Prediction of a reactor's isotopic inventory Build Status

Authors : Benjamin Dechenaux, Jean-Baptiste Clavel, Cécilia Damon (IRSN) with the help of François Caud and Alexandre Gramfort (Université Paris-Saclay)

Institut de Radioprotection et de Sûreté Nucléaire (IRSN)

31 avenue de la division Leclerc

92260 - Fontenay-aux-Roses

Getting started Install To run a submission and the notebook you will need the dependencies listed in requirements.txt. You can install install the dependencies with the following command-line:

pip install -U -r requirements.txt If you are using conda, we provide an environment.yml file for similar usage.

Challenge description Get started with the dedicated notebook

Test a submission The submissions need to be located in the submissions folder. For instance for my_submission, it should be located in submissions/my_submission.

To run a specific submission, you can use the ramp-test command line:

ramp-test --submission my_submission You can get more information regarding this command line:

ramp-test --help To go further You can find more information regarding ramp-workflow in the dedicated documentation

```
!unzip "nuclear_inventory_open (2).zip"
```

```
Archive:  nuclear_inventory_open (2).zip
  inflating: environment.yml
  inflating: .gitignore
  inflating: README.md
  inflating: download_data.py
  inflating: requirements.txt
  inflating: extra_libraries.txt
  inflating: nuclear_inventory_starting_kit.html
  inflating: DATAIA-h.png
  inflating: nuclear_inventory_starting_kit.ipynb
  inflating: problem.py
  inflating: submissions/mlp/regressor.py
  inflating: submissions/starting_kit/regressor.py
  inflating: .github/workflows/check_deps.py
  inflating: .github/workflows/install.yml
  inflating: .github/workflows/testing.yml
  inflating: .git/ORIG_HEAD
  inflating: .git/FETCH_HEAD
```

```
inflating: .git/HEAD
inflating: .git/description
inflating: .git/packed-refs
inflating: .git/index
inflating: .git/config
inflating: .git/info/exclude
inflating: .git/refs/remotes/origin/HEAD
inflating: .git/refs/remotes/origin/main
inflating: .git/refs/heads/main
inflating: .git/hooks/pre-receive.sample
inflating: .git/hooks/commit-msg.sample
inflating: .git/hooks/prepare-commit-msg.sample
inflating: .git/hooks/pre-rebase.sample
inflating: .git/hooks/update.sample
inflating: .git/hooks/pre-push.sample
inflating: .git/hooks/pre-commit.sample
inflating: .git/hooks/applypatch-msg.sample
inflating: .git/hooks/fsmonitor-watchman.sample
inflating: .git/hooks/post-update.sample
inflating: .git/hooks/pre-applypatch.sample
inflating: .git/logs/HEAD
inflating: .git/logs/refs/remotes/origin/HEAD
inflating: .git/logs/refs/remotes/origin/main
inflating: .git/logs/refs/heads/main
inflating: .git/objects/f2/c8aceef33db27bf750d950c0b989f361a13f0c
inflating: .git/objects/aa/232ada1af6acb57562e8716e3e12a2af791c53
inflating: .git/objects/2f/5d8d91a6a90b160295c238f8037407647a7b28
inflating: .git/objects/48/17fbf9c0c67a00f7f0b0a14a5b403effdc6717
inflating: .git/objects/ff/dfff84528b88cd9dc8e24653a56d0685ca2ea8
inflating: .git/objects/64/7a9532ac5c05af625a44ed9153aef279d2bf3b
inflating: .git/objects/66/3be0f50df5fd6432aec940a89b69359a42e327
inflating: .git/objects/d5/f52f1d279000dedf010feb64470e5d3420c929
inflating: .git/objects/a0/75c20614442bcc8d3001dba914c5574d6742f4
inflating: .git/objects/d3/5dd3234e33096d2d9453e56e288f87299efdaa
inflating: .git/objects/5c/979fec9e1a4e48cba75b7fc3c91cc48b28a111
inflating: .git/objects/f7/50a5af0c888ba237b9b01530e653fd48fbbfed
inflating: .git/objects/ea/63297bdc8ef49d00d103b41d3b31d12b37c935
inflating: .git/objects/92/114031f24072a0939ead9d60d931500a8558f5
inflating: .git/objects/96/003bb04d2f27af2eaec68a6e3a5a71120e6ae5
inflating: .git/objects/a1/81d49cb7081ecd54b4bdebc5aabbe938c62986
inflating: .git/objects/6e/b4c63feeaa22ed191e26c36134013b9f66f23e
inflating: .git/objects/87/7f29b28e8db1278b8c3d952e40a06c537493e3
inflating: .git/objects/97/bb3a9be86f364706e2819d594731f6638631a5
inflating: .git/objects/db/3765648be6b72fdd0bb8bce52bdc870d0508de
inflating: .git/objects/68/953f23cc6dc51d0e18a7b0fd62a74359907d8c
inflating: .git/objects/2d/83e9a141ced6b92f81c1e9c14424589d1185d2
inflating: .git/objects/1b/c2f3816ec4ee0ea1e52caa9656d1bf2739c9a8
inflating: .git/objects/90/e6f04f540e7fd66b2e04a6826210ceae98ac03
inflating: .git/objects/cb/0c60ab587f1911576fdc7d98994af7e2b4a870
inflating: .git/objects/80/0d8feb5d2ba2f4cab7b7c619313bf4e0b806aa
```

```
  inflating: .git/objects/9d/2ed90fa562be9a9b77b3d259ebf2c4b0f087af
  inflating: .git/objects/9d/0005fa41ce1c0b2329bec69a9bd193b8ae05b9
  inflating: .git/objects/38/47a998d33fa8576bc7e4599b1a02d6021d3228
  inflating: .git/objects/77/5a2ba4f74459e1f16559298972a39217f34c70
  inflating: .git/objects/f8/f93addaaff2160ce6231c413e19c32b064b867
  inflating: .git/objects/01/4ff7725c9038262f9cf48344d2e30967225708
  inflating: .git/objects/04/3e54272c3561cab9b8ce2aa73fa5be06ae81c3
  inflating: .git/objects/84/e4c6418e6950f04ec25ef9b129d106af93992f
  inflating: .git/objects/pack/pack-
e79343b4baf94f547d223c1b0916fbbd1b30222c.pack
  inflating: .git/objects/pack/pack-
e79343b4baf94f547d223c1b0916fbbd1b30222c.idx
  inflating: .git/objects/32/954e190db50e02b606b75f6d733a0096921eb3
  inflating: .git/objects/32/71618521df4123481987dcfff58b1500254b43
  inflating: .git/objects/30/b480f77775df1e6ab91cf684a13a01561e1ce0
  inflating: .git/objects/30/f335a8a757742d54115b0d328537689bb13c4b
  inflating: .git/objects/6a/76a547609951a6697b502c02a4bd1dc9614fc4
  inflating: .git/objects/85/a3e4719d488e5833df9d457108a1697189d4bf
  inflating: .git/objects/f9/15a99e05ec25c8ccbc4c3520635806407d4491
  inflating: .git/objects/f9/ea60642855f9183b04421582cd534981081f4d
  inflating: .git/objects/58/cea230fb7bc2d134459ffd4e1c69211cafedc6
  inflating: .git/objects/e1/2c6711eebc5712f9fe9d22f4da95990246256d
  inflating: .git/objects/08/27dfe250bcd718c00b7584286a31b42356350e
  inflating: .git/objects/12/178da32eeb8c8f7515e67d044756e8b9eb30ce
  inflating: .git/objects/1f/47eae6f759ed882af000ff21f815c2b31edeed
  inflating: .git/objects/81/11ef5923d8fc16f8ff7aeeae8172c83e4462e1
  inflating: .git/objects/21/7c228e2a78723bb16fdba190df87759633908a
  inflating: __pycache__/problem.cpython-39.pyc
  inflating: __pycache__/problem.cpython-37.pyc
```

```
!pip install osfclient
```

```
Collecting osfclient
  Downloading osfclient-0.0.5-py2.py3-none-any.whl (39 kB)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-
packages (from osfclient) (1.15.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-
packages (from osfclient) (4.62.3)
Requirement already satisfied: requests in
/usr/local/lib/python3.7/dist-packages (from osfclient) (2.23.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from requests->osfclient)
(2021.10.8)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from requests->osfclient)
(3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1
in /usr/local/lib/python3.7/dist-packages (from requests->osfclient)
(1.24.3)
Requirement already satisfied: idna<3,>=2.5 in
/usr/local/lib/python3.7/dist-packages (from requests->osfclient)
```

```
(2.10)
Installing collected packages: osfclient
Successfully installed osfclient-0.0.5
```

```
#mlp
import numpy as np
from sklearn.neural_network import MLPRegressor
from sklearn.base import BaseEstimator

class Regressor(BaseEstimator):
    def __init__(self):
        self.model = MLPRegressor(
            solver="adam",
            hidden_layer_sizes=(100, 100, 100),
            max_iter=300,
            batch_size=100,
            random_state=57,
        )

    def fit(self, X, Y):
        self.X_scaling_ = np.max(X, axis=0, keepdims=True)
        self.Y_scaling_ = np.max(Y, axis=0, keepdims=True)
        self.model.fit(X / self.X_scaling_, Y / self.Y_scaling_)

    def predict(self, X):
        res = self.model.predict(X / self.X_scaling_) *
self.Y_scaling_
        return res
```

```
#starting_kit
from sklearn.linear_model import LinearRegression
from sklearn.base import BaseEstimator


class Regressor(BaseEstimator):
    def __init__(self):
        self.model = LinearRegression()

    def fit(self, X, Y):
        self.model.fit(X, Y)

    def predict(self, X):
        res = self.model.predict(X)
        return res
```

name: ramp-nuclear-inventory channels:

- conda-forge
- defaults dependencies:
- numpy

- scikit-learn>=0.22
- scipy
- click
- pandas
- matplotlib
- seaborn

# Extra requirements

- tensorflow-gpu
- pytorch
- keras
- lightgbm
- pip
- pip:
    - osfclient
    - ramp-workflow
    - ramp-utils

### Extra requirements pip

- xgboost
- catboost

Prediction of the isotopic inventory in a nuclear reactor core

Benjamin Dechenaux, Jean-Baptiste Clavel, Cécilia Damon (IRSN), François Caud, Alexandre Gramfort (DATAIA, Univ. Paris-Saclay)

This challenge was done with the support of DATAIA in collaboration with IRSN:

```
# Required for running the notebook
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
import seaborn as sns
import string
import os
# sklearn dependences are used to build a baseline model
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
```

```
!python download_data.py
```

```
Checking the data URL...Ok.
Downloading the data...
100% 9.86M/9.86M [00:00<00:00, 48.5Mbytes/s]
Extracting now...Ok.
Removing the archive...Ok.
Checking the data...Ok.
```

```
%matplotlib inline
# Exploratory data analysis
#First, download the data executing this python script:
!python download_data.py
### Loading the data

#The 920 simulations have been split into two *training* and *testing*
datasets.

#* The *training* dataset is composed of 690 simulations and is
accessible in CSV format under the __train__ folder
#* The *testing* dataset is composed of 230 simulations and is
accessible in CSV format under the __test__ folder

#Training and testing sets can be loaded as follows:

from glob import glob

def get_file_list_from_dir(*, path, datadir):
    data_files = sorted(glob(os.path.join(path, "data", datadir,
"*.csv.gz")))
    return data_files
```

```
data directory is not empty. Please empty it or select another
destination for LOCAL_DATA if you wish to proceed
```

```
train_files = get_file_list_from_dir(path=".", datadir="train")
len(train_files)
```

```
690
```

```
dtrain = pd.concat((pd.read_csv(f) for f in train_files))
dtrain
```

```
     times         A         B  ...        p3        p4        p5
0      0.0  0.317729  3.265168  ...  0.008332  0.035934  0.016353
1     20.0  0.316280  3.263388  ...  0.008332  0.035934  0.016353
2     40.0  0.314831  3.261604  ...  0.008332  0.035934  0.016353
3     60.0  0.313387  3.259812  ...  0.008332  0.035934  0.016353
```

```
4     80.0  0.311945  3.258014  ...  0.008332  0.035934  0.016353
..     ...       ...       ...  ...       ...       ...       ...
76  1700.0  0.169739  3.433316  ...  0.003044  0.016795  0.001108
77  1720.0  0.169704  3.433221  ...  0.003044  0.016795  0.001108
78  1740.0  0.169668  3.433125  ...  0.003044  0.016795  0.001108
79  1760.0  0.169632  3.433030  ...  0.003044  0.016795  0.001108
80  1825.0  0.169636  3.433026  ...  0.003044  0.016795  0.001108

[55890 rows x 32 columns]
```

```
# In these dataframes, data have been concatenated one on top of the
other.
# We have 690 samples * 81 (length of each time series) = 55890 rows.

test_files = get_file_list_from_dir(path=".", datadir="test")
len(test_files)
```

```
230
```

```
dtest = pd.concat((pd.read_csv(f) for f in test_files))
dtest
```

```
     times         A         B  ...        p3        p4        p5
0      0.0  0.133810  4.776655  ...  0.007466  0.002743  0.040281
1     20.0  0.132701  4.772421  ...  0.007466  0.002743  0.040281
2     40.0  0.131602  4.768180  ...  0.007466  0.002743  0.040281
3     60.0  0.130507  4.763926  ...  0.007466  0.002743  0.040281
4     80.0  0.129422  4.759661  ...  0.007466  0.002743  0.040281
..     ...       ...       ...  ...       ...       ...       ...
76  1700.0  0.349419  5.362785  ...  0.035835  0.048688  0.008009
77  1720.0  0.349006  5.362041  ...  0.035835  0.048688  0.008009
78  1740.0  0.348592  5.361306  ...  0.035835  0.048688  0.008009
79  1760.0  0.348177  5.360567  ...  0.035835  0.048688  0.008009
80  1825.0  0.348189  5.360571  ...  0.035835  0.048688  0.008009

[18630 rows x 32 columns]
```

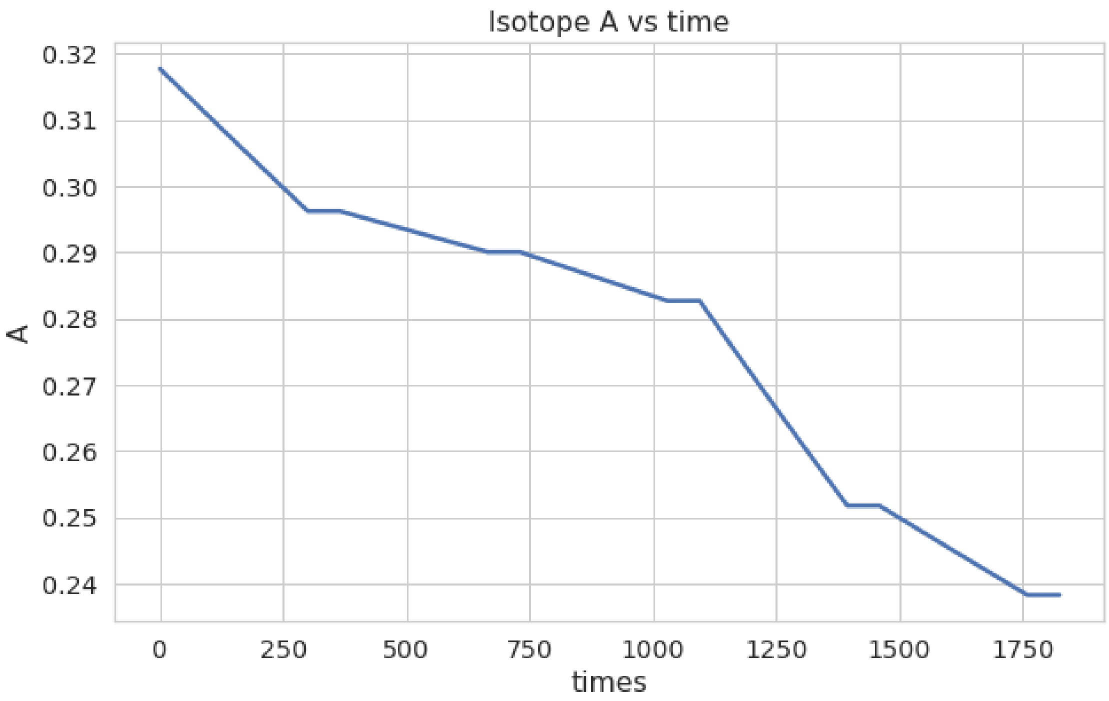
```
### One sample
#Let us first take a look at one of the 690 train samples (or
simulations). We will put times as the index to ease plotting.

smpl1 = dtrain.reset_index(drop=True).iloc[:81]
smpl1 = smpl1.set_index('times')
smpl1
```

```
               A         B         C  ...        p3        p4
p5
times                                 ...

0.0     0.317729  3.265168  0.205470  ...  0.008332  0.035934
0.016353
```

```
20.0    0.316280  3.263388  0.204874  ...  0.008332  0.035934
0.016353
40.0    0.314831  3.261604  0.204346  ...  0.008332  0.035934
0.016353
60.0    0.313387  3.259812  0.203877  ...  0.008332  0.035934
0.016353
80.0    0.311945  3.258014  0.203466  ...  0.008332  0.035934
0.016353
...          ...       ...       ...  ...       ...       ...       ..
.
1700.0  0.240961  3.157960  0.208606  ...  0.008332  0.035934
0.016353
1720.0  0.240071  3.156546  0.208798  ...  0.008332  0.035934
0.016353
1740.0  0.239182  3.155125  0.208985  ...  0.008332  0.035934
0.016353
1760.0  0.238295  3.153707  0.209168  ...  0.008332  0.035934
0.016353
1825.0  0.238300  3.153707  0.210896  ...  0.008332  0.035934
0.016353

[81 rows x 31 columns]
```

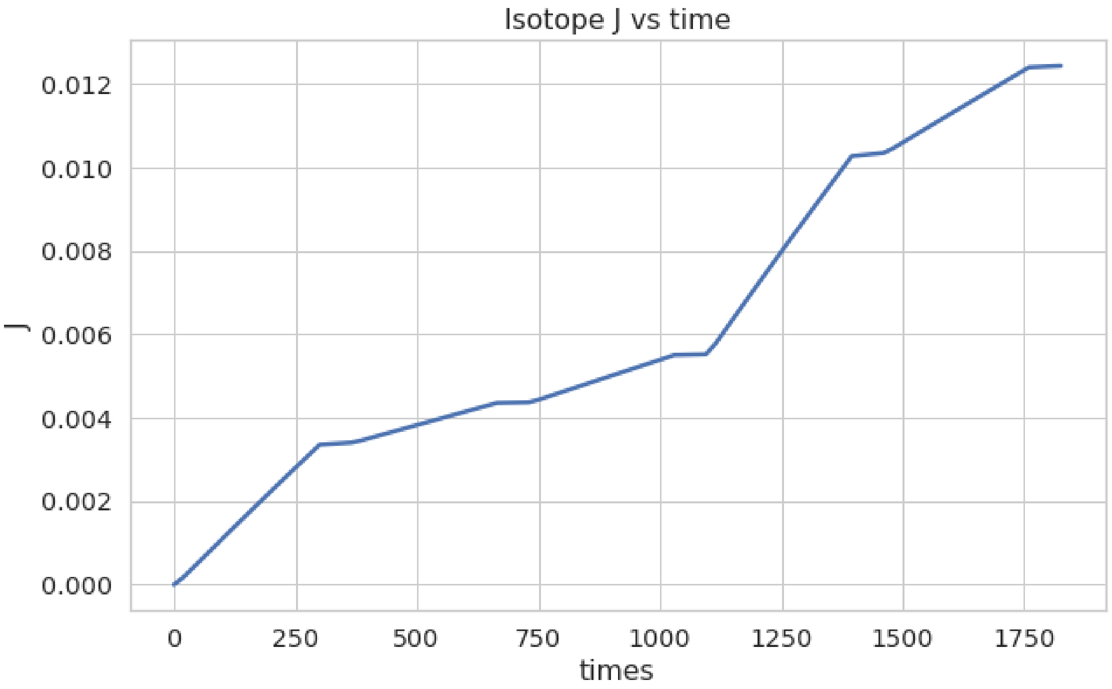
```
#We can plot each individual isotope content vs times:
sns.set(rc={'figure.figsize': (10, 6)})
sns.set_style("whitegrid")
sns.set_context("notebook", font_scale=1.3, rc={"lines.linewidth":
2.5})

sns.lineplot(data=smpl1['A']).set(title="Isotope A vs time")
```

```
[Text(0.5, 1.0, 'Isotope A vs time')]
```



```
sns.lineplot(data=smpl1['J']).set(title="Isotope J vs time")
```

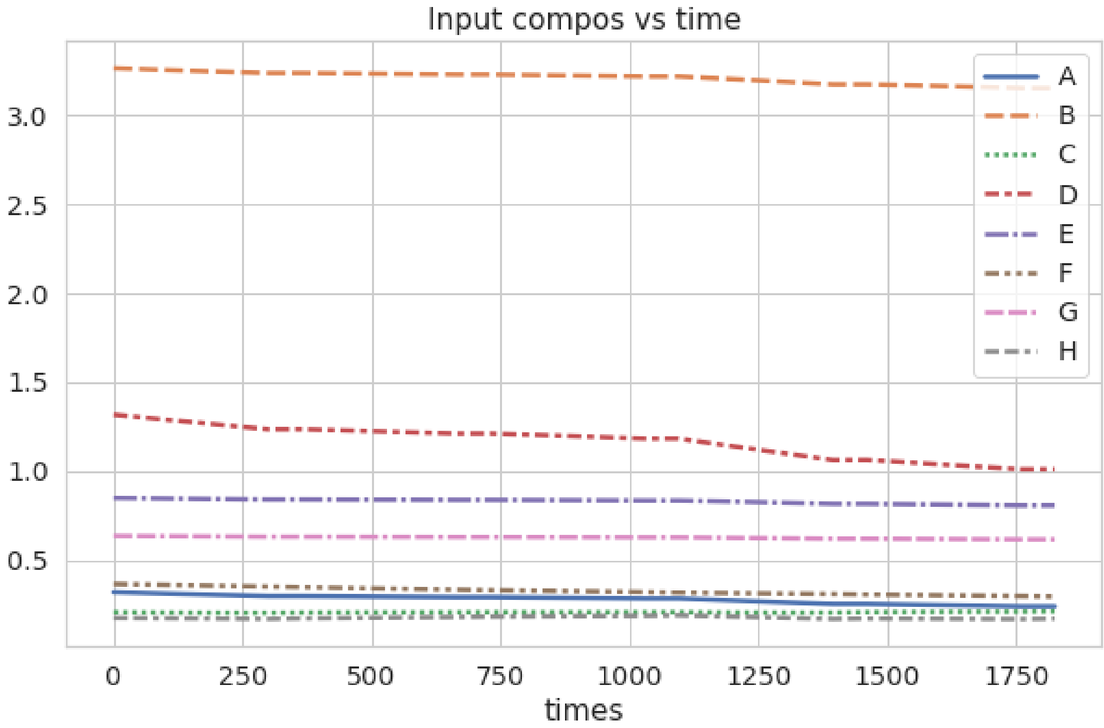
```
[Text(0.5, 1.0, 'Isotope J vs time')]
```



```
#Plots reveal irradiation and maintenance cycles.

#Let's plot all input composition:
alphabet = list(string.ascii_uppercase)  # to ease the manipulation of
```

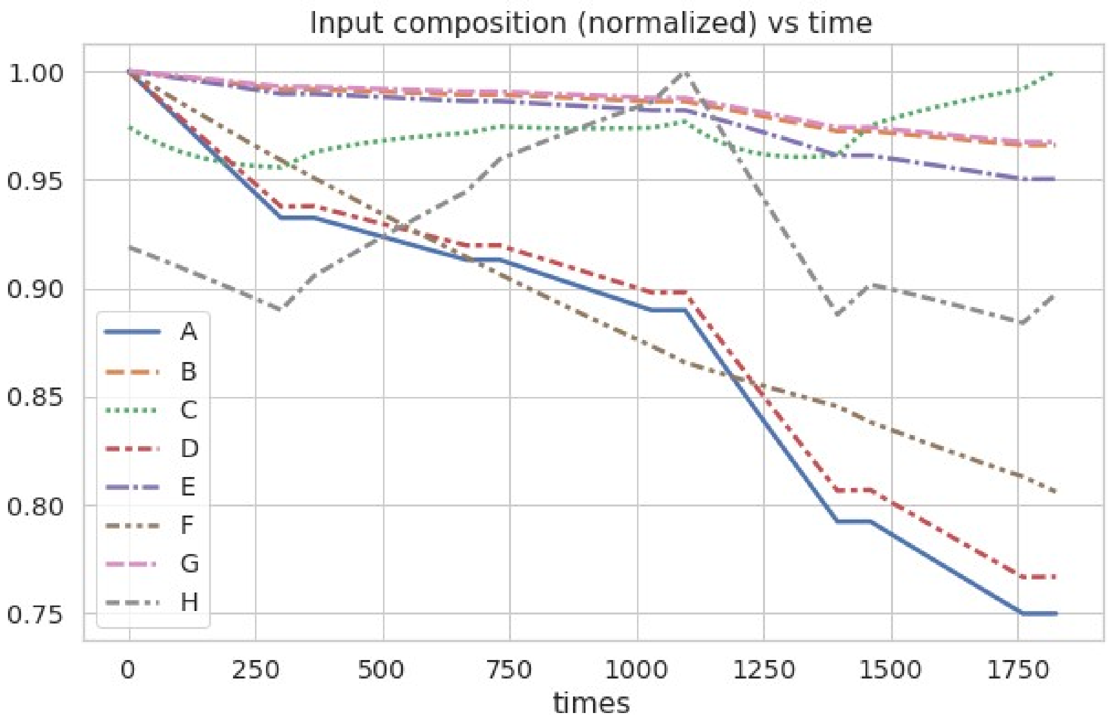
```
the data

# The input compositions are isotopes A -> H
input_compos = alphabet[:8]
sns.lineplot(data=smpl1[input_compos]).set(title="Input compos vs
time")
```

```
[Text(0.5, 1.0, 'Input compos vs time')]
```



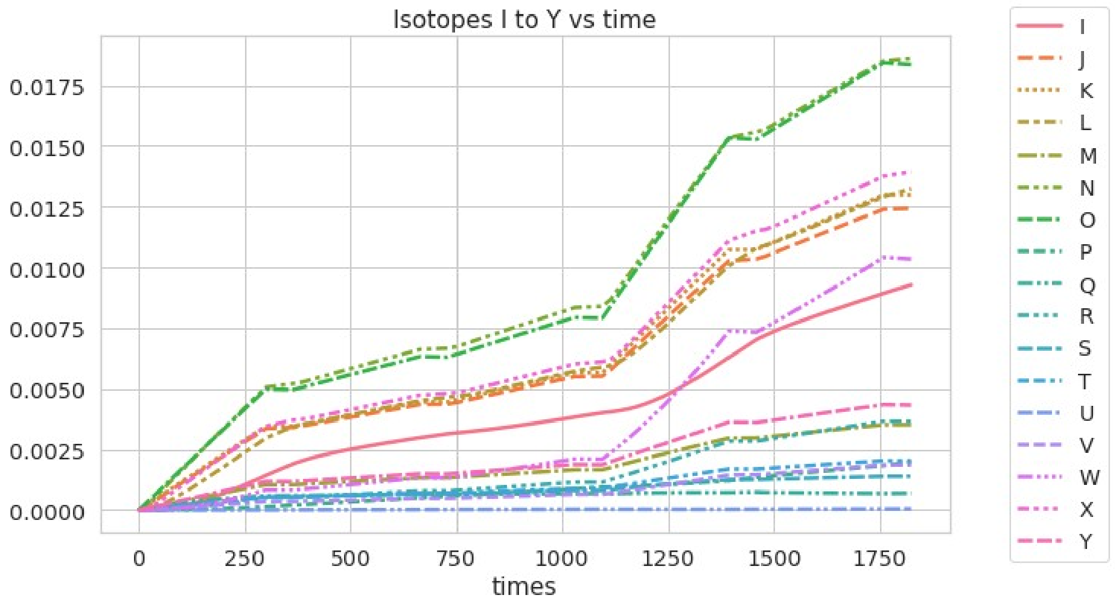
```
#We need a data scaling to better see all the curves. Isotope B is for
example much higher than the other isotopes.
smpl1_max = smpl1.max()
smpl1_norm = smpl1 / smpl1_max

sns.lineplot(data=smpl1_norm[input_compos]).set(
    title="Input composition (normalized) vs time")
```

```
[Text(0.5, 1.0, 'Input composition (normalized) vs time')]
```



```
#And the rest of the isotopes (except Z):
g = sns.lineplot(data=smpl1[alphabet[8: -1]])
g.set(title="Isotopes I to Y vs time")
g.legend(loc='center right', bbox_to_anchor=(1.2, 0.5), ncol=1)
```

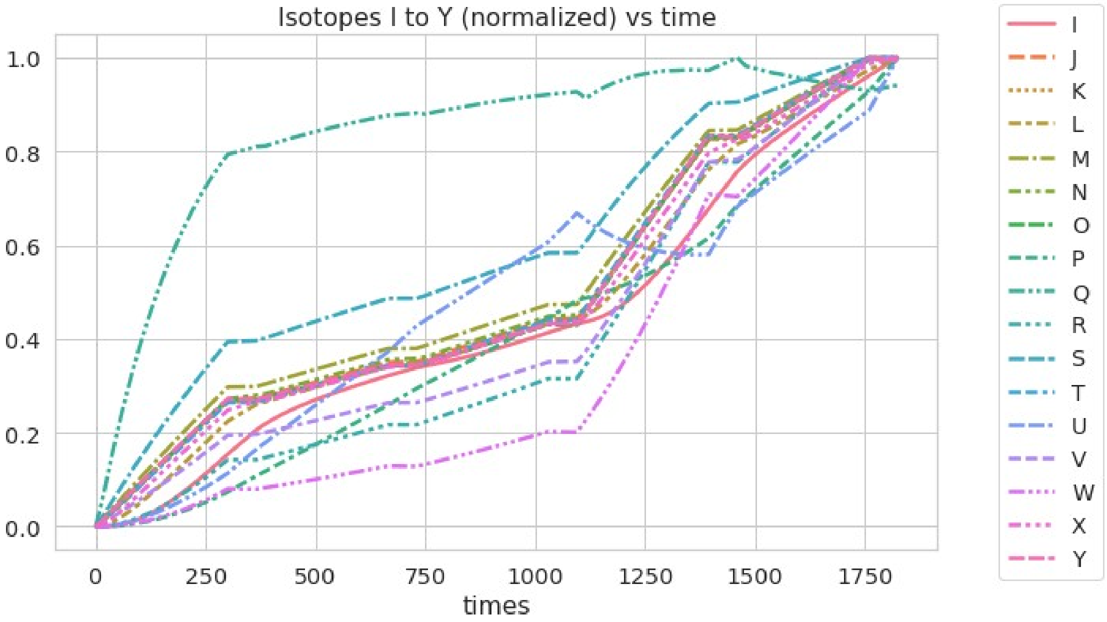
```
<matplotlib.legend.Legend at 0x7fab7ad06650>
```



```
#Normalized:
g = sns.lineplot(data=smpl1_norm[alphabet[8: -1]])
```

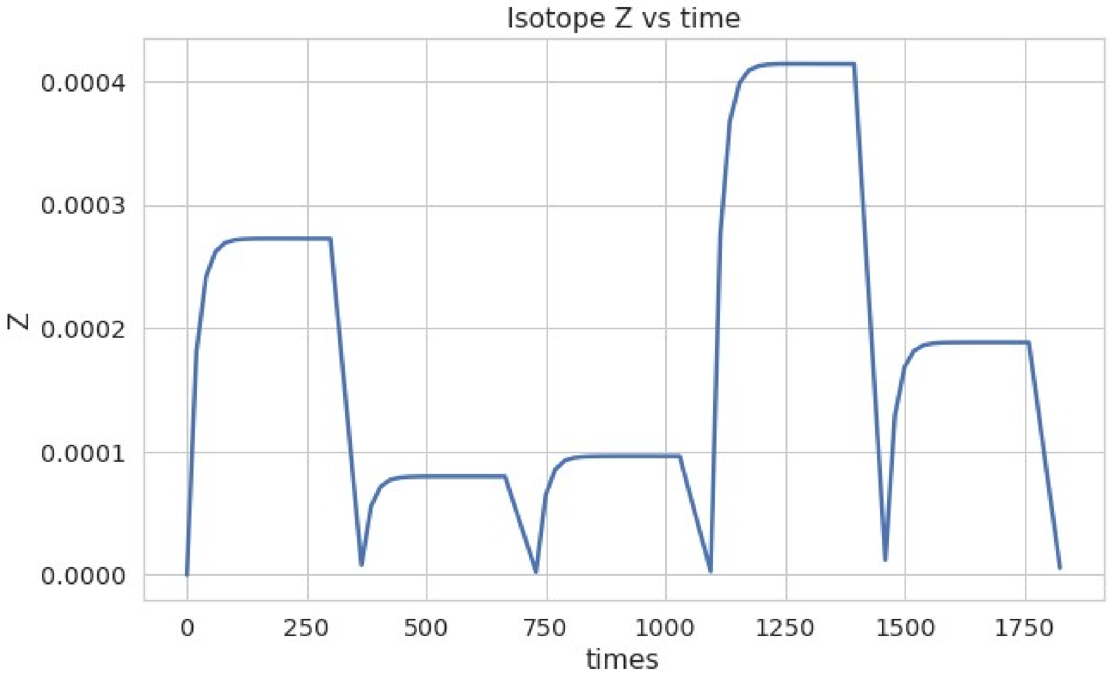
```
g.set(title="Isotopes I to Y (normalized) vs time")
g.legend(loc='center right', bbox_to_anchor=(1.2, 0.5), ncol=1)
```

```
<matplotlib.legend.Legend at 0x7fab7ad509d0>
```



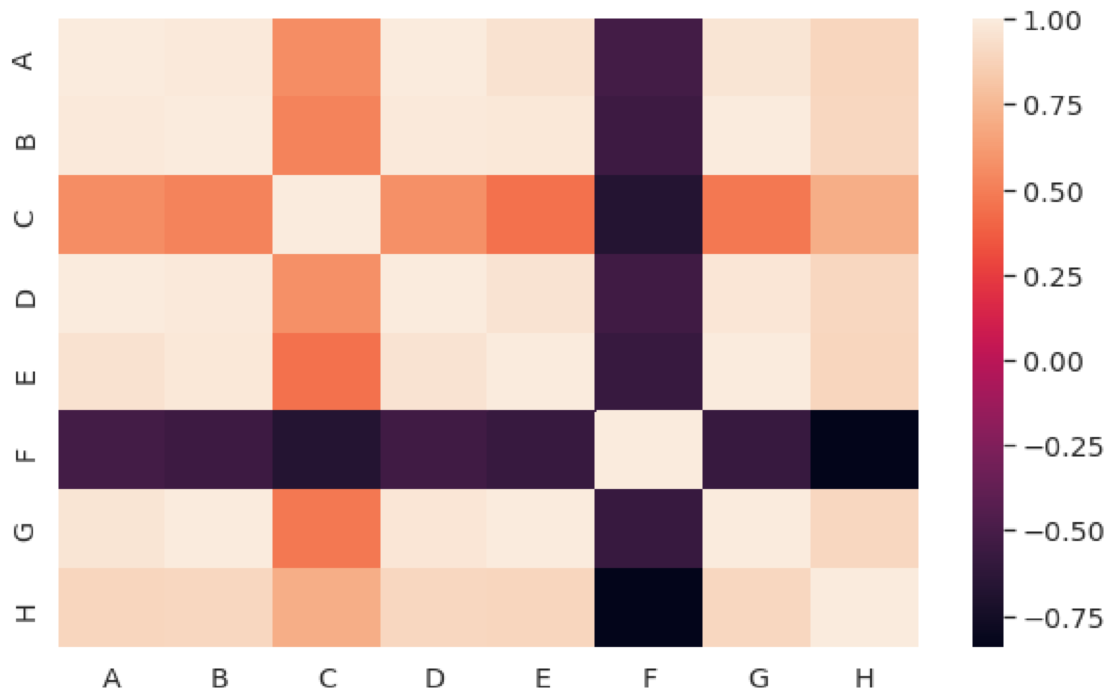
```
#Z alone:
sns.lineplot(data=smpl1['Z']).set(title="Isotope Z vs time")
```

```
[Text(0.5, 1.0, 'Isotope Z vs time')]
```



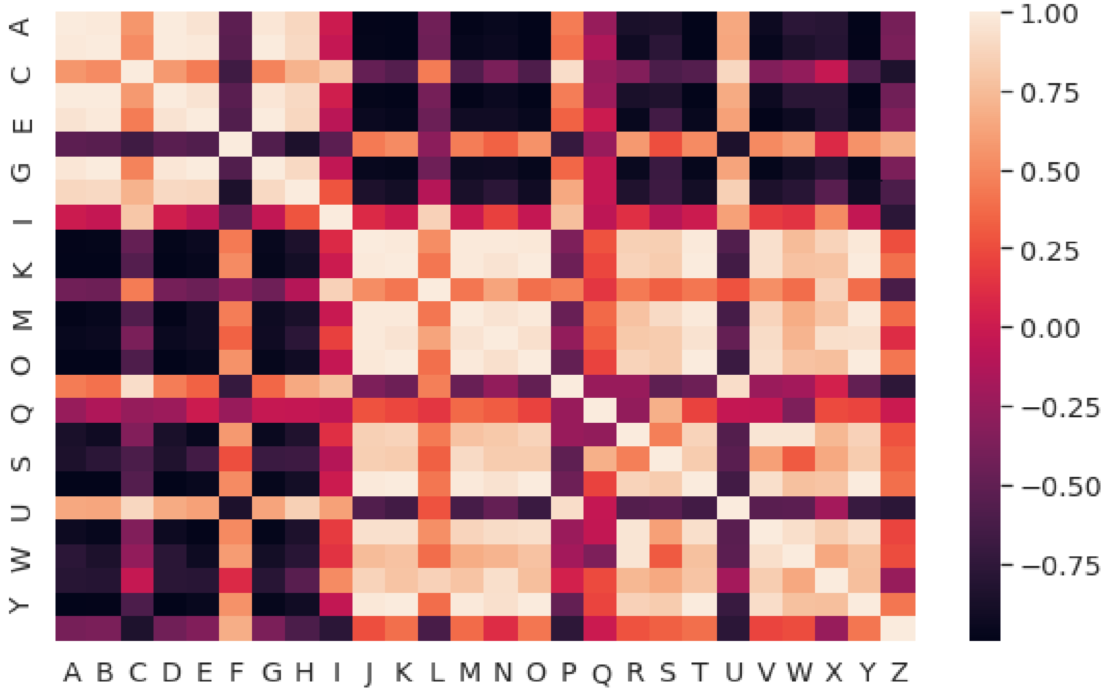
```
#We want to see if there is a pairwise correlation between the time
variation of isotopes.
#Let's compute the first discrete difference of element and plot the
heatmap of the correlation between first isotopes:
smpl1_diff = smpl1.diff()
sns.heatmap(smpl1_diff[input_compos].corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fab79a32050>
```



```
#Here we see that for example A and D are positively correlated in
their temporal variation (coefficient 1.00) and F and H are negatively
correlated (coeff < -0.75).
#And between all isotopes:
sns.heatmap(smpl1_diff[alphabet].corr())
#It shows that the "input" isotopes that mainly decline have a
strongly negative correlation in their time variation with almost all
the other isotopes which are created in the five years time. Except
for example for C and F which behave differently.
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fab799cdb50>
```



```
### Input and output data
#To separate input data from output data, use the index of the
dataframes or the "times" column.
#The input data for instance is composed of the value of each T=0
entry in the datasets. To obtain those values, simply select every
entry having T=0 in the train or test datasets :
dtrain.loc[0].shape  # equivalent to dtrain.loc[dtrain["times"] == 0.]
```

```
(690, 32)
```

```
#As said before, the train dataset regroup a total of 690 simulations
that were performed varying the 13 input parameters listed above. Here
the dataset is found to have 32 parameters at T=0, but only 13 are non
zero, as can be seen using :
dtrain.iloc[0]
```

```
times    0.000000
A        0.317729
B        3.265168
C        0.205470
D        1.316402
E        0.848286
F        0.364322
G        0.635073
H        0.173054
I        0.000000
J        0.000000
K        0.000000
L        0.000000
```

```
M        0.000000
N        0.000000
O        0.000000
P        0.000000
Q        0.000000
R        0.000000
S        0.000000
T        0.000000
U        0.000000
V        0.000000
W        0.000000
X        0.000000
Y        0.000000
Z        0.000000
p1       0.023597
p2       0.006906
p3       0.008332
p4       0.035934
p5       0.016353
Name: 0, dtype: float64
```

```
#As adverstised, the initial compositions for isotopes "I" -> "Z" are
zero, leaving only 13 input parameters.
#At T= 1825 days the (81th timestep), the composition has evolved :
dtrain.iloc[[0, 80]]
```

```
     times         A         B  ...        p3        p4        p5
0      0.0  0.317729  3.265168  ...  0.008332  0.035934  0.016353
80  1825.0  0.238300  3.153707  ...  0.008332  0.035934  0.016353

[2 rows x 32 columns]
```

```
#To get all of the unique timesteps, one can do:
timesteps = dtrain["times"].unique()
print(timesteps)
```

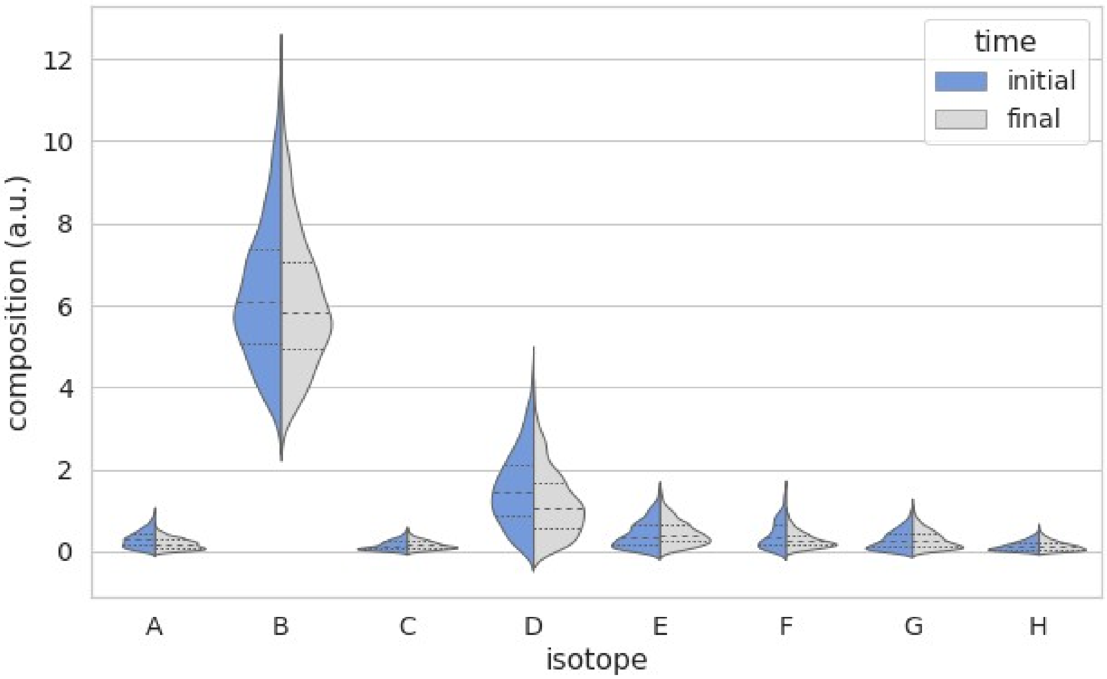
```
[   0.   20.   40.   60.   80.  100.  120.  140.  160.  180.  200.
220.
  240.  260.  280.  300.  365.  385.  405.  425.  445.  465.  485.
505.
  525.  545.  565.  585.  605.  625.  645.  665.  730.  750.  770.
790.
  810.  830.  850.  870.  890.  910.  930.  950.  970.  990. 1010.
1030.
 1095. 1115. 1135. 1155. 1175. 1195. 1215. 1235. 1255. 1275. 1295.
1315.
 1335. 1355. 1375. 1395. 1460. 1480. 1500. 1520. 1540. 1560. 1580.
1600.
 1620. 1640. 1660. 1680. 1700. 1720. 1740. 1760. 1825.]
```

```
# The input parameters are composed of the input_compos + parameters
p1 to p5
input_params = input_compos + ["p1", "p2", "p3", "p4", "p5"]
input_params
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'p1', 'p2', 'p3', 'p4', 'p5']
```

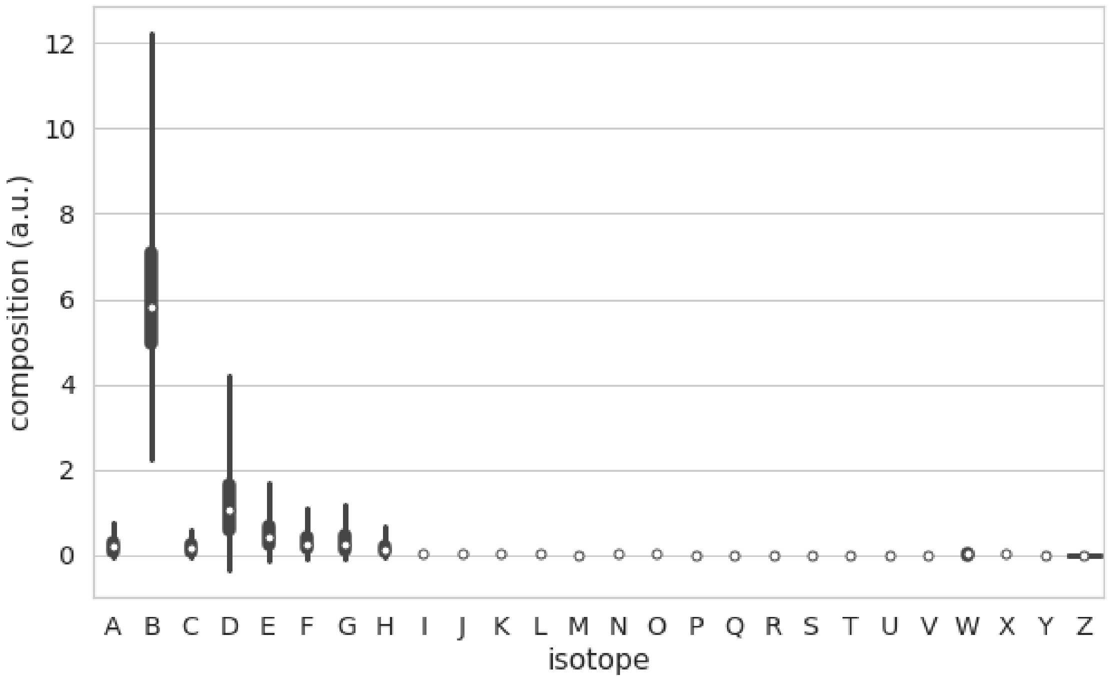
```
## Normalization of the data
#It is important to note that the composition data that make up the
database are very heterogeneous. Indeed, the isotopes present in this
database __have typical compositions that can present orders of
magnitude of differences__. This can pose serious problems of
normalization to succeed in learning the data at best.
#Let us for instance plot the distributions of the isotopes that makes
up the input parameter composition (i.e. isotopes A to H) at the
initial T=0 and final T=1825 times :

temp_initial = pd.DataFrame(
    dtrain.loc[0, ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']].melt(
    value_name="composition (a.u.)", var_name="isotope")
)
temp_final = pd.DataFrame(
    dtrain.loc[80, ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']].melt(
    value_name="composition (a.u.)", var_name="isotope")
)
temp_initial['time'] = 'initial'
temp_final['time'] = 'final'

temp = pd.concat([temp_initial, temp_final], axis=0)

# plot a violin plot for both the initial and final compositions of
the input_compos
sns.set_style("whitegrid")
sns.set_context("notebook", font_scale=1.3, rc={"lines.linewidth":
2.5, 'figure.figsize':(14, 8)})
sns.violinplot(
    data=temp,
    x="isotope",
    y="composition (a.u.)",
    hue="time",
    split=True,
    inner="quartile",
    linewidth=1,
    palette={"initial": "cornflowerblue", "final": ".85"},
)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fab77029410>
```



```
#As can be seen from the figure, most matter inside the reactor is
made up of isotope B.
#The difficulty lies in the fact that we want, in the end, to have a
precise __relative__ error on each individual isotope because it can
happen that an isotope which is present in small quantities still has
a non-negligible impact on the safety of the reactor and on the
mitigation of an accident.
#So we can't just be good in the absolute overall composition of the
matter: we need to be sufficiently precise for each individual isotope
!__
#The differences between the isotopes for the final compositions,
adding all of the 26 isotopes, is even more visible:

temp = pd.DataFrame(
    dtrain.loc[80, alphabet].melt(
    value_name="composition (a.u.)", var_name="isotope")
)
sns.violinplot(data=temp, x="isotope", y="composition (a.u.)",
inner="box")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fab76f65710>
```



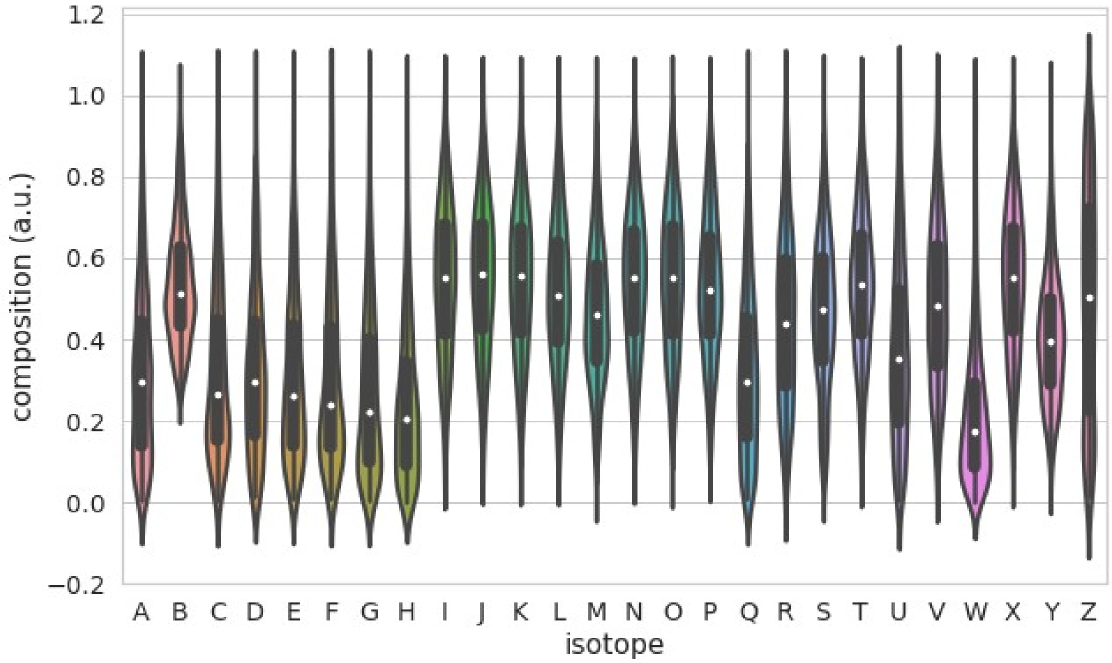
```
#There, we want to be sufficiently precise on each individual
isotopes, even if there exist ~ up to 5 orders of magnitudes between
some isotopes !

dtrain.B.max(), dtrain.Z.max()
```

```
(11.71932, 0.0006211064)
```

```
#To achieve good overall performances for this challenge, it is
believed that the first key is to find a clever way to normalize the
input and output data.
#In this notebook, we propose to normalize data just before feeding it
into the models, i.e. normalize X and Y matrices. But just to see the
effect of normalization, let's visualize isotope composition at a
given time step after scaling.
#You can change timestep in loc[ ] to see compositions at different
times.

# dtrain at a certain timestep
temp = dtrain.loc[50, alphabet]
# scaling
temp = temp / temp.max()

temp = pd.DataFrame(
    temp.melt(
    value_name="composition (a.u.)", var_name="isotope")
)
```

```
sns.violinplot(data=temp,x="isotope",y="composition (a.u.)",
inner="box")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fab76d54150>
```



```
## A simple baseline algorithm
#To benchmark the performances of a machine learning algorithm, we
first try to build a simple baseline method.
#Our goal is to predict the composition of matter inside the reactor
**at any given time** by just using its initial composition (isotopes
A --> H) and the parameters p1, ..., p5.
#The baseline algorithm presented here simply performs a multioutput
linear regression for each isotope. Here we have  80 timesteps as
output in the predictive model.

#Let us first reshape the data into a form that will be more useful
for the rest of this notebook
train_data = dtrain[alphabet].add_prefix('Y_')
train_data["times"] = dtrain["times"]
train_data = train_data[ train_data["times"] > 0.]
temp = dtrain.loc[0][input_params].reset_index(drop=True)
temp = temp.loc[temp.index.repeat(80)].reset_index(drop=True)
train_data = pd.concat([temp, train_data.reset_index(drop=True)],
axis=1)
train_data

test_data = dtest[alphabet].add_prefix('Y_')
test_data["times"] = dtest["times"]
test_data = test_data[test_data["times"] > 0.]
temp = dtest.loc[0][input_params].reset_index(drop=True)
temp = temp.loc[temp.index.repeat(80)].reset_index(drop=True)
test_data = pd.concat([temp, test_data.reset_index(drop=True)],
axis=1)
test_data
```

```
              A         B         C  ...       Y_Y       Y_Z   times
0      0.133810  4.776655  0.401556  ...  0.000141  0.000331    20.0
1      0.133810  4.776655  0.401556  ...  0.000282  0.000443    40.0
2      0.133810  4.776655  0.401556  ...  0.000422  0.000481    60.0
3      0.133810  4.776655  0.401556  ...  0.000562  0.000494    80.0
4      0.133810  4.776655  0.401556  ...  0.000702  0.000498   100.0
...         ...       ...       ...  ...       ...       ...     ...
18395  0.457563  5.532067  0.350256  ...  0.006538  0.000094  1700.0
18396  0.457563  5.532067  0.350256  ...  0.006558  0.000094  1720.0
18397  0.457563  5.532067  0.350256  ...  0.006578  0.000094  1740.0
18398  0.457563  5.532067  0.350256  ...  0.006597  0.000094  1760.0
18399  0.457563  5.532067  0.350256  ...  0.006569  0.000003  1825.0

[18400 rows x 40 columns]
```

```
### Train and test data:
#From these dataframe, let us extract X_train, y_train, X_test and
y_test for each isotope:

#For isotope A:
train_target_A = train_data.groupby(input_params)
['Y_A'].apply(list).apply(pd.Series).rename(
    columns=lambda x: 'A' + str(x + 1)).reset_index()
train_target_A
```

```
            A          B         C  ...       A78       A79       A80
0    0.005192   4.990765  0.126878  ...  0.004700  0.004694  0.004703
1    0.006727   4.876231  0.174605  ...  0.006456  0.006449  0.006461
2    0.007188   6.176172  0.109065  ...  0.005089  0.005055  0.005060
3    0.008880   4.154490  0.077639  ...  0.007385  0.007375  0.007387
4    0.011813   5.558284  0.438273  ...  0.010605  0.010602  0.010611
..        ...        ...       ...  ...       ...       ...       ...
685  0.904926   9.835680  0.103366  ...  0.517594  0.513942  0.513945
686  0.934583   9.466261  0.027023  ...  0.501101  0.497381  0.497383
687  0.957282  10.214110  0.008380  ...  0.434517  0.426852  0.426853
688  0.959747  10.057000  0.111717  ...  0.488471  0.484538  0.484540
689  0.968041   9.749631  0.002714  ...  0.344674  0.341766  0.341767

[690 rows x 93 columns]
```

```
test_target_A = test_data.groupby(input_params)
['Y_A'].apply(list).apply(pd.Series).rename(
    columns=lambda x: 'A' + str(x + 1)).reset_index()
test_target_A
```

```
            A          B         C  ...       A78       A79       A80
0    0.007282   5.636715  0.412482  ...  0.006935  0.006947  0.006956
1    0.018082   4.220983  0.253170  ...  0.014393  0.014316  0.014322
2    0.018187   8.435444  0.017857  ...  0.001101  0.001053  0.001053
3    0.024323   7.804812  0.109733  ...  0.007746  0.007722  0.007723
4    0.027303   3.948096  0.222217  ...  0.020762  0.020677  0.020683
..        ...        ...       ...  ...       ...       ...       ...
225  0.877807  10.121250  0.056728  ...  0.545612  0.537980  0.537983
226  0.897279  10.004340  0.027873  ...  0.327512  0.320783  0.320784
227  0.916215  10.383010  0.007526  ...  0.250743  0.246337  0.246338
228  0.972700   9.890109  0.029839  ...  0.581238  0.573178  0.573179
229  0.999898  10.007760  0.038949  ...  0.626794  0.618565  0.618567

[230 rows x 93 columns]
```

```
X_train = train_target_A[input_params]
# scale X_train
max_X_train = X_train.max()
X_train = X_train / max_X_train
X_train
```

```
            A         B         C  ...        p3        p4        p5
0    0.005363  0.425858  0.233685  ...  0.047443  0.385955  0.832317
1    0.006949  0.416085  0.321590  ...  0.091628  0.285955  0.939493
2    0.007425  0.527008  0.200877  ...  0.377074  0.563385  0.890780
3    0.009173  0.354499  0.142997  ...  0.949347  0.572730  0.324807
4    0.012203  0.474284  0.807217  ...  0.477787  0.711153  0.880216
..        ...       ...       ...  ...       ...       ...       ...
685  0.934802  0.839271  0.190380  ...  0.142299  0.711716  0.454782
686  0.965438  0.807748  0.049772  ...  0.175744  0.545873  0.418075
687  0.988886  0.871562  0.015433  ...  0.489252  0.128342  0.622672
688  0.991432  0.858156  0.205763  ...  0.732292  0.825660  0.488552
689  1.000000  0.831928  0.004998  ...  0.292049  0.930102  0.253247

[690 rows x 13 columns]
```

```
y_train_A = train_target_A.iloc[:, len(input_params):]
# scale y_train_A
max_y_train_A = y_train_A.max()
y_train_A = y_train_A / max_y_train_A
y_train_A
```

```
           A1        A2        A3  ...       A78       A79       A80
0    0.005417  0.005439  0.005448  ...  0.007089  0.007085  0.007099
1    0.007030  0.007071  0.007093  ...  0.009736  0.009734  0.009752
2    0.007485  0.007501  0.007498  ...  0.007676  0.007630  0.007637
3    0.009284  0.009342  0.009375  ...  0.011138  0.011131  0.011149
4    0.012348  0.012421  0.012463  ...  0.015995  0.016003  0.016016
..        ...       ...       ...  ...       ...       ...       ...
685  0.941855  0.943366  0.942399  ...  0.780632  0.775709  0.775711
686  0.968933  0.966641  0.961778  ...  0.755757  0.750712  0.750714
```

```
687  0.997376  0.999992  1.000000  ...  0.655336  0.644262  0.644261
688  0.998681  1.000000  0.998694  ...  0.736709  0.731328  0.731329
689  1.000000  0.994033  0.985478  ...  0.519834  0.515839  0.515838

[690 rows x 80 columns]
```

```
X_test = test_target_A[input_params]
# scale X_test the same way X_train was scaled
X_test = X_test / max_X_train
X_test
```

```
            A         B         C  ...        p3        p4        p5
0    0.007523  0.480976  0.759716  ...  0.865575  0.576617  0.589161
1    0.018679  0.360173  0.466292  ...  0.694389  0.880378  0.965078
2    0.018788  0.719790  0.032890  ...  0.155534  0.389847  0.417847
3    0.025126  0.665978  0.202108  ...  0.968925  0.715934  0.141899
4    0.028204  0.336888  0.409283  ...  0.926961  0.330878  0.550075
..        ...       ...       ...  ...       ...       ...       ...
225  0.906787  0.863638  0.104482  ...  0.350386  0.631457  0.897085
226  0.926902  0.853662  0.051338  ...  0.093961  0.718253  0.763929
227  0.946463  0.885974  0.013861  ...  0.895170  0.351277  0.484065
228  1.004813  0.843915  0.054957  ...  0.702929  0.035355  0.676715
229  1.032909  0.853954  0.071737  ...  0.139562  0.454501  0.876515

[230 rows x 13 columns]
```

```
y_test_A = test_target_A.iloc[:, len(input_params):]
y_test_A
```

```
           A1        A2        A3  ...       A78       A79       A80
0    0.007246  0.007210  0.007175  ...  0.006935  0.006947  0.006956
1    0.018060  0.018038  0.018016  ...  0.014393  0.014316  0.014322
2    0.017000  0.015852  0.014747  ...  0.001101  0.001053  0.001053
3    0.023882  0.023448  0.023021  ...  0.007746  0.007722  0.007723
4    0.027292  0.027281  0.027270  ...  0.020762  0.020677  0.020683
..        ...       ...       ...  ...       ...       ...       ...
225  0.873388  0.868957  0.864537  ...  0.545612  0.537980  0.537983
226  0.885411  0.873605  0.861860  ...  0.327512  0.320783  0.320784
227  0.906209  0.896269  0.886418  ...  0.250743  0.246337  0.246338
228  0.969906  0.967113  0.964323  ...  0.581238  0.573178  0.573179
229  0.993294  0.986688  0.980099  ...  0.626794  0.618565  0.618567

[230 rows x 80 columns]
```

```
##We can now apply a multioutput algorithm:
model = LinearRegression()
cv = ShuffleSplit(n_splits=10, test_size=0.25, random_state=57)
scores = cross_val_score(model, X_train, y_train_A, cv=cv,
scoring='neg_mean_absolute_percentage_error', n_jobs=-1)
errors = -scores
```

```
print(errors)
print(errors.mean())
```

```
[0.25346852 0.20156435 0.21988226 0.17279623 0.14846828 0.23597558
 0.16959568 0.22835256 0.20292559 0.26815275]
0.21011817948222147
```

```
model.fit(X_train, y_train_A)
y_pred_A = model.predict(X_test)
y_pred_A = pd.DataFrame(y_pred_A, columns=y_train_A.columns) *
max_y_train_A
y_pred_A
```

```
           A1        A2        A3  ...       A78       A79       A80
0    0.006891  0.006502  0.006115  ...  0.034296  0.034801  0.034810
1    0.019652  0.021211  0.022758  ...  0.035758  0.034816  0.034822
2    0.014498  0.010851  0.007249  ... -0.055374 -0.055763 -0.055763
3    0.021342  0.018394  0.015486  ... -0.080519 -0.079800 -0.079800
4    0.029087  0.030858  0.032614  ...  0.042192  0.042062  0.042068
..        ...       ...       ...  ...       ...       ...       ...
225  0.872249  0.866714  0.861206  ...  0.519698  0.514883  0.514886
226  0.889980  0.882714  0.875487  ...  0.481429  0.476818  0.476818
227  0.909943  0.903698  0.897486  ...  0.497787  0.493749  0.493748
228  0.967548  0.962412  0.957295  ...  0.590898  0.586030  0.586031
229  0.993329  0.986782  0.980262  ...  0.599773  0.594481  0.594483

[230 rows x 80 columns]
```

```
MAPE = mean_absolute_percentage_error(y_test_A, y_pred_A)
print(f"MAPE={MAPE:.4f}")
```

```
MAPE=0.3392
```

```
##Treating all isotopes at once:
y_train_all = []
for i in alphabet:
    y_train_all.append(
        train_data.groupby(input_params)
['Y_'+i].apply(list).apply(pd.Series).rename(
        columns=lambda x: i + str(x + 1)).reset_index().iloc[:,
len(input_params):]
    )

y_train_all = pd.concat(y_train_all, axis=1)
# scale y_train_all
max_y_train_all = y_train_all.max()
y_train_all = y_train_all / max_y_train_all
y_train_all
```

```
           A1        A2        A3  ...       Z78       Z79       Z80
0    0.005417  0.005439  0.005448  ...  0.796821  0.797535  0.797532
1    0.007030  0.007071  0.007093  ...  0.897639  0.898526  0.898522
```

```
2    0.007485  0.007501  0.007498  ...  0.872320  0.873218  0.879094
3    0.009284  0.009342  0.009375  ...  0.310309  0.310587  0.310586
4    0.012348  0.012421  0.012463  ...  0.846158  0.846919  0.846917
..        ...       ...       ...  ...       ...       ...       ...
685  0.941855  0.943366  0.942399  ...  0.467043  0.467328  0.470475
686  0.968933  0.966641  0.961778  ...  0.432508  0.432802  0.435715
687  0.997376  0.999992  1.000000  ...  0.662416  0.662534  0.662535
688  0.998681  1.000000  0.998694  ...  0.502053  0.502394  0.502394
689  1.000000  0.994033  0.985478  ...  0.268515  0.268661  0.268661

[690 rows x 2080 columns]
```

```
y_test_all = []
for i in alphabet:
    y_test_all.append(
        test_data.groupby(input_params)
['Y_'+i].apply(list).apply(pd.Series).rename(
        columns=lambda x: i + str(x + 1)).reset_index().iloc[:,
len(input_params):]
    )

y_test_all = pd.concat(y_test_all, axis=1)
y_test_all
```

```
           A1        A2        A3  ...       Z78       Z79       Z80
0    0.007246  0.007210  0.007175  ...  0.000334  0.000334  0.000010
1    0.018060  0.018038  0.018016  ...  0.000548  0.000548  0.000016
2    0.017000  0.015852  0.014747  ...  0.000239  0.000239  0.000007
3    0.023882  0.023448  0.023021  ...  0.000081  0.000081  0.000002
4    0.027292  0.027281  0.027270  ...  0.000319  0.000319  0.000009
..        ...       ...       ...  ...       ...       ...       ...
225  0.873388  0.868957  0.864537  ...  0.000541  0.000541  0.000016
226  0.885411  0.873605  0.861860  ...  0.000465  0.000464  0.000014
227  0.906209  0.896269  0.886418  ...  0.000294  0.000293  0.000009
228  0.969906  0.967113  0.964323  ...  0.000419  0.000419  0.000012
229  0.993294  0.986688  0.980099  ...  0.000533  0.000532  0.000016

[230 rows x 2080 columns]
```

```
model = LinearRegression()
cv = ShuffleSplit(n_splits=10, test_size=0.25, random_state=57)
scores = cross_val_score(model, X_train, y_train_all, cv=cv,
scoring='neg_mean_absolute_percentage_error', n_jobs=-1)
errors = -scores
errors.mean()
```

```
0.5278524309265935
```

```
model.fit(X_train, y_train_all)
y_pred_all = model.predict(X_test)
y_pred_all = pd.DataFrame(y_pred_all, columns=y_train_all.columns) *
```

```
max_y_train_all
y_pred_all
```

```
           A1        A2        A3  ...       Z78       Z79       Z80
0    0.006891  0.006502  0.006115  ...  0.000333  0.000333  0.000010
1    0.019652  0.021211  0.022758  ...  0.000552  0.000552  0.000016
2    0.014498  0.010851  0.007249  ...  0.000240  0.000240  0.000007
3    0.021342  0.018394  0.015486  ...  0.000079  0.000079  0.000002
4    0.029087  0.030858  0.032614  ...  0.000313  0.000313  0.000009
..        ...       ...       ...  ...       ...       ...       ...
225  0.872249  0.866714  0.861206  ...  0.000529  0.000529  0.000016
226  0.889980  0.882714  0.875487  ...  0.000454  0.000454  0.000013
227  0.909943  0.903698  0.897486  ...  0.000293  0.000293  0.000009
228  0.967548  0.962412  0.957295  ...  0.000405  0.000405  0.000012
229  0.993329  0.986782  0.980262  ...  0.000520  0.000520  0.000015

[230 rows x 2080 columns]
```

```
MAPE = mean_absolute_percentage_error(y_test_all, y_pred_all)
print(f"MAPE={MAPE:.4f}")
```

```
MAPE=1.3653
```

```
%tensorflow_version 2.x
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

```
X_test.columns
```

```
Index(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'p1', 'p2', 'p3', 'p4',
'p5'], dtype='object')
```

```
max_X_train
```

```
A      0.968041
B     11.719320
C      0.542943
D      4.514485
E      1.528256
F      1.549679
G      1.160611
H      0.539508
p1     0.049956
p2     0.049783
p3     0.049947
p4     0.049941
```

```
p5     0.049851
dtype: float64
```

```
X_train.shape[1]
```

```
13
```

```
for i in range(np.shape(X_train)[1]):
  plt.plot(np.array(pd.DataFrame(X_train).drop([2,7],axis=1))[:][i])
```

```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()
X_train_stand = scaler.fit_transform(X_train)
for i in range(np.shape(X_train_stand)[1]):
  plt.plot(X_train[:][i])
```

Most effective model that i found for the MAPE error. Resulting MAPE = 4.8e-02

```
import numpy as np
import pandas as pd
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingRandomSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.base import BaseEstimator
from sklearn.neural_network import MLPRegressor
from sklearn.compose import TransformedTargetRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import SplineTransformer
from catboost import CatBoostRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import PolynomialFeatures
import lightgbm as lgb

#test
class Regressor(BaseEstimator):
  def __init__(self):
    self.splineTransformer = SplineTransformer(degree=3, n_knots=4,
knots='quantile', extrapolation='continue')
    self.columns_dropped = []
    self.poly = PolynomialFeatures(interaction_only=True)
    self.model = LinearRegression()
    self.scalerinput = StandardScaler()
    self.modelsreg = {}
    self.scaleroutputs = {}

  def fit(self, X, Y):
    X = np.delete(X, self.columns_dropped, axis=1)
    X = 
self.splineTransformer.fit_transform(self.scalerinput.fit_transform(X)
)
    X = self.poly.fit_transform(X)
    for i in range(Y.shape[1]):
      outputs_i = Y[:, i:i+1]
      self.modelsreg[i] = LinearRegression()
      self.scaleroutputs[i] = StandardScaler()
      outputs_i = (outputs_i)**(1/3)
      outputs_i = self.scaleroutputs[i].fit_transform(outputs_i)
      self.modelsreg[i].fit(X,outputs_i)

    return self

  def predict(self, X):
    X = np.delete(X, self.columns_dropped, axis=1)
    X = 
self.splineTransformer.transform(self.scalerinput.transform(X))
    X = self.poly.fit_transform(X)
    res = np.concatenate([
        scaler.inverse_transform(
          self.modelsreg[i].predict(X))**3 for i, scaler in 
self.scaleroutputs.items()
      ], axis=1)
    return res
```

Other models that i tried

```
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingRandomSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.base import BaseEstimator
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import TransformedTargetRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.preprocessing import RobustScaler
```

```
mlp = MLPRegressor(random_state=57)
param_grid = {'hidden_layer_sizes': [(2000, 2000),(3500, 3500), (5000,
5000)],
              'batch_size' : [3],
              'activation': ['tanh'],
              'solver': ['adam'],
              'power_t': [0.5],
              'alpha': [0.0001],
              'max_iter': [400],
              'early_stopping': [False],
              'warm_start': [True],
              'epsilon' : [1e-8]
              }

model = HalvingRandomSearchCV(mlp,
                            scoring=mean_absolute_percentage_error,
                            n_jobs=-1,
                            param_distributions = param_grid,
                            factor=3,
                            cv = 5,
                            verbose=1)

transformer = RobustScaler().fit(X_train)
X_train = transformer.transform(X_train)
model.fit(X_train, y_train_all)

X_test = transformer.transform(X_test)
y_pred_all = model.predict(X_test)
MAPE = mean_absolute_percentage_error(y_test_all, y_pred_all)

print(f"MAPE={MAPE:.4f}")

print("Best parameters set found on development set:")
print(model.best_params_)

# LSTM for international airline passengers problem with regression
framing
import numpy
import keras
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras import layers
from tensorflow.keras import activations
import tensorflow as tf
```

```
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

# create and fit the LSTM network
model = Sequential()
model.add(LSTM(512,return_sequences=True,
input_shape=(X_train.shape[1], X_train.shape[-1])))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.3))
model.add(layers.Dense(256, activation=tf.nn.tanh))
model.add(layers.BatchNormalization())
model.add(layers.Dense(256))
model.add(layers.Activation('tanh'))
model.add(LSTM(128,return_sequences=False))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(128, activation=tf.nn.tanh))
model.add(layers.Dropout(0.3))
model.add(Dense(256))
model.add(layers.Dropout(0.3))
model.add(layers.Activation.tanh)
model.add(LSTM(512,return_sequences=True,
input_shape=(X_train.shape[1], X_train.shape[-1])))
model.add(layers.BatchNormalization())
model.add(layers.Activation.tanh)
model.add(layers.Dropout(0.5))
model.add(LSTM(256,return_sequences=False))
model.add(layers.Dropout(0.5))
model.add(layers.BatchNormalization())
model.add(layers.Activation.relu)
model.add(layers.Dense(128, activation=tf.nn.tanh))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(64, activation=tf.nn.tanh))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(1))
model.compile(loss='mean_absolute_percentage_error', optimizer='adam')
model.fit(X_train, y_train_all)

# make predictions
trainPredict = model.predict(X_train)
testPredict = model.predict(X_test)

# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
testPredict = scaler.inverse_transform([testPredict])

testPredict = pd.DataFrame(y_pred_all, columns=testPredict.columns) *
max_testPredict
```

```
MAPE = mean_absolute_percentage_error(y_test_all, testPredict)
print(f"MAPE={MAPE:.4f}")

import sklearn
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingRandomSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.base import BaseEstimator
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import TransformedTargetRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.preprocessing import RobustScaler
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import ensemble
from sklearn.metrics import mean_squared_error

xgb = XGBRegressor(random_state=57)
param_grid_2 = {'max_depth': [i for i in range(6,13)],
              'n_estimators' : [100,1000,10000],
              'min_child_weight': [0.5],
              'colsample_bytree': [0.8],
              'subsample': [0.8],
              'eta': [0.1],
              'seed': [57]
              }

model2 = HalvingRandomSearchCV(xgb,
                              scoring='neg_mean_absolute_error',
                              n_jobs=-1,
                              param_distributions = param_grid_2,
                              factor=3,
                              cv = 10,
                              verbose=1)

transformer2 = RobustScaler().fit(X_train)
transformer2.transform(X_train)
transformer2.transform(X_test)

model2.fit(
    X_train,
    y_train_all,
    eval_metric="mape",
    eval_set=[(X_train, y_train_all), (X_test, y_test_all)],
    verbose=True,
    early_stopping_rounds = 20)
```

```
y_pred_all_2 = model2.predict(X_test)
y_pred_all_2 = pd.DataFrame(y_pred_all_2, columns=y_train_all.columns)
* max_y_train_all

MAPE = mean_absolute_percentage_error(y_test_all, y_pred_all_2)
print(f"MAPE={MAPE:.4f}")

import sklearn
sorted(sklearn.metrics.SCORERS.keys())

# Training
model9 = XGBRegressor(
    max_depth=10,
    n_estimators=1000,
    min_child_weight=0.5,
    colsample_bytree=0.8,
    subsample=0.8,
    eta=0.1,
    seed=42)

model9.fit(
    X_train,
    Y_train,
    eval_metric="rmse",
    eval_set=[(X_train, Y_train), (X_valid, Y_valid)],
    verbose=True,
    early_stopping_rounds = 20)

# Import the model we are using
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import ensemble
from sklearn.metrics import mean_squared_error

x = XGBRegressor(random_state = 44, n_jobs = 8, n_estimators =
10000,min_child_weight = 5,subsample = 0.6,  max_depth=10, verbosity =
3)
transformer = RobustScaler().fit(X_train)
transformer.transform(X_train)
x.fit(X_train, y_train_all)
print('xgboost train score: ', x.score(X_train, y_train_all))
transformer2 = RobustScaler().fit(X_train)
transformer2.transform(X_test)
predictions = x.predict(X_test)
print('xgboost test score: ', x.score(X_test, predictions))
predictions = pd.DataFrame(predictions, columns=y_train_all.columns) *
max_y_train_all
```

```
MAPE = mean_absolute_percentage_error(y_test_all, predictions)
print(f"MAPE={MAPE:.4f}")

from sklearn.linear_model import LinearRegression
from sklearn.base import BaseEstimator
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import TransformedTargetRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_percentage_error

#test
class Regressor(BaseEstimator):
    def __init__(self):
        self.model = MLPRegressor()
        pipeline = Pipeline(steps=[('normalize', MinMaxScaler()),
('model', MLPRegressor(random_state=57))])
        model_mlp = TransformedTargetRegressor(regressor=pipeline,
transformer=MinMaxScaler())

        param_grid = {
            'hidden_layer_sizes': [(300,300,300,300,),
(200,200,200,200,), (200,200,200,), (300,300,300,), (400,400,400,),
(200,200,), (300,300,)],
            'batch_size' : [5, 10, 20, 40, 60, 80, 100],
            'activation': ['identity', 'logistic', 'tanh', 'relu'],
            'solver': ['lbfgs', 'sgd', 'adam'],
            'learning_rate': ['constant', 'invscaling', 'adaptive'],
            'learning_rate_init': [0.01,0.001],
            'power_t': [0.005,0.05,0.5],
            'alpha': [0.1,0.001,0.0001],
            'max_iter': [400,800,1000],
            'early_stopping': [False,True],
            'warm_start': [False,True],
            'epsilon' : [1e-8,1e-4,1e-10]
            }
        self.model = GridSearchCV(model_mlp,

scoring=mean_absolute_percentage_error,
                                  n_jobs=-1,
                                  param_grid = param_grid,
                                  cv = 10)
    def fit(self, X, Y):
        self.model.fit(X, Y)

    def predict(self, X):
        res = self.model.predict(X)
        return res

# -*- coding: utf-8 -*-
```

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.base import BaseEstimator
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import TransformedTargetRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.preprocessing import RobustScaler

#test
class Regressor(BaseEstimator):
    def __init__(self):
      mlp = MLPRegressor(random_state=57)
      param_grid = {
          'hidden_layer_sizes': [(300,300,300,300,),
(200,200,200,200,), (200,200,200,), (300,300,300,), (400,400,400,),
(200,200,), (300,300,)],
          'batch_size' : [5, 10, 20, 40, 60, 80, 100],
          'activation': ['identity', 'logistic', 'tanh', 'relu'],
          'solver': ['lbfgs', 'sgd', 'adam'],
          'learning_rate': ['constant', 'invscaling', 'adaptive'],
          'learning_rate_init': [0.01,0.001],
          'power_t': [0.005,0.05,0.5],
          'alpha': [0.1,0.001,0.0001],
          'max_iter': [400,800,1000],
          'early_stopping': [False,True],
          'warm_start': [False,True],
          'epsilon' : [1e-8,1e-4,1e-10]
        }
      self.model = GridSearchCV(mlp,

scoring=mean_absolute_percentage_error,
                                  n_jobs=-1,
                                  param_grid = param_grid,
                                  cv = 10)
    def fit(self, X, Y):
      transformer = RobustScaler().fit(X)
      transformer.transform(X)
      self.model.fit(X, Y)

    def predict(self, X):
      res = self.model.predict(X)
      return res

# summarize the results of the grid search
print(grid.best_score_)
print(grid.best_estimator)
```

```
for key, value in parameters.items():
  model = MLPRegressor(solver="sgd", hidden_layer_sizes=(50,50,50),
max_iter=400, batch_size=i, random_state=57)
  model.fit(X_train, y_train_all)
  y_pred_all = model.predict(X_test)
  y_pred_all = pd.DataFrame(y_pred_all, columns=y_train_all.columns) *
max_y_train_all
  MAPE = mean_absolute_percentage_error(y_test_all, y_pred_all)
  tab.append(MAPE)
  print(f"MAPE={MAPE:.4f}")

parameters = {'solver':['lbfgs', 'sgd', 'adam'],
              'max_iter':[100,200,300],
              'hidden_layer_sizes':[100,150,200,240],
              'batch_size':[1,5,10, 20, 40, 60, 80, 100],
              'activation': ['identity', 'logistic', 'tanh', 'relu']}
from sklearn.neural_network import MLPRegressor
for key, value in parameters.items():
  print(key)
  print(value)

model.get_params().keys()

model.fit(X_train, y_train_all)

y_pred_all = model.predict(X_test)
y_pred_all = pd.DataFrame(y_pred_all, columns=y_train_all.columns) *
max_y_train_all
y_pred_all

MAPE = mean_absolute_percentage_error(y_test_all, y_pred_all)
print(f"MAPE={MAPE:.4f}")

## Quick submission test
#Finally, make sure the local processing goes through by running the
!ramp-test --submission <submission folder>
#If you want to quickly test the that there are no obvious code
errors, use the `--quick-test` flag to only use a small subset of the
data (training and testing sets with 5 elements each).
!ramp-test --submission <submission folder> --quick-test
#See the [online
documentation](https://paris-saclay-cds.github.io/ramp-docs/ramp-
workflow/stable/using_kits.html) for more details.

!python problem.py

from sklearn.model_selection import GridSearchCV

# define the grid search parameters
hidden_layer_tab = [100,150,200,240]
batch_size_tab = [10, 20, 40, 60, 80, 100]
activation_tab = ['identity', 'logistic', 'tanh', 'relu']
```

```
max_iter_tab = [100,200,300]
epochs = [10, 50, 100]
param_grid = dict(batch_size=batch_size_tab,
                  epochs=epochs,
                  hidden_layer_sizes = hidden_layer_tab,
                  activation = activation_tab,
                  max_iter = max_iter_tab)

grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1,
cv=3)
grid_result = grid.fit(X_train, y_train_all)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

from sklearn.preprocessing import RobustScaler
from sklearn.pipeline import Pipeline
model2 = Pipeline([
            # SVM or NN work better if we have scaled the data in the
first
            # place. MinMaxScaler is the simplest one. RobustScaler or
            # StandardScaler could be an alternative.
            ("scaler", RobustScaler(quantile_range=(15, 85))),
            # The "real" estimator:
            ("model2", MLPRegressor(max_iter=600)),
        ])
# To optimize the results, we try different hyper parameters by
# using a grid search
hyper_parameter = [
            {   # Hyper parameter for lbfgs solver
                'model2__solver': ['lbfgs', 'sgd', 'adam'],
                'model2__activation': ['identity', 'logistic', 'tanh',
'relu'],
                'model2__hidden_layer_sizes': [(100,150,200,240),
(50,100,150,200)],
                'model2__random_state': [0, 42, 100, 3452],
                'model2__alpha': [0.1, 0.001, 0.0001],
                'model2__batch_size':[10, 20, 40, 60, 80, 100],
                'model2': [10, 50, 100]
            },
        ]
GridSearchCV(model2, hyper_parameter, refit=True, n_jobs=-1, cv=10)
#grid = GridSearchCV(estimator=model2, param_grid=param_grid, n_jobs=-
1, cv=3)
grid = GridSearchCV(model2, hyper_parameter, refit=True, n_jobs=-1,
cv=10)
grid_result = grid.fit(X_train, y_train_all)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

# -*- coding: utf-8 -*-
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.base import BaseEstimator
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import TransformedTargetRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.preprocessing import RobustScaler

#test
class Regressor(BaseEstimator):
    def __init__(self):
      mlp = MLPRegressor(random_state=57)
      param_grid = {
          'hidden_layer_sizes': [(300,300,300,300,),
(200,200,200,200,), (200,200,200,), (300,300,300,), (400,400,400,),
(200,200,), (300,300,)],
          'batch_size' : [5, 10, 20, 40, 60, 80, 100],
          'activation': ['identity', 'logistic', 'tanh', 'relu'],
          'solver': ['lbfgs', 'sgd', 'adam'],
          'learning_rate': ['constant', 'invscaling', 'adaptive'],
          'learning_rate_init': [0.01,0.001],
          'power_t': [0.005,0.05,0.5],
          'alpha': [0.1,0.001,0.0001],
          'max_iter': [400,800,1000],
          'early_stopping': [False,True],
          'warm_start': [False,True],
          'epsilon' : [1e-8,1e-4,1e-10]
        }
      self.model = GridSearchCV(mlp,

scoring=mean_absolute_percentage_error,
                                  n_jobs=-1,
                                  param_grid = param_grid,
                                  cv = 10)
    def fit(self, X, Y):
      transformer = RobustScaler().fit(X)
      transformer.transform(X)
      self.model.fit(X, Y)

    def predict(self, X):
      res = self.model.predict(X)
      return res
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingRandomSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.base import BaseEstimator
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import TransformedTargetRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.preprocessing import RobustScaler

mlp = MLPRegressor(random_state=57)
param_grid = {'hidden_layer_sizes': [(2000, 2000),(3500, 3500), (5000,
5000)],
              'batch_size' : [3],
              'activation': ['tanh'],
              'solver': ['adam'],
              'power_t': [0.5],
              'alpha': [0.0001],
              'max_iter': [400],
              'early_stopping': [False],
              'warm_start': [True],
              'epsilon' : [1e-8]
              }

model = HalvingRandomSearchCV(mlp,
                            scoring=mean_absolute_percentage_error,
                            n_jobs=-1,
                            param_distributions = param_grid,
                            factor=3,
                            cv = 5,
                            verbose=1)

transformer = RobustScaler().fit(X_train)
transformer.transform(X_train)
model.fit(X_train, y_train_all)

transformer.transform(X_test)
y_pred_all = model.predict(X_test)
y_pred_all = pd.DataFrame(y_pred_all, columns=y_train_all.columns) *
max_y_train_all

MAPE = mean_absolute_percentage_error(y_test_all, y_pred_all)
```

```
print(f"MAPE={MAPE:.4f}")
# LSTM for international airline passengers problem with regression
framing
import numpy
import keras
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras import layers
from tensorflow.keras import activations

# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

# create and fit the LSTM network
model = Sequential()
model.add(LSTM(32,return_sequences=True,
input_shape=(X_train.shape[1], X_train.shape[-1])))
model.add(layers.Dropout(0.5))
model.add(layers.BatchNormalization())
model.add(layers.Dense(128))
model.add(layers.Activation.tanh)
model.add(LSTM(64,return_sequences=False))
model.add(layers.Dropout(0.5))
model.add(Dense(128))
model.add(layers.Activation.tanh)
model.add(LSTM(128,return_sequences=True,
input_shape=(X_train.shape[1], X_train.shape[-1])))
model.add(layers.BatchNormalization())
model.add(layers.Activation.tanh)
model.add(layers.Dropout(0.5))
model.add(LSTM(64,return_sequences=False))
model.add(layers.BatchNormalization())
model.add(layers.Activation.relu)
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1))
model.compile(loss='mean_absolute_percentage_error', optimizer='adam')
model.fit(X_train, y_train_all)

# make predictions
trainPredict = model.predict(X_train)
testPredict = model.predict(X_test)
```

```
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])

from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingRandomSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.base import BaseEstimator
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import TransformedTargetRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from catboost import CatBoostRegressor
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.ensemble import RandomForestRegressor
import lightgbm as lgb

#test
class Regressor(BaseEstimator):
  def __init__(self):
    reg1 = KNeighborsRegressor(n_jobs=-1)
    reg2 = XGBRegressor(n_jobs=-1)
    reg3 = LGBMRegressor(n_jobs=-1)
    reg4 = CatBoostRegressor()
    reg5 = RandomForestRegressor()

    param1 = {
      'algorithm': ['auto'],
      'leaf_size': [30],
      'metric': ['mape'],
      'metric_params': None,
      'n_jobs': None,
      'n_neighbors': [5],
      'p': [2],
      'weights': ['uniform']
    }
    param2 = {}
    param3 = {
      'objective': ['regression'],
      'metric' : ['rmse'],
      'num_leaves' : [64],
      'learning_rate' : [0.005],
      'bagging_fraction' : [0.7],
      'feature_fraction' : [0.5],
      'bagging_frequency' : [6],
      'bagging_seed' : [42],
      'verbosity' : [1],
      'seed': [42],
    }
    param4 = {
      'depth': [4,6,8,10,12,14,16],
      'learning_rate' : [0.01, 0.05, 0.1],
      'l2_leaf_reg' : [0.1,0.01,0.001],
      'iterations' : [1000,2000,4000],
      'grow_policy' : ['SymmetricTree','Depthwise','Lossguide'],
      'min_child_samples': [1,2,4],
      'eval_metric': ['mape'],
      'random_state': [57]
    }
    param5 = {
      'n_estimators' = [int(x) for x in np.linspace(start = 200, stop
= 2000, num = 10)],
      'max_features' = ['auto', 'sqrt'],
      'max_depth' = [int(x) for x in np.linspace(10, 110, num = 11)],
      'min_samples_leaf' = [1, 2, 4],
      'min_samples_split' = [2, 5, 10],
      'bootstrap': [True, False],
      'criterion': ["gini", "entropy"]
    }

    rmse_error = []
    for i,model in enumerate(models):
      model.fit(X_train,Y_train)
      rmse = check_rmse(model,X_val,Y_val)
      rmse_error.append(rmse)
      print(f"Model : {models_name[i]}   rmse = {rmse}")

    model_CBR = CatBoostRegressor()
    self.model = model_CBR.randomized_search(scoring='mape',
param_grid = parameters, cv = 10, n_jobs=-1)

  def fit(self, X, Y):
    lgtrain = lgb.Dataset(train_X, label=train_y)
    lgval = lgb.Dataset(val_X, label=val_y)
    evals_result = {}
    model = lgb.train(params, lgtrain, 5000,
                      valid_sets=[lgval],
                      early_stopping_rounds=100,
                      verbose_eval=50,
                      evals_result=evals_result)

    pred_test_y = np.expm1(model.predict(test_X,
num_iteration=model.best_iteration))
    self.model.fit(X, Y)

  def predict(self, X):
    res = self.model.predict(X)
    return res
```