DIRECT CAPSTONE PROJECT SUMMARY
JUNE 2019
RYAN BECK, LAUREN KOULIAS, LINNETTE TEO

DopeDefects: Predicting impurity energy levels of semiconductors using machine learning

https://github.com/dopedefects/dopedefects/

DopeDefects is an open source python package licensed under the MIT license. It contains several features for extracting and predicting information.

When provided with information arranged as POSCARs and .csv files the init_data function is able to read in information and save it all to a .hdf5 file. The ability to save the information between runs allows for the user to not have to run the data collection and cleaning processes each time the package is started; and allows for consistency and ease of communication between collaborators.

The package is also able to read in the position of atoms from VASP (Vienna *Ab initio* Simulation Package) and determine the defect type and the defect location based off the file name and the provided structure. The package is able to calculate the change in the bond lengths and angles for the atoms surrounding the defect in comparison to the pure system, and is able to calculate the Coulomb matrix for the unit cell system.

In order to call the data extraction and dataframe initialization function execute: import data_build; df = data_build.init_data('NAME_FOR_HDF5_FILE', data_dir='DATA_DIRECTORY', csvs=csvs). Which will return a Pandas dataframe object containing the information from the .csv files specified in the csvs list and the information determined from the POSCAR files located in the DATA_DIRECTORY. Note that if the .hdf5 file already exists the package will not rebuild it, and The package expects the VASP POSCAR files to be located in the given data directory, and anticipates them having the general structure of Crystal type (CdS, CdSe_0.5S0.5, CdS, etc.) / Defect type (M_Cd, M_i_Cd_site, etc.) / Doping atom type (Ag, B, etc.).

The package is also able to append atomic properties not originally specified in the csvs object. The package will match the atomic type of the properties and append them to the proper row(s) in the original pandas dataframe corresponding to a dopant of that type. To do so, specify the file_in flag to the init_data function (Default is a file named Elemental_properties.csv located within the main project folder).

Neural Network

The neural networks were trained on the same descriptors used in the published paper (base descriptors + elemental properties + unit cell defect properties; 22 in total).

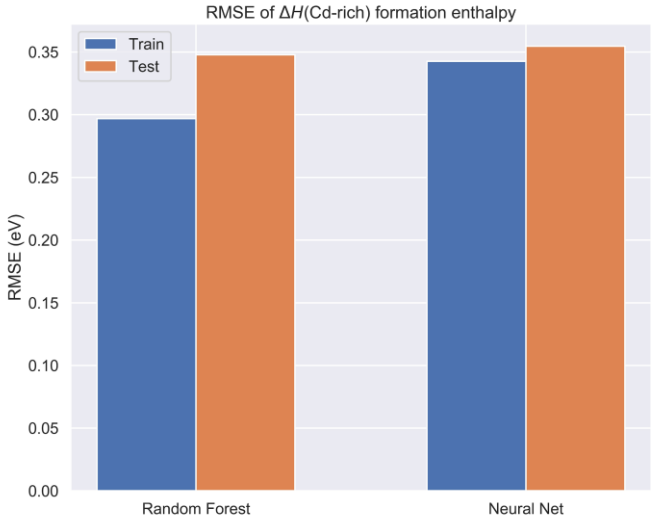
All 425 known data points were used.

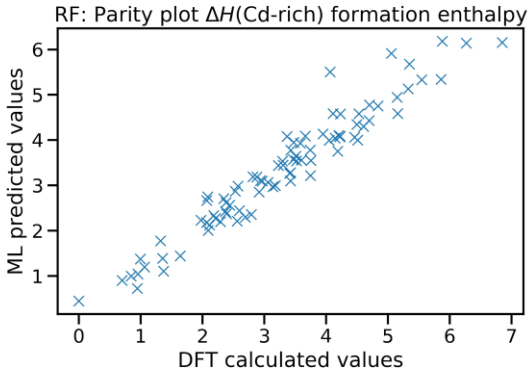
We split the data 6:2:2 for training, validation, and testing.

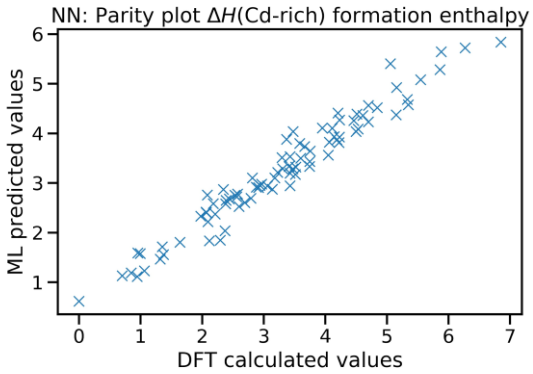
We did a grid search for hyperparameter tuning and explored 3 values for each of the 6 hyperparameters - number of epochs (250, 500, 700), batch size (50, 100, 400), number of nodes in hidden layers (8, 16, 32 - there are two hidden layers), dropout rate (0.05, 0.1, 0.25), and learning rate (0.001, 0.01, 0.1). This gives 3^6 = 729 separate models. The optimizer used is Adam. We used keras with tensorflow backend to build the neural network models. The neural network with the lowest validation RMSE is taken to be the best set of hyperparameters and we used these hyperparameters to train the final model. Other ways to do this that might show an improvement in optimization would be taking an average of the top 30 models for example and explore the hyperparameter space in greater detail, or do more sophisticated gradient descent optimization methods

For each of the 9 values we are trying to predict, we optimized and trained separate neural networks. Here we show results for a few of the predicted values.

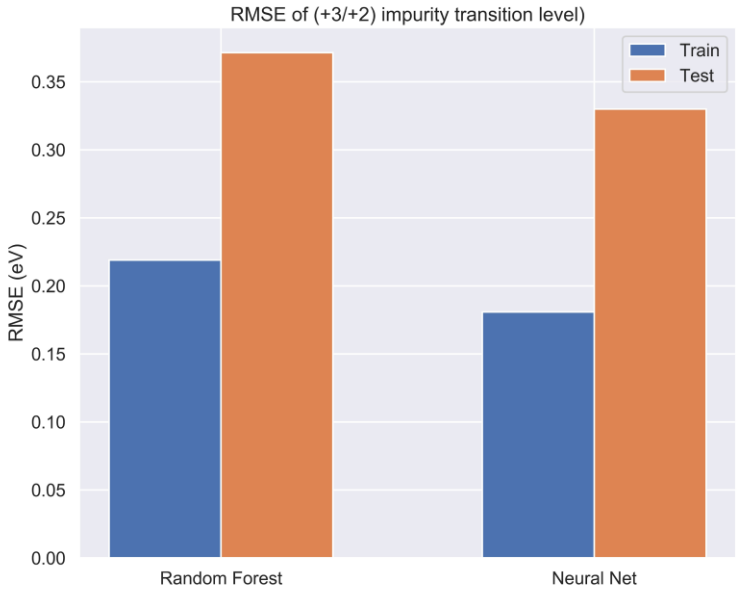
Formation enthalpy (Cd-rich)

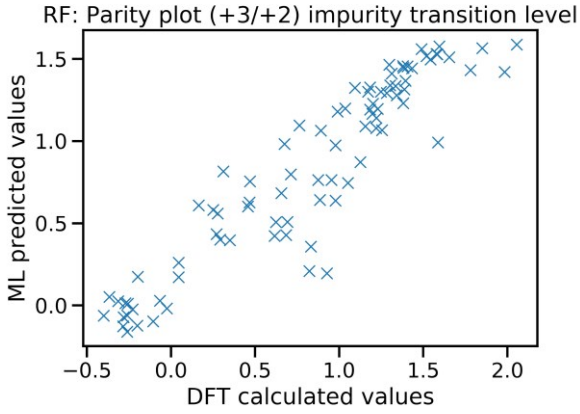


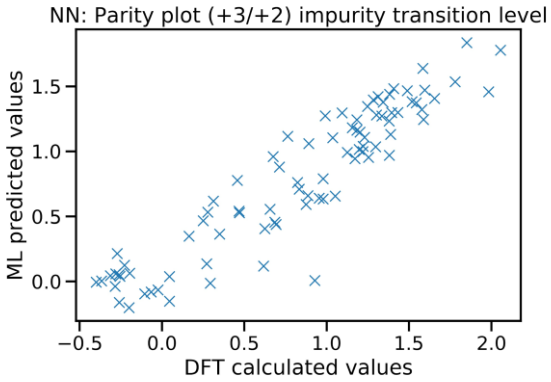




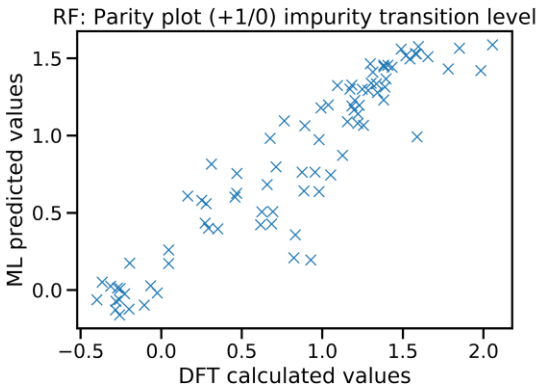
(+3/+2) impurity transition level

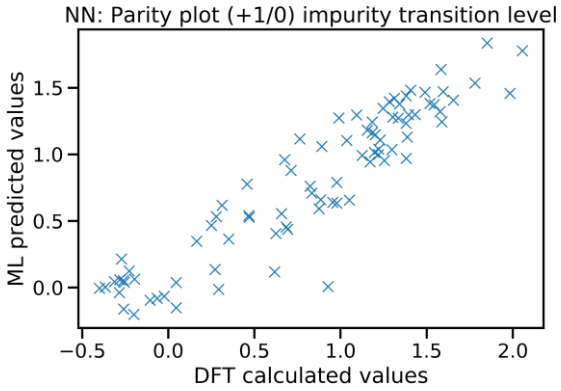




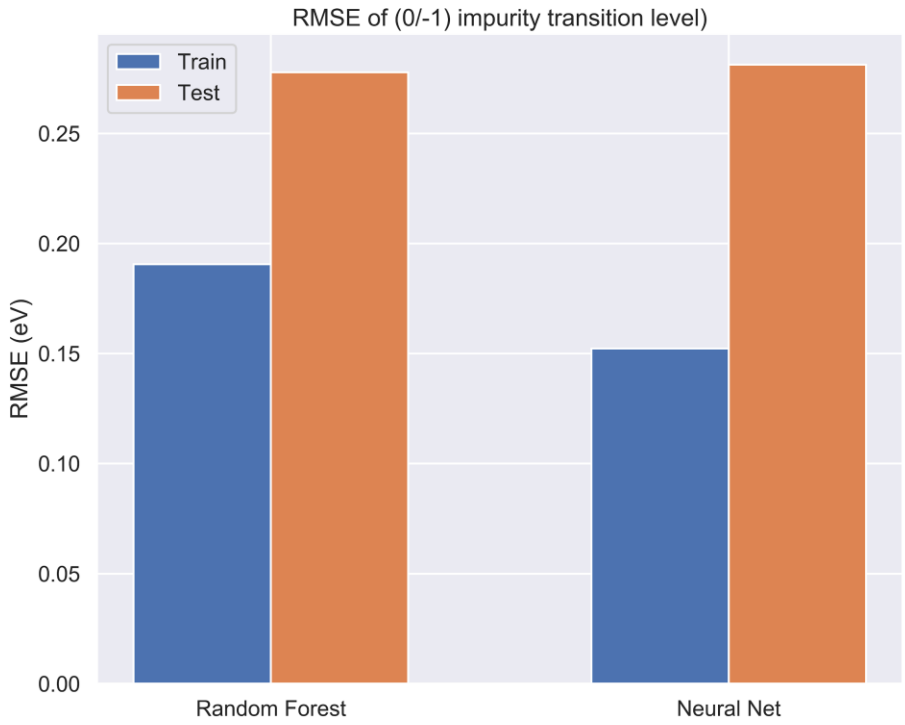


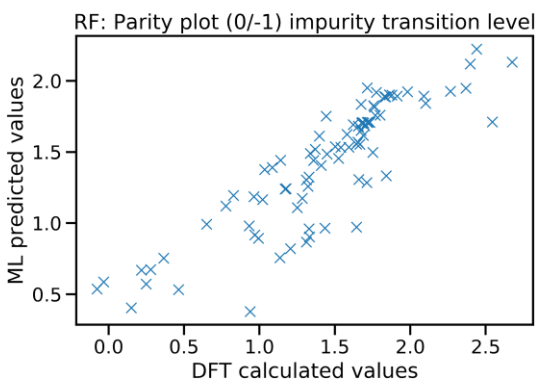
(+1/0) impurity transition level

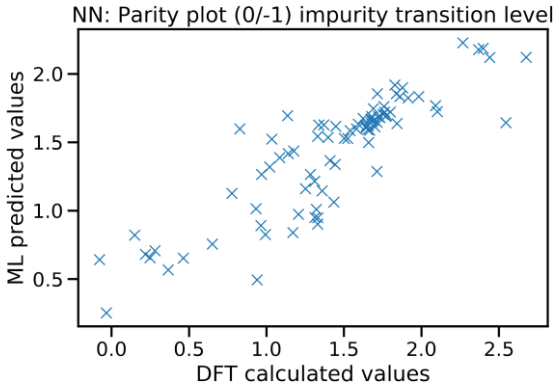




(0/-1) impurity transition level







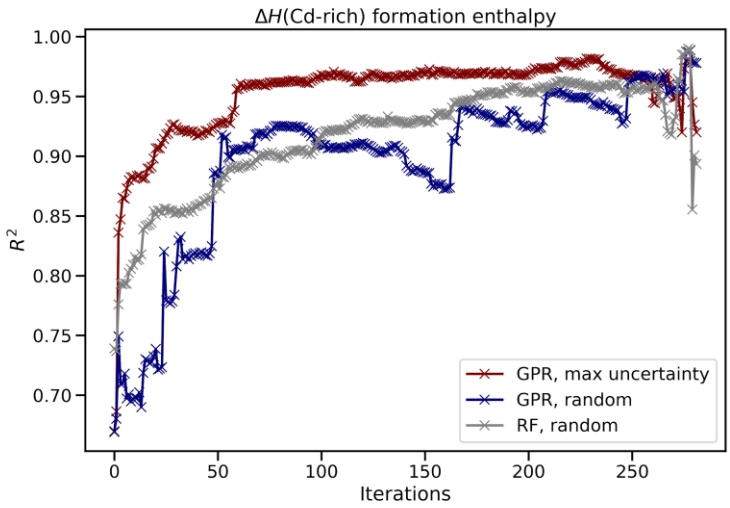
Neural networks do not perform significantly better than random forest - need further optimization, and more training data. Random forests shown above can also be further optimized to prevent overfitting.

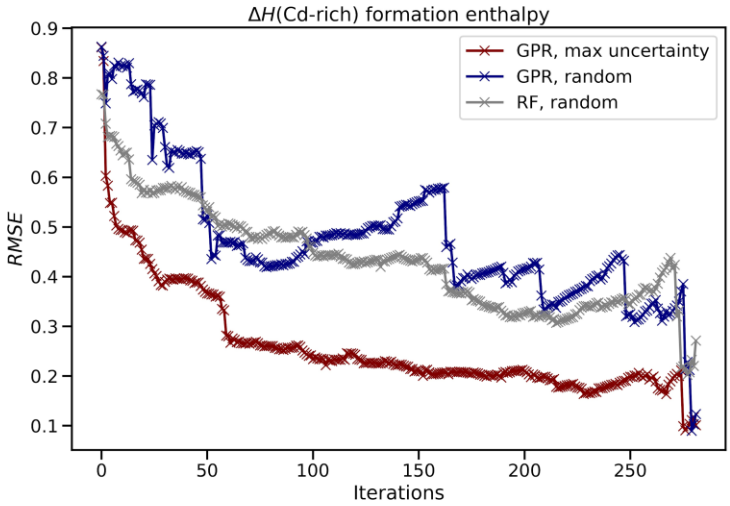
Iterative Method using Gaussian Process Regression

- Used CdTe structures only (315 data points)
- Start with small subset of data (10%) as training data
- Use GPR (from sklearn) to predict mean and uncertainty (standard deviation) on remaining test points
- Choose a test point that maximizes uncertainty
- Add test point (with calculated value) to model and retrain
- Iterate - keep adding points until satisfied

The iterative method using GPR is shown in red below for Cd-rich formation enthalpy. The line in blue is also using GPR model but each successive point is chosen randomly rather than to maximize uncertainty. The line is grey is using a random forest regressor with each successive point also chosen randomly.

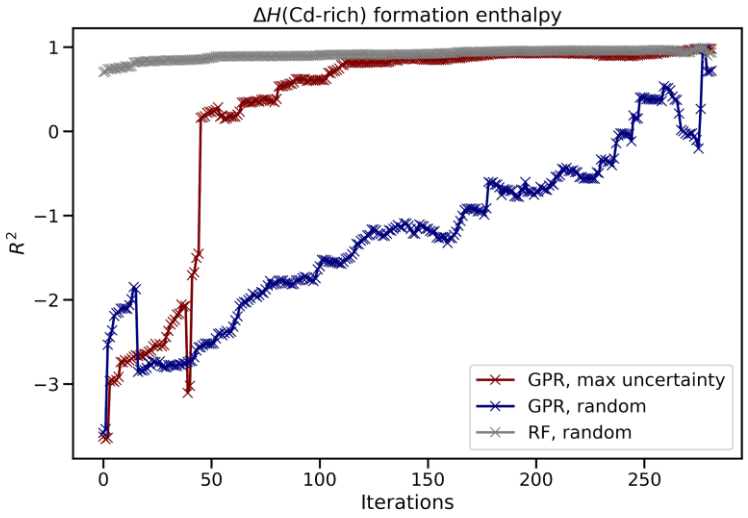
Descriptors are similar to those used in published paper -- base + unit cell defect properties.

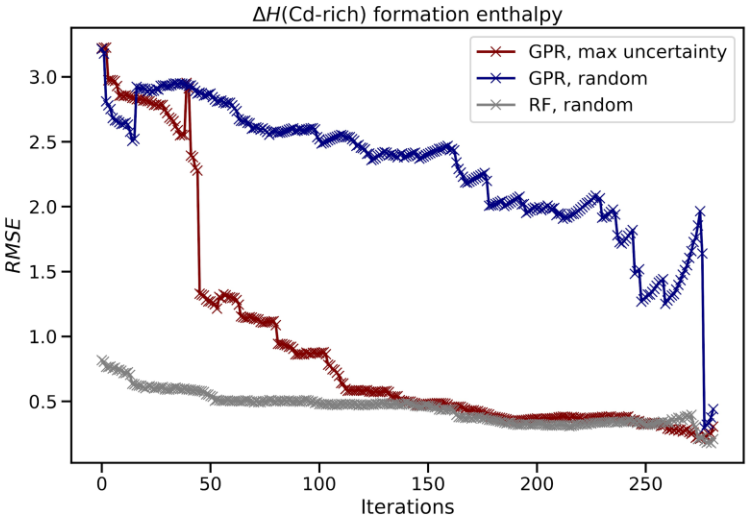




Maximizing uncertainty using GPR vs random search helps reduce initial number of known points needed.

We also tried adding more descriptors -- base + unit cell defect + elemental properties. This greatly worsened the effectiveness of the GPR model. R-squared (coefficient of determination) values can be negative here because it is defined as 1-(sum of squares of residuals/total sum of squares). R-squared values being negative means that the model is worse than simply predicting the mean. This is probably because R-squared value is for the entire data set, not the subset of data that the model is trained on.





All code used to train models are uploaded on github:

data_preprocess.py - organize data from dataframe generated into inputs/descriptors and outputs/predictors for models; includes functions to scale and unscale X and y values

Nn.py - code used to train neural net model. Also contains code for hyperparameter tuning

Iterative.py - code used for iterative method - GPR and RF

dopedefects/doc/fig folder contains all plots
dopedefects/doc/hyperparamter_runs folder contains all npy files for hyperparameter tuning

dopedefects/doc/notebooks folder contains plotting notebooks for examples on using the above .py scripts and generating plots

Feature Selection and Random Forest

The descriptors were divided into the following categories:

- **Elemental**: Period, Group, Site, Ionic_radius, Boiling_point, Melting_point, Density, Atomic_weight, ICSD_volume, Cov_radius, Atomic_radius, Electron_affinity, Atomic_vol, Mendeleev_number, Ionization_pot_1, Ionization_pot_2, Ionization_pot_3, Therm_expn_coef', Sp_heat_cap, Therm_cond, Heat_of_fusion, Heat_of_vap, Electronegativity, At_num, Valence, and Ox_state

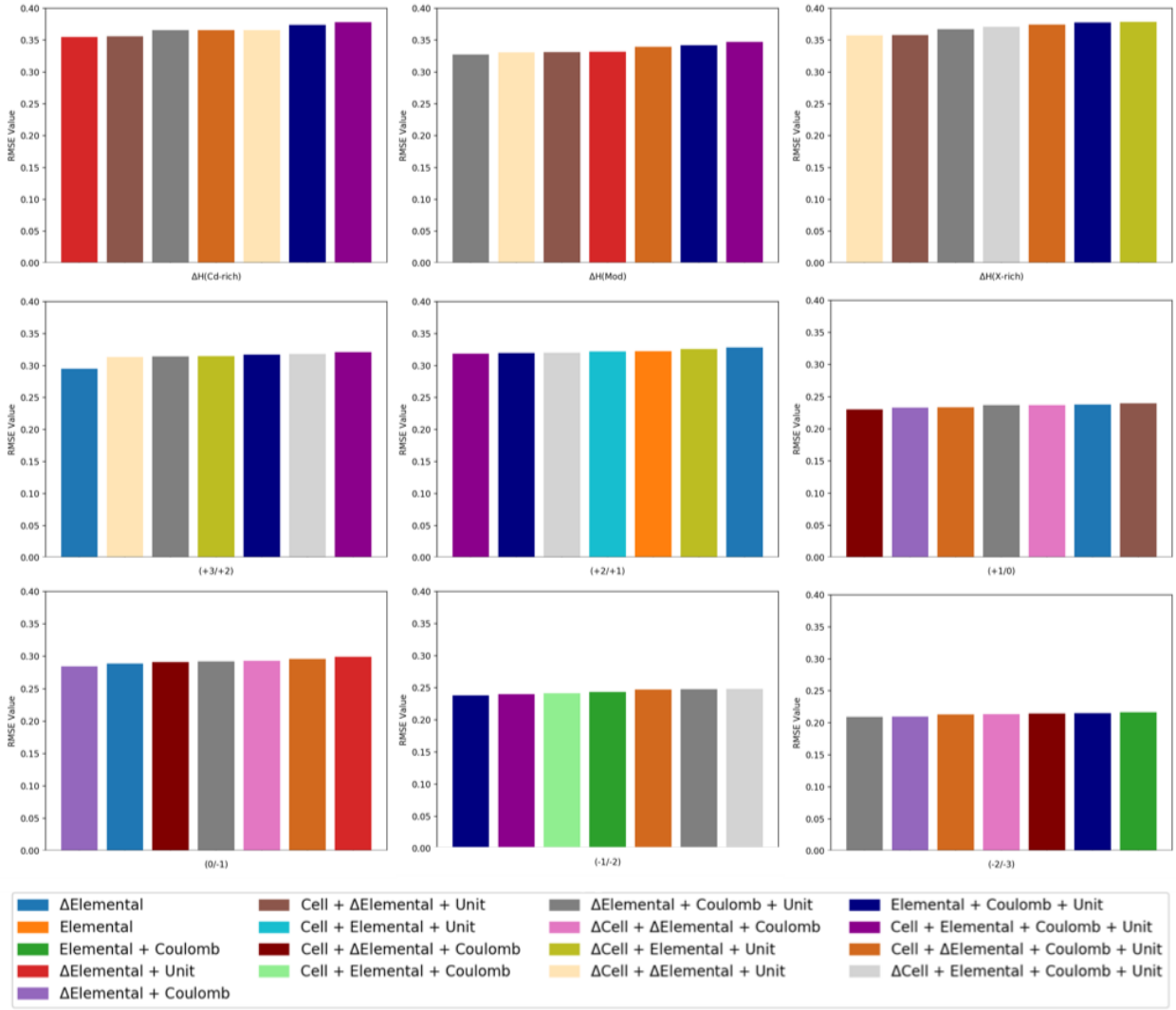
- **ΔElemental**: Period, Group, Site, Delta Ion. Rad., Delta At. Wt., Delta Cov. Rad., Delta Ion. En., Delta At. Rad., Delta EA, Delta EN, Delta At. Num., Delta Val., # Cd Neighbors, # S Neighbors, # Se Neighbors, # Te Neighbors, # Se/Te Neighbors, Corrected VBM (eV), and Corrected CBM (eV)

- **Cell**: bond angles and bond lengths for all atoms in the unit cell

- **ΔCell**: the change in bond angles and bond lengths in the doped versus undoped unit cell
- **Unit**: 'dH(Cd-rich) UC', 'dH(Mod) UC', 'dH(X-rich) UC'
- **Coulomb**: the coulomb matrix

The code used to create this figures is all on the github in descriptor_analysis.py and examples of how to use the descriptor_analysis functions can be found in the jupyter notebook located in doc/notebooks/descriptor_analysis_examples.ipynb

Here are the top 7 best groups of descriptors and their RMSE:


RMSE Value
ΔH(Cd-rich)
ΔH(Mod)
ΔH(X-rich)
(+3/+2)
(+2/+1)
(+1/0)
(0/-1)
(-1/-2)
(-2/-3)
ΔElemental
Elemental
Elemental + Coulomb
ΔElemental + Unit
ΔElemental + Coulomb
Cell + ΔElemental + Unit
Cell + Elemental + Unit
Cell + ΔElemental + Coulomb
Cell + Elemental + Coulomb
ΔElemental + Coulomb + Unit
ΔCell + ΔElemental + Coulomb
ΔCell + Elemental + Unit
ΔCell + ΔElemental + Unit
Elemental + Coulomb + Unit
Cell + Elemental + Coulomb + Unit
Cell + ΔElemental + Coulomb + Unit
ΔCell + Elemental + Coulomb + Unit