 : 0.979
MAE : 0.000128
RMSE : 0.000172

*******
```

(c) Predict from xyz files

Predictions can also be done directly from xyz files.

In the following, structures from TE-2 are used for predictions.

For MBTR

In [35]:
```
model_MBTR = pickle.load(open('Models/MBTR_ERT_model.pkl','rb'))  ## Load MBTR trained model
```

In [36]:
```
predicted = predict_MBTR(model_MBTR,'XYZ_files/TE-2/PTMA-1/chainlength_6ptma_16_4.xyz') # predict_MBTR function can be found in Scrip
DFT_calculated = 2.0051969


print('DFT calcualted g = ',DFT_calculated)
print('ML predicted g = ',predicted)
print('Difference  = ',DFT_calculated-predicted)
```

```
DFT calcualted g =  2.0051969
ML predicted g =  [2.00516016]
Difference  =  [3.67408333e-05]
```

```
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.0s finished
```

For DAD

In [37]:
```
model_DAD = pickle.load(open('Models/DAD_ERT_model.pkl','rb'))  ## Load DAD trained model
```

In [38]:
```
predicted = predict_DAD(model_DAD,'XYZ_files/TE-2/PTMA-1/chainlength_6ptma_16_4.xyz') # predict_DAD function can be found in Scripts/
DFT_calculated = 2.0051969


print('DFT calcualted g = ',DFT_calculated)
print('ML predicted g = ',predicted)
print('Difference  = ',DFT_calculated-predicted)
```

```
DFT calcualted g =  2.0051969
ML predicted g =  [2.00541722]
Difference  =  [-0.00022032]
```

```
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.0s finished
```

XYZ files for TE-2 test data set for each radical density are provided under the folder XYZ_files/TE-2/

(d) TE-2 dataset ERT-MBTR

TE-2 predictions from the manuscript are shown below. As an example, PTMA-1 structures are used. Other radical densities can be selected by changing the files names specified in X_test_te2 and Y_test_te2 variables. All required files can be found in the folder *Datasets/TE-2/*.

In [39]:
```python
X_test_te2 = np.load('Datasets/TE-2/PTMA_1_structure_data.npy')  ## structure data stored in MBTR representation
Y_test_te2 = np.load('Datasets/TE-2/PTMA_1_giso_DFT_data.npy')   ## g_iso values calculated using DFT
```

In [40]:
```python
predictions_te2 = model_MBTR.predict(X_test_te2) ## g_iso values predicted
```

```
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.0s finished
```

In [41]:
```python
#```python
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
print('*** TE-2 error metrics ***\n')
print('R2 : {:.4f}'.format(r2_score(Y_test_te2,predictions_te2)))
print('MAE : {:.6f}'.format(mean_absolute_error(Y_test_te2,predictions_te2)))
print('RMSE : {:.6f}'.format(mean_squared_error(Y_test_te2,predictions_te2,squared=False)))

print('\n*******')
#```
```

```
*** TE-2 error metrics ***

R2 : 0.9922
MAE : 0.000095
RMSE : 0.000114

*******
```