Establishing predicting model to predict the bandgap of crystals

UG Project Report

Introduction

Since the late 20th century, heavy growth of the electronics and communication sector has led to numerous innovations in the semiconductor industry. Every day newer and more efficient semiconductors are being discovered. The main property the scientists are targeting is the material's conductivity, which is directly related to the bandgap.

The concept of bandgap emerge from solid-state physics: Every solid contain electrons, The critical part is electrical conductivity, the response of electron under applied electric field as electrons are arranged in energy band separated by region for which no wavelike orbital exists such forbidden region is known as energy gap or band gaps or in other simpler words band gaps can be referred as the energy difference between the top of the valence band and the bottom of the conduction band in insulators and semiconductors.

COMPLEXITIES IN BAND GAP PREDICTION

The bandgap being an intrinsic property is calculated using experiments like UV spectroscopy or differential and cyclic voltammetry. These experiments require large, expensive equipment and are tedious. Some materials are difficult to handle and need particular environments to perform these experiments. Calculating band gaps accurately is still one of the unsolved problems in solid-state physics. There is no equation or immediate solution to calculate band gaps accurately. To help overcome these difficulties, we propose machine learning models to recognize meaningful patterns in bandgap values across thousands of compounds and their chemical properties. Predicting the bandgap, or more explicitly predicting the range in which the bandgap of new material would lie between, would give researchers and students a good idea of what to expect in the experiments before going for more elaborate ways bandgap calculation.

Descriptors taken into account

Atomic number : The atomic number (represented by the letter Z) *of an element is the number of protons in the nucleus of each atom of that element*.
{ `atomic_numbers` }

Lattice parameter : Refers to the physical dimension of unit cells in a crystal lattice. Lattices in three dimensions generally have three lattice constants, referred to as *a*, *b*, and *c*.

```
{ length:  a_parameters, b_parameters ,c_parameters
 Angles : alpha_parameters, beta_parameters, gamma_parameters }
```

Van der Waals radius is a measure for the size of an atom that is not chemically (ionically or covalently) bound. In general a van der Waals radius is defined as half the closest distance of two equal, non-covalently bound, atoms.
{ van_der_waals_radius }

Electrical resistivity is a fundamental property of a material that quantifies how strongly it resists electric current. Its inverse, called electrical conductivity, quantifies how well a material conducts electricity.
{ electrical_resistivity }

Poisson's ratio is a measure of the Poisson effect, the deformation (expansion or contraction) of a material in directions perpendicular to the specific direction of loading.
{ poissons_ratio }

Molar Volume is the volume of one mole of a substance at a specified pressure and temperature.
{ molar_volume }

Thermal conductivity refers to the intrinsic ability of a material to transfer or conduct heat.
{ thermal_conductivity }

Melting point is usually defined as the point at which materials changes from a solid to a liquid.
{ melting_point}

Critical temperature is the temperature of a substance that can be defined as the highest temperature at which the substance can exist as a liquid.
{ critical_temperature }

Bulk modulus of a substance is a measure of how resistant to compression that substance is. It is defined as the ratio of the infinitesimal pressure increase to the resulting *relative* decrease of the volume.
{ bulk_modulus }

Young's modulus is a mechanical property that measures the tensile stiffness of a solid material. It gives a relationship between tensile stress and axial strain in the linear elastic region of a material.
{youngs_modulus}

Brinell hardness number (HB) is the load divided by the surface area of the indentation. The diameter of the impression is measured with a microscope with a superimposed scale.
{ brinell_hardness }

Rigidity modulus is a measure of the elastic shear stiffness of a material and is defined as the ratio of shear stress to the shear strain.
{ `rigidity_modulus` }

Vickers hardness is a measure of the hardness of a material, calculated from the size of an impression produced under load by a pyramid-shaped diamond indenter.
{ `vickers_hardness` }

Density of substance is its mass per unit volume.
{ `density_of_solid` }

Coefficient of Linear Thermal Expansion is a material property which characterizes the ability of a plastic to expand under the effect of temperature elevation. It tells you how much the developed part will remain dimensionally stable under temperature variations.
{ `coefficient_of_linear_thermal_expansion` }

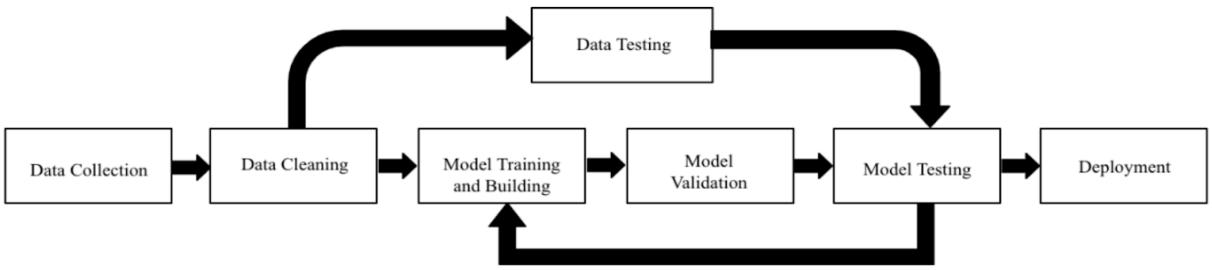
Ionic radius is a measure of the size of the atom's ion, the average value is taken in account. Similarly average value of cationic radius and anionic radius is also taken.
{`average_ionic_radius, average_cationic_radius,average_anionic_radius`}

OBJECTIVES

The objective is to build a decent (near accurate) model to determine materials properties using machine learning algorithms.

Major steps taken

- Data extraction
- Data cleaning and visualization
- Factor dependencies
- Building model



Machine Learning Process Sequence

Data extraction

Data is mined from open sources enabling us to programmatically access several thousand training examples whose properties were calculated properly and consistently by a single research group [MaterialsProject (MP)] database; we can find a massive amount of material science data generated using density functional theory (DFT). About 131,000+ inorganic crystals are currently present in the database, along with the DFT-calculated properties of these materials.

The second part is more concerned with selecting the subset of all available data that will be considered working with; the data regarding the various dependencies and its bandage energy is obtained from a REST API of MaterialsProject (MP). The obtained subset data has 36000 rows of different systems. The data of compound systems and their bandgap energy is collected in a CSV format for further processing and usage. The generated CSV file contains 36000 compound systems and their bandgap energy.

Data cleaning and visualization

Cleaning data is the removal of noisy data or fixing missing samples/datasets. Incomplete datasets and outliers are addressed before it is used to build models. There may be sensitive information in some of the attributes and these attributes may need to be anonymized or removed from the data entirely.

Data transformation: The compound system is transformed into various features that depicts the model. The following are the features that are considered:

- All the descriptors are taken into account has been discussed earlier.

To derive these features from a single compound, Pymatgen is used. Pymatgen (Python Materials Genomics) is a robust, open-source Python library for materials analysis.
The following are some of the main features of Pymatgen:

- Highly flexible classes for the representation of Element, Site, Molecule, Structure objects.
- Powerful analysis tools, including generation of phase diagrams, Pourbaix diagrams, diffusion analyses, reactions, etc.
- Electronic structure analyses, such as density of states and band structure.
- Integration with the Materials Project REST API.

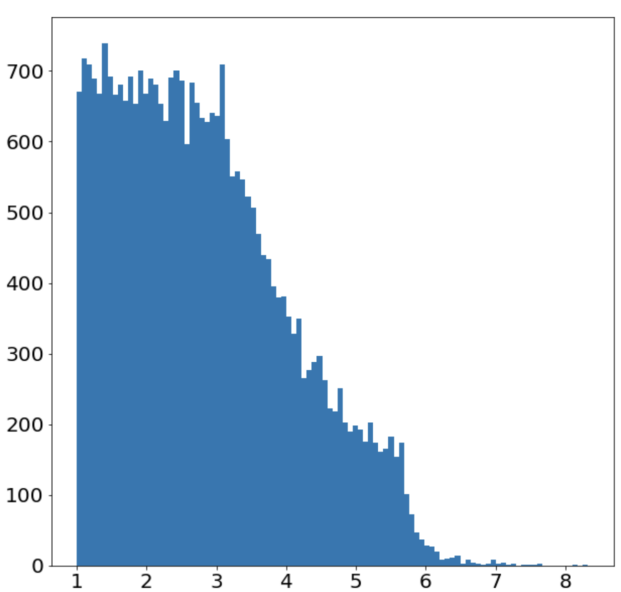
NumPy is used for data exploration and manipulation. NumPy is the fundamental package needed for scientific computing with Python. This package contains:

- A powerful N-dimensional array object.
- Sophisticated (broadcasting) functions.
- Tools for integrating C/C++ and Fortran code.
- Useful linear algebra, Fourier transform, and random number capabilities.

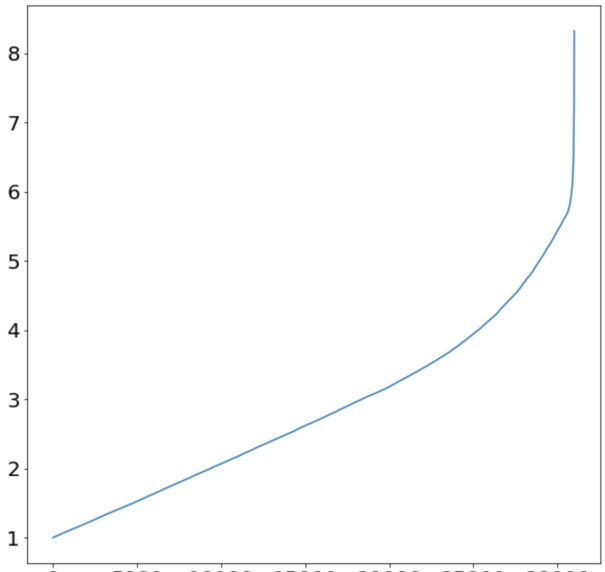
Matplotlib is a python 2D plotting and data visualization library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.

The following are some of the main features of Matplotlib:

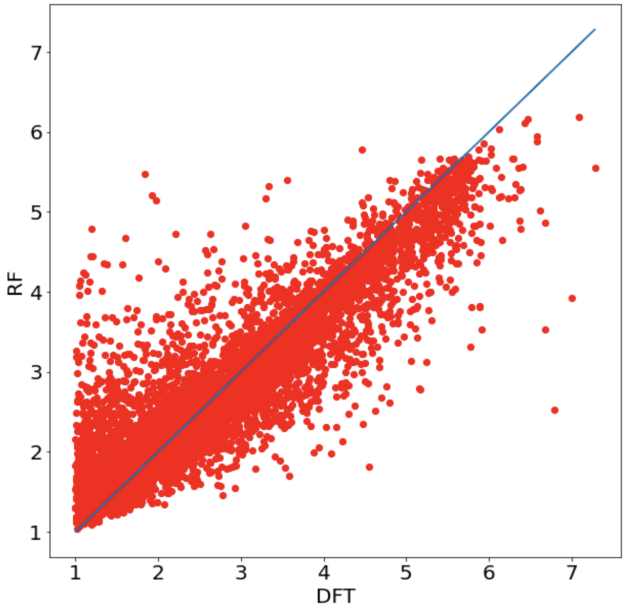
- Very fast when processing large datasets.
- Easier to manipulate plot details.
- Standalone program: no external dependencies.
- Deep integration with Python.



1



2



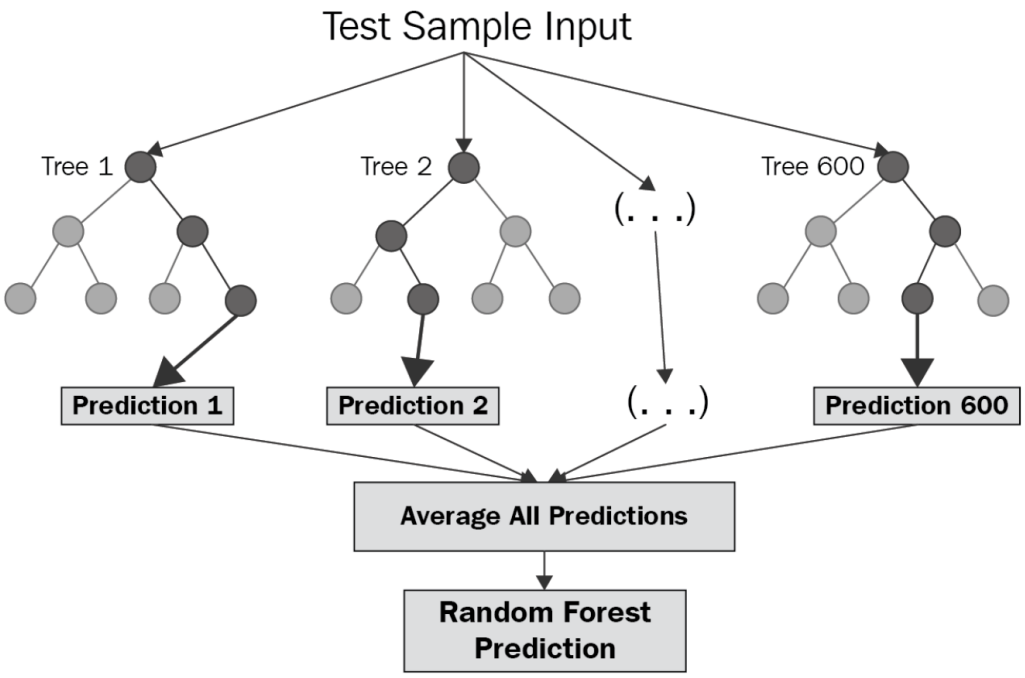
3

Fig (1,2): Distribution of data according to band gaps .
FIg 3: Correlation between predicted value and DFT value.

Building model

Random Forest Regression

Random forests is a notion of the general technique of random decision forests that are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set.



Random Forest Structure

Features

The main features contributing to the model were selected and passed to Random forest Regression, the feature that are taken in account are based on the simple mathematical principles consisting of :

- Mean value of each descriptor.
- Maximum value of each descriptor
- Minimum value of each descriptor.
- standard deviation value of each descriptor.

Result and Conclusion

Ever since the advent of data science and efficient machine learning tools, diverse fields have taken advantage and have been using them to solve domain-specific problems. There are a lot of unsolved problems in material science, and this provides us an opportunity to experiment with unique ways of approaching the issues. This is one such attempt, this project does not target replacing experimentation to calculate bandgap with a simple mathematical model; instead, it explores the possibility of the use of machine learning to assist and validate experimentation.

The result obtained from the model is fascinating. It receives an accuracy of about **78.3001**%.

Machine learning is indeed a potent tool for analyzing materials data; when used correctly, it can produce powerful, original insights. But it is just that: another tool in the materials scientist's toolbox. Machine learning (or any other computational technique) is not a substitute for scientific judgment or common sense.

However, with a combination of robust training data and insightful features, we can build speedy and very effective models of materials behavior without a supercomputer and without lengthy and expensive scientific procedures.

References

- https://research.ijcaonline.org/volume81/number12/pxc3892245.pdf
- https://statistics.berkeley.edu/sites/default/files/tech-reports/666.pdf
- Introduction to solid state physics (Charles Kittle)
- Solid state physics (Neil W Ashcroft, N David Mermin)
- https://pymatgen.org
- https://materialsproject.org
- https://numpy.org

Appendix: Source code

```
!pip3 install pymatgen
!pip3 install xgboost
!pip3 install sklearn pandas
from pymatgen.io.cif import CifParser
from urllib.request import urlopen
import pandas as pd
from pymatgen.ext.matproj import MPRester
from pymatgen.ext.matproj import MPRestError
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
import numpy as np

m = MPRester("XXXXXXXXXXXXXXX")
results = m.query({"elements":{"$all": ["O"]},"band_gap": {"$gt": 1}},
   properties=["cif", "band_gap"])


atomic_numbers = []
a_parameters = []
b_parameters = []
c_parameters = []
alpha_parameters = []
beta_parameters = []
gamma_parameters = []
distance_matrix = []
band_gaps = []
van_der_waals_radius = []
electrical_resistivity = []
velocity_of_sound = []
reflectivity = []
```

```
poissons_ratio = []
molar_volume = []
thermal_conductivity = []
melting_point = []
critical_temperature = []
superconduction_temperature = []
liquid_range = []
bulk_modulus = []
youngs_modulus = []
brinell_hardness = []
rigidity_modulus = []
vickers_hardness = []
density_of_solid = []
coefficient_of_linear_thermal_expansion = []
average_ionic_radius = []
average_cationic_radius = []
average_anionic_radius = []

imp = SimpleImputer(missing_values=None, strategy='constant',fill_value=0)

for r in results:
   cif = r['cif']
   bg = r['band_gap']
   if bg > 0.1:
     parser = CifParser.from_string(cif)

     structure = parser.get_structures()
     structure = structure[0]

     numElements = len(structure.atomic_numbers)

     atomic_numbers +=
   [[np.mean(structure.atomic_numbers),np.max(structure.atomic_numbers),np
   .min(structure.atomic_numbers),np.std(structure.atomic_numbers)]]

     # Lattice parameters:
     a_parameters += [structure.lattice.abc[0]]
     b_parameters += [structure.lattice.abc[1]]
     c_parameters += [structure.lattice.abc[2]]
     alpha_parameters += [structure.lattice.angles[0]]
```

```
  beta_parameters += [structure.lattice.angles[1]]
  gamma_parameters += [structure.lattice.angles[2]]

  distance_matrix +=
[[np.mean(structure.distance_matrix),np.max(structure.distance_matrix),
np.min(structure.distance_matrix),np.std(structure.distance_matrix)]]


e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,e11,e12,e13,e14,e15,e16,e17,e18,e19,e20,
e21,e22,e23=[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]
,[],[],[]
  for e in structure.species:
      e1 +=[e.van_der_waals_radius]
      e2 +=[e.electrical_resistivity]
      e3 +=[e.velocity_of_sound]
      e4 +=[e.reflectivity]
      e6 +=[e.poissons_ratio]
      e7 +=[e.molar_volume]
      e8 +=[e.thermal_conductivity]
      e9 +=[e.melting_point]
      e10 +=[e.critical_temperature ]
      e11 +=[e.superconduction_temperature ]
      e12 +=[e.liquid_range ]
      e13 +=[e.bulk_modulus ]
      e14 +=[e.youngs_modulus ]
      e15 +=[e.brinell_hardness ]
      e16 +=[e.rigidity_modulus ]
      # e17 +=[e.mineral_hardness ]
      e18 +=[e.vickers_hardness ]
      e19 +=[e.density_of_solid ]
      e20 +=[e.coefficient_of_linear_thermal_expansion ]
      e21 +=[e.average_ionic_radius ]
      e22 +=[e.average_cationic_radius ]
      e23 +=[e.average_anionic_radius ]

  e1=[0 if v is None else v for v in e1]
  e2=[0 if v is None else v for v in e2]
  e3=[0 if v is None else v for v in e3]
  e4=[0 if v is None else v for v in e4]
  # e5=[0 if v is None else v for v in e5]
```

```
  e6=[0 if v is None else v for v in e6]
  e7=[0 if v is None else v for v in e7]
  e8=[0 if v is None else v for v in e8]
  e9=[0 if v is None else v for v in e9]
  e10=[0 if v is None else v for v in e10]
  e11=[0 if v is None else v for v in e11]
  e12=[0 if v is None else v for v in e12]
  e13=[0 if v is None else v for v in e13]
  e14=[0 if v is None else v for v in e14]
  e15=[0 if v is None else v for v in e15]
  e16=[0 if v is None else v for v in e16]
  # e17=[0 if v is None else v for v in e17]
  e18=[0 if v is None else v for v in e18]
  e19=[0 if v is None else v for v in e19]
  e20=[0 if v is None else v for v in e20]
  e21=[0 if v is None else v for v in e21]
  e22=[0 if v is None else v for v in e22]
  e23=[0 if v is None else v for v in e23]

  van_der_waals_radius +=
[[np.mean(e1),np.max(e1),np.min(e1),np.std(e1)]]
  electrical_resistivity +=
[[np.mean(e2),np.max(e2),np.min(e2),np.std(e2)]]
  velocity_of_sound += [[np.mean(e3),np.max(e3),np.min(e3),np.std(e3)]]
  reflectivity += [[np.mean(e4),np.max(e4),np.min(e4),np.std(e4)]]
  poissons_ratio += [[np.mean(e6),np.max(e6),np.min(e6),np.std(e6)]]
  molar_volume += [[np.mean(e7),np.max(e7),np.min(e7),np.std(e7)]]
  thermal_conductivity +=
[[np.mean(e8),np.max(e8),np.min(e8),np.std(e8)]]
  melting_point += [[np.mean(e9),np.max(e9),np.min(e9),np.std(e9)]]
  critical_temperature +=
[[np.mean(e10),np.max(e10),np.min(e10),np.std(e10)]]
  superconduction_temperature +=
[[np.mean(e11),np.max(e11),np.min(e11),np.std(e11)]]
  liquid_range += [[np.mean(e12),np.max(e12),np.min(e12),np.std(e12)]]
  bulk_modulus += [[np.mean(e13),np.max(e13),np.min(e13),np.std(e13)]]
  youngs_modulus +=
[[np.mean(e14),np.max(e14),np.min(e14),np.std(e14)]]
  brinell_hardness +=
[[np.mean(e15),np.max(e15),np.min(e15),np.std(e15)]]
```

```
      rigidity_modulus +=
    [[np.mean(e16),np.max(e16),np.min(e16),np.std(e16)]]
      vickers_hardness +=
    [[np.mean(e18),np.max(e18),np.min(e18),np.std(e18)]]
      density_of_solid +=
    [[np.mean(e19),np.max(e19),np.min(e19),np.std(e19)]]
      coefficient_of_linear_thermal_expansion +=
    [[np.mean(e20),np.max(e20),np.min(e20),np.std(e20)]]
      average_ionic_radius +=
    [[np.mean(e21),np.max(e21),np.min(e21),np.std(e21)]]
      average_cationic_radius +=
    [[np.mean(e22),np.max(e22),np.min(e22),np.std(e22)]]
      average_anionic_radius +=
    [[np.mean(e23),np.max(e23),np.min(e23),np.std(e23)]]

      band_gaps += [bg]


print(len(band_gaps))

plt.rcParams.update({'font.size': 20})

plt.figure(figsize=(10, 10))
plt.hist(band_gaps, bins=100)
plt.savefig('Histogram_PDF_NoMetals', bbox_inches='tight')

band_gaps_sorted=sorted(band_gaps)

# Scatter plot
plt.figure(figsize=(10,10))
plt.plot(band_gaps_sorted)
plt.ylabel('')
plt.xlabel('')
plt.savefig('ScatterPlot_NoMetals', bbox_inches='tight')



# This histogram shows that almost half of our data are metals (zero
   bandgap).
```

```
# Next, we create a pandas DataFrame object
a_parameters=np.array(a_parameters)
b_parameters=np.array(b_parameters)
c_parameters=np.array(c_parameters)
atomic_numbers=np.array(atomic_numbers).transpose()
distance_matrix=np.array(distance_matrix).transpose()
van_der_waals_radius=np.array(van_der_waals_radius).transpose()
electrical_resistivity=np.array(electrical_resistivity).transpose()
velocity_of_sound=np.array(velocity_of_sound).transpose()
reflectivity=np.array(reflectivity).transpose()
poissons_ratio=np.array(poissons_ratio).transpose()
molar_volume=np.array(molar_volume).transpose()
thermal_conductivity=np.array(thermal_conductivity).transpose()
melting_point=np.array(melting_point).transpose()
critical_temperature=np.array(critical_temperature).transpose()
superconduction_temperature=np.array(superconduction_temperature).transpos
   e()
liquid_range=np.array(liquid_range).transpose()
bulk_modulus=np.array(bulk_modulus).transpose()
youngs_modulus=np.array(youngs_modulus).transpose()
brinell_hardness=np.array(brinell_hardness).transpose()
rigidity_modulus=np.array(rigidity_modulus).transpose()
vickers_hardness=np.array(vickers_hardness).transpose()
density_of_solid=np.array(density_of_solid).transpose()
coefficient_of_linear_thermal_expansion=np.array(coefficient_of_linear_the
   rmal_expansion).transpose()
average_ionic_radius=np.array(average_ionic_radius).transpose()
average_cationic_radius=np.array(average_cationic_radius).transpose()
average_anionic_radius=np.array(average_anionic_radius).transpose()

dataset_df = pd.DataFrame({\
                           "mean_atomic_numbers": atomic_numbers[0],
                           "max_atomic_numbers": atomic_numbers[1],
                           "min_atomic_numbers": atomic_numbers[2],
                           "std_atomic_numbers": atomic_numbers[3],
                           "a_parameters": a_parameters,
                           "b_parameters": b_parameters,
                           "c_parameters": c_parameters,
                           "V":a_parameters*b_parameters*c_parameters,
                           "alpha_parameters": alpha_parameters,
```

```
                        "beta_parameters": beta_parameters,
                        "gamma_parameters": gamma_parameters,
                        "mean_distance_matrix": distance_matrix[0],
                        "max_distance_matrix": distance_matrix[1],
                        "min_distance_matrix": distance_matrix[2],
                        "std_distance_matrix": distance_matrix[3],

    "mean_van_der_waals_radius":van_der_waals_radius[0],

    "max_van_der_waals_radius":van_der_waals_radius[1],

    "min_van_der_waals_radius":van_der_waals_radius[2],

    "std_van_der_waals_radius":van_der_waals_radius[3],

    "mean_electrical_resistivity":electrical_resistivity[0],

    "max_electrical_resistivity":electrical_resistivity[1],

    "min_electrical_resistivity":electrical_resistivity[2],

    "std_electrical_resistivity":electrical_resistivity[3],
                        "mean_velocity_of_sound":velocity_of_sound[0],
                        "max_velocity_of_sound":velocity_of_sound[1],
                        "min_velocity_of_sound":velocity_of_sound[2],
                        "std_velocity_of_sound":velocity_of_sound[3],
                        "mean_reflectivity":reflectivity[0],
                        "max_reflectivity":reflectivity[1],
                        "min_reflectivity":reflectivity[2],
                        "std_reflectivity":reflectivity[3],
                        "mean_poissons_ratio":poissons_ratio[0],
                        "max_poissons_ratio":poissons_ratio[1],
                        "min_poissons_ratio":poissons_ratio[2],
                        "std_poissons_ratio":poissons_ratio[3],
                        "mean_molar_volume":molar_volume[0],
                        "max_molar_volume":molar_volume[1],
                        "min_molar_volume":molar_volume[2],
                        "std_molar_volume":molar_volume[3],

    "mean_thermal_conductivity":thermal_conductivity[0],
```

```
"max_thermal_conductivity":thermal_conductivity[1],

"min_thermal_conductivity":thermal_conductivity[2],

"std_thermal_conductivity":thermal_conductivity[3],
                        "mean_melting_point":melting_point[0],
                        "max_melting_point":melting_point[1],
                        "min_melting_point":melting_point[2],
                        "std_melting_point":melting_point[3],

"mean_critical_temperature":critical_temperature[0],

"max_critical_temperature":critical_temperature[1],

"min_critical_temperature":critical_temperature[2],

"std_critical_temperature":critical_temperature[3],

"mean_superconduction_temperature":superconduction_temperature[0],

"max_superconduction_temperature":superconduction_temperature[1],

"min_superconduction_temperature":superconduction_temperature[2],

"std_superconduction_temperature":superconduction_temperature[3],
                        "mean_liquid_range":liquid_range[0],
                        "max_liquid_range":liquid_range[1],
                        "min_liquid_range":liquid_range[2],
                        "std_liquid_range":liquid_range[3],
                        "mean_bulk_modulus":bulk_modulus[0],
                        "max_bulk_modulus":bulk_modulus[1],
                        "min_bulk_modulus":bulk_modulus[2],
                        "std_bulk_modulus":bulk_modulus[3],
                        "mean_youngs_modulus":youngs_modulus[0],
                        "max_youngs_modulus":youngs_modulus[1],
                        "min_youngs_modulus":youngs_modulus[2],
                        "std_youngs_modulus":youngs_modulus[3],
                        "mean_brinell_hardness":brinell_hardness[0],
                        "max_brinell_hardness":brinell_hardness[1],
```

```
                        "min_brinell_hardness":brinell_hardness[2],
                        "std_brinell_hardness":brinell_hardness[3],
                        "mean_rigidity_modulus":rigidity_modulus[0],
                        "max_rigidity_modulus":rigidity_modulus[1],
                        "min_rigidity_modulus":rigidity_modulus[2],
                        "std_rigidity_modulus":rigidity_modulus[3],
                        "mean_vickers_hardness":vickers_hardness[0],
                        "max_vickers_hardness":vickers_hardness[1],
                        "min_vickers_hardness":vickers_hardness[2],
                        "std_vickers_hardness":vickers_hardness[3],
                        "mean_density_of_solid":density_of_solid[0],

"mean_coefficient_of_linear_thermal_expansion":coefficient_of_linear_th
ermal_expansion[0],

"mean_coefficient_of_linear_thermal_expansion":coefficient_of_linear_th
ermal_expansion[0],

"mean_coefficient_of_linear_thermal_expansion":coefficient_of_linear_th
ermal_expansion[0],

"mean_coefficient_of_linear_thermal_expansion":coefficient_of_linear_th
ermal_expansion[0],

"mean_average_ionic_radius":average_ionic_radius[0],

"max_average_ionic_radius":average_ionic_radius[1],

"min_average_ionic_radius":average_ionic_radius[2],

"std_average_ionic_radius":average_ionic_radius[3],

"mean_average_cationic_radius":average_cationic_radius[0],

"max_average_cationic_radius":average_cationic_radius[1],

"min_average_cationic_radius":average_cationic_radius[2],

"std_average_cationic_radius":average_cationic_radius[3],
```

```
    "mean_average_anionic_radius":average_anionic_radius[0],

    "max_average_anionic_radius":average_anionic_radius[1],

    "min_average_anionic_radius":average_anionic_radius[2],

    "std_average_anionic_radius":average_anionic_radius[3]
                            })

# We need to normalize the data using a scaler
# Define the scaler
scaler = StandardScaler().fit(dataset_df)

# Scale the train set
scaled_dataset_df = scaler.transform(dataset_df)
# Then we do a 80/20 split of data: training set and test set

X_train_scaled, X_test_scaled, y_train, y_test = train_test_split(
    scaled_dataset_df, band_gaps, test_size=.2, random_state=None)


regr = RandomForestRegressor(n_estimators=300, max_depth=200,
    random_state=0)
regr.fit(X_train_scaled, y_train)
y_predicted = regr.predict(X_test_scaled)

print('RF MSE\t'+str(mean_squared_error(y_test, y_predicted))+'\n')
print('RF R2\t'+str(r2_score(y_test, y_predicted))+'\n')
```