Active learning for materials discovery

In this module
• Introduction to active learning for materials science
• Hands on tutorial using nanoHUB: high Li+ conductivity oxides (this document)
  • Homework assignment

Alejandro Strachan, Juan Carlos Verduzco
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:

• Be able to use and modify active learning workflows
• Evaluate different information acquisition functions
• Use active learning to reduce the number of experiments in materials discovery or design

Pre-requisites:
• Basic Python programming
• Querying materials repositories
• Random forests including uncertainty quantification
Data science for design of experiments

Can ML help reduce the number of experiments to achieve a desired goal?

Goal achieved?
YES: SUCCESS
NO: Add data & restart

Existing data

Predictive model of state of knowledge

Query information source

Information acquisition function


Sequential Learning tool in nanoHUB

From your browser go to link: https://nanohub.org/tools/citrinetools

**Citrine Tools for Materials Informatics**

By Juan Carlos Verdugo Gastelum¹, Alejandro Strachan
1. Purdue University

Jupyter notebooks for sequential learning in the context of materials design. Run your own models, explore various methods and adapt the notebooks to your needs.

Click *Launch Tool* to begin

Remember: we will use active learning to find the oxide with the highest ionic conductivity in the smallest number of experiments
Materials Informatics resources from Citrine Informatics, powered by nanoHUB
Juan Carlos Verdezúco and Alejandro Strachan, Materials Engineering, Purdue University

References
Based on work published on: Julia Luo et al.
Original notebooks from: Logan Ward, Argonne National Laboratory, and Max Hutchinson, Citrine Informatics. Their Github implementation can be found here.

Overview
The following Jupyter Notebooks implement methods developed by Citrine Informatics for materials design. Users can modify the notebooks to explore different models, try new ideas and adapt them for their own problems.

The examples under Sequential Learning seek to solve materials design problems (posed as a maximization or minimization problem) with the fewest number of experiments possible. In the examples below, we start with a given number of previously performed experiments, a random subset of which is revealed at the beginning of the exercise. The goal is the find the optimal experiment in the smallest possible number of trials. There are three main steps in the sequential learning approach used:

- **Step 1.** Establish the state of knowledge from the data available using machine learning. Capturing uncertainties is important at this step.
- **Step 2.** Use the model in Step 1 to decide what experiment to carry out next (in our toy examples, this will be revealing the information in one of the pre-formed experiments). This step uses a so-called information acquisition function that seeks to maximize the amount of information gained by the selected experiment. The notebooks below compare various approaches.
- **Step 3.** The information source selected in Step 2 is queried and, if the optimal material is not selected, the new test is added to the available data and the process starts again in Step 1.

The notebooks below use sequential learning to identify the material with the highest bulk modulus and highest ionic conductivity. They all obtain their data from citrination databases (https://citrination.com/), build models using random forest or neural networks, and compare different information acquisition strategies against random searches.

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.
- These notebooks require you to have a Citrine API Key, which you can get by going [here](https://citrination.com/) where you’ll need to create an account. Your API key is listed in “Account Settings”.
- 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.

Loading your Citrine API Key
In order to load your API key, you can insert it in the input box below and we will store it for your convenience. Remember to hit "Enter".

- We will use data from an online repository
- To access the repository you will need to create an account (free) and obtain an API key
- Learn more about data queries in the Data module in the series
- Enter your key in the box
Launch the ceramic oxides notebook

Loading your Citrine API Key
In order to load your API key, you can insert it in the input textbox below and we will store it for your convenience. Remember to hit “Enter”.

Success

Notebooks

Sequential learning approach to find the best candidate material for bulk modulus
- Query databases from Citrinon
- Obtaining K-Frame Cross-validation residuals using lioopy
- Comparison of Information Acquisition Functions for Sequential Learning.

Machine learning guided design of ceramic oxides for batteries
- Query databases from Citrinon
- Featureizers from Matminer
- Comparison: Neural Networks vs Random Forests
- Visual representation of Information Acquisition Functions.

Important Notes:
- These are Jupyter notebooks running Python
- Remember to run each cell by clicking “Shift+Enter”
- Learn more about Jupyter and python
  - Intro to Jupyter and Data: https://nanohub.org/resources/33266
- Read every line of code, comments, and the text cells for detailed explanations
Load libraries for querying, manipulating, modeling, and plotting data
After loading libraries, we query citrination to get that data providing the data set id

• This is a Pandas data frame (think of it as excel in Jupyter)
• Learn more about data frames in the data querying module in this series
Data processing before modeling

Invariable you will have to pre-process and organize your raw data before modeling

```
In [2]:
def get_composition(c):
    try:
        return Composition(c)
    except:
        return None

We will use the utility function to transform the chemicalFormula column, and we'll typecast relevant columns into numeric types.

For this specific application, we'll introduce some filters for the dataframe. We are interested in measurements in structures that are cubic and more.

```
In [4]:
data['composition'] = data['chemicalFormula'].apply(get_composition)  # Transformation of chemical formula string

data['Measuring Temperature'] = pd.to_numeric(data['Measuring Temperature'], errors='coerce')  # Transformation of temperature

data['property_interest'] = pd.to_numeric(data['property_interest'], errors='coerce')  # Transformation of our property

data['Crystallographic Structure'] == 'Cubic'  # Filter all non-cubic structures

data[data['Measuring Temperature']<30]  # Filter all high temperature measurements (over room temperature)

data[data['Measuring Temperature']>10]  # Filter all low temperature measurements (over room temperature)

data.reset_index(drop=True, inplace=True)  # Reindexing of dataframe rows

In order to reduce noise in the neural network and deal with the inconsistencies in the data, we will filter repeated composition values from different values. A similar approach was implemented in this paper.

```
In [1]:
dup_indexes = data[data.duplicated(subset = data.columns.tolist()[0], keep=False)].index.tolist()

dup_dataframe = data[data.duplicated(subset = data.columns.tolist()[0], keep=False)]

duplicates = [dup_dataframe.loc[x[0]], dup_dataframe.loc[x[1]], dup_indexes[x]] for x in range(len(dup_dataframe.duplicated_compositions) == k] for k in set([dup_dataframe.loc[x[0]], dup_indexes[x] for x in range(len(dup_dataframe.duplicated_compositions) == k])

for _ in duplicates:
    duplicate_compositions[0].append([1])
    duplicate_indexes[0].append([2])

for k in duplicate_compositions:
    duplicate_compositions[k] = np.median(duplicate_compositions[k])
    data[duplicate_indexes[k][0], 'Ionic Conductivity'] = duplicate_compositions[k]

data = data.drop(duplicate_indexes[k][1:], axis = 0)

data = data.drop(['index'], axis = 1)
```

Convert the chemical formula to composition

Transform strings into numbers

Filter results:

- Only keep oxides with cubic phase (tetragonal structures result in lower ionic conductivity)
- Only keep results around room temperature

Remove duplicates and replace them with the mean conductivity values
To create predictive models of IC from composition, it is convenient to use materials descriptors that can correlate with the property of interest.

We use matminer* to extract descriptors that can easily obtained from composition:

- Fractions of elements present
- Valence attributes
- Statistical elemental properties:
  - Atomic number, atomic weight, melting point, electronegativity...

* https://hackingmaterials.lbl.gov/matminer/
Descriptors and additional pre-processing

2. Obtaining features/descriptors from Matminer

Matminer offers a variety of features that can be derived from composition. They can be found [here](https://hackingmaterials.lbl.gov/matminer/).

In this example, we will query the descriptors listed in the paper by [Ward et al.](https://hackingmaterials.lbl.gov/matminer/). We will use the `ElementFraction()` feature from Matminer and the experimental measuring temperature. All columns that have a standard deviation of zero are dropped, as they don’t contribute new information to the models.

```python
In [ ]:
X = np.array(df.feature_names['composition'], ignore_errors=True)  # Array to store such features
X = np.array(df['measuring_temp'] - np.mean(np.array(df['measuring_temp']), ignore_warnings=True))  # Into the features previously calculated to add it as a descriptor.
y = df['property_inter', 'values']  # separate the value we want to predict to use as labels.
```

3. Processing and Organizing Data

For the use in machine learning, we need to split our data into a training set and a testing set. We are establishing a fixed length of 10% of the entries as testing.

Normalization is a standard score norm where we substract the mean and divide by the standard deviation.

```python
from sklearn.utils import shuffle
all_values = list(x_df.loc[i] for i in range(len(x_df.index)));
all_values = np.array(all_values, dtype = float)
all_labels = y.copy()
train_percent = 0.9
index_split_at = int(train_percent * len(all_labels))
all_values, all_labels = shuffle(all_values, all_labels, random_state=1)
train_values, test_values = np.split(all_values, [index_split_at])
train_labels, test_labels = np.split(all_labels, [index_split_at])
```

Both training and testing datasets use mean/std of the training set

From our initial dataset, we split it into two sets:
- Training: data the model will see and learn from
- Testing: data hidden from the model to evaluate it

Some properties have very different scales for values. We normalize the data using a standard score normalization.

\[
Z = \frac{X - \mu}{\sigma}
\]
Testing predictive models

Before running the active learning workflow the tools creates and tests a neural network network and a random forest model. Please run the cells.

4. Regression Models

We will start by creating models for regression with all these entries and descriptors.

4.1 Neural Networks

We set the architecture of the sequential feed-forward neural network we’ll test. Weights are initialized with a Random Normal. You can find a summary of the layers, neurons and free parameters to optimize by running the `model.summary()` command.

4.2 Random Forest (Scikit-learn)

A random forest from the scikit-learn library is implemented and an equivalent match plot is shown below.
Active learning: data preparation

5. Active Learning

Active learning is the use of algorithms not for regression, but for the improvement of the input sample space that guides the ‘experiments’ to make predictions for a specific material, its main task is the selection of the most likely candidate to be in a global maxima. Global optimization algorithms “choose” which point to query next. This is the approach introduced in the paper by Julia Ling et al.

5.3 Approach

We will select an initial set of 10 entries, and we'll make sure the highest value is not in it. If it is, the cell will throw an error.

```python
X = all_values.copy()
y = all_labels.copy()
model = RandomForestRegressor()
entry_number_init = 10
in_train = np.zeros(len(data), dtype=np.bool)
in_train[np.random.choice(len(data), entry_number_init, replace=False)] = True
print('Picked {} training entries'.format(in_train.sum()))
assert not np.isclose(max(y), max(y[in_train]))
```

Select 10 initial experiments at random to start the active learning
Active learning loop

### MEI
Pick the experiment with the highest expected value

### MLI
Pick the experiment with the maximum expected improvements normalized by uncertainty

### MU
Pick the experiment with the highest uncertainty

### UCB
Pick the experiment with the highest expected value + uncertainty
The simulation outputs a video that shows how each information acquisition function selects experiments and how long they take to converge.
Summary

• Predictive models can be used to reduce the number of experiments or simulations in materials discovery or optimization processes

• Knowledge of uncertainties in the model is highly desirable for decision making

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

**Homework assignment**: to reinforce concepts and help students modify the workflow and adapt it for their needs