Supplementary Information

of the paper

"Improved gas selectivity based on carbon modified SnO_2 nanowires"

Below is the Python code used to obtain improved selectivity from carbon modified SnO_2 nanowires.

Dependencies

```
%matplotlib inline
from collections import defaultdict
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# ---- Scikit --------
from sklearn import svm
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler
from sklearn.model_selection import StratifiedKFold, train_test_split,
cross_val_score, KFold
from sklearn.metrics import mean_squared_error
# ---- Classifier ------
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier,
AdaBoostClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
# ---- Regressor --------
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor, BaggingRegressor,
AdaBoostRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor

# ---- Keras ---------
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.utils import to_categorical
```

```
# ---- PyDrive -------
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# ----- Umap ---------
!pip install -U -q umap-learn
import umap

import warnings
warnings.simplefilter("ignore")
```

```
Using TensorFlow backend.
  Building wheel for PyDrive (setup.py) ... done
```

Loading files

```
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# Load file
sensor_data_id = '0B7EQzoMtbLBSRE96d0k4OFp0X2RfUzc3elZmbTRiU2o2blNB'
drive.CreateFile({'id':sensor_data_id}).GetContentFile('sensor_data.csv')
sensor_data = pd.read_csv('sensor_data.csv', index_col=0)
```

Data exploration

```
sensor_data
```

```
sensor_data.keys()
```

```
Index(['T200', 'T250', 'T300', 'T350', 'T400', 'Gas', ' "Color"',
       ' "Concentration"'],
      dtype='object')
```

```
sensor_data.rename(columns={
    ' "Concentration"': 'Concentration',
    ' "Color"':'Color'
}, inplace=True)
```

```
sensor_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 60 entries, 1 to 60
```

```
Data columns (total 8 columns):
T200             60 non-null float64
T250             60 non-null float64
T300             60 non-null float64
T350             60 non-null float64
T400             60 non-null float64
Gas              60 non-null object
Color            60 non-null object
Concentration    60 non-null int64
dtypes: float64(5), int64(1), object(2)
memory usage: 4.2+ KB
```

```
sensor_data[['T200', 'T250', 'T300', 'T350', 'T400']].describe()
```

```
for key in sensor_data['Gas'].unique():
  print(f'{key}: {sensor_data[sensor_data["Gas"] == key].shape[0]} occourances')
```

```
Ethanol: 10 occourances
Hydrogen: 10 occourances
CO: 10 occourances
Acetone: 10 occourances
Ammonia: 10 occourances
Toluene: 10 occourances
```

Pre-processing

```
label_encoder = LabelEncoder()
X_data = StandardScaler().fit_transform(sensor_data[['T200', 'T250', 'T300',
'T350', 'T400']])
Y_clas = label_encoder.fit_transform(sensor_data['Gas'])
Y_regr = sensor_data['Concentration'].values
```

```
X_data[:5]
```

```
Y_clas
```

```
array([3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 1, 1,
       1, 1, 1, 5, 5, 5, 5, 5, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 2, 2, 2, 2,
       2, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 5, 5, 5, 5, 5])
```

Classification

Testing multiple models

```
scores = defaultdict(list)
```

```
# Stratified k-Fold keeps the proportions of the classes
skf = StratifiedKFold(n_splits=10)
```

```
for train_index, test_index in skf.split(X_data, Y_clas):
  x_train, x_test = X_data[train_index], X_data[test_index]
  y_train, y_test = Y_clas[train_index], Y_clas[test_index]

  models = [
    {"name":"Random Forest", "model":RandomForestClassifier(n_estimators=10,
random_state=10)},
    {"name":"SVC\t", "model":svm.LinearSVC(random_state=10)},
    {"name":"Ada Boost", "model":AdaBoostClassifier(random_state=10)},
    {"name":"Bagging (KNN)", "model":BaggingClassifier(KNeighborsClassifier(),
random_state=10)},
    {"name":"Extra trees", "model":ExtraTreesClassifier(n_estimators=10,
random_state=10)},
    {"name":"Decision trees", "model":DecisionTreeClassifier(random_state=10)},
    {"name":"MLP\t", "model":MLPClassifier(random_state=10)},
  ]

  # Models are tested on the split and result is saved
  for model in models:
    # Train model
    model['model'].fit(x_train, y_train)

    # Save validation data
    scores[model['name']].append(model['model'].score(x_test, y_test))


for model, score in scores.items():
  print(f'Model: {model} \t Score: {np.mean(score):.2f} (+/-)
{np.std(score):.2f}')
```

UMAP

```
embedding = umap.UMAP(n_neighbors=5,
                      min_dist=0.3,
                      metric='correlation').fit_transform(X_data)
x = StandardScaler().fit_transform(embedding[:, 0].reshape(-1, 1))
y = embedding[:, 1]

class_color = {
    0:'b',
    1:'g',
    2:'r',
    3:'c',
    4:'m',
    5:'y'
}
c = list(map(lambda i: class_color[i], Y_clas))
```

```
plt.figure(facecolor='w',figsize=(16, 8), dpi=600)
plt.scatter(x, y, c=c)
plt.grid()
plt.savefig('umap');
scores = defaultdict(list)

# Stratified k-Fold keeps the proportions of the classes
skf = StratifiedKFold(n_splits=10)
for train_index, test_index in skf.split(X_data, Y_clas):

  # Embedding
  umap_obj = umap.UMAP(
      n_neighbors=5,
      min_dist=0.3,
      metric='correlation'
  )

  # Separate data
  x_train = umap_obj.fit_transform(X_data[train_index])
  x_test = umap_obj.transform(X_data[test_index])
  y_train, y_test = Y_clas[train_index], Y_clas[test_index]

  # Scale
  scaler = StandardScaler()
  x_train = scaler.fit_transform(x_train)
  x_test = scaler.transform(x_test)

  # Models
  models = [
    {"name":"Random Forest", "model":RandomForestClassifier(n_estimators=10,
random_state=10)},
    {"name":"SVC\t", "model":svm.LinearSVC(random_state=10)},
    {"name":"Ada Boost", "model":AdaBoostClassifier(random_state=10)},
    {"name":"Bagging (KNN)", "model":BaggingClassifier(KNeighborsClassifier(),
random_state=10)},
    {"name":"Extra trees", "model":ExtraTreesClassifier(n_estimators=10,
random_state=10)},
    {"name":"Decision trees", "model":DecisionTreeClassifier(random_state=10)},
    {"name":"MLP\t", "model":MLPClassifier(random_state=10)},
  ]

  # Models are tested on the split and result is saved
  for model in models:
    # Train model
    model['model'].fit(x_train, y_train)

    # Save validation data
    scores[model['name']].append(model['model'].score(x_test, y_test))
```

```
for model, score in scores.items():
  print(f'Model: {model} \t Score: {np.mean(score):.2f} (+/-)
{np.std(score):.2f}')
```

Regression

```
y_temp = StandardScaler().fit_transform(Y_clas.reshape(-1, 1))
X_regr = np.hstack((X_data,Y_clas.reshape(-1, 1)))
X_regr[:5]
```

```
Y_regr = sensor_data['Concentration'].values
Y_regr
```

```
array([ 1,  5, 10, 20, 50,  1,  5, 10, 20, 50,  1,  5, 10, 20, 50,  1,  5,
       10, 20, 50,  1,  5, 10, 20, 50,  1,  5, 10, 20, 50,  1,  5, 10, 20,
       50,  1,  5, 10, 20, 50,  1,  5, 10, 20, 50,  1,  5, 10, 20, 50,  1,
        5, 10, 20, 50,  1,  5, 10, 20, 50])
```

```
data = defaultdict(dict)
for class_n in range(6):
  idx = Y_clas == class_n
  data[class_n]['data'] = X_data[idx]
  data[class_n]['label'] = Y_regr[idx]
  data[class_n]['idx'] = idx
```

```
sensor_data.shape
```

```
(60, 8)
```

Regression with class

```
for key, value in data.items():
  label = label_encoder.inverse_transform((key,))[0]
  print(f'{label}')
  r_sq = defaultdict(list)
  mse = defaultdict(list)

  # Stratified k-Fold keeps the proportions of the classes
  skf = StratifiedKFold(n_splits=2)
  for train_index, test_index in skf.split(value['data'], value['label']):
    x_train, x_test = value['data'][train_index], value['data'][test_index]
    y_train, y_test = value['label'][train_index], value['label'][test_index]

    models = [
      {"name":"Random Forest", "model":RandomForestRegressor(n_estimators=10,
random_state=10)},
      {"name":"SVR\t", "model":svm.LinearSVR(random_state=10)},
```

```
      {"name":"Ada Boost", "model":AdaBoostRegressor(random_state=10)},
      {"name":"Bagging (KNN)", "model":BaggingRegressor(KNeighborsRegressor(),
random_state=10)},
      {"name":"Extra trees", "model":ExtraTreesRegressor(n_estimators=10,
random_state=10)},
      {"name":"Decision trees", "model":DecisionTreeRegressor(random_state=10)},
      {"name":"MLP\t", "model":MLPRegressor((50, 100, 50), random_state=10)},
      {"name":"Lasso\t", "model":Lasso(random_state=10)}
    ]

    # Models are tested on the split and result is saved
    for model in models:
      # Train model
      model['model'].fit(x_train, y_train)

      # Save r^2 data
      r_sq[model["name"]].append(model['model'].score(x_test, y_test))

      # Save mse data
      y_pred = model['model'].predict(x_test)
      mse[model["name"]].append(mean_squared_error(y_test, y_pred))


  for model, score in r_sq.items():
    print(
        f'Model: {model} \t R^2: {np.mean(score):.2f} (+/-) {np.std(score):.2f}'
\
        f' \t RMSE: {np.mean(np.sqrt(mse[model])):.2f} (+/-)
{np.std(np.sqrt(mse[model])):.2f}'
    )

models = [
    {"name":"Random Forest", "model":RandomForestRegressor(n_estimators=10,
random_state=10)},
    {"name":"Extra trees", "model":ExtraTreesRegressor(n_estimators=10,
random_state=10)},
    {"name":"Ada Boost", "model":AdaBoostRegressor(random_state=10)},
    {"name":"Decision trees", "model":DecisionTreeRegressor(random_state=10)},
    {"name":"Decision trees", "model":DecisionTreeRegressor(random_state=10)},
    {"name":"Extra trees", "model":ExtraTreesRegressor(n_estimators=10,
random_state=10)},
]

full_prediction = np.zeros(sensor_data.shape[0])


plt.figure(figsize=(16, 24))
for key, value in data.items():
  true_concentration = value['label']
```

```
  estimated = np.zeros(shape=true_concentration.shape)

  # Plotting predicted vs real concentration
  skf = StratifiedKFold(n_splits=2)
  for train_index, test_index in skf.split(value['data'], value['label']):
    x_train, x_test = value['data'][train_index], value['data'][test_index]
    y_train, y_test = value['label'][train_index], value['label'][test_index]

    model = models[key]

    # Train model
    model['model'].fit(x_train, y_train)

    # Get estimated data
    y_pred = model['model'].predict(x_test)
    estimated[test_index] = y_pred


  label = label_encoder.inverse_transform((key,))[0]
  full_prediction[value['idx']] = estimated

  # Line 1:1
  x = np.arange(51)
  # Plot
  plt.subplot(3, 2, key + 1)
  plt.style.use('ggplot')
  plt.scatter(true_concentration, estimated)
  plt.plot(x, x, label='1:1 Line')
  plt.title(f'{model["name"]} regression on {label}')
  plt.ylabel('Estimated concentration [ppm]')
  plt.xlabel('Real concentration [ppm]')
  plt.legend()

  sensor_data['Predicted concentration'] = full_prediction
plt.show()


sensor_data.to_csv('sensor_prediction.csv',index=False)
```

Ensamble

```
true_concentration = Y_regr
estimated = np.zeros(shape=Y_regr.shape)

r_sq = defaultdict(list)
mse = defaultdict(list)

# Stratified k-Fold keeps the proportions of the classes
skf = StratifiedKFold(n_splits=10)
```

```
for train_index, test_index in skf.split(X_regr, Y_regr):
  x_train, x_test = X_regr[train_index], X_regr[test_index]
  y_train, y_test = Y_regr[train_index], Y_regr[test_index]

  models = [
    {"name":"Ada Boost", "model":AdaBoostRegressor(random_state=10)},
    {"name":"Bagging (KNN)", "model":BaggingRegressor(KNeighborsRegressor(),
random_state=10)},
    {"name":"Extra trees", "model":ExtraTreesRegressor(n_estimators=10,
random_state=10)},
    {"name":"MLP\t", "model":MLPRegressor((50, 100, 50), random_state=10)}
  ]

  ensamble_models = [
#     {"name":"Random Forest", "model":RandomForestRegressor(n_estimators=10,
random_state=10)},
#     {"name":"SVR\t", "model":svm.LinearSVR(random_state=10)},
#     {"name":"Ada Boost", "model":AdaBoostRegressor(random_state=10)},
#     {"name":"Bagging (KNN)", "model":BaggingRegressor(KNeighborsRegressor(),
random_state=10)},
#     {"name":"Extra trees", "model":ExtraTreesRegressor(n_estimators=10,
random_state=10)},
#     {"name":"Decision trees", "model":DecisionTreeRegressor(random_state=10)},
#     {"name":"MLP\t", "model":MLPRegressor((50, 50), random_state=10)},
    {"name":"Lasso\t", "model":Lasso(random_state=10)}
  ]
  ensamble_train = {}
  ensamble_test = {}

  for model in models:
    # Train model
    model['model'].fit(x_train, y_train)

    ensamble_train[model['name']] = model['model'].predict(x_train)
    ensamble_test[model['name']] = model['model'].predict(x_test)

  ensamble_train = pd.DataFrame(ensamble_train)
  ensamble_test = pd.DataFrame(ensamble_test)

  for ensamble_model in ensamble_models:
    ensamble_model['model'].fit(ensamble_train, y_train)

    # Save r^2 data

r_sq[ensamble_model['name']].append(ensamble_model['model'].score(ensamble_test,
y_test))

    # Save mse data
    y_pred = ensamble_model['model'].predict(ensamble_test)
```

```
    estimated[test_index] = y_pred
    mse[ensamble_model['name']].append(mean_squared_error(y_test, y_pred))


for model, score in r_sq.items():
  print(
      f'Model: {model} \t R^2: {np.mean(score):.2f} (+/-) {np.std(score):.2f}' \
      f' \t RMSE: {np.mean(np.sqrt(mse[model])):.2f} (+/-)
{np.std(np.sqrt(mse[model])):.2f}'
  )

plt.figure(figsize=(8, 8))
plt.style.use('ggplot')
plt.scatter(estimated, true_concentration)
plt.plot(x, x, label='1:1 Line')
plt.title('Ensamble regression')
plt.xlabel('Estimated concentration [ppm]')
plt.ylabel('Real concentration [ppm]')
plt.legend()
plt.show()
```

Regression on UMAP with class

```
r_sq = defaultdict(list)
mse = defaultdict(list)

# Stratified k-Fold keeps the proportions of the classes
skf = StratifiedKFold(n_splits=10)
for train_index, test_index in skf.split(X_regr, Y_regr):
    # Embedding
  umap_obj = umap.UMAP(
      n_neighbors=5,
      min_dist=0.3,
      metric='correlation'
  )

  # Separate data
  x_train = umap_obj.fit_transform(X_regr[train_index])
  x_test = umap_obj.transform(X_regr[test_index])
  y_train, y_test = Y_regr[train_index], Y_regr[test_index]

  # Scale
  scaler = StandardScaler()
  x_train = scaler.fit_transform(x_train)
  x_test = scaler.transform(x_test)

  # Models
  models = [
```

```
    {"name":"Random Forest", "model":RandomForestRegressor(n_estimators=10,
random_state=10)},
    {"name":"SVR\t", "model":svm.LinearSVR(random_state=10)},
    {"name":"Ada Boost", "model":AdaBoostRegressor(random_state=10)},
    {"name":"Bagging (KNN)", "model":BaggingRegressor(KNeighborsRegressor(),
random_state=10)},
    {"name":"Extra trees", "model":ExtraTreesRegressor(n_estimators=10,
random_state=10)},
    {"name":"Decision trees", "model":DecisionTreeRegressor(random_state=10)},
    {"name":"MLP\t", "model":MLPRegressor(random_state=10)},
    {"name":"Lasso\t", "model":Lasso(random_state=10)}
  ]

  # Models are tested on the split and result is saved
  for model in models:
    # Train model
    model['model'].fit(x_train, y_train)

    # Save r^2 data
    r_sq[model["name"]].append(model['model'].score(x_test, y_test))

    # Save mse data
    y_pred = model['model'].predict(x_test)
    mse[model["name"]].append(mean_squared_error(y_test, y_pred))


for model, score in r_sq.items():
  print(
      f'Model: {model} \t R^2: {np.mean(score):.4f} (+/-) {np.std(score):.2f}' \
      f' \t RMSE: {np.mean(np.sqrt(mse[model])):.2f} (+/-)
{np.std(np.sqrt(mse[model])):.2f}'
  )
```

Regression without class

```
r_sq = defaultdict(list)
mse = defaultdict(list)

# Stratified k-Fold keeps the proportions of the classes
skf = StratifiedKFold(n_splits=10)
for train_index, test_index in skf.split(X_data, Y_regr):
  x_train, x_test = X_data[train_index], X_data[test_index]
  y_train, y_test = Y_regr[train_index], Y_regr[test_index]

  models = [
    {"name":"Random Forest", "model":RandomForestRegressor(n_estimators=10,
random_state=10)},
    {"name":"SVR\t", "model":svm.LinearSVR(random_state=10)},
```

```
    {"name":"Ada Boost", "model":AdaBoostRegressor(random_state=10)},
    {"name":"Bagging (KNN)", "model":BaggingRegressor(KNeighborsRegressor(),
random_state=10)},
    {"name":"Extra trees", "model":ExtraTreesRegressor(n_estimators=10,
random_state=10)},
    {"name":"Decision trees", "model":DecisionTreeRegressor(random_state=10)},
    {"name":"MLP\t", "model":MLPRegressor(random_state=10)},
  ]

  # Models are tested on the split and result is saved
  for model in models:
    # Train model
    model['model'].fit(x_train, y_train)

    # Save r^2 data
    r_sq[model["name"]].append(model['model'].score(x_test, y_test))

    # Save mse data
    y_pred = model['model'].predict(x_test)
    mse[model["name"]].append(mean_squared_error(y_test, y_pred))


for model, score in r_sq.items():
  print(
      f'Model: {model} \t R^2: {np.mean(score):.2f} (+/-) {np.std(score):.2f}' \
      f' \t RMSE: {np.mean(np.sqrt(mse[model])):.2f} (+/-)
{np.std(np.sqrt(mse[model])):.2f}'
  )
```

Regression on UMAP without class

```
r_sq = defaultdict(list)
mse = defaultdict(list)

# Stratified k-Fold keeps the proportions of the classes
skf = StratifiedKFold(n_splits=10)
for train_index, test_index in skf.split(X_data, Y_regr):

  # Embedding
  umap_obj = umap.UMAP(
      n_neighbors=5,
      min_dist=0.3,
      metric='correlation'
  )

  # Separate data
  x_train = umap_obj.fit_transform(X_data[train_index])
  x_test = umap_obj.transform(X_data[test_index])
  y_train, y_test = Y_regr[train_index], Y_regr[test_index]
```

```
  # Scale
  scaler = StandardScaler()
  x_train = scaler.fit_transform(x_train)
  x_test = scaler.transform(x_test)

  # Models
  models = [
    {"name":"Random Forest", "model":RandomForestRegressor(n_estimators=10,
random_state=10)},
    {"name":"SVR\t", "model":svm.LinearSVR(random_state=10)},
    {"name":"Ada Boost", "model":AdaBoostRegressor(random_state=10)},
    {"name":"Bagging (KNN)", "model":BaggingRegressor(KNeighborsRegressor(),
random_state=10)},
    {"name":"Extra trees", "model":ExtraTreesRegressor(n_estimators=10,
random_state=10)},
    {"name":"Decision trees", "model":DecisionTreeRegressor(random_state=10)},
    {"name":"MLP\t", "model":MLPRegressor(random_state=10)},
  ]

  # Models are tested on the split and result is saved
  for model in models:
    # Train model
    model['model'].fit(x_train, y_train)

    # Save r^2 data
    r_sq[model["name"]].append(model['model'].score(x_test, y_test))

    # Save mse data
    y_pred = model['model'].predict(x_test)
    mse[model["name"]].append(mean_squared_error(y_test, y_pred))


for model, score in r_sq.items():
  print(
      f'Model: {model} \t R^2: {np.mean(score):.4f} (+/-) {np.std(score):.2f}' \
      f' \t RMSE: {np.mean(np.sqrt(mse[model])):.2f} (+/-)
{np.std(np.sqrt(mse[model])):.2f}'
  )
```

Regression with class on UMAP concatenated with data

```
nsr_sq = defaultdict(list)
mse = defaultdict(list)

# Stratified k-Fold keeps the proportions of the classes
skf = StratifiedKFold(n_splits=10)
for train_index, test_index in skf.split(X_regr, Y_regr):
```

```
  # Embedding
  umap_obj = umap.UMAP(
      n_neighbors=5,
      min_dist=0.3,
      metric='correlation'
  )

  # Separate data
  x_train = umap_obj.fit_transform(X_regr[train_index])
  x_test = umap_obj.transform(X_regr[test_index])
  y_train, y_test = Y_regr[train_index], Y_regr[test_index]

  # Append data
  x_train = np.hstack((X_regr[train_index], x_train))
  x_test  = np.hstack((X_regr[test_index ], x_test ))

  # Scale
  scaler = StandardScaler()
  x_train = scaler.fit_transform(x_train)
  x_test = scaler.transform(x_test)

  # Models
  models = [
    {"name":"Random Forest", "model":RandomForestRegressor(n_estimators=10,
random_state=10)},
    {"name":"SVR\t", "model":svm.LinearSVR(random_state=10)},
    {"name":"Ada Boost", "model":AdaBoostRegressor(random_state=10)},
    {"name":"Bagging (KNN)", "model":BaggingRegressor(KNeighborsRegressor(),
random_state=10)},
    {"name":"Extra trees", "model":ExtraTreesRegressor(n_estimators=10,
random_state=10)},
    {"name":"Decision trees", "model":DecisionTreeRegressor(random_state=10)},
    {"name":"MLP\t", "model":MLPRegressor(random_state=10)},
    {"name":"Lasso\t", "model":Lasso(random_state=10)}
  ]

  # Models are tested on the split and result is saved
  for model in models:
    # Train model
    model['model'].fit(x_train, y_train)

    # Save r^2 data
    r_sq[model["name"]].append(model['model'].score(x_test, y_test))

    # Save mse data
    y_pred = model['model'].predict(x_test)
    mse[model["name"]].append(mean_squared_error(y_test, y_pred))
```

```
for model, score in r_sq.items():
  print(
      f'Model: {model} \t R^2: {np.mean(score):.4f} (+/-) {np.std(score):.2f}' \
      f' \t RMSE: {np.mean(np.sqrt(mse[model])):.2f} (+/-)
{np.std(np.sqrt(mse[model])):.2f}'
  )
```