A Practical Tutorial for Jupyter Notebooks

Extracted from Logan Ward's notes

Postdoctoral Scholar

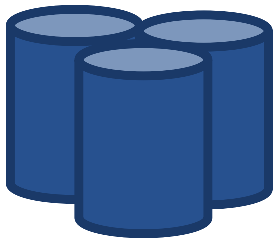
University of Chicago

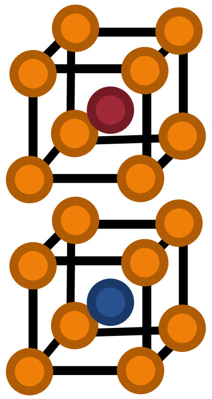
17 October 2018

globus labs

Materials Informatics Workflow

Collect → Process → Represent → Learn





$\Delta H_f = -1.0$

$\Delta H_f = -0.5$

$\vec{X}$ $\vec{y}$

Z_A	Z_B	$\Delta \mathrm{H_f}$
3	4	-1.0
3	5	-0.5

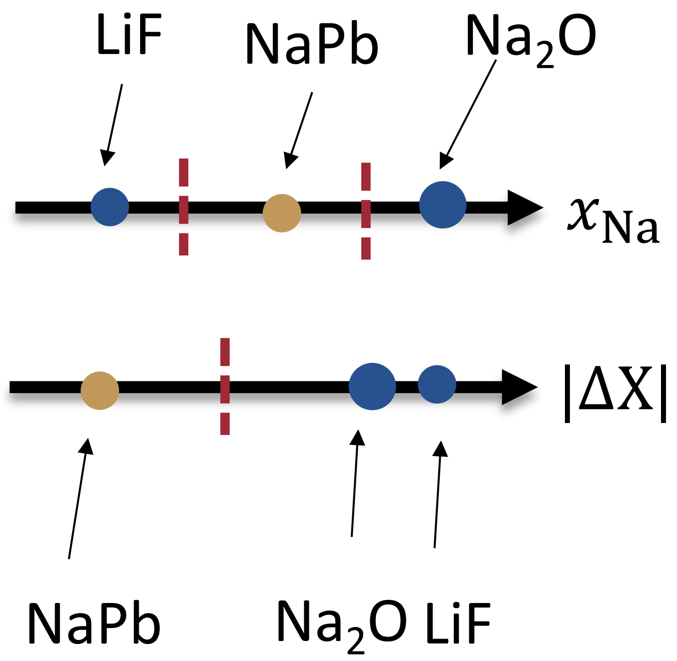
$$\Delta H_f = f(Z_A, Z_B)$$

Data Representation

Collect → Process → **Represent** → Learn

Encode domain knowledge into inputs

$$Property = \boldsymbol{f}(Attributes)$$



Representation of material

Ex: *Attributes* = $\boldsymbol{g}(x_H, x_{He}, \ldots)$

What does a representation need?

Completeness: Differentiate materials

Efficiency: Quick to compute

Accuracy: Capture important effects

End product: Machine-learning compatible inputs

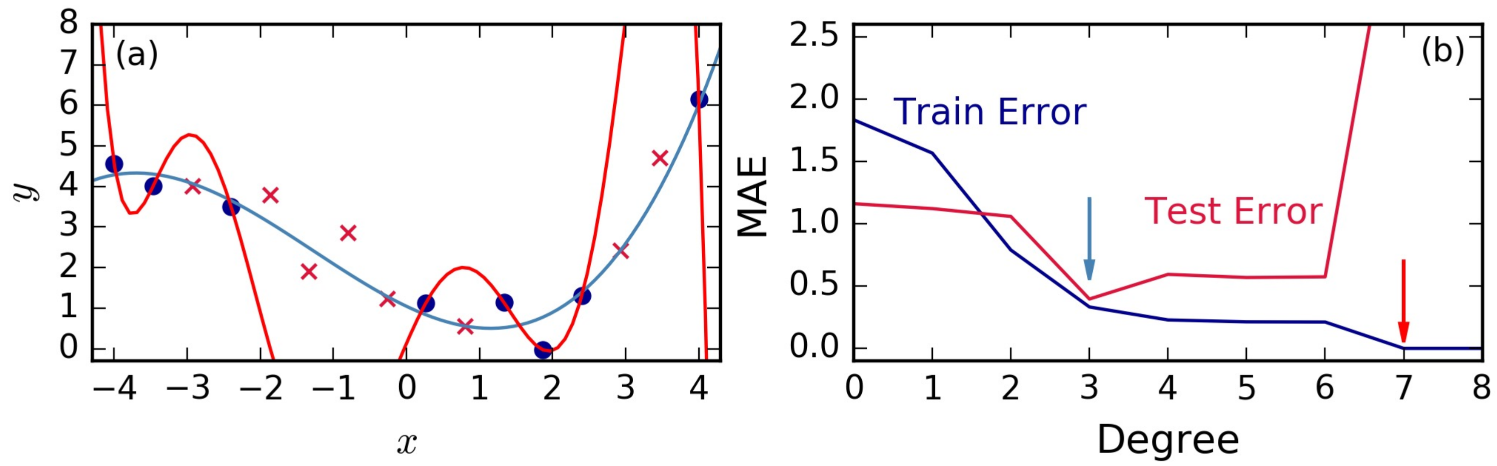
Fit a model to the data

Collect → Process → Represent → **Learn**

There are many algorithms, how do you decide?

1. Identify algorithms with desired properties.
   Ex: differentiability
2. Use cross-validation to optimize parameters
3. Pick the one with the lowest "loss"

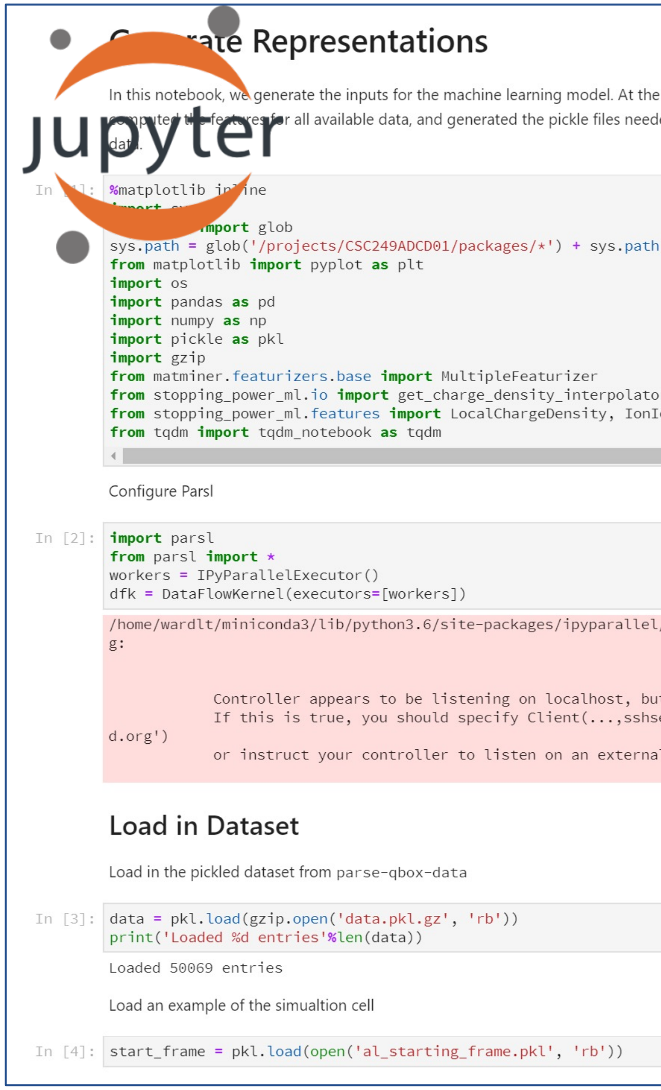
- **Numpy**
- **Scipy**
- **Scikit-learn**
- **Theano**
- **TensorFlow**
- **Keras**
- **PyTorch**
- **Pandas**
- **Matplotlib**



How to Organize? [Jupyter]

What is Jupyter? Why care?

- Jupyter is environment designed for reproducible computational science
- Stores code with outputs and documentation in a single notebook
  - Modeled after Mathematica
  - https://www.theatlantic.com/science/archive/2018/04/the-scientific-paper-is-obsolete/556676/
- Why do *I* care? Jupyter lets me…
  - organize my code
  - easily run on a remote system
  - keep track of results and rationale
  - communicate my research better



Generate Representations

In this notebook, we generate the inputs for the machine learning model. At the
computed the features for all available data, and generated the pickle files neede
data.

```
In [1]: %matplotlib inline
        import sys
        from glob import glob
        sys.path = glob('/projects/CSC249ADCD01/packages/*') + sys.path
        from matplotlib import pyplot as plt
        import os
        import pandas as pd
        import numpy as np
        import pickle as pkl
        import gzip
        from matminer.featurizers.base import MultipleFeaturizer
        from stopping_power_ml.io import get_charge_density_interpolato
        from stopping_power_ml.features import LocalChargeDensity, IonI
        from tqdm import tqdm_notebook as tqdm
```

Configure Parsl

```
In [2]: import parsl
        from parsl import *
        workers = IPyParallelExecutor()
        dfk = DataFlowKernel(executors=[workers])
```

```
/home/wardlt/miniconda3/lib/python3.6/site-packages/ipyparallel/
g:


            Controller appears to be listening on localhost, bu
            If this is true, you should specify Client(...,sshse
d.org')
            or instruct your controller to listen on an externa
```

Load in Dataset

Load in the pickled dataset from `parse-qbox-data`

```
In [3]: data = pkl.load(gzip.open('data.pkl.gz', 'rb'))
        print('Loaded %d entries'%len(data))
```

```
Loaded 50069 entries
```

Load an example of the simualtion cell

```
In [4]: start_frame = pkl.load(open('al_starting_frame.pkl', 'rb'))
```

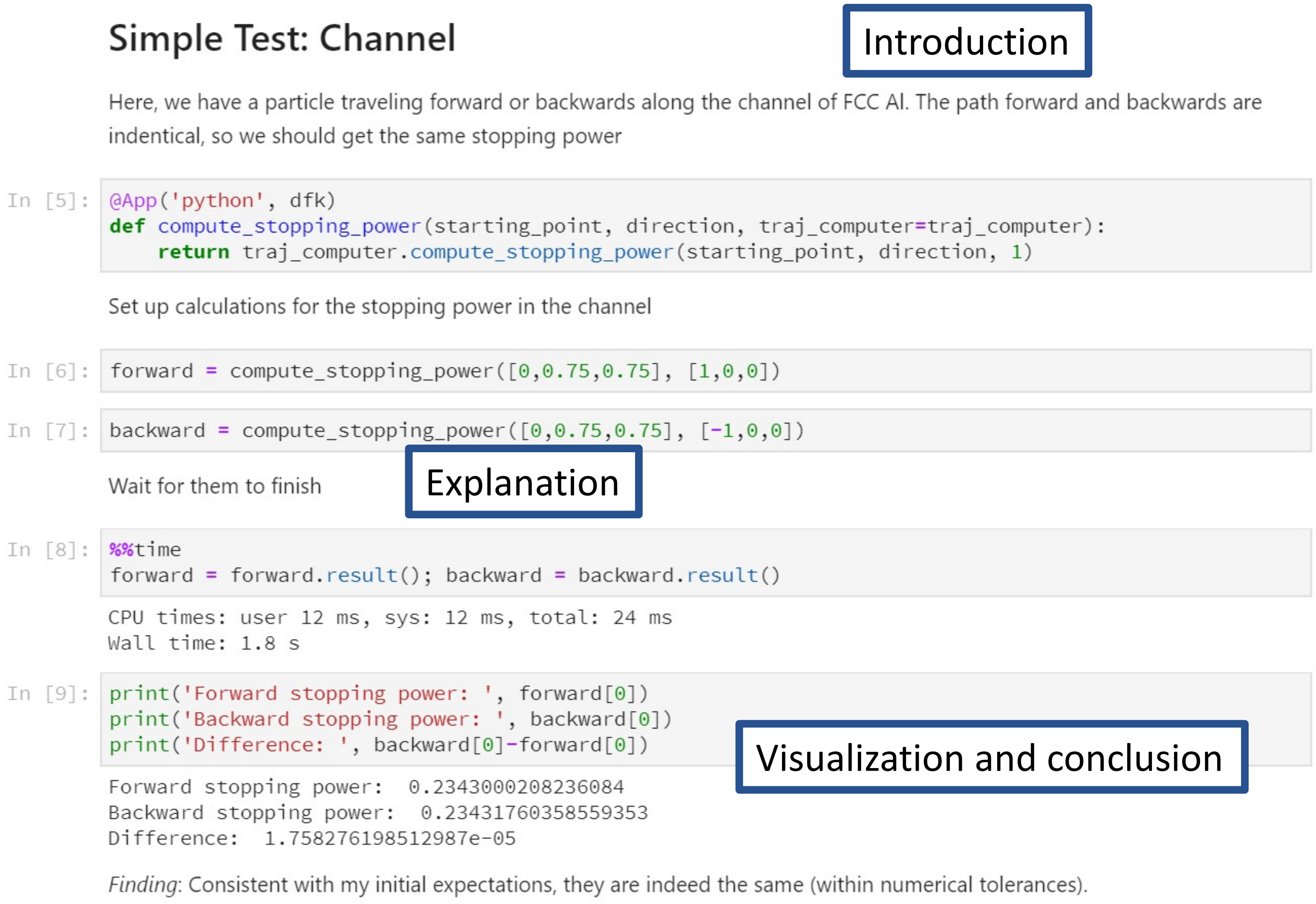
How I write a notebook

One notebook per "experiment" or "idea"

Outline:

1. Title and short abstract
    1. "What am I doing here and why"
2. Load in libraries
    1. Put them up front, so it crashes early
3. Load in data from disk
    1. Ex: training data, results from other notebook
4. Each step in their own block
    1. Introduction
    2. Code and explanation
    3. Figure/visualization
    4. Explanation of finding

Example Step



Simple Test: Channel

Introduction

Here, we have a particle traveling forward or backwards along the channel of FCC Al. The path forward and backwards are indentical, so we should get the same stopping power

```
In [5]: @App('python', dfk)
        def compute_stopping_power(starting_point, direction, traj_computer=traj_computer):
            return traj_computer.compute_stopping_power(starting_point, direction, 1)
```

Set up calculations for the stopping power in the channel

```
In [6]: forward = compute_stopping_power([0,0.75,0.75], [1,0,0])
```

```
In [7]: backward = compute_stopping_power([0,0.75,0.75], [-1,0,0])
```

Wait for them to finish

Explanation

```
In [8]: %%time
        forward = forward.result(); backward = backward.result()
```

```
CPU times: user 12 ms, sys: 12 ms, total: 24 ms
Wall time: 1.8 s
```

```
In [9]: print('Forward stopping power: ', forward[0])
        print('Backward stopping power: ', backward[0])
        print('Difference: ', backward[0]-forward[0])
```

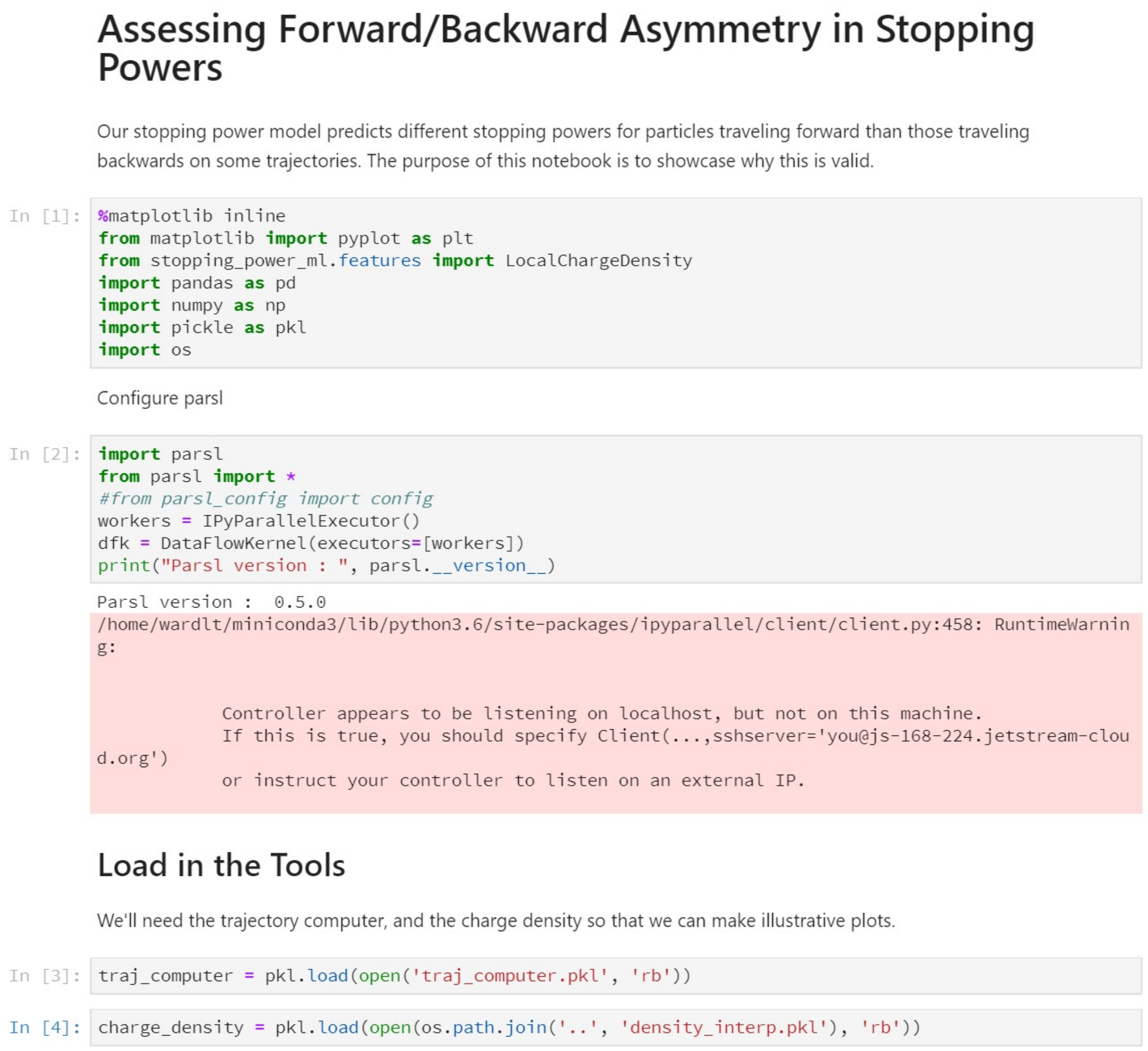
Visualization and conclusion

```
Forward stopping power:  0.2343000208236084
Backward stopping power:  0.23431760358559353
Difference:  1.758276198512987e-05
```

Finding: Consistent with my initial expectations, they are indeed the same (within numerical tolerances).

Someone should be able to understand this without knowing Python!

The Full Narrative



Assessing Forward/Backward Asymmetry in Stopping Powers

Our stopping power model predicts different stopping powers for particles traveling forward than those traveling backwards on some trajectories. The purpose of this notebook is to showcase why this is valid.

In [1]:
```
%matplotlib inline
from matplotlib import pyplot as plt
from stopping_power_ml.features import LocalChargeDensity
import pandas as pd
import numpy as np
import pickle as pkl
import os
```

Configure parsl

In [2]:
```
import parsl
from parsl import *
#from parsl_config import config
workers = IPyParallelExecutor()
dfk = DataFlowKernel(executors=[workers])
print("Parsl version : ", parsl.__version__)
```

```
Parsl version :  0.5.0
/home/wardlt/miniconda3/lib/python3.6/site-packages/ipyparallel/client/client.py:458: RuntimeWarnin
g:


            Controller appears to be listening on localhost, but not on this machine.
            If this is true, you should specify Client(...,sshserver='you@js-168-224.jetstream-clou
d.org')
            or instruct your controller to listen on an external IP.
```

Load in the Tools

We'll need the trajectory computer, and the charge density so that we can make illustrative plots.

In [3]:
```
traj_computer = pkl.load(open('traj_computer.pkl', 'rb'))
```

In [4]:
```
charge_density = pkl.load(open(os.path.join('..', 'density_interp.pkl'), 'rb'))
```

Link to GitHub Page

Pitfalls When Making Notebooks

1. Not including documentation

   Advice: Write what you’re going to do first

2. Notebook not capturing entire process

   Problem: Jupyter lets you execute cells out of order

   Advice: Periodically “Restart Kernel and Run All Cells”

3. Duplicate code between notebooks

   Advice: Make a separate module for common code

4. Library conflicts

   Advice: Run mature projects in container, separate machines

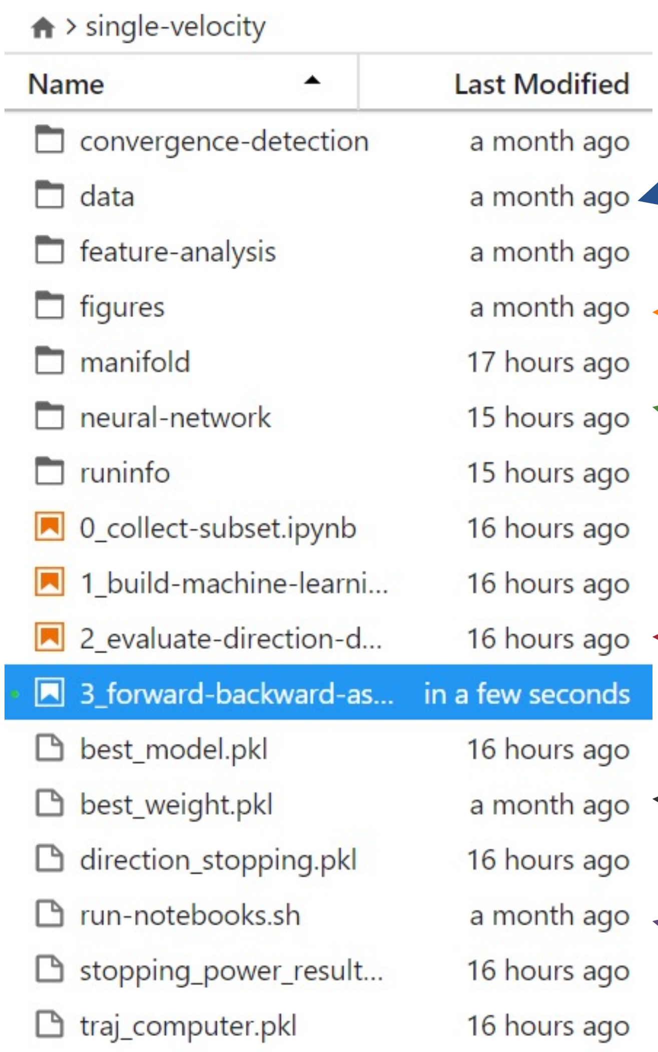
   Advice: Make a “requirements.txt” file

5. The “one cell notebooks”

   Advice: <10 lines of code per cell

See “I Don’t Like Jupyter Notebooks”: https://t.co/30peBFwTbv

Organizing Multiple Notebooks



> single-velocity

Name	Last Modified
convergence-detection	a month ago
data	a month ago
feature-analysis	a month ago
figures	a month ago
manifold	17 hours ago
neural-network	15 hours ago
runinfo	15 hours ago
0_collect-subset.ipynb	16 hours ago
1_build-machine-learni...	16 hours ago
2_evaluate-direction-d...	16 hours ago
3_forward-backward-as...	in a few seconds
best_model.pkl	16 hours ago
best_weight.pkl	a month ago
direction_stopping.pkl	16 hours ago
run-notebooks.sh	a month ago
stopping_power_result...	16 hours ago
traj_computer.pkl	16 hours ago

Re-usable datasets in separate folder

Figures in separate folder
(with meaningful names!)
(making pub-ready figures!)

Dependent tasks in separate folder

Notebooks numbered by
execution order

Results shared between
steps using pickled files

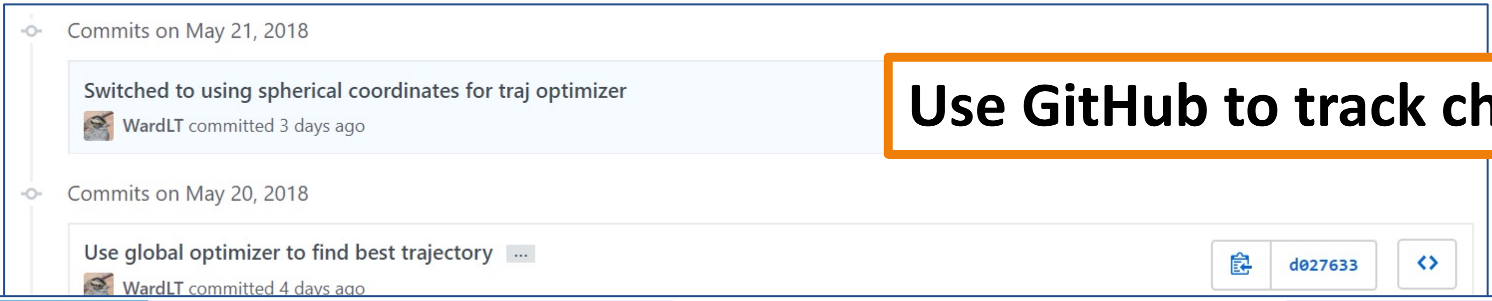
A bash script to re-run all notebooks

Running Notebooks from Terminal

```
#! /bin/bash

# This script just runs through all of the current notebooks in the proper order

runJupyter() {
   jupyter nbconvert --execute --inplace --ExecutePreprocessor.timeout=-1 $1
}

# If we get too many files, we should probably create a file containing
#  names of notebooks and the order in which they should be executed
for n in `seq 0 1`; do
   runJupyter ${n}_*nb
done
```

Rerun your project often, put in “assert” statements for key results!

Keeping Track of Project



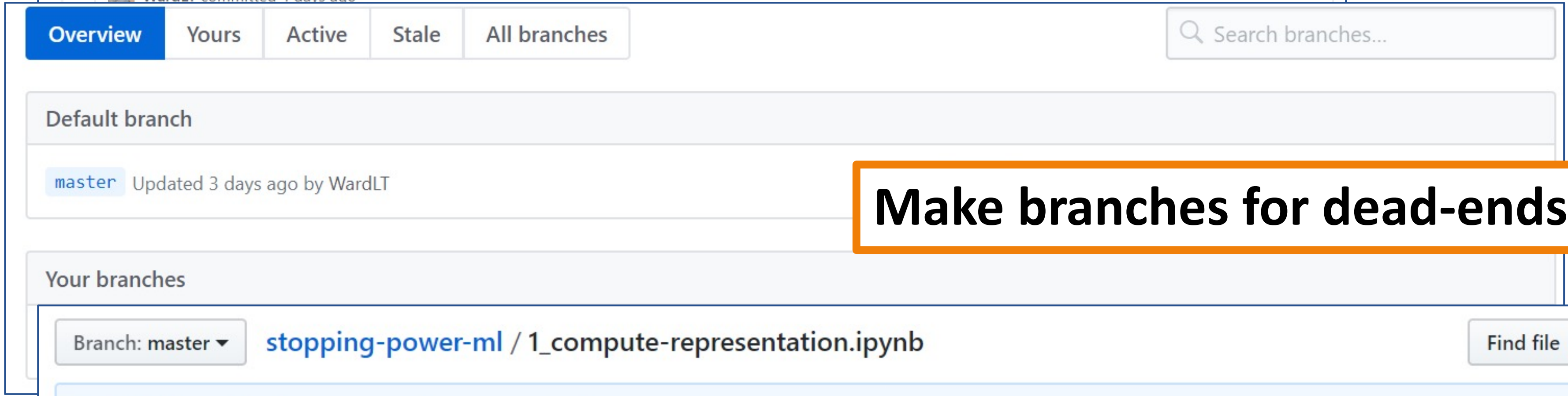
Commits on May 21, 2018

Switched to using spherical coordinates for traj optimizer
WardLT committed 3 days ago

Commits on May 20, 2018

Use global optimizer to find best trajectory …
WardLT committed 4 days ago

Use GitHub to track changes

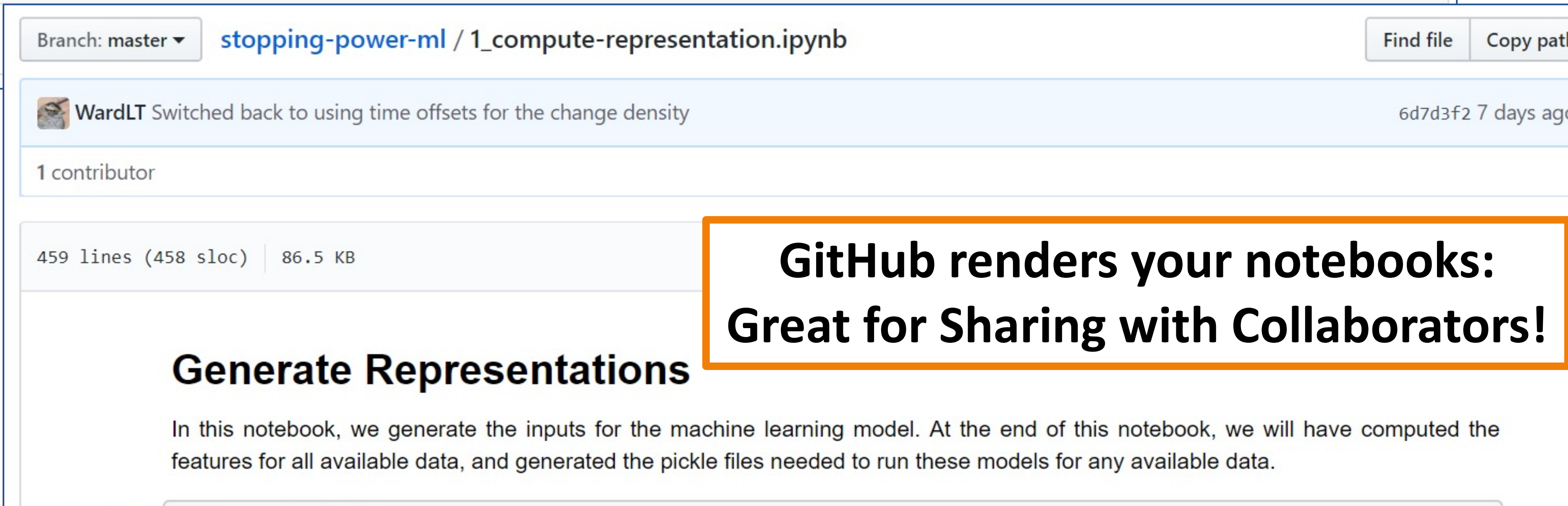


Default branch

master Updated 3 days ago by WardLT

Your branches

Make branches for dead-ends



stopping-power-ml / 1_compute-representation.ipynb

WardLT Switched back to using time offsets for the change density 6d7d3f2 7 days ag

1 contributor

459 lines (458 sloc) 86.5 KB

Generate Representations

In this notebook, we generate the inputs for the machine learning model. At the end of this notebook, we will have computed the features for all available data, and generated the pickle files needed to run these models for any available data.

GitHub renders your notebooks: Great for Sharing with Collaborators!

Take-Away Points

1. Write your notebooks like papers

   Explain what you are doing, why, and what you found

   Learn Markdown to include links, equations, pictures

2. Break up complex projects into multiple steps

   Each notebook should tell a single story

3. Notebooks should be easy to re-run

   Use "Restart and Run," bash scripts

4. Track your changes with Git

   Explain what changes you made, and why

Publication: Not Just Papers

Is my 2016 paper reproducible?

npj Computational Materials www.nature.com/npjcompumats

ARTICLE OPEN

A general-purpose machine learning framework for predicting properties of inorganic materials

Logan Ward[1], Ankit Agrawal[2], Alok Choudhary[2] and Christopher Wolverton[1]

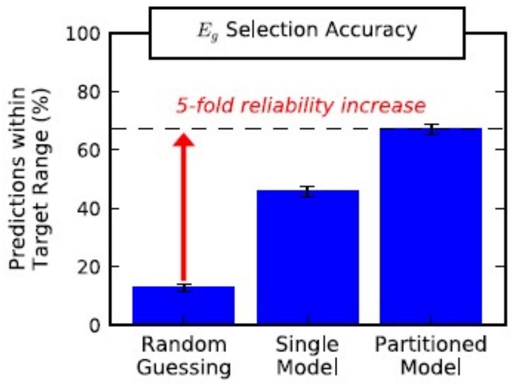


Figure 1. Performance of three different strategies to locate compounds with a band gap energy within a desired range: randomly selecting nonmetal-containing compounds, and two ased method presented in ategy used a single model rgies of 22,667 compounds

Method Comparison

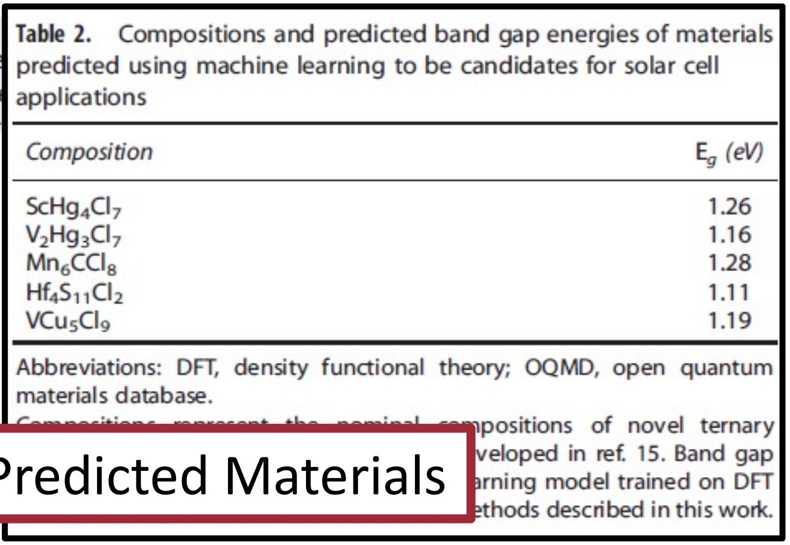


Table 2. Compositions and predicted band gap energies of materials predicted using machine learning to be candidates for solar cell applications

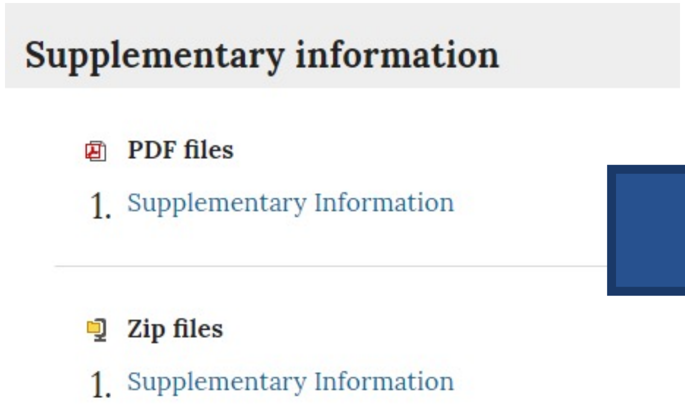
Composition	E_g *(eV)*
$ScHg_4Cl_7$	1.26
$V_2Hg_3Cl_7$	1.16
Mn_6CCl_8	1.28
$Hf_4S_{11}Cl_2$	1.11
VCu_5Cl_9	1.19

Abbreviations: DFT, density functional theory; OQMD, open quantum materials database.
Compositions represent the nominal compositions of novel ternary veloped in ref. 15. Band gap arning model trained on DFT ethods described in this work.

Predicted Materials

I released the code, but not *all of it*

From Original Source Code



Supplementary information

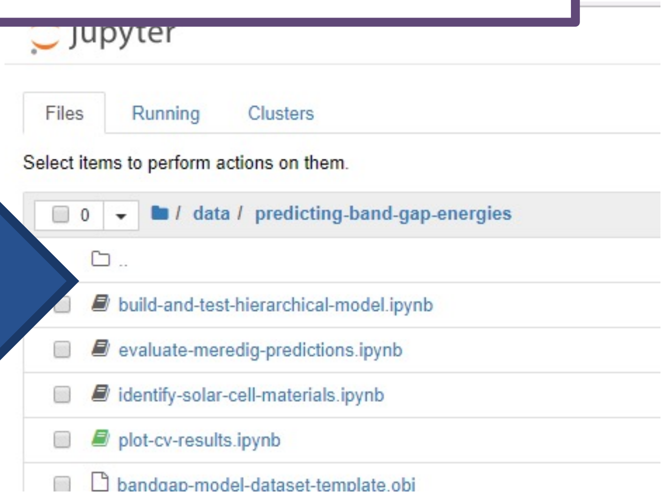
PDF files

1. Supplementary Information

Zip files

1. Supplementary Information

… to New Notebooks



Select items to perform actions on them.

/ data / predicting-band-gap-energies

..

build-and-test-hierarchical-model.ipynb

evaluate-meredig-predictions.ipynb

identify-solar-cell-materials.ipynb

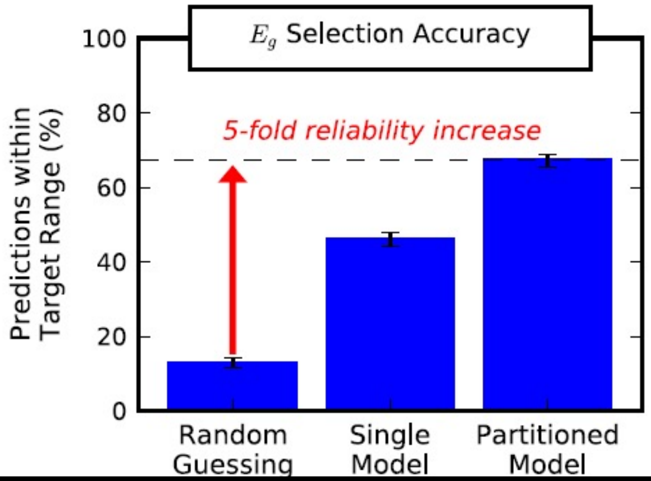
plot-cv-results.ipynb

bandgap-model-dataset-template.obj

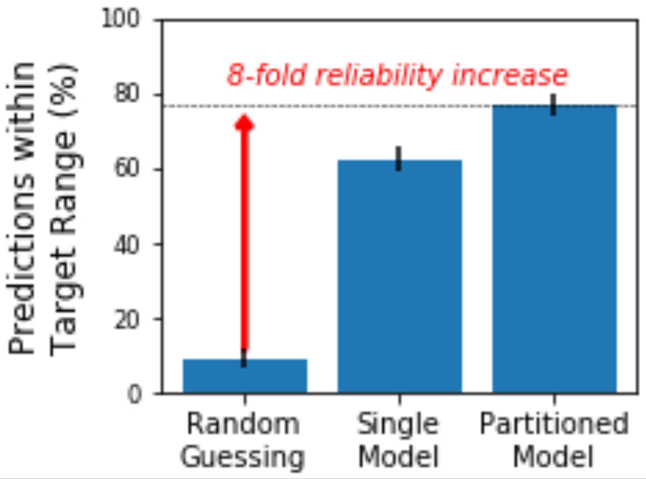
… and there are differences

Reported Results in 2016(?)

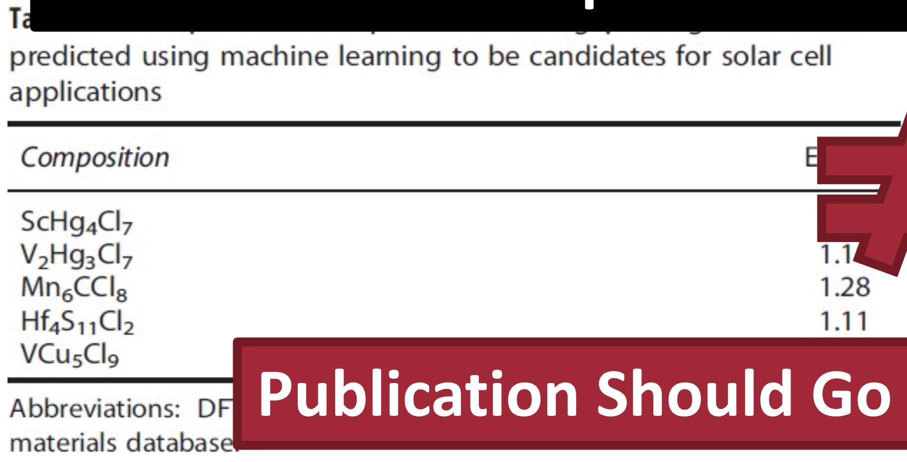
Replication in 2017



≠



Without Exact Scripts From <2016, Cannot Verify Results



Ta

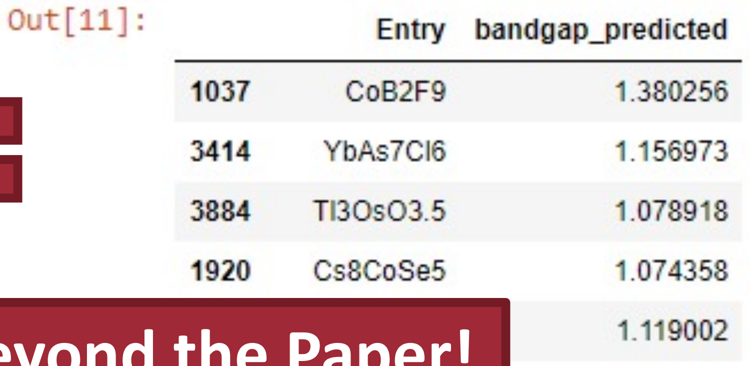
predicted using machine learning to be candidates for solar cell applications

Composition	E
$ScHg_4Cl_7$	
$V_2Hg_3Cl_7$	1.1
Mn_6CCl_8	1.28
$Hf_4S_{11}Cl_2$	1.11
VCu_5Cl_9	

Abbreviations: DF

materials database

≠



Out[11]:

	Entry	bandgap_predicted
1037	CoB2F9	1.380256
3414	YbAs7Cl6	1.156973
3884	Tl3OsO3.5	1.078918
1920	Cs8CoSe5	1.074358
		1.119002

Publication Should Go Beyond the Paper!

How Do We Avoid this Problem?

What do I need to publish?

- *Datasets*: In a well-described format
- *Scripts*: Not just the core methods
- *Outputs:* Exact version from the paper
- *Models:* In a user-friendly way

Save Anaconda/Python versions and Environments/ library version!

WholeTale

DLHub