cam_som_test

March 30, 2022

1 SOMPY Experimentation

1.1 Description

This notebook contains code for generating hierarchical data files (`*.h5`) that can be loaded in `SOM_Visualization.ipynb`.

1.2 Workflow

1. Choose the columns you wish to train on in the som column selection cell, and run the cell in jupyter to update the variable reference.
2. Navigate to the som training cell.
3. Press the “Train” button to begin training the SOM. Depending on the size of your dataset, and how many columns you are training on, this may take longer or shorter.
4. When the process is complete, optionally give the SOM a uniquely identifying name by entering it in the text box, and then hit the “Save” button to write a hierarchical data file to be loaded later.

[1]:
```
import logging
import ipywidgets as widgets
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import sompy
from sompy.decorators import timeit
from sompy.sompy import SOMFactory
logging.getLogger().setLevel(logging.WARNING)
import itertools
import functools
import datetime as dt
# import skimage
from IPython.display import display
```

[2]:
```
# Group specific package
from tfprop_sompy.tfprop_vis import render_posmap_to_axes, kmeans_clust, ViewTFP
```

[3]:
```
from IPython.display import display
```

[4]:
```
logging.getLogger().setLevel(logging.WARNING)
```

[5]:
```
# File location and name
fin = 'general_main.csv'

# The product of the values in mapsize needs to be a square and larger than the␣
↪dataset
# Warning: the larger the dataset the longer it will take to run
mapsize = (25, 25)
n_job = 1

# Reads CSV as data frame
data_df = pd.read_csv(fin)
```

[6]:
```
data_df
```

[6]:
```
     Row  modulus  hardness  carb   crys    fluo  age tooth_label mammal  \
0      1    76.07      4.17  0.08  12.21  962.65    6          6j      h
1      2    75.07      4.45  0.09  12.53  962.60    6          6j      h
2      3    74.10      3.91  0.10  12.77  962.61    6          6j      h
3      4    68.91      3.94  0.11  12.95  962.51    6          6j      h
4      5    63.81      3.11  0.12  13.18  962.68    6          6j      h
..   …      …         …    …      …   …       …    …
133  134    64.80      3.47  0.10  13.06  960.71   10          10      o
134  135    59.40      3.32  0.11  13.35  960.65   10          10      o
135  136    56.92      2.89  0.12  13.53  960.57   10          10      o
136  137    57.43      3.09  0.12  13.66  960.50   10          10      o
137  138    59.56      3.29  0.13  13.83  960.47   10          10      o

    age_group position  depth    kc       b
0           p    outer   1.00  1.02  258.34
1           p    outer   0.80  0.93  277.70
2           p   middle   0.60  1.02  243.59
3           p   middle   0.40  1.00  244.44
4           p    inner   0.20  0.92  264.45
..        …       …     …     …       …
133       wad    outer   0.77  1.02  352.11
134       wad   middle   0.59  1.04  304.30
135       wad   middle   0.39  1.07  305.00
136       wad    inner   0.21  1.14  259.27
137       wad    inner   0.02  1.20  227.67

[138 rows x 14 columns]
```

[7]:
```
data_P = data_df[data_df.age_group == 'p']
data_Y = data_df[data_df.age_group == 'y']
data_O = data_df[data_df.age_group == 'o']
```

```
data_bb = data_df[data_df.age_group == 'bb']
data_gw = data_df[data_df.age_group == 'gw']
data_lion = data_df[data_df.age_group == 'lion']
data_sl = data_df[data_df.age_group == 'sl']
data_wad = data_df[data_df.age_group == 'wad']
```

[8]:
```
data_sl
```

[8]:
```
     Row  modulus  hardness  carb   crys    fluo  age tooth_label mammal  \
126  127    81.42      3.84  0.10  12.16  961.36   11          11      o
127  128    82.69      3.66  0.10  12.70  961.14   11          11      o
128  129    79.87      3.38  0.11  12.93  961.06   11          11      o
129  130    75.12      2.95  0.11  13.21  961.00   11          11      o
130  131    78.53      3.10  0.13  13.44  960.95   11          11      o
131  132    71.03      3.20  0.14  13.75  960.96   11          11      o

    age_group position  depth    kc       b
126        sl    outer   0.95  0.71  615.28
127        sl    outer   0.80  0.82  464.81
128        sl   middle   0.59  0.84  353.18
129        sl   middle   0.34  0.84  348.44
130        sl    inner   0.13  0.87  309.45
131        sl    inner   0.00  0.88  299.60
```

[9]:
```
#Makes data frame of index values
name_df = pd.DataFrame(data_df.index)
km_cluster = 3
```

[10]:
```
# Names of columns you want to train on
# Columns with values that are not numerical should be excluded
som_columns = [
    "modulus",
    "hardness",
    "carb",
    "crys",
    "fluo",
    "depth",
    "kc",
    "b",
]
```

[11]:
```
# All the data values for indicated som_columns
#descr = data_Y[som_columns].values
#descr = data_B[som_columns].values
descr = data_df[som_columns].values
```

```
# Shows size of data
# Sample should be (560,4)
descr.shape
```

[11]:

```
(138, 8)
```

[12]:

```
# Builds a SOM model
sm = SOMFactory.build(descr,
                      mapsize=mapsize,
                      normalization='var',
                      initialization='pca',
                      component_names=som_columns
                     )
```

[13]:

```
# Updates training block to show epoch, topographic error, and quantization␣
 ↪error
def _batchtrain_monkeypatch(self, trainlen, radiusin, radiusfin, njob=1,␣
 ↪shared_memory=False):
    from time import time
    radius = np.linspace(radiusin, radiusfin, trainlen)
    if shared_memory:
        data = self._data
        data_folder = tempfile.mkdtemp()
        data_name = os.path.join(data_folder, 'data')
        dump(data, data_name)
        data = load(data_name, mmap_mode='r')
    else:
        data = self._data
    bmu = None
    fixed_euclidean_x2 = np.einsum('ij,ij->i', data, data)
    logging.info(" radius_ini: %f , radius_final: %f, trainlen: %d\n" %
                 (radiusin, radiusfin, trainlen))

    for i in range(trainlen):
        t1 = time()
        neighborhood = self.neighborhood.calculate(
            self._distance_matrix, radius[i], self.codebook.nnodes)
        bmu = self.find_bmu(data, njb=njob)
        self.codebook.matrix = self.update_codebook_voronoi(data, bmu,
                                                             neighborhood)
        qerror = (i + 1, round(time() - t1, 3),
                  np.mean(np.sqrt(bmu[1] + fixed_euclidean_x2)))
        logging.info(
            " epoch: %d ---> elapsed time:  %f, quantization error: %f\n" %
            qerror)
        update_sm_info(*qerror)
        if np.any(np.isnan(qerror)):
```

```
            logging.info("nan quantization error, exit train\n")

    bmu[1] = np.sqrt(bmu[1] + fixed_euclidean_x2)
    self._bmu = bmu

# Variable for training cell to update and incorporate _batchtrain_monkeypatch
sompy.sompy.SOM._batchtrain = _batchtrain_monkeypatch
```

[14]:
```
%matplotlib notebook

# Creates the Training and save box
b = widgets.Button(description="Train")
out = widgets.Output(layout={'border': '1px solid black'})
hm_output = widgets.Output()

# Saves the trained som data for use in SOM_Visualization
def save_som_data(sm: sompy.sompy.SOM, name: str):
    # This will overwrite the old hd5 file, so be aware
    with pd.HDFStore(name, mode="w") as store:
        store['sm_codebook_matrix'] = pd.DataFrame(sm.codebook.matrix,␣
 ↪columns=som_columns)
        store['sm_data'] = data_df.drop("Row", axis='columns')
        store['sm_codebook_mapsize'] = pd.Series(mapsize)
        columns_group = store._handle.create_group(store._handle.root,␣
 ↪'sm_codebook_columns')
        stored_columns_array = store._handle.create_array(columns_group,␣
 ↪"property_names", list(som_columns), "Material property names")
        matfamilies_group = store._handle.create_group(store._handle.root,␣
 ↪'sm_codebook_matfamilies')
        stored_matfamilies_array = store._handle.
 ↪create_array(matfamilies_group, "material_families", list(data_df["Row"]),␣
 ↪"Material families")
    with out:
        print(f"Saved to {name}")

# Trains the data
def do_training(*args):
    out.clear_output()
    with out:
        sm.train(n_job=n_job, verbose='debug', train_rough_len=0,
                 train_finetune_len=0)

        topographic_error = sm.calculate_topographic_error()
        quantization_error = np.mean(sm._bmu[1])
        print("Topographic error = {:.5f}; Quantization error = {:.5f};"
              .format(topographic_error, quantization_error))
b.on_click(do_training)
```

```
# Produces text for the widget box
epoch_text_widget = widgets.Label(value="Epoch: 0")
topo_err_text_widget = widgets.Label(value="Topographic error: 0")
quantization_err_text_widget = widgets.Label(value="Quantization error: 0")
warning_txt = widgets.Label(value="Clicking save will overwrite the old hd5␣
↪file, so be aware")
infobox = widgets.VBox([warning_txt, epoch_text_widget, topo_err_text_widget,␣
↪quantization_err_text_widget])

# Gives file name and saves it
today = dt.date.today()
outname = widgets.Text(description="Output file",␣
↪value=f"som_codemat_{len(som_columns)}props_{today.strftime('%y-%m-%d')}.h5")
savebtn = widgets.Button(description="Save")
savebox = widgets.VBox([outname, savebtn], layout={'border': '1px solid black'})

savebtn.on_click(lambda *args: save_som_data(sm, outname.value))

# Displays the widgets below
graph_display = widgets.Output()
with graph_display:
    display(hm_output)

# Updates as data gets trained
def update_sm_info(epoch, topographic_err, quantization_err):
    epoch_text_widget.value = "Epoch: {}".format(epoch)
    topo_err_text_widget.value = "Topographic error: {}".format(topographic_err)
    quantization_err_text_widget.value = "Quantization error: {}".
↪format(quantization_err)

widgets.VBox([graph_display, widgets.Box([widgets.VBox([savebox, b, infobox]),␣
↪out])])

# When training with dummy data is done, epoch: 65
# Click Save and switch to SOM_Visualization for visualization of trained data
```

```
VBox(children=(Output(),␣
↪Box(children=(VBox(children=(VBox(children=(Text(value='som_codemat_8props_22-03-30.
↪h…
```

[ ]:

```
```

[ ]:

```
```