Site Specific Descriptor for Oxygen Evolution Reaction Activity on Single Atom Catalysts Using QMML

Erakulan E. Siddharthan[a], *Sourav Ghosh*[a], *Ranjit Thapa*[a,b,*]

[a]*Department of Physics, SRM University—AP, Amaravati, Andhra Pradesh 522 240, India*

[b]*Center for Computational and Integrative Sciences, SRM University—AP, Amaravati, Andhra Pradesh 522 240, India*

*Corresponding Author: ranjit.t@srmap.edu.in

Free energy calculation:

The free energy of each molecule and intermediate is G = E+ZPE-TS-neU, where E is the DFT energy, ZPE is the zero-point energy, TS is the entropic term, n is the number of electrons transferred and U is the applied potential at the electrode. The ZPE and TS terms for adsorbed intermediates are small and negligible. The equilibrium potential (U_0) is 0.40 V vs SHE in the alkaline medium at the pH of 14.

The following free energy relations of ion/molecules to fix the total free energy of the overall reaction at 4.92 eV as given by Norskov et al[1].,

$$G_{H_2O(l)} = G_{H_2O(g)} + RT\ln\left(\frac{p}{p_0}\right) \quad \ldots(1)$$

$$G_{O_{2(g)}} = 2G_{H_2O(l)} - 2G_{H_2} + 4.92 \quad \ldots(2)$$

$$G_{OH^-} = G_{H_2O(l)} - G_{H^+} \quad \ldots(3)$$

$$G_{H^+} = \frac{1}{2}G_{H_2} - k_B\mathrm{T}\ln 10 \times \mathrm{pH} \quad \ldots(4)$$

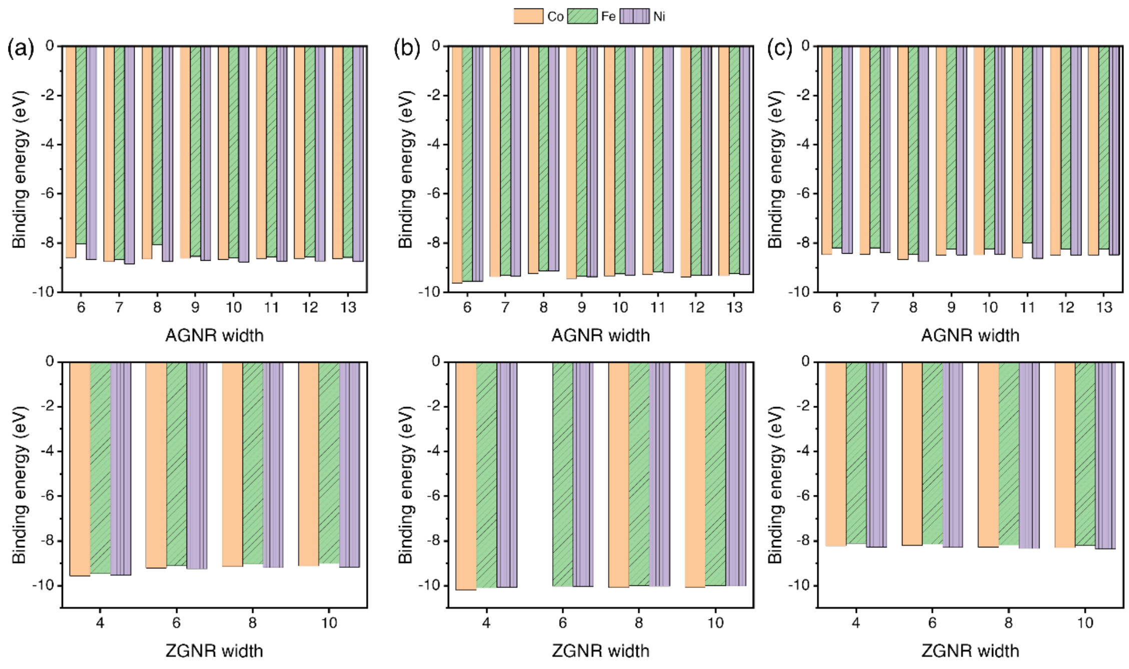
where R is the gas constant, T= 298.15 K, p =0.035 bar and p_0 = 1 bar.

The DFT energies, entropy terms and zero point energies for free molecules are given in table.

Molecule	**E (eV)**	**TS (eV)**	**ZPE (eV)**	**G (eV)**
H_2	-6.76	0.41	0.27	-6.9
H_2O (g)	-14.229	0.67	0.59	-14.307

Table S1: Relative DFT energies of Co, Fe and Ni on pristine AGNRs and ZGNRs for three N4 configurations considered.

System	**Relative average total energy (eV)**		
	N_4	**N_{4T}**	**N_{4E}**
aCo	1.36	0.52	0
aFe	1.47	0.73	0
aNi	1.20	0.46	0
zCo	0.84	1.68	0
zFe	0.86	1.68	0
zNi	0.73	1.53	0



Bulk cohesive energy (eV)	
Co	4.39
Fe	4.28
Ni	4.44

Figure S1. Binding energies of Co, Fe and Ni on (a) N_4 (b) N_{4E} and (c) N_{4T} configurations on AGNRs.

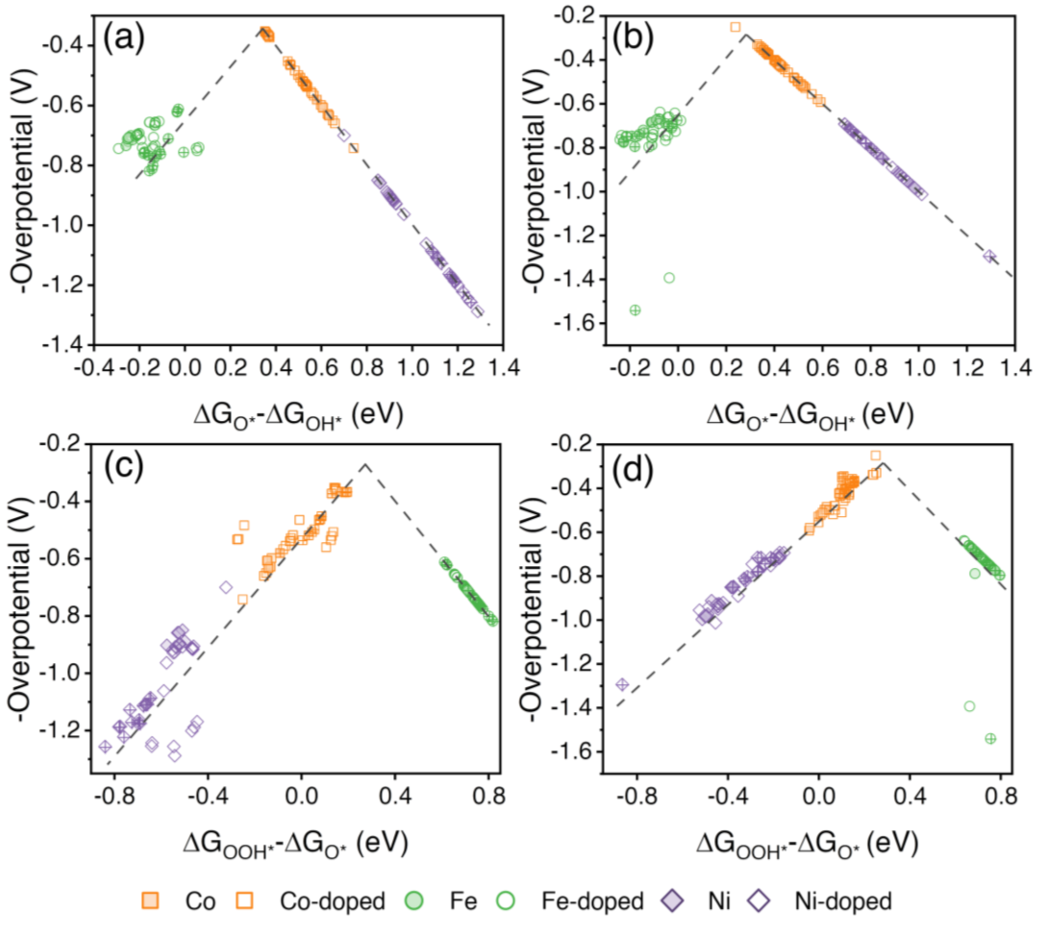


Figure S2. ΔG_{O*}-ΔG_{OH*} versus negative of overpotentials for configurations (a) N_{4E} and (b) N_{4T}. ΔG_{OOH*}-ΔG_{O*} versus negative of overpotentials for configurations (c) N_{4E} and (d) N_{4T}. Shapes without '+' sign indicate AGNRs and with '+' sign indicate ZGNRs.

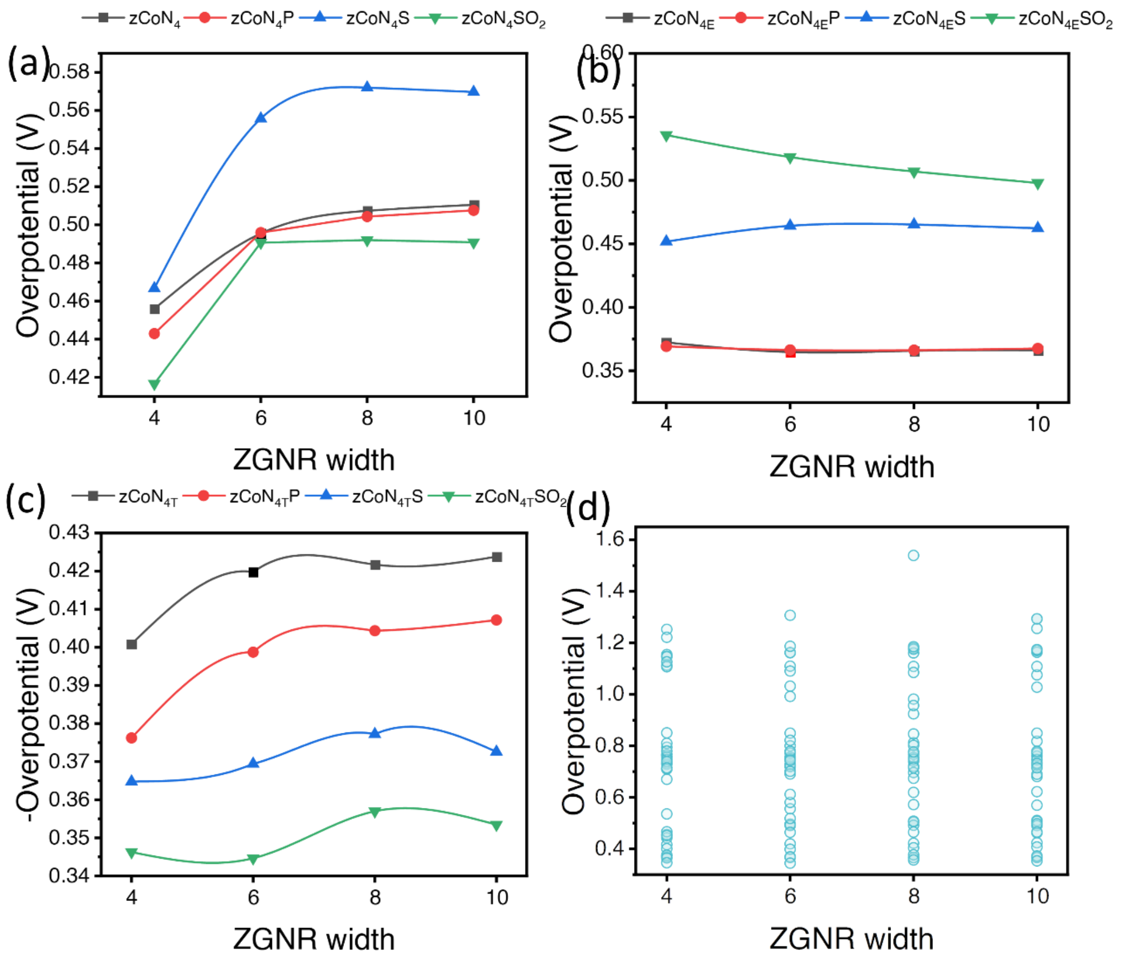


Figure S3. Overpotential variation with width of the pristine and doped ZGNR for the metals considered in (a) N_4 (b) N_{4E} and (c) N_{4T} configurations. (d) Overpotential distribution over the widths of the ZGNR.

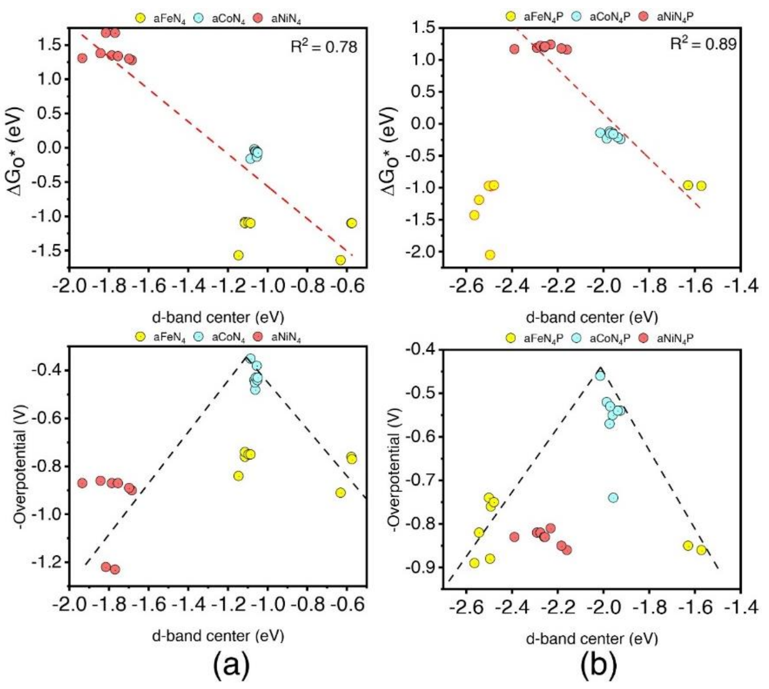


Figure S4. (a) d-band center vs ΔG_{O^*} and OER overpotentials of Fe, Co and Ni on aN_4 hosts. (b) d-band center vs ΔG_{O^*} and OER overpotentials of Fe, Co and Ni on aN_4P hosts.

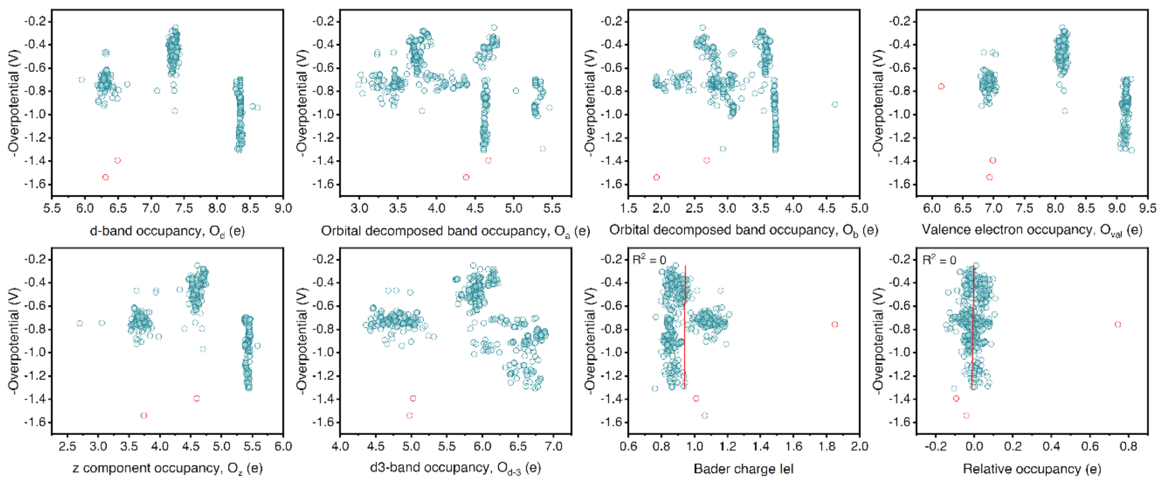


Figure S5. Occupancy and Bader charge based descriptors vs overpotentials.

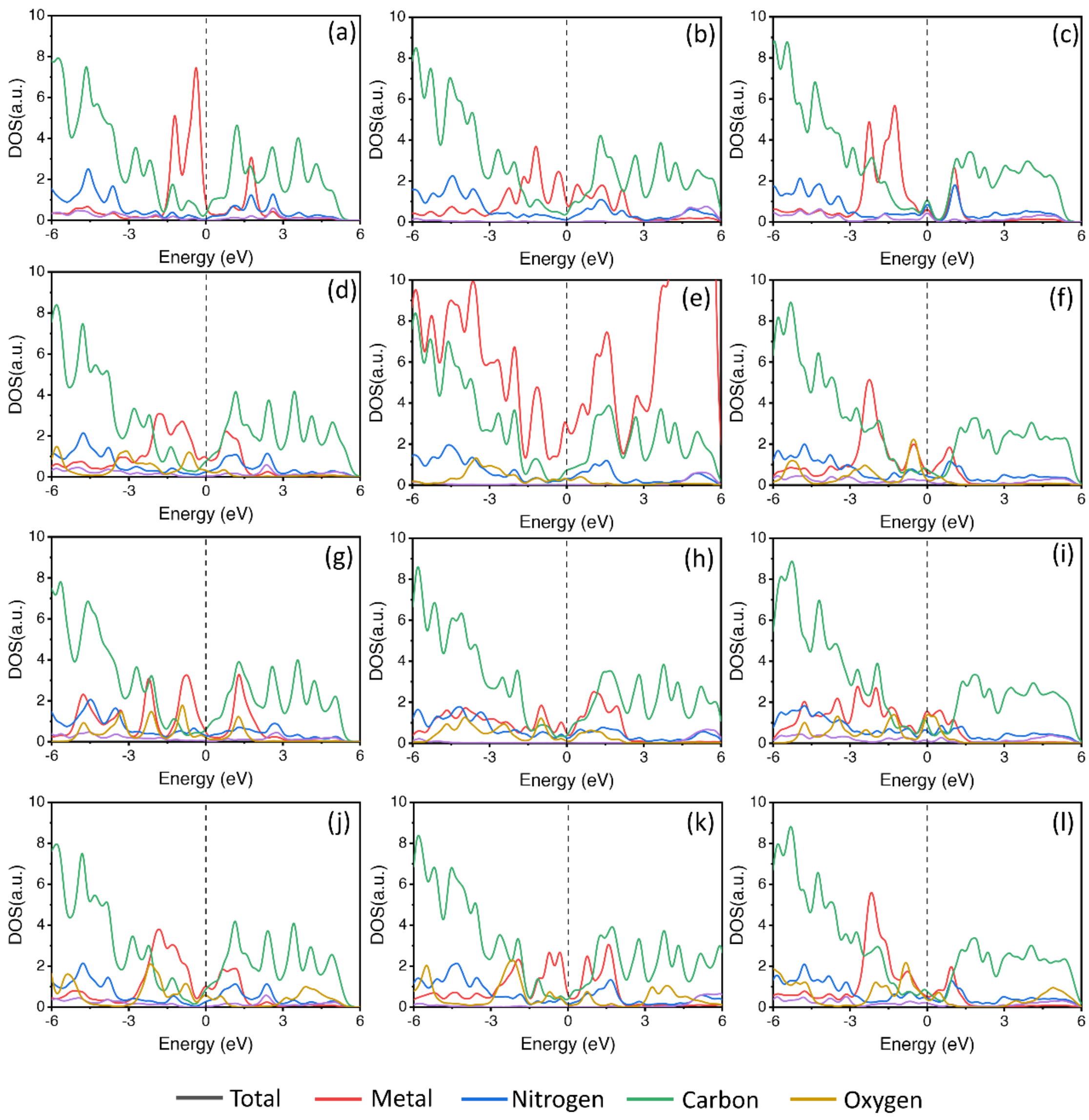


Figure S6. Atom projected density of states the systems (a) 6aCoN_4S (b) 6aFeN_{4E}SO2 and (c) 6aNiN_{4E}S. Atom projected density of states for the same system after adsorption of (d), (e) and (f) OH*, (g), (h) and (i) O* and (g), (h) and (i) OOH*. Dashed vertical line represents Fermi level.

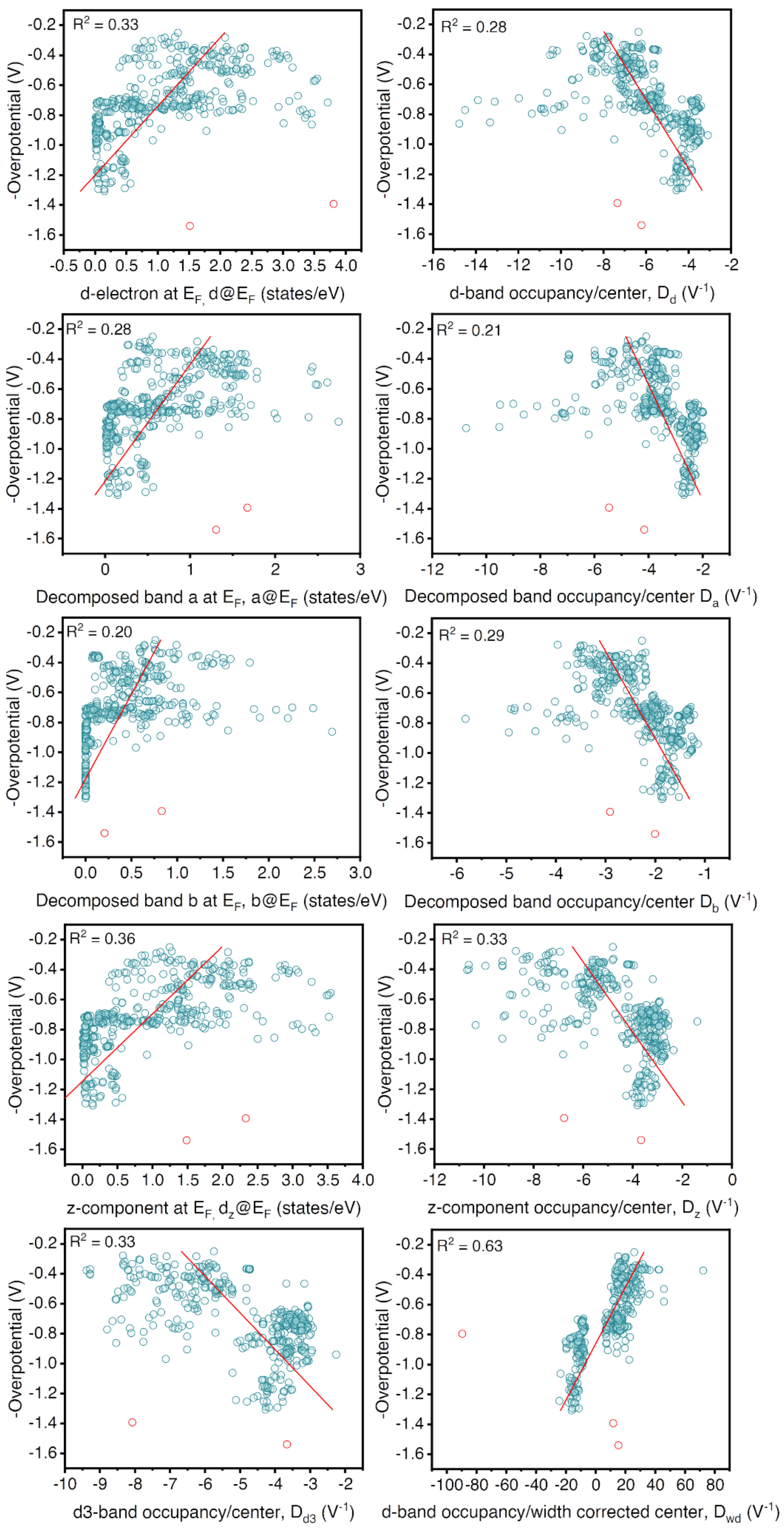


Figure S7. Density of states at Fermi level and band center scaled occupancy based descriptors vs overpotentials.

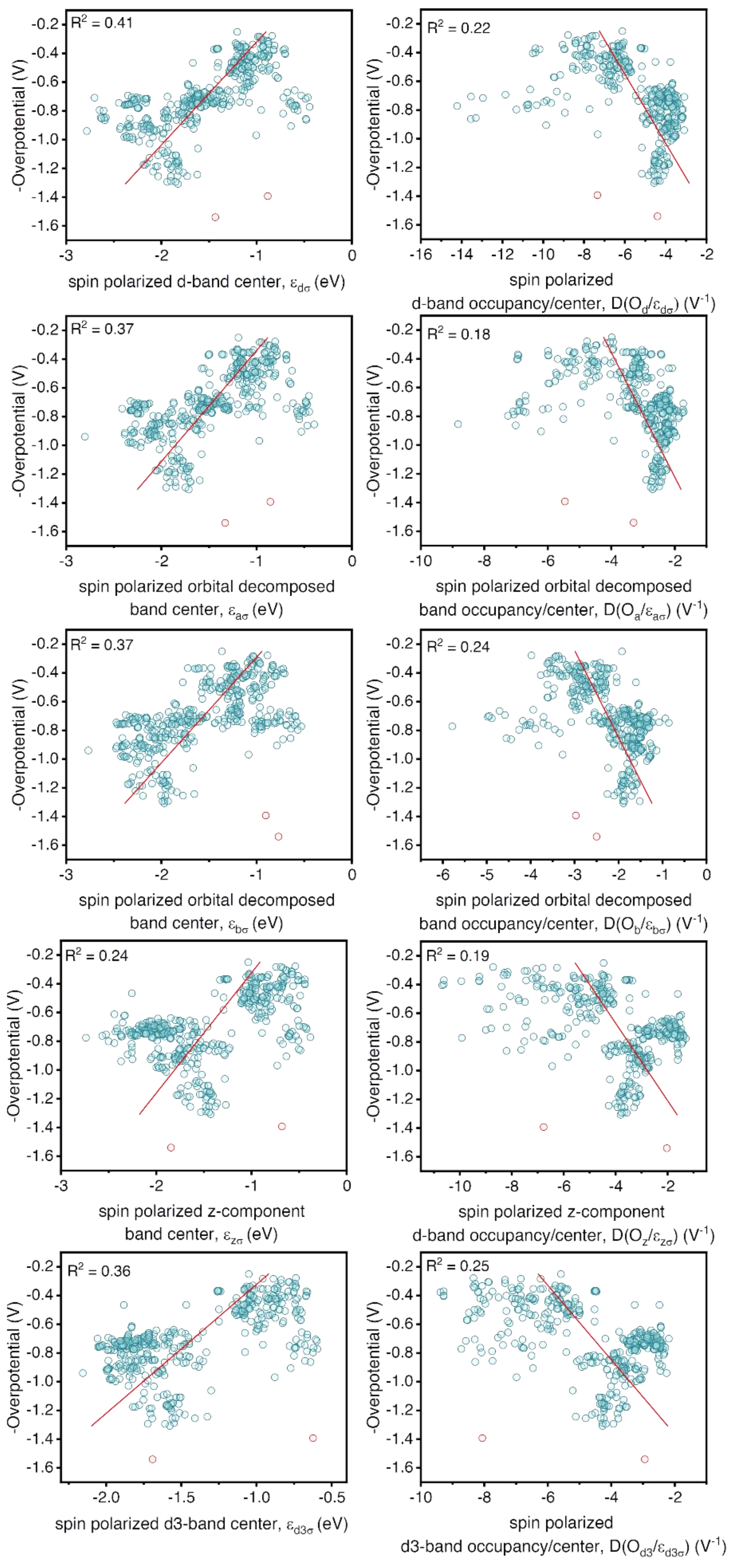


Figure S8. Spin effects included band centers and center scaled occupancy descriptors vs overpotentials.

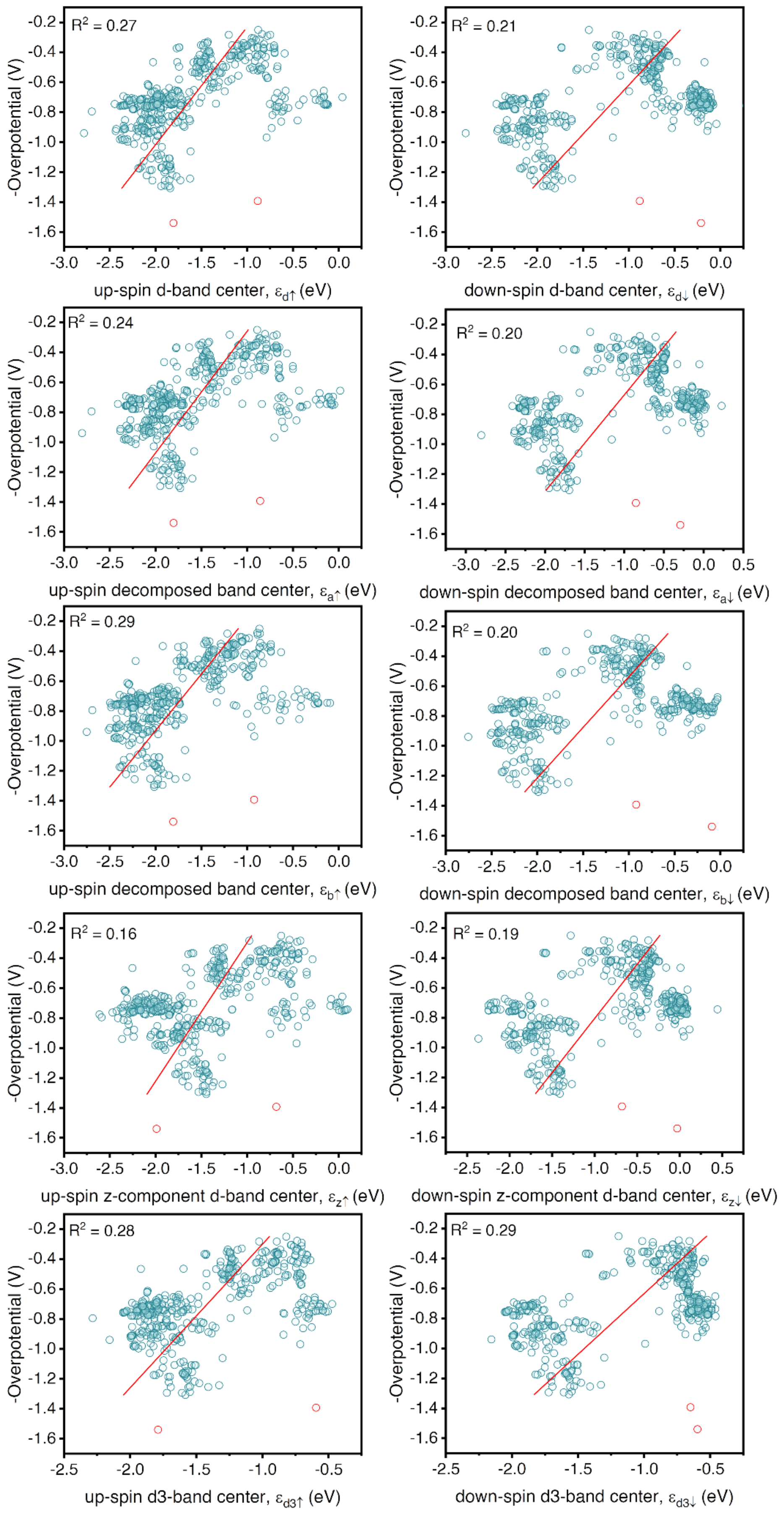


Figure S9. Individual spin component band centers vs overpotentials.

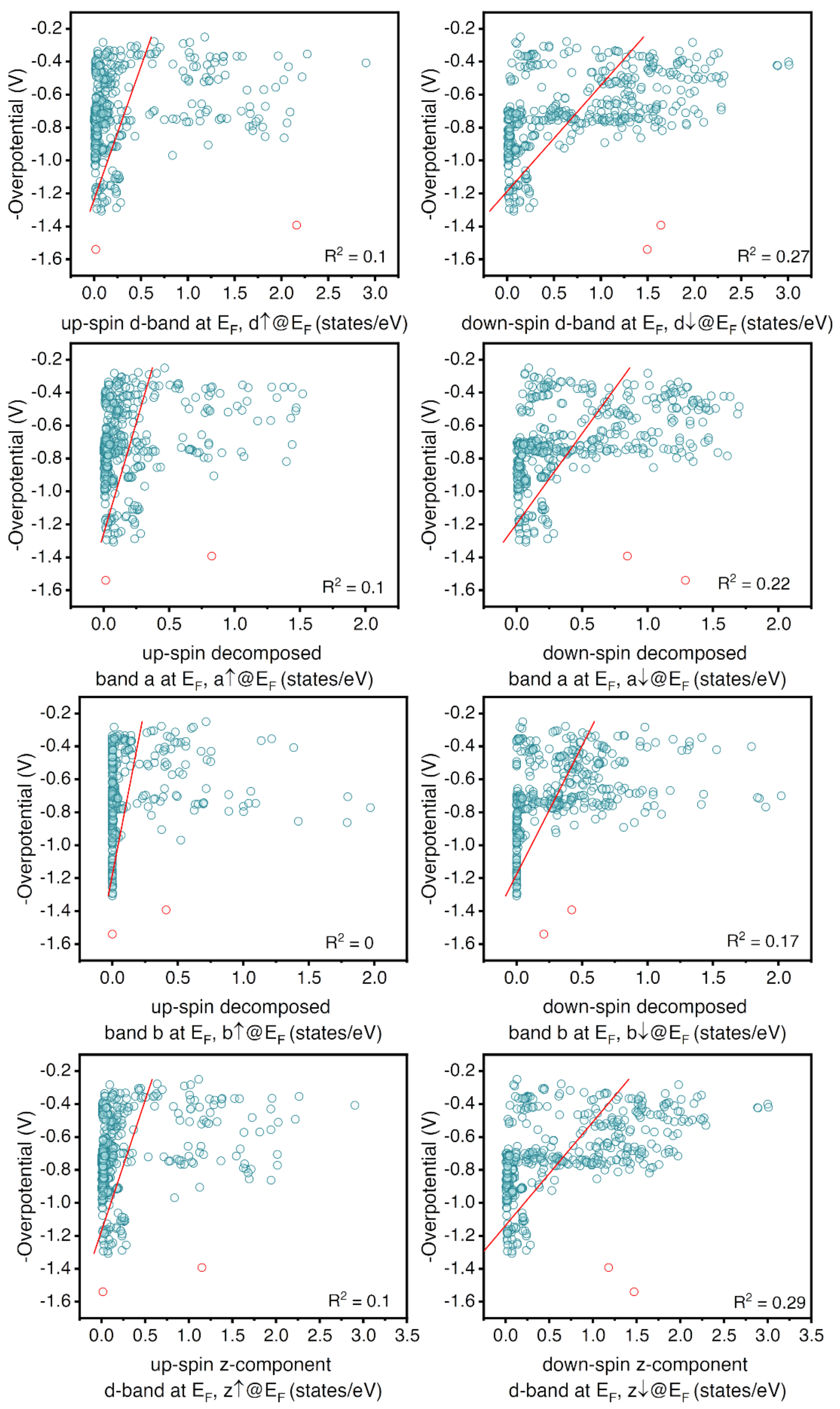


Figure S10. Individual spin component density of states of Fermi level descriptors vs overpotentials.

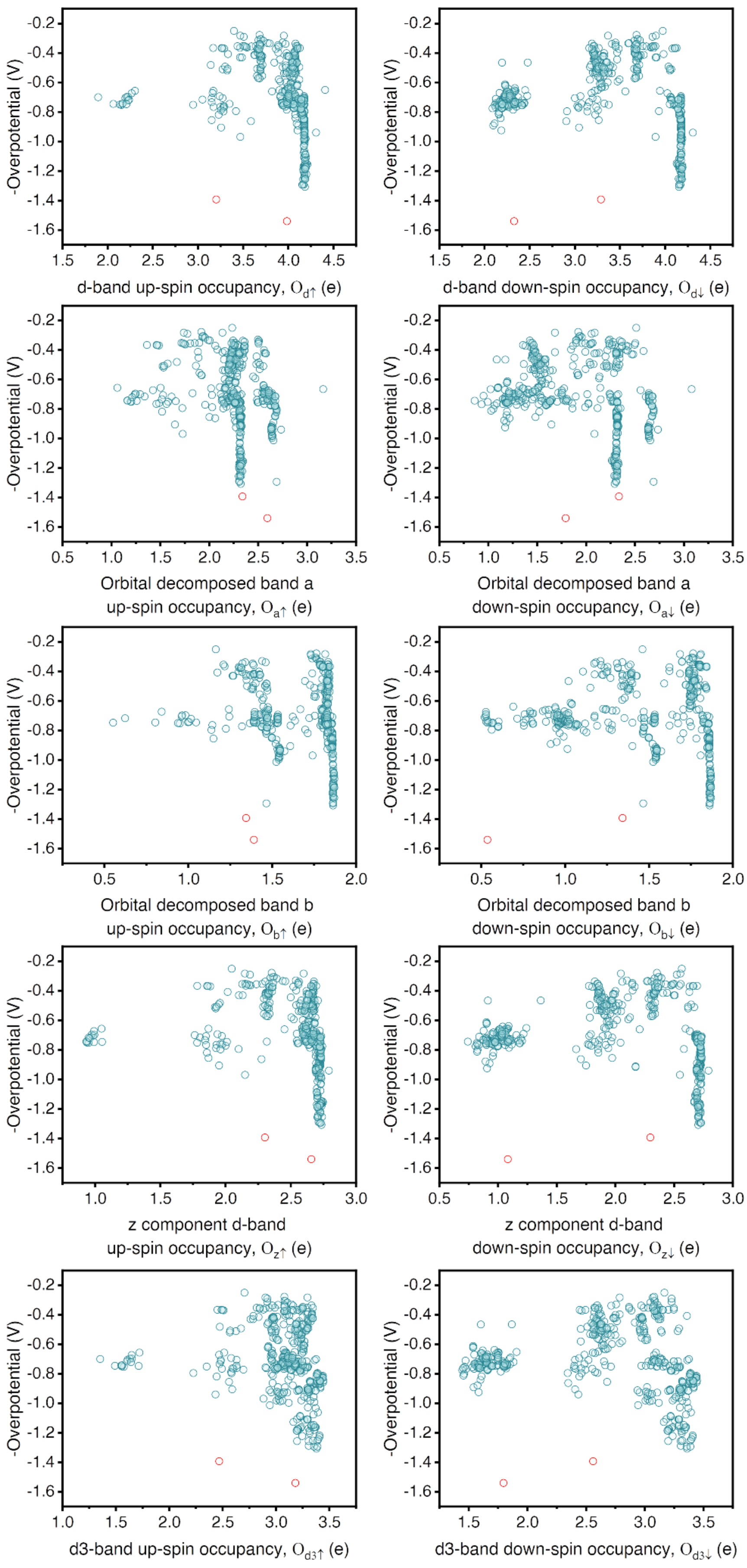


Figure S11. Individual spin component occupancy descriptors vs overpotentials.

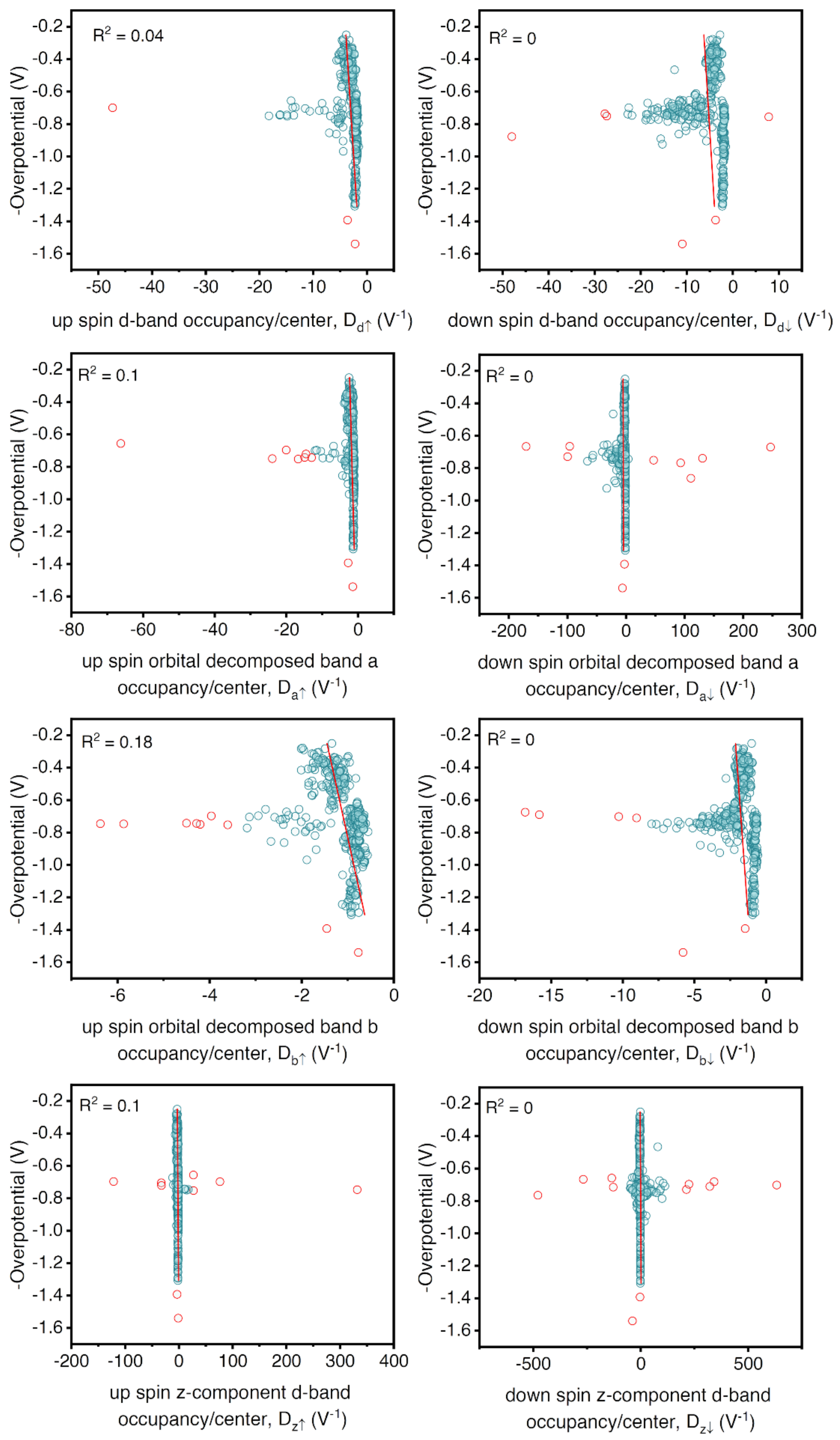


Figure S12. Individual spin component band center scaled occupancy descriptors vs overpotentials.

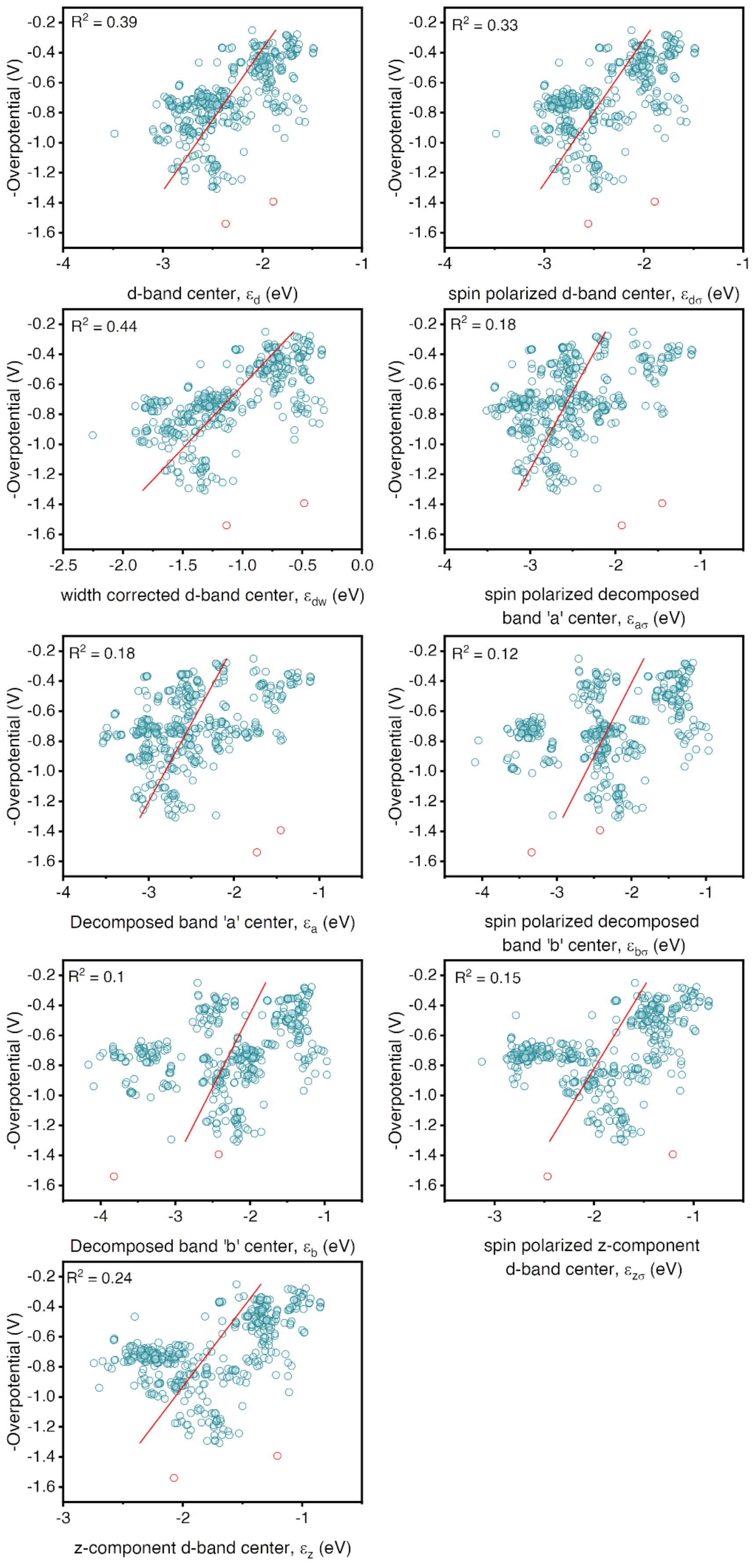


Figure S13. Band center and spin effects included band centers averaged from $-\infty$ to E_F vs overpotentials.

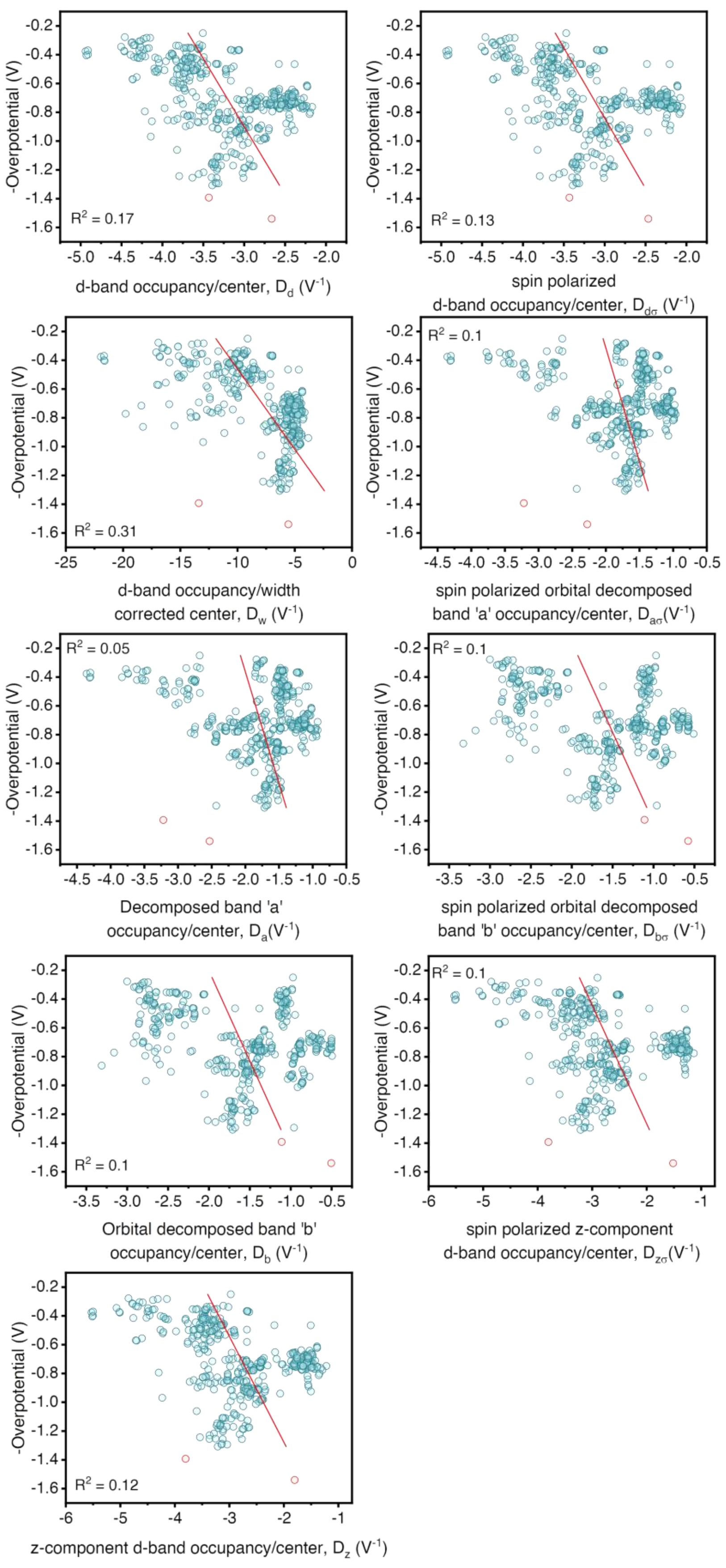


Figure S14. Band center scaled occupancy averaged from -∞ to E_F vs overpotentials.

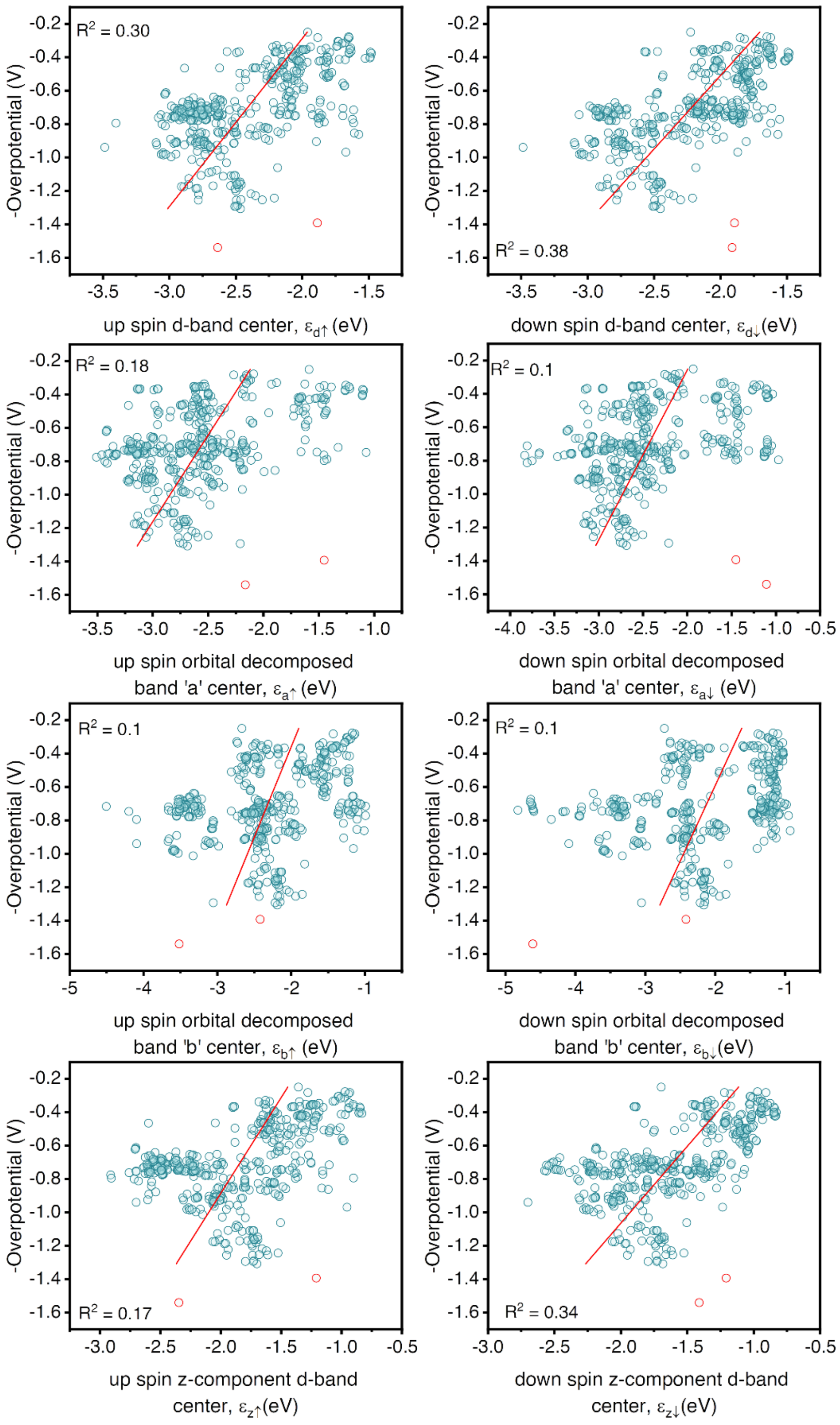


Figure S15. Individual spin component band centers averaged from -∞ to E_F vs overpotentials.

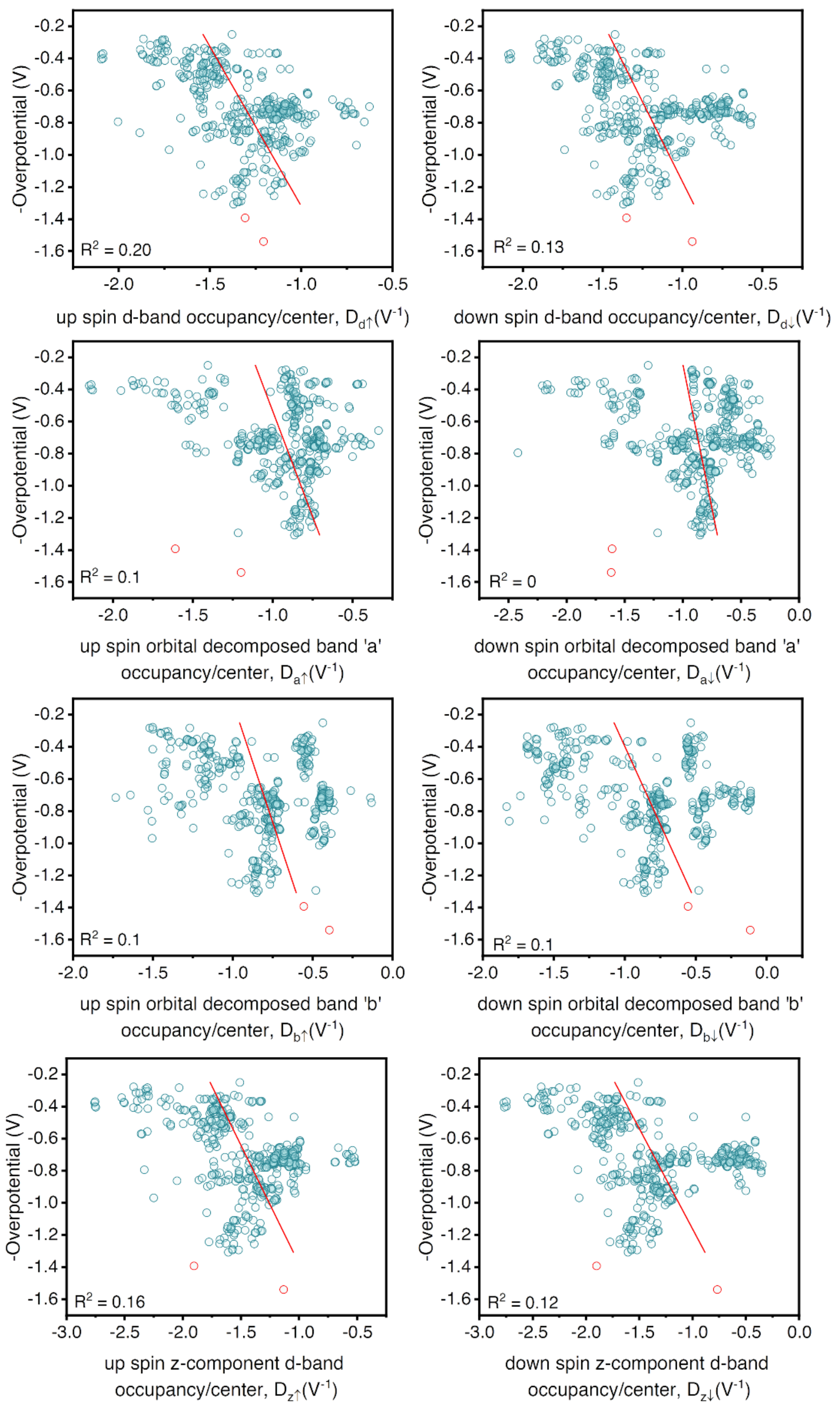


Figure S16. Individual spin component band center scaled occupancy descriptors averaged from -∞ to E_F vs overpotentials.

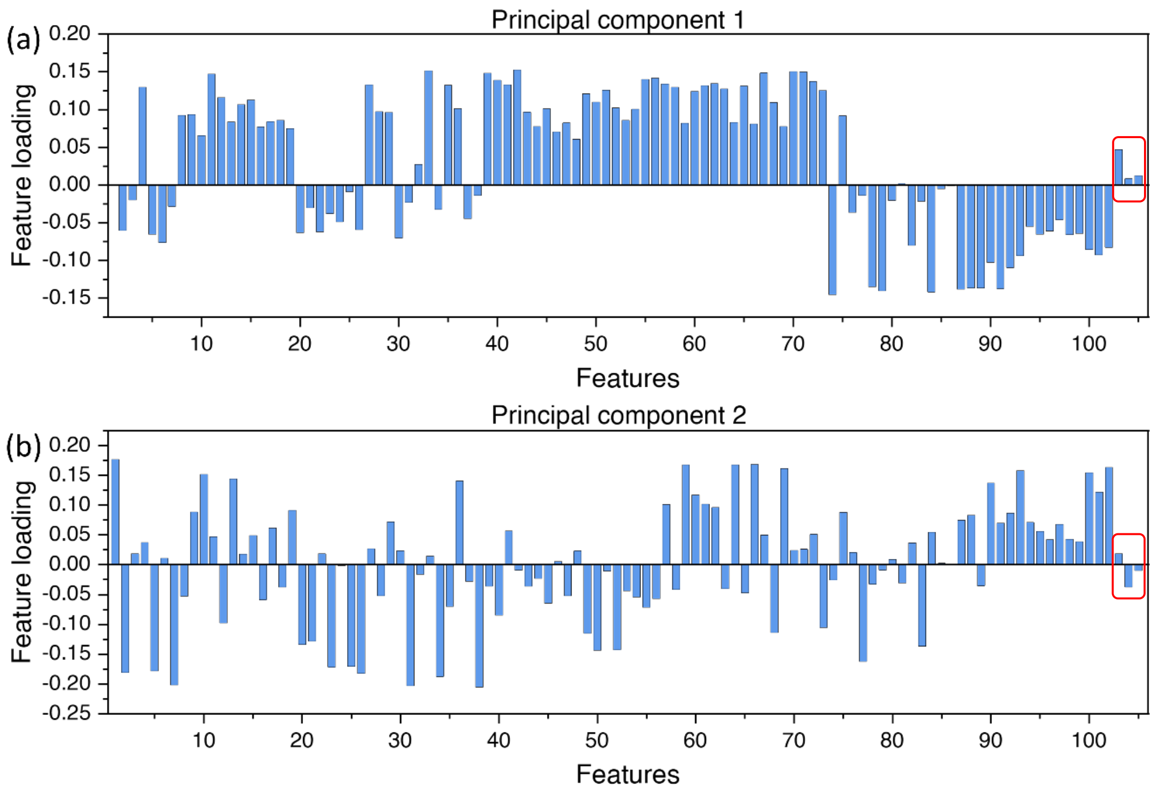


Figure S17. Features loading of the 105 features in the principal components (a) 1 and (b) 2.

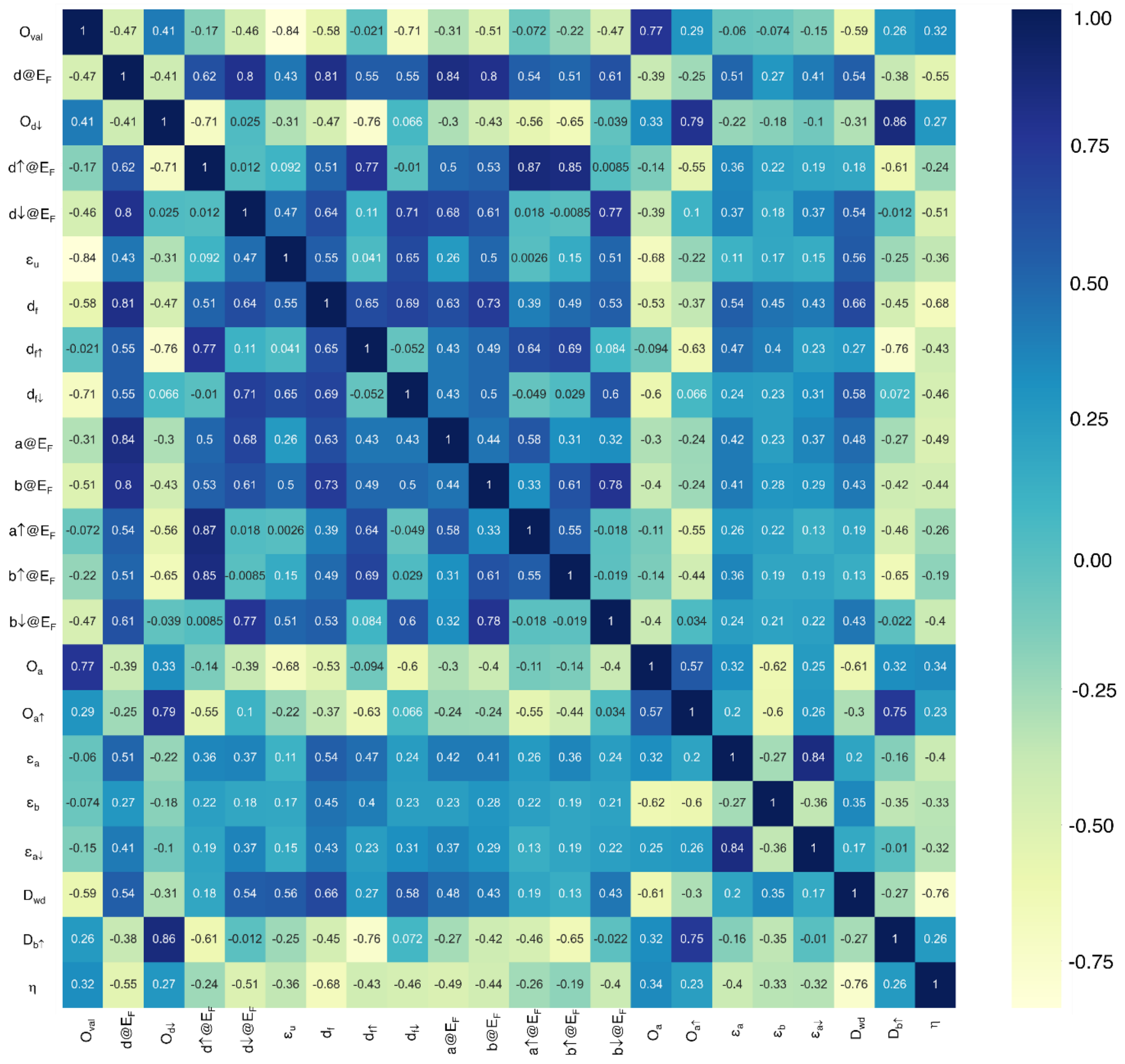


Figure S18. Correlation plot of the features filtered after PC analysis with overpotential.

Python code for machine learning algorithms is as follows,

```
#!/usr/bin/env python
# coding: utf-8

# ## Train the dataset on simple regression models , We will use the
following models
#
# * Ridge regression
# * Lasso regression
# * Random forest regressor
# * KNeighbours regressor
# * Support vector regressor



### import libraries
import pandas as pd
import matplotlib.pylab as plt
import numpy as np
plt.rc('font', family='Helvetica')
plt.rc('axes', linewidth=2.5)
```

```
#read and visualize the data
df=pd.read_csv('pca_corr_filtered_data.csv')
cols=df.columns
df=df.drop(cols[0],axis=1)
df.head()

df.describe()

##sep[erate the features

df_X=df.drop('y',axis=1)
df_X.head()

#seperate the target
df_y=df['y']
df_y.head()

#training and testing set prepare
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_X,df_y,
test_size=0.20, random_state=40)
print(X_train.shape)
print(X_test.shape)

#scale the features
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
print(X_train)
print(X_test)
# X_train = normalize(X_train)
# X_test = normalize(X_test)
#X_train[0]



##import the regression/classifier models

from time import time

from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import BayesianRidge

from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error


#define a dataframe where we will store the result

df_simple_reg =
pd.DataFrame(columns=['model_name','model_name_pretty','model_params','fit_
time','r2_train','mae_train',
                                    'rmse_train'])
```

```
df_simple_reg


# Build a dictionary of model names
from collections import OrderedDict
simple_reg_model_names = OrderedDict({
    'rr': Ridge,
    'lr': Lasso,
    'rfr': RandomForestRegressor,
    'svr': SVR,
    'knr': KNeighborsRegressor,
    'mlr': LinearRegression,
    'bayesR':BayesianRidge
})


#define a function so that when we will call that function it will do the
regression and predict the result
def model_train_eval(model,X_train,Y_train):
    #Fit the model
    t = time()
    model=model()
    model.fit(X_train,Y_train)
    time_to_fit = time()-t

    #Evaluate the model
    y_actual = Y_train
    y_predicted = model.predict(X_train)

    #Get the scores
    r2_Score = r2_score(y_actual,y_predicted)
    mae = mean_absolute_error(y_actual,y_predicted)
    rmse = mean_squared_error (y_actual,y_predicted,squared=False)

    #Results
    result_dict = {
        'model_name': model_name,
        'model_name_pretty': type(model).__name__,
        'model_params': model.get_params(),
        'fit_time': time_to_fit,
        'r2_train': r2_Score,
        'mae_train': mae,
        'rmse_train': rmse}
    return result_dict




simple_reg_models = OrderedDict()
# Loop through each model type, fit and predict, and evaluate and store
results
for model_name, model in simple_reg_model_names.items():
    print(f'Currently running {model_name}: {model.__name__}')
    result_dict = model_train_eval(model, X_train, y_train)
    df_simple_reg = df_simple_reg.append(result_dict, ignore_index=True)

#df_simple_reg
## Get the best fit model based on r2score
df_simple_reg = df_simple_reg.sort_values('r2_train', ignore_index=True)
df_simple_reg

# Get the parameters of the best performing model
best_r2score_train= df_simple_reg.iloc[-1, :].copy()
```

```
model =
simple_reg_model_names[best_r2score_train['model_name']](**best_r2score_tra
in['model_params'])
model.fit(X_train, y_train)

best_r2score_train['model_params']

##print r2_score_tests
y_act_test = y_test
y_pred_test = model.predict(X_test)


r2 = r2_score(y_act_test, y_pred_test)
mae = mean_absolute_error(y_act_test, y_pred_test)
rmse = mean_squared_error(y_act_test, y_pred_test, squared=False)

print(f'r2: {r2:0.4f}')
print(f'mae: {mae:0.4f}')
print(f'rmse: {rmse:0.4f}')




##define a function which will show the visualization of plots of
prediction
def plot_pred_vs_act(act, pred, model, label=''):
    xy_max = np.max([np.max(act), np.max(pred)])
    plot = plt.figure(figsize=(10,10))
    plt.plot(act, pred, 'o', ms=15, mec='b', mfc='silver', alpha=0.8)
    plt.plot([0, xy_max], [0, xy_max], 'k--', label='ideal')
    plt.axis('scaled')
    plt.xlabel(f'Actual',fontsize=16)
    plt.ylabel(f'Predicted',fontsize=16)
    plt.title(f'{type(model).__name__}, r2: {r2_score(act,
pred):0.4f}',fontsize=16)
    plt.legend(loc='upper left',fontsize=14)
    plt.tick_params(which='major',left = True, right = False , labelleft =
True ,
                labelbottom = True, bottom = True,direction="in",width =
2,length = 4)
    plt.xticks(fontsize = 16)
    plt.yticks(fontsize = 16)

    return plot

plot = plot_pred_vs_act(y_act_test, y_pred_test, model)


# ## Linear Regression

def plot_pred_vs_act2(act, pred, model,r2_score, label=''):
    xy_max = np.max([np.max(act), np.max(pred)])
    plot = plt.figure(figsize=(10,10))
    plt.plot(act, pred, 'o', ms=15, mec='b', mfc='silver', alpha=0.8)
    plt.plot([0, xy_max], [0, xy_max], 'k--', label='ideal')
    plt.axis('scaled')
    plt.xlabel(f'Actual',fontsize=16)
    plt.ylabel(f'Predicted',fontsize=16)
    ##plt.title(f'{model}, r2: {r2_score:0.4f}',fontsize=16)
    plt.legend(loc='upper left',fontsize=14)
    plt.tick_params(which='major',left = True, right = False , labelleft =
True ,
                labelbottom = True, bottom = True,direction="in",width =
2,length = 4)
```

```
    plt.xticks(fontsize = 16)
    plt.yticks(fontsize = 16)

    return plot

cols1=['x1','x2','target','coef1','coef2','intercept','r2_test','r2_train']
res1=pd.DataFrame(columns=cols1)
def do_lin_reg(X_train,y_train,X_test,y_test):

    # Create an instance of the LinearRegression class
    reg = LinearRegression()


    # Fit the model to the data
    reg.fit(X_train, y_train)
    #Coefficients
    #print(reg.coef_)
    #Intercept
    #print(reg.intercept_)
    #r2_score
    y_pred=reg.predict(X_test)
    r2_test=r2_score(y_test, y_pred)
    y_pred_train=reg.predict(X_train)
    r2_train=r2_score(y_train, y_pred_train)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    #print(r2)
    y_pred_df=pd.DataFrame(data=y_pred,columns=[df.columns[-1]])
    #plot = plot_pred_vs_act2(y_test,y_pred,'mlr',r2_score)
    return [reg.coef_,reg.intercept_,r2_test,r2_train,mae,rmse]


cols_x=df_X.columns
#cols_x
#########m mlr with combination of 2
for i in range(len(cols_x)):
    for j in range(i+1,len(cols_x)):
        X_train_new=[]
        X_test_new=[]
        for ele in X_train:
            small_arr=[ele[i],ele[j]]
            X_train_new.append(small_arr)
        for ele in X_test:
            small_arr=[ele[i],ele[j]]
            X_test_new.append(small_arr)
        res_reg=do_lin_reg(X_train_new,y_train,X_test_new,y_test)

row_to_append=[cols_x[i],cols_x[j],'overpotential',res_reg[0][0],res_reg[0]
[1],res_reg[1],res_reg[2],res_reg[3]]
        res1.loc[len(res1.index)] = row_to_append

res1.sort_values(by="r2_test",ascending=False)


#checking with linear regression so that if there is a good linear
correlation we can predict some equation

# Create an instance of the LinearRegression class
reg = LinearRegression()
# Fit the model to the data
reg.fit(X_train, y_train)

print(reg.coef_)
print(reg.intercept_)

y_pred=reg.predict(X_test)
r2=r2_score(y_test, y_pred)
#r2
#adjusted_r2
adjustd_r_sqrd = 1.0 - (1.0 - r2)*(len(y_test)-1)/(len(y_test)-
X_test.shape[1]-1)
#adjustd_r_sqrd
y_pred_df=pd.DataFrame(data=y_pred,columns=[df.columns[-1]])

#changing the previously defined plotting function little bit according to
the convenience
def plot_pred_vs_act2(act, pred, model,r2_score, label=''):
    xy_max = np.max([np.max(act), np.max(pred)])
    plot = plt.figure(figsize=(10,10))
    plt.plot(act, pred, 'o', ms=15, mec='b', mfc='silver', alpha=0.8)
    plt.plot([0, xy_max], [0, xy_max], 'k--', label='ideal')
    plt.axis('scaled')
    plt.xlabel(f'Actual',fontsize=37,family='Helvetica')
    plt.ylabel(f'Predicted',fontsize=37)
#     plt.title(f'{model}, r2: {r2_score}',fontsize=37)
    ##plt.title(f'{model}, r2: {r2_score:0.4f}',fontsize=16)
#     plt.legend(loc='upper left',fontsize=30)
    plt.tick_params(which='major',left = True, right = False , labelleft =
True ,
                labelbottom = True, bottom = True,direction="out",width =
2,length = 4)
    plt.text(0.01,1.2,f'R$^{2}$= {format(r2_score, ".2f")}', fontsize = 32)
    plt.xticks(fontsize = 30)
    plt.yticks(fontsize = 30)

    return plot

plot = plot_pred_vs_act2(y_test,y_pred,'mlr',r2)


#selecting the best random_state
seeds=[i for i in range(100)]
dat_col=['seed','coef1','coef2','intercept','r2_test','r2_train','mae','rms
e']
dat=pd.DataFrame(columns=dat_col)
for seed in seeds:
    X_train, X_test, y_train, y_test = train_test_split(df_X,df_y,
test_size=0.20, random_state=seed)
    #print(X_train.shape)
    #print(X_test.shape)
    res_reg=do_lin_reg(X_train,y_train,X_test,y_test)

row_to_append=[int(seed),res_reg[0][0],res_reg[0][1],res_reg[1],res_reg[2],
res_reg[3],res_reg[4],res_reg[5]]
    dat.loc[len(dat.index)] = row_to_append
dat_sort=dat.sort_values(by="r2_test",ascending=False)

dat_sort.to_csv("dat_col.csv")
df_seed_based_df=pd.read_csv("dat_col.csv")
##df_seed_based_df.head()
seed=int(df_seed_based_df['seed'][0])
#seed


##splitting the dataset according to best random state and repeating the
previous steps
```

```
X_train, X_test, y_train, y_test = train_test_split(df_X,df_y,
test_size=0.20, random_state=seed)
print(X_train.shape)
print(X_test.shape)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# print(X_train)
# print(X_test)

new_result_train_df=pd.DataFrame()
new_result_test_df=pd.DataFrame()
new_result_train_df['Actual']=y_train.tolist()
new_result_test_df['Actual']=y_test.tolist()
#new_result_train_df
# X_train

r2_train=[]
r2_test=[]
def
give_the_model_and_get_plot(X_train,X_test,y_train,y_test,model,model_name)
:
    # Create an instance of the model class
    reg = model()
    reg.fit(X_train, y_train)
    y_pred=reg.predict(X_test)
    y_pred_train=reg.predict(X_train)
    new_result_train_df[f'{model_name}_pred']=y_pred_train.tolist()
    new_result_test_df[f'{model_name}_pred']=y_pred.tolist()
    r2_test_val=r2_score(y_test, y_pred)
    r2_train_val=r2_score(y_train, y_pred_train)
    #plot = plot_pred_vs_act2(y_test,y_pred,model_name,r2)
    return r2_test_val,r2_train_val

for model_name, model in simple_reg_model_names.items():

r2_test_val,r2_train_val=give_the_model_and_get_plot(X_train,X_test,y_train
,y_test,model,model_name)
    r2_train.append(r2_train_val)
    r2_test.append(r2_test_val)
r2_df=pd.DataFrame()
#r2_df.columns=["train","test"]
r2_df.index=["Ridge","Lasso","RandomForestRegressor","SVR","KNeighborsRegre
ssor","LinearRegression","BayesianRidge"]
r2_df["train"]=r2_train
r2_df["test"]=r2_test
r2_df

#trying with xgboost also
import xgboost as xgb
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_train = le.fit_transform(y_train)


# Use "hist" for constructing the trees, with early stopping enabled.
clf = xgb.XGBClassifier(tree_method="hist", early_stopping_rounds=50)
# Fit the model, test sets are used for early stopping.
clf.fit(X_train, y_train, eval_set=[(X_test, y_test)])
# Save model into JSON format.
clf.save_model("clf.json")
```

```
from sklearn.metrics import mean_squared_error

preds1 = clf.predict(X_test)
preds2 = clf.predict(X_train)
xgb_test_pred=le.inverse_transform(preds1)
xgb_train_pred=le.inverse_transform(preds2)
new_result_train_df['xgb']=xgb_train_pred.tolist()
new_result_test_df['xgb']=xgb_test_pred.tolist()
r2_xgb_train=r2_score(y_train, xgb_train_pred)
r2_xgb_test=r2_score(y_test, xgb_test_pred)
r2_index=r2_df.index.tolist()
r2_index.append('xgb')
#r2_df.index=r2_index
print([r2_xgb_train,r2_xgb_test])
#r2_df
learn = XGBClassifier()
learn.fit(X_train, y_train)
print (learn)


#saving the prediction of training and testing datasets
new_result_train_df.to_csv("train_pred.csv")
new_result_test_df.to_csv("test_pred.csv")
#saving the r2_score of each model
r2_df.to_csv("r2.csv")
r2_df
```

References:

(1) Nørskov, J. K.; Rossmeisl, J.; Logadottir, A.; Lindqvist, L.; Kitchin, J. R.; Bligaard, T.; Jónsson, H. Origin of the Overpotential for Oxygen Reduction at a Fuel-Cell Cathode. *J. Phys. Chem. B* **2004**, *108* (46), 17886–17892. https://doi.org/10.1021/jp047349j.