Materials Descriptors for Data Science

In this module
• Enhance data using descriptors (this lecture)
• Analyze descriptors, calculate and visualize correlations (this lecture)
• Hands on tutorial using nanoHUB: modeling melting temperatures
  • Homework assignment

Juan C. Verduzco, Zachary D. McClure, and Alejandro Strachan
jverduzc@purdue.edu || zmcclure@purdue.edu || strachan@purdue.edu
School of Materials Engineering & Network for Computational Nanotechnology
Purdue University
West Lafayette, Indiana USA
Learning objectives and prerequisites

After completing this lecture you will:

• Enhance materials data using descriptors
  • Periodic table data
  • Surrogate properties and physics-based models
• Analyze descriptors, calculate correlations to rank descriptors

Pre-requisites:
• Basic programming skills
Use of descriptors vs. deep learning

Descriptors → Deep Learning → Output

Launch *featureselect* tool in nanoHUB

Materials descriptors for predictive machine learning models: application to melting temperature predictions
Zachary D. McClure and Alejandro Strachan,
Materials Engineering, Purdue University.

Overview
These notebooks explore the importance of using appropriate descriptors (or features) of materials to develop predictive models using machine learning. Ideal descriptors capture important aspects of the output quantity of interest and are easy to calculate. Appropriate descriptors significantly improve model accuracy and enable the development of models where data is scarce.

These notebooks introduce materials descriptors in the context of building models to predict the melting temperature of oxides. In addition to simple features, we will explore the addition of materials properties expected to correlate with the quantity of interest and physical laws as inputs to the models to improve accuracy.

Getting started and hints
- Click on the links below to access each notebook.
- Remember to click “Shift+Return” or click “Run” from the top bar to run each cell.
- To exit individual notebooks and return to this page, use File -> Close and Halt. “Terminate Session” (top right) will kill your entire Jupyter session.
- The notebooks will be read only, you can make your own copy by downloading the notebook file (Menu File -> Download as... -> notebook.ipynb). You can upload this notebook to your main Jupyter directory within nanoHUB to run it again.

Notebooks
- **Role of periodic table descriptors and properties in models for melting temperature**
  - Train Random Forest models to predict the melting temperature of oxides
  - The notebook shows how adding meaningful descriptors improves model accuracy
- **Quantifying descriptors: Pearson correlations**
  - Perform Pearson Correlation analysis for features associated with the melting point
  - Split Materials features into separate groups, and rank all descriptors at the end

Launch first notebook
Obtaining and exploring initial data

Read dataset and display it

```
In [2]:
# Use Pandas to read data in included csv file
# ZD McClure, A Strachan JOM, 1-13 (2020)
oxide_melting = pd.read_csv('Oxide_Melting_Point_Training.csv')
# Remove duplicate entries
oxide_melting.drop_duplicates(subset ="Formula", keep = 'first', inplace = True)
# Let's visualize the data, analyze the various columns
display(oxide_melting)
```

Elements and formula

Experimental melting temperature: this is what we want to predict
Train a model with modest descriptors

2. Train a model using few descriptors available in the input data

For this first model we will use only four descriptors included in the input data (in the csv file):

- Ionic packing fraction,
- Density,
- Space group (crystal structure info),
- Molar volume

```python
In [4]:
MAE_FINAL = []
FEATURE_LEN = []

In [5]:
# Create an array of output properties (this is what we want to predict, i.e. melting temp) from the pandas dataframe
# Note that T_melt is the name of the column with the melting temps
output_properties = np.array([oxide_melting.T_melt])

# Create an array with the input properties
# These are the properties in our data file:
# IPF is packing fraction, density, space group (crystal structure info), and molar volume
input_properties = np.array([oxide_melting.IPF, oxide_melting.Density, oxide_melting['spacegroup.number'],
                             oxide_melting['Molar.Volume']])

# It is common practice to normalize input properties so that they all have similar magnitudes
# Here we subtract the mean and divide by the standard deviation
means = np.mean(input_properties, axis=0)
stdevs = np.std(input_properties, axis=0)
for j in range(0, len(input_properties)):
    input_properties[j] = (input_properties[j] - means)/stdevs;

# nan_to_num: Replace NaN with zero and infinity with large finite numbers
input_properties = np.nan_to_num(input_properties)

# Create a dictionary with the data
properties_dictionary = {
    'inputs': input_properties,
    'outputs': output_properties,
    'labels': oxide_melting.Formula,
    'length': output_properties.shape[0]
}

# This prints the size of the input array:
# We have 4 descriptors for each oxide and 157 oxides
print(np.shape(input_properties))
(157, 4)
```

Create arrays of input and output data

Normalize data

Input array is 157x4
(number of datapoints x number of descriptors)
Train a model with modest descriptors

We train 10 independent random to collect statistics

We will train RFs to try to predict melting temperatures from the initial four inputs
We will do this several times with different random sets of training and testing data

```python
import numpy as np

# Generate 10 random numbers to be used as seeds to shuffle our data for the 10 independent models
np.random.seed(0)
s = np.random.randint(100, size=10)
for seed in s:
    print(seed)
    seed

# Input validation
if (train_percent == 1):
    raise ValueError()

from sklearn.utils import shuffle

# We will sklearn to shuffle our data before dividing it into training and sets
properties_dictionary['outputs'], properties_dictionary['labels'] = shuffle(properties_dictionary['inputs'],
properties_dictionary['outputs'],
properties_dictionary['labels'],
random_state=seed)

# Split dictionary into training (80%) and testing (20%) sets
index_split_at = int(train_percent * properties_dictionary['length'])
train_inputs, test_inputs = np.split(properties_dictionary['inputs'], [index_split_at])
train_outputs, test_outputs = np.split(properties_dictionary['outputs'], [index_split_at])
train_labels, test_labels = np.split(properties_dictionary['labels'], [index_split_at])

# Dictionary to return to user
train_test_props_dict = {
    'inputs_train': train_inputs,
    'inputs_test': test_inputs,
    'outputs_train': train_outputs,
    'outputs_test': test_outputs
}

... train_test_model takes a dictionary with training and testing data and builds and trains a model on that data...

# Split data from dictionary into training and testing data
train_inputs = train_test_props_dict['inputs_train']
train_outputs = train_test_props_dict['outputs_train']
test_inputs = train_test_props_dict['inputs_test']
test_outputs = train_test_props_dict['outputs_test']
```

Shuffle data and create different groups of training and testing data
Train a model with modest descriptors

• Mean average error (MAE) over 10 random forests is 500+ K
• Can we do better with better descriptors?
3. Enhance the input data performing featurization based on periodic table data

We will convert the formula to composition, and do featurization based on the composition. The featurization uses periodic table data, such as electronic structure, electronegativity, ionic properties.

```python
x := oxide

oxide_elements = oxide["elements"]

# Convert formula to composition
composition = oxide[schema].apply(get_composition)

# List the features we would like to extract, these are properties that can easily be obtained from periodic table data
# Some of these could be important for predicting melting temperatures
# For example, the difference in electronegativity between atoms is related to the strength of ionic bonding and the stronger
# the bonding the higher the melting temperature
features = MultipleFeaturizer(stoichiometry(), valence_orbital(prop="avg"), ion_property(prop=True), metal_fraction(), valence_orbital(prop=True), bond_center(), oxidation_state(prop=True), element_fraction(), element_property(prop=True, name="mp behaving"))

# X_modelDescriptors will have all the descriptors listed in f
X_modelDescriptors = np.array(features(x=oxide_elements, ignore_errors=True))
```

```python
Matminer: An open source toolkit for materials data mining
Logan Ward3, Alexander Dunn3, Alireza Faghahinia1, Nils E.R. Zimmermann1, Saurabh Bajaj1, Qi Wang1, Joseph Montoya1, Jimmy Chen1, Kyle Bystrom1, Maxwell Dylla1, Kyle Chard1, Mark Asta1, Kristin A. Persson1, G. Jeffrey Snyder1, Ian Foster2, Anubhav Jain2,3

Magpie: A Materials-Agnostic Platform for Informatics and Exploration
We are also invested in making the capabilities and models our group creates available the materials community as a whole. As part of this effort, we have created a Java-based software library named Magpie (short for Materials-Agnostic Platform for Informatics and Exploration). Magpie lets users create, test, and use machine learning models all through a simple text interface or even interactive webpages (such as this). The Magpie platform is available on BitBucket under a permissive, open-source license. Our goal is for other groups to be able to replicate our results and, hopefully, to use our techniques and models in ways that we have not envisioned.

http://wolverton.northwestern.edu/research/machine-learning
```
Train a model with periodic table featurizers

- Mean average error (MAE) over 10 random forests is ~360 K – much better!
- Can we do even better with physics?
Add stiffness and Lindemann melting law

Lindemann: physics based expression for melting temperatures

\[ T_m = \frac{k_b}{9\hbar^2} f^2 a^2 mT_D^2 \]

- Mean average error (MAE) over 10 random forests is \(~360\ \text{K} – much better!
- Can we do even better with physics?
Analyzing descriptors: Pearson correlations

Materials descriptors for predictive machine learning models: application to melting temperature predictions
Zachary D. McCue and Alejandro Strachan,
Materials Engineering, Purdue University.

Overview
These notebooks explore the importance of using appropriate descriptors (or features) of materials to develop predictive models using machine learning. Ideal descriptors capture important aspects of the output quantity of interest and are easy to calculate. Appropriate descriptors significantly improve model accuracy and enable the development of models where data is scarce.

These notebooks introduce materials descriptors in the context of building models to predict the melting temperature of oxides. In addition to simple features, we will explore the addition of materials properties expected to correlate with the quantity of interest and physical laws as inputs to the models to improve accuracy.

Getting started and hints
- Click on the links below to access each notebook.
- Remember to click “Shift+Return” or click “Run” from the top bar to run each cell.
- To exit individual notebooks and return to this page, use File -> Close and Halt. “Terminate Session” (top right) will kill your current Jupyter session.
- The notebooks will be read only; you can make your own copy by downloading the notebook file (Menu File -> Download as... > notebook .py file). You can upload this notebook to your main Jupyter directory within nanoHUB to run it again.

Notebooks
Role of periodic table descriptors and properties in models for melting temperature
- Train Random Forest models to predict the melting temperature of oxides
- The notebook shows how adding meaningful descriptors improves model accuracy

Quantifying descriptors: Pearson correlations
- Perform Pearson Correlation analysis for features associated with the melting point
- Split Matermier features into separate groups, and rank all descriptors at the end

Launch second notebook
Analyzing descriptors: Pearson correlations

After selecting a sub-set of descriptors, calculate and display Pearson correlations

4. Explore correlations between descriptors and melting temperature

We will calculate Pearson correlations between various descriptors with each other and with the melting temperature.

4.1 Crystal properties and Lindemann law

```python
# Create a dataframe with first set of properties to study
df_xtal = pd.DataFrame(np.column_stack((oxide_melting['IPF'], oxide_melting['Density'], oxide_melting['spacegroup.number'], oxide_melting['Molar Volume'], oxide_melting['G_VAH'], oxide_melting['K_VAH'], T_lind, oxide_melting['T_melt'])), columns=['IPF', 'Density', 'SpaceGroup', 'Molar Volume', 'Shear Mod', 'Bulk Mod', 'T_Lind', 'T_melt'])

# Compute and visualize Pearson correlations
pearsoncorr = df_xtal.corr(method='pearson')
plt.figure(figsize=(20,20))
sb.heatmap(pearsoncorr, xticklabels=pearsoncorr.columns, yticklabels=pearsoncorr.columns,
cmap='RdBu_r', annot=True, linewidth=0.25)
plt.show()
```

Properties

Visualizing correlations
Summary

- Use modern tools to compute materials descriptors from composition

- Analyze descriptors using Pearson correlations

- The code in https://nanohub.org/tools/featureselect can be easily modified to meet needs in a wide range of problems
Next steps

Homework assignment: designed to reinforce concepts and help you modify the workflow and adapt it for your needs