Graph Kernel

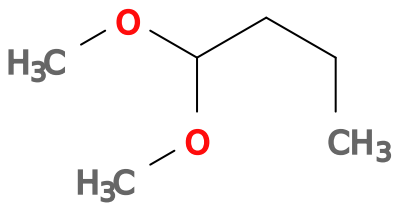
Objectives Implement a first graph kernel based on bags of patterns. The enumeration of patterns is made using an external library.

Preliminaries A paper, mine : `https://www.sciencedirect.com/science/article/abs/pii/S016786551200102X` or here `http://pagesperso.litislab.fr/~bgauzere/GauzereAl-PRL2012-Two_New_Graphs_Kernels_In_Chemoinformatics.pdf`

Also, the code provided is from this github : `https://github.com/jajupmochi/py-graph`

1 Loading dataset

On Moodle, you can download the dataset `Acyclic`. This dataset is composed of 183 chemical compounds, encoded as graphs, associated to their boiling point. Each node encodes an atom and is labeled by its chemical element. Each edge encodes an atomic bond and is labeled by its valence, encoded as an integer.



Questions

1. Load the dataset and check the data

```
#Chargement du dataset
#!pip install graphkit-learn # uncomment to install graphkit-learn
import numpy as np

%matplotlib inline
import matplotlib.pyplot as plt

import networkx as nx
from gklearn.utils.graph_files import load_dataset

# Load dataset
G,y,info= load_dataset("/path/to/the/dataset/dataset_bps.ds")

y = np.array(y)
N = len(G)
```

2 First contact with graphs

Contents of dataset are graphs as `NetworkX` format. `NetworkX` proposes some functions to visualize graphs.

Questions

1. Take two graphs and visualize them using NetworkX
2. Feel free to check NetworkX library documentation

```
x1 = 92
G1 = G[x1]
print(y[x1])
print(nx.get_node_attributes(G1, 'label'))
nx.draw_networkx(G1)
plt.show()

x2 = 90
G2 = G[x2]
print(y[x2])
print(nx.get_node_attributes(G2, 'label'))
nx.draw_networkx(G2)
plt.show()
```

3 Treelet Kernel

Treelet kernel is a graph kernel based on bags of patterns. Similarity between two graphs is defined as a sum over the similarity of patterns extracted from the graphs. This similarity is computed using a kernel between number of occurences, generally a Gaussian Kernel.

The Treelet kernel is implemented in `gklearn` library [1].

Question

1. What function computes the kernel value between two graphs ?
2. Compute the kernel value between two arbitrary graphs. Compare it to the difference of associated boiling points.

```
from gklearn.kernels import treeletKernel
def linear_kernel(x,y):
return np.dot(x,y)

def rbf_kernel(x,y, sigma=1):
return np.exp(-(np.linalg.norm(x-y)**2)/(2*sigma**2))

x1 = 100
```

[1] See `https://graphkit-learn.readthedocs.io/en/master/modules.html` for the docs.

```
G1 = G[x1]

x2 = 101
G2 = G[x2]
kernel_value,run_time = treeletKernel.treeletkernel(G1, G2,
sub_kernel = rbf_kernel, n_jobs=1, parallel=None)
print(kernel_value)
print(y[x1],y[x2])
```

The same function can be used to compute the Gram Matrix over a particular dataset.

Question

1. Compute the Gram Matrix of our dataset and display it.
2. Check if this kernel is a valid kernel

```
#Calcul matrice de Gram

K,run_time = treeletKernel.treeletkernel(G,
sub_kernel = rbf_kernel,n_jobs=1, parallel=None)
plt.imshow(K)
np.min(np.linalg.eigvalsh(K))
```

4 Learn a graph kernel machine

Since we have now a valid Gram Matrix, we are able to combine it with kernel machines such as SVM.

Question

1. What kind of problem do we have here ? What method you will use ?
2. Without tuning any hyperparameter, learn a first predictor and evaluate its performance correctly.

```
from sklearn.svm import SVR
import numpy as np
from sklearn.model_selection import ShuffleSplit
clf = SVR( kernel ="precomputed")

rs = ShuffleSplit (n_splits=5, test_size=.33, random_state=0)
errors = []

dataset = np.arange(len(G))
for train_index, val_index in rs.split(dataset):
        Ktrain = K[train_index,:][:,train_index]
```

```
            Kval = K[val_index,:][:,train_index]
            clf.fit(Ktrain , y[train_index ])
            y_pred = clf.predict ( Kval )
            errors.extend (np.abs( y_pred - y[ val_index ]))

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

3. Identify the hyperparameters and tune them using an unbiased method

5 Improve the kernel

When applying `treeletKernel` function, we hinge on a gaussian keernel with a σ value equals to 1 to compare the number of occurences of each treelet in the compared graphs.

The aim of this section is to modify this kernel between number of occurences. To do this, we first need to get the bags of patterns from both graphs.

Question

1. Extract and visualize the bag of patterns extracted from one graph.
2. Extract bags of patterns for all the graphs in the dataset.

```
from gklearn.kernels.treeletKernel import get_canonkeys

treelets = get_canonkeys(G1, node_label='atom_symbol', edge_label='bond
print(treelets)

X = []
for graph in G:
        X.append(get_canonkeys(graph,
        node_label='atom_symbol', edge_label='bond_type',
         labeled=True, is_directed=False))
```

Now, we want to compare the number of occurences of a same treelet in two different graphs.

Question

1. What are the input space for our kernel ?
2. Propose and implement kernels to compare number of occurences
3. Implement a function computing the kernel value between two bags of patterns.
4. Check that your new kernel is semi positive definite

```
def my_graph_kernel(T1,T2,sigma=1):
    #T1 and T2 are dict encoding the number of occurences of each tree
    codes = set(T1.keys()) & set(T2.keys())

    kernel = 0
    for c in codes:
        t1 = T1.get(c,0)
        t2 = T2.get(c,0)
        sub_kernel = np.exp((-(t1 - t2)**2) / (sigma**2))
        kernel = kernel + sub_kernel
    return kernel

def my_tani_kernel(T1,T2,sigma=1):
    codes = set(T1.keys())
    codes.update(set(T2.keys()))

    kernel = 0
    for c in codes:
        t1 = T1.get(c,0)
        t2 = T2.get(c,0)
        sub_kernel = min(t1,t2)
        kernel = kernel + sub_kernel
    return kernel

def compute_gram(dataset,kernel,param_kernel):
    N = len(dataset)
    my_K = np.zeros((N,N))
    for i in range(0,N):
        for j in range(i,N):
            my_K[i,j] = kernel(X[i],X[j],param_kernel)
            my_K[j,i] =  my_K[i,j]
    return my_K

my_K = compute_gram(G, my_graph_kernel, 5)
plt.imshow(my_K)
print(np.min(np.linalg.eigvalsh(my_K)))
```

The sub kernel is now a new hyperparameter. Tune the selection of this kernel together with hyperparameter related to your classifier.

```
from sklearn.svm import SVR
from sklearn.kernel_ridge import KernelRidge

import numpy as np
from sklearn.model_selection import ShuffleSplit
```

```
clf = SVR(kernel='precomputed')

K = compute_gram(G,my_tani_kernel,10)
rs = ShuffleSplit(n_splits=5, test_size=.33, random_state=0)
errors = []
for train_index, val_index in rs.split(G):
    K_train = K[train_index,:][:,train_index]
    K_val = K[val_index,:][:,train_index]

    clf.fit(K_train, y[train_index])
    y_pred = clf.predict(K_val)
    rmse_val = np.sqrt(np.mean((y_pred - y[val_index])**2))
    errors.extend(np.abs(y_pred - y[val_index]))
print(np.mean(errors))
```

Comment the obtained results.

References

[CV94] D. Cherqaoui and D. Villemin. Use of a neural network to determine the boiling point of alkanes. *J. Chem. Soc. Faraday Trans.*, 90:97–102, 1994.

[GBV11] Benoit Gaüzère, Luc Brun, and Didier Villemin. Two new graph kernels and applications to chemoinformatics. *Pattern Recognition Letters*, 2011.

[MV08] P. Mahé and J.-P. Vert. Graph kernels based on tree patterns for molecules. *Machine Learning*, 75(1):3–35, October 2008.

[RG03] J. Ramon and T. Gärtner. Expressivity versus efficiency of graph kernels. In *1st Int. Workshop on Mining Graphs, Trees and Sequences*, pages 65–74, 2003.

[VBKS10] S.V.N. Vishwanathan, K. M. Borgwardt, I. R. Kondor, and N. N. Schraudolph. Graph Kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.