Benedikt Hopf
Alireza Ketabdari

1	2	3	4	Σ

Exercise Sheet Nr. 3

(Deadline December 07, 2021)

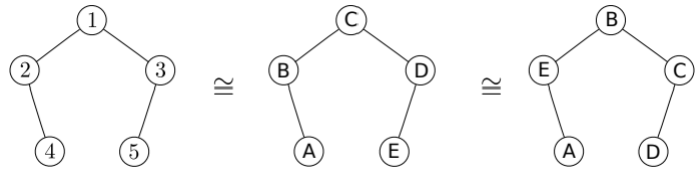
Problem 1

a) It is undeniable that we encounter structured input data in many biomedical domains, such as Protein 3D structures, Metabolic networks, and protein protein interaction networks. The reason for comparing two data structures is the same as in other domains. We use similarity in data science for measuring distance between two or more data points. Different kinds of regression, classification, and other machine learning algorithms use similarity. For example, predict the function of a protein based on the 3D crystal structure. We can use such measures of similarity in kernel methods, since the value of a kernel function is specifically supposed to be a similarity (requiring, that this similarity function is positive semi-definite of course).
One application of structured data in biomedical domains can be in MRI or CT scan photos. The structure of the photo and the elements in the photo can be interpreted as different nodes and the relation between each element with the others can be shown as edges. A more precise example can be groups of tumors (each tumor considered as a node) and the effect of each on the others (can be considered as an edge) to realize if this tumors are sarcoma or normal.

b) Complexity is one of the issues in data science and also other branches of Computer Science, since it directly increases cost of time and space (computer memory and storage). Similarity analysis of graphs in both isomorphism and sub-graph isomorphism (which is basically isomorphism) are expensive (NP-complete).
Using the current solution for this problem is to either compare approximations, that are computable in polynomial time or to compare substructures of graphs, that are small enough, to be doable, despite the NP-completeness (even an NP-complete problem is solvable, if the problem size is small enough).
Sampling Graphlet kernel: The graphlet kernel compares graphlets of some specified size k from both graphs. If k is sufficiently small, then isomorphism can be checked without approximation. In order to do this properly one would have to look at all size-k-graphlets, which can often be very many. So the approximation one then uses, is to just sample these graphlets and only check a fixed number (e.g. 1000).
Weisfeiler-Lehman: Another way to do that is the Weisfeiler-Lehman algorithm (this is the algorithm used in problem 3). This is an iterative way to compare two graphs, which can be done in polynomial time. The downside is, that the algorithm can fail (see Problem 3c)).
In Problem 4 we use both approaches, and there we get the better results using Weisfeiler-Lehman (similar to the results on slide 33 of the graph-kernels slides). This of course also depends on the number of samples you use in the Graphlet sampling kernel.

c) Isomorphism is a term used for graphs, that are basically identical, up to the naming of their nodes. Isomorphism is defined by the existence of a mapping f of the vertices of two graphs $G1$ and $G2$ in a way that for every edge like (x, y) that belongs to $G1$

can be correspond to edge $f(x), f(y)$ under function f of the other graph $G2$. Then the mapping function f is an isomorphism.
A relabeling of vertices of a graph is isomorphic to the graph itself. Consider the three isomorphic graphs illustrated below. The first two graphs illustrate a change of using letters to using numbers to label the graphs. The second pair of graphs are also isomorphic as only the labels were changed. We can match vertices in the second graph with those in the third graph to satisfy the isomorphism requirements.



d) Given a graph $G(n, m)$ of n nodes and m edges, a k-graphlet is a sub-graph g on any set of k nodes from G, where k.
Graphlet kernels are introduced to measure the similarity of the graphs when we are receiving structured input data in our application. It can be more fast and beneficial specially when we use sampling approach rather than Random walk for example. Moreover, there were other Kernels in the course that are not specialized for structured input data like Gaussian, String, Weighted Degree Kernels. Meanwhile, the most important similarity among all Kernels is to measure the similarity between two or more data points.

e) The general approach in Weisfeiler-Lehman kernels is the same and they are trying to measure similarity of the graphs by considering the labels on the vertices. For example, Weisfeiler-Lehman kernel and Weisfeiler-Lehman sub-tree kernel.
The Weisfeiler-Lehman kernel is based on a so called base kernel k. The value of the Weisfeiler-Lehman kernel is then given by the sum of the base kernels values on every iteration of the Weisfeiler-Lehman algorithm (of height h).

$$k_{WL}^{(h)} = k(G_0, G_0') + ... + k(G_h, G_h')$$

The Weisfeiler-Lehman subtree kernel is also based on the Weisfeiler Lehman algorithm. There on creates a vector, whith the counts of every label in every iteration of the WL-algorithm. This is then computed for both graphs and the final kernel value is given by the inner product of these two vectors.
Which one of these kernels is better, does of course depend on the base kernel used in the WD-kerel. On slide 33 of the slides 6 there is a table, comparing the WD subtree kernel to the WD kernel with shortest-path base kernel. The performance is relatively similar, but the WD stortest-path kernel seems to be a little bit better almost every time.

f) Integrative analysis describes a way of tackling a problem, where one uses several datasets/datatypes at once. This makes a lot of sense for cancer analyses, because there are often several measurements availabe (Clinical data, DNA methylation, ...), so it makes sense, to use them in a combined way.
One usecase I could imagine is in early recognition of type 1 diabetes. There one might have similar measurements like in cancer concerning DNA (e.g. methylation) as well as some information on the constitution of the patients pankreas, which is responsible for producing insulin. So here it would make sense to do integrative

analysis on these different types of data.
I am not a medical computer scientist, however, so here is another non-medical example. In astrophysics, one might want to classify stars in terms of if there could be live on one of its planets. For that there are also several types of data available (wavelength-spectrum of the star, age of the star, proximity to other objects like quasars, ...). So here integrative analysis might help as well.

g) The assumption in gaussian latent variable models is, that there exist a set of latent variables Z, which are unique to each patient (or datapoint in a more general sense) and actually determine all measurements. The individual measurements are then created by multiplying these latent variables with some coefficients and adding gaussian noise. The coefficients and the amount/shape of the noise is specific to each measurement and therefore explains, that measurements (DNA-methylation, clinical data, ...) result in different values.
iCluster is based on a gaussian latent variable model. It used the different measurements to try to recover the original parameters Z of each patient, and then cluster based on that Z. This is computationally very expensive, however, so feature selection is necessary, but can also strongly influence the results.

Problem 2

AlphaFold 2 Video Summary

Good news in engineering and science in 2020:

- Space X: Launching new era of space exploration.
- DeepMind: Second version of AphaFold solved 50 years old grand challenge of protein folding (in prediction performance last iteration scored 58 but now it scored 87 on the hardest class of proteins) which is considered as the one of the most considerable achievements in structural biology and artificial intelligence.

Competition among subjective breakthroughs (criteria: how much the real-world direct impact the achievement has)
These are listed below and also their area of technology are mentioned.

- Alex Net: Simplified data set, on the other hand, deep neural network on the big amount of data in supervised way)
- Alpha Zero: Reinforcement learning self-play
- GPT-3: Transform in NLP space
- Tesla autopilot: Deployment of robots in the field used by real human and massive machine learning in safety critical systems
- Waymo systems: Self-driving car
- Smart speakers in home: natural language processing
- Boston Dynamics: In robots and hard ware space

Protein and Protein folding
Proteins, chain of amino acids (basic building box of life), are workhorses of living organisms of cells they have structural and functional properties.
Folding process: Amino acid sequence to 3D structure. main properties are listed below:

- Property of uniqueness: A lot of ways for the proteins to fold based on the sequence of the amino acids (10^{143})
- 3D structure determines the function of the protein
- Some disease can be result of misfolding of proteins
- There are 200 millions of protein and just 170000 3D structures are determined (So these are our training data for learning data approaches for protein folding problems and many many structures are not determined yet)
- One of the most accurate methods among experimental methods in determining 3D structures is x-ray crystallography (cost 120 000 Dollar per protein and takes 1 year to determine 3D structure)
- The most important thing that the AlphaFold 2 does: for the large class of protein determines the 3D structure with high accuracy

How AlphaFold 1 is working: Two steps in process. First include machine learning and the second not.
First step is convolutional Neural Network, input is amino acid sequence + different features and the output is distance matrix whose rows and columns are amino acid residues. The result is the distance between two amino acids in final geometric 3d structure of the protein. The second step is then to try to fold the protein into a shape, that fits with the previously calculated distance matrix (not learning based).
How AlphaFold 2 is working: We are not sure since there is only blog post about that but the thing that is obvious is convolutional neural network is out and transformers are in (similar to NLP and deep learning spaces, attention mechanism). Other features and assumptions listed below:

- The rest is in speculation space.
- Multiple sequence alignment is part of learning process (in AlphaFold 1 it was part of feature engineering)
- There is a constant pass of learned information between sequence residue presentation and amino acid residue distance
- The iterative part is unknown
- Evolutionary related sequences are part of learning process
- Spatial graph instead of distance matrix in AlphaFold 1

Why is this breakthrough important? Since Protein structure causes protein function and then:

- Figuring out the structure of the millions of proteins, allow us to learn unknown functions of genes encoded in DNA

- Understand the cause of many diseases result of misfolded proteins
- Designing protein in different function from other existing ones in drugs and curing diseases result of unfolded protein
- Agriculture
- Tissue regeneration
- Supplements in health and anti-aging
- Bio-material
- Long term impact: learning very complicated problems in life sciences, which causes:
  - Predict multi protein interactions
  - Protein complex formation
  - Physics and bio physics → physic base simulations of biological systems

What was the big news?

An artificial intelligence (AI) network developed by Google AI offshoot DeepMind has made a gargantuan leap in solving one of biology's grandest challenges — determining a protein's 3D shape from its amino-acid sequence.
DeepMind's program, called AlphaFold2, Second version of AlphaFold solved 50 years old grand challenge of protein folding. Which is considered as the one of the most considerable achievements in structural biology and artificial intelligence. By this achievement the functionality of protein structures can be predictable and then it can be start of many progresses in diverse area like curing diseases, health, agriculture, Bio-material, and so on.

Why is this topic of such huge interest and why can't it be solved using a deterministic approach?

According to the video and the article, this achievement could stand over the other achievements in competitions and could solve many problems like Lupas's team's problem after many years or predictions of the structures of a handful of SARS-CoV-2 proteins and everything about proteins and their functions.
Since we have (10^{143}) 3D-structures to fold based on the sequence of the amino acids and we cannot try out the functionality of every single version in deterministic way and we need approaches like AlphaFold 2 which is based on prediction of protein 3D structures.

What is (probably) the underlying machine learning approach?

As stated above, one thing, that is likely to have helped is the use of transformers instead of convolutions. Also there seems to be more learning and less feature engineering (multiple squence alignment), which also generally helps with performance.

How can this achievement probably improve biomedical research?

Proteins are the building blocks of life, responsible for most of what happens inside cells. How a protein works and what it does is determined by its 3D shape — 'structure is function' is an axiom of molecular biology. Predicting 3D structure of protein means learning the functionality of that protein. Thus, understanding behavior of them can be

trigger of new areas in many fields of science including biology. For example, quicker and more advanced drug discovery.

Do you think the "protein folding problem" (see reference article) is solved given this new approach?

According to words of scientists in the article the problem has been solved.
"This is a problem that I was beginning to think would not get solved in my lifetime," says Janet Thornton
"This is going to empower a new generation of molecular biologists to ask more advanced questions," says Lupas
"The model from group 427 gave us our structure in half an hour, after we had spent a decade trying everything," Lupas says.
"It's a breakthrough of the first order, certainly one of the most significant scientific results of my lifetime.", Mohammed AlQuraishi
Of course it is not perfect, yet, so it cannot yet replace experimental approaches like x-ray crystallography, but according to the scientists, performance is impressive enough, to get close do doing that.

Task 3 We have the following two graphs:

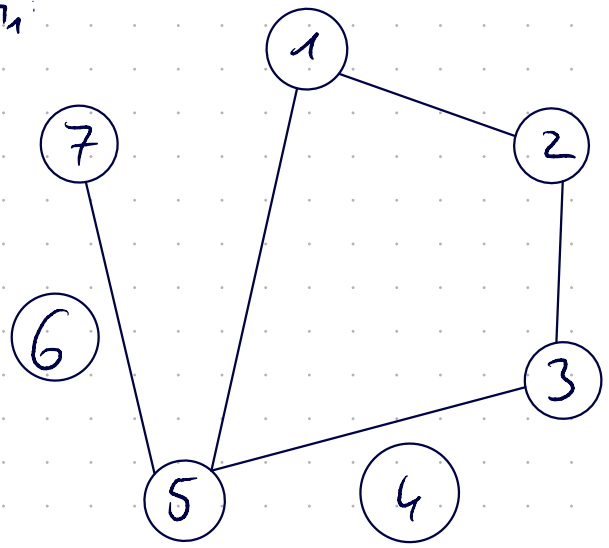
$$G_1 = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix}$$

and

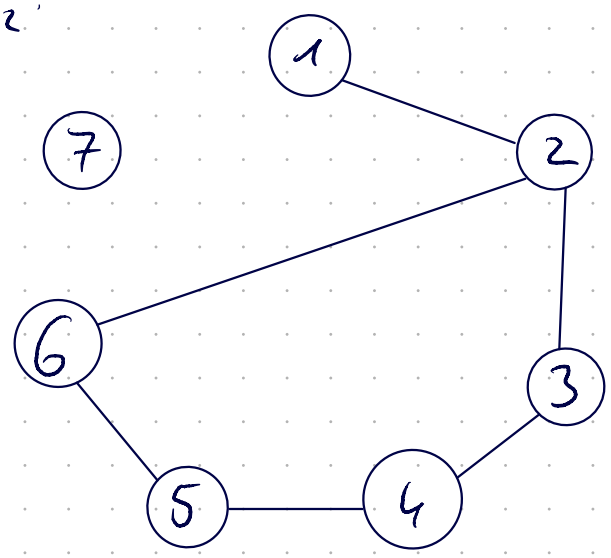
$$G_2 = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix}$$

Using column numbers as labels:

G_1:

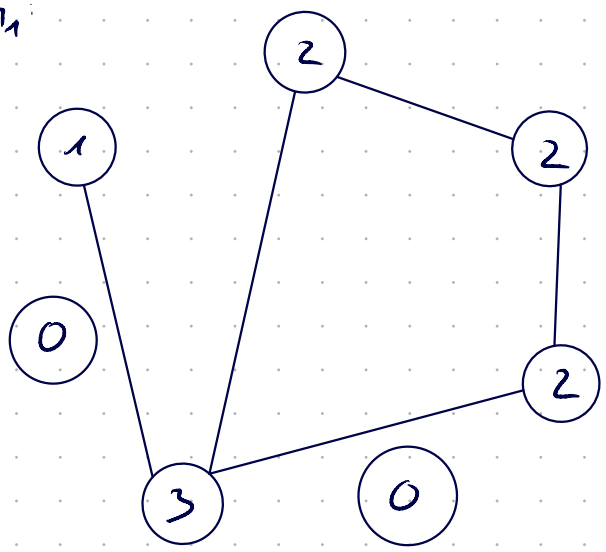


G_2:

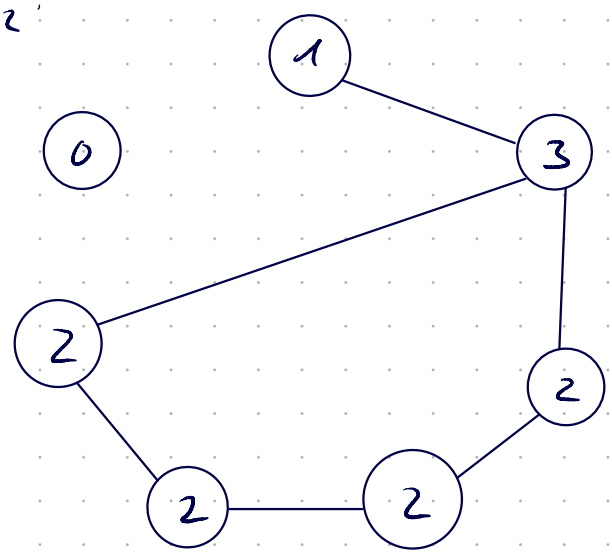


a) Draw with degrees as labels:

G_1:

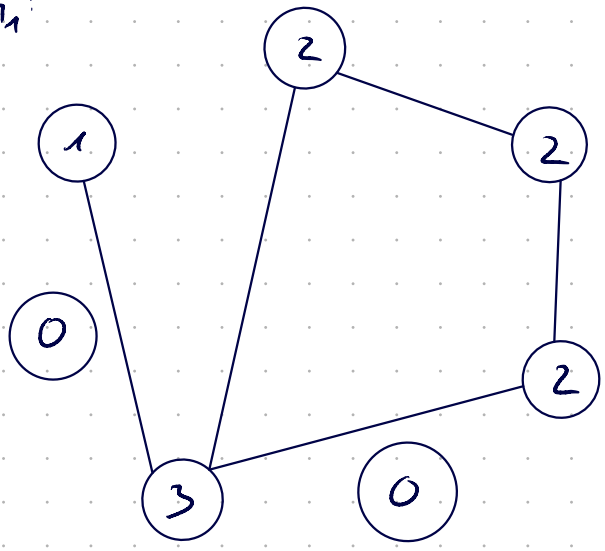


G_2:

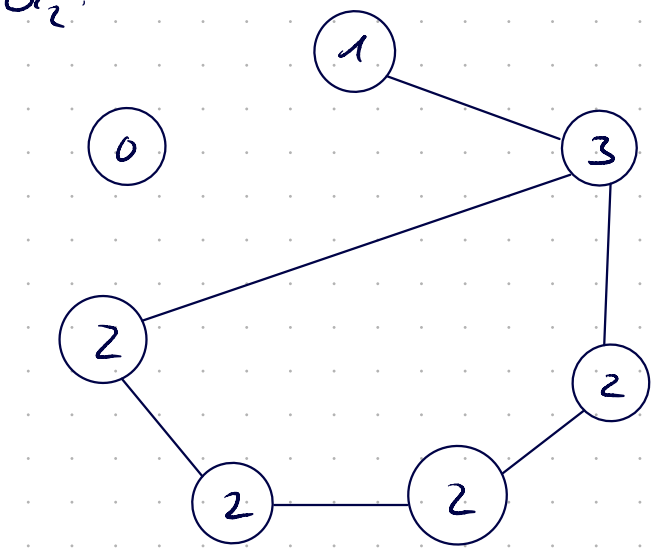


b) Weisfeiler - Lehmann

G_1:

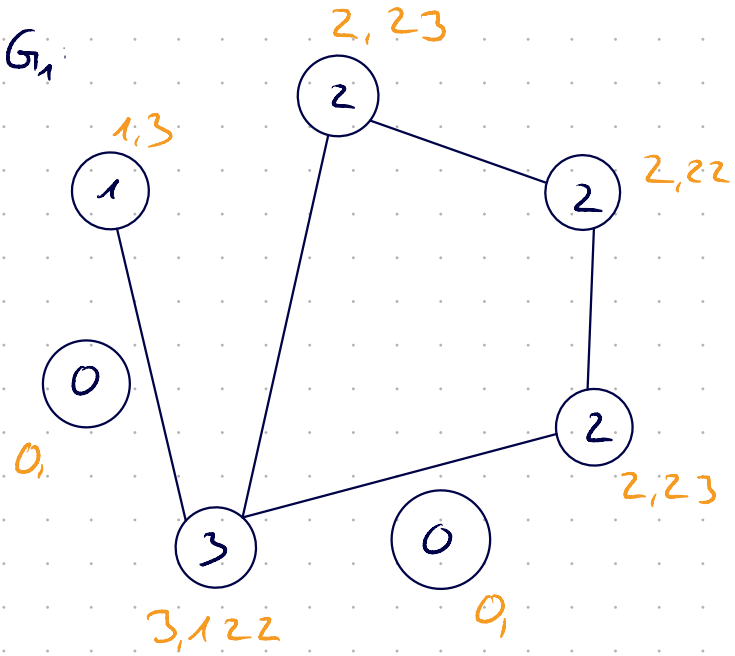


G_2:

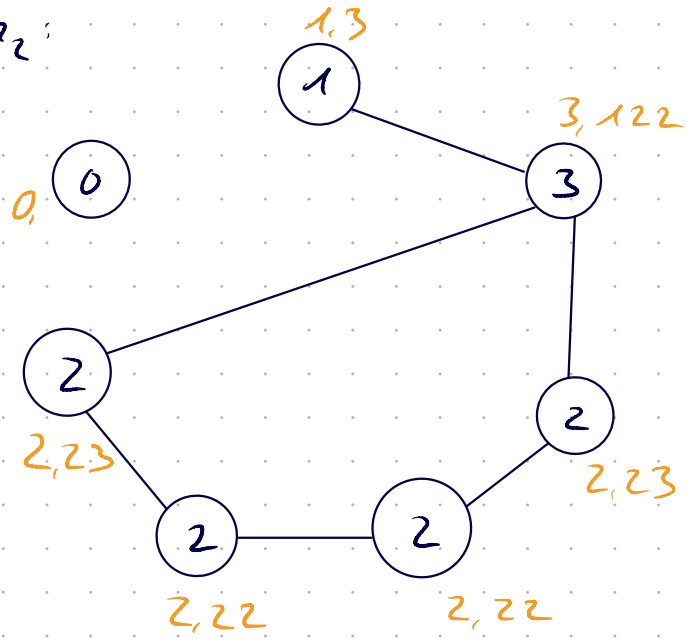


first iteration:

G_1:



G_2:

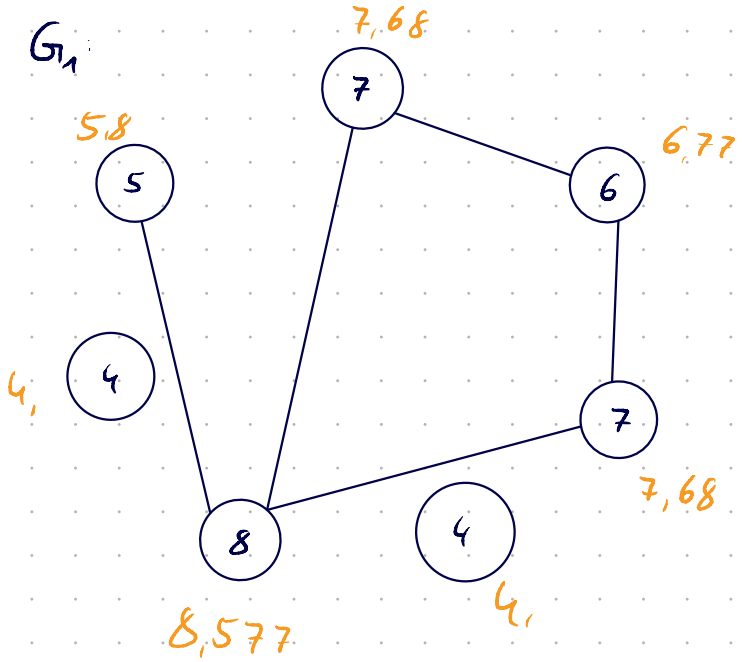


rename labels:

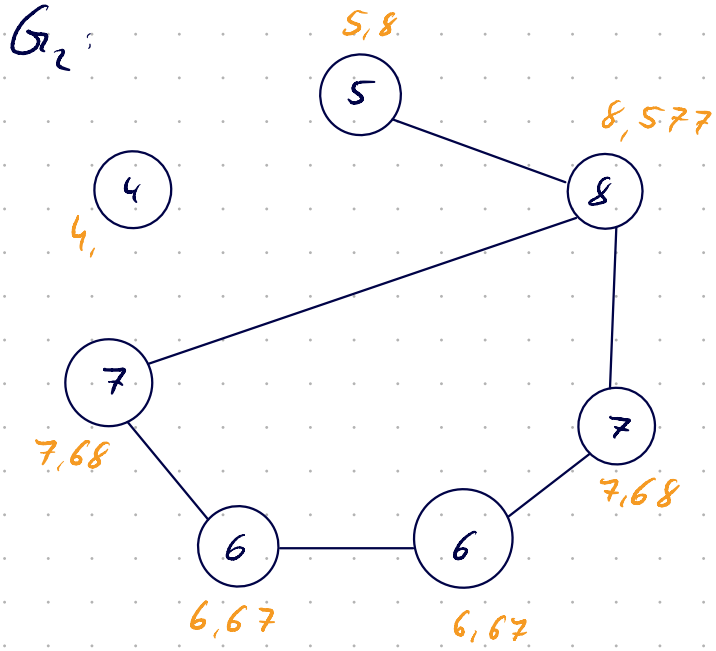
old	0,	1, 3	2, 22	2, 23	3, 122
new	4	5	6	7	8

second iteration

G_1:



G_2:

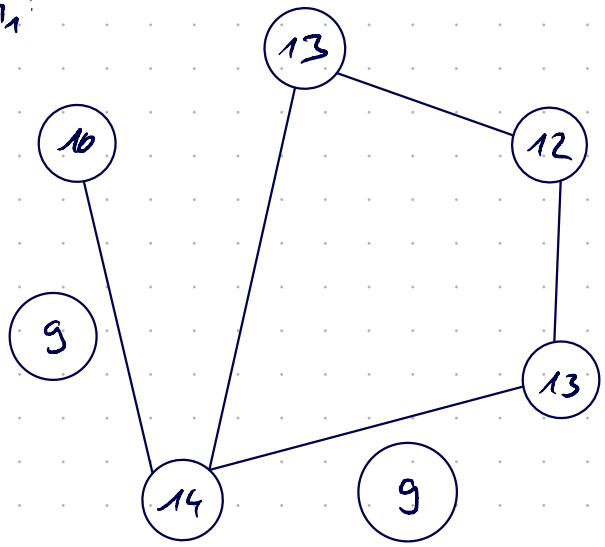


rename labels:

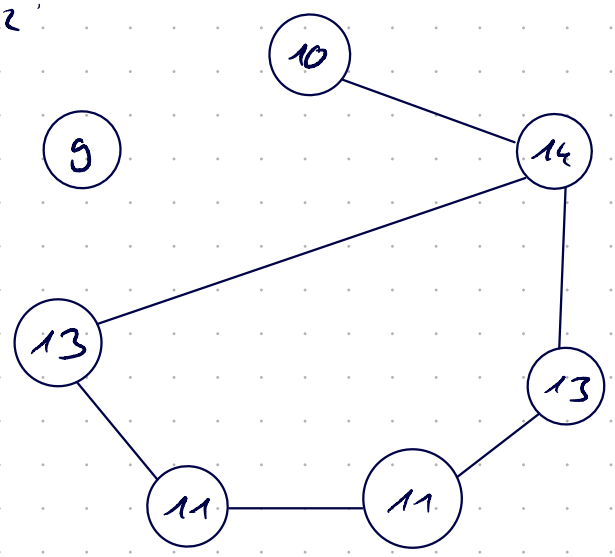
old	4,	5,8	6,67	6,77	7,68	8,577
new	9	10	11	12	13	14

Redraw graph

G_1:



G_2:



Now we can stop since

$$L_1 := \{ l_2(v) \mid v \in V_1 \} \neq \{ l_2(v) \mid v \in V_2 \} =: L_2$$

where V_1 is the set of edges of G_1 and V_2 the set of edges of G_2.

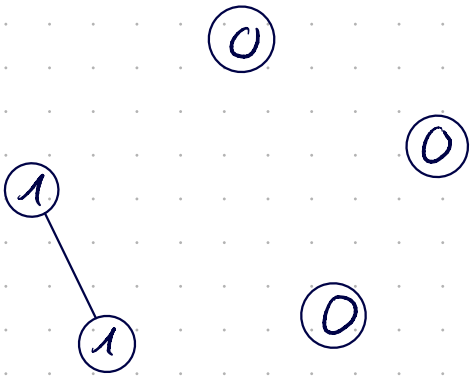
This is because

$$12 \in L_1 \qquad 12 \in L_2$$
$$11 \notin L_1 \qquad 11 \in L_2$$

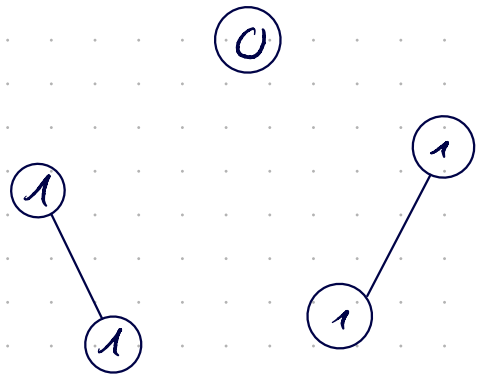
Therefore the algorithm terminates and G_1 and G_2 are <u>not</u> isomorphic.

c)

G_1



G_2



These graphs are obviously not isomorphic. However the algorithm will not detect that. This is because there are only two types of nodes: Those with degree 0 and those with 1. And those with degree 0 are (obviously) unconnected, and those with 1 have one connection to another node with

degree 1. So the 0-nodes will all be renamed the same and the 1-nodes will all be renamed the same. Since both graphs have both types of nodes, the labelsets will always be the same.
So the algorithm will stay in an infinite loop despite the fact, that the graphs are not isomorphic.

Problem4

December 5, 2021

1 Problem 4

[1]:
```
import pandas as pd
import grakel
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.model_selection import KFold
from scipy.special import binom
```

1.1 Data Loading

First we will read in the files and convert them to numpy, in order to then build the graphs. The arrays that only have one column anyways, will be flattened, to avoid lists of length 1.

[2]:
```
edges = pd.read_csv("MUTAG/MUTAG_A.txt", header=None).to_numpy()
graph_indicator = pd.read_csv("MUTAG/MUTAG_graph_indicator.txt", header=None).
 ↪to_numpy().flatten()
labels = pd.read_csv("MUTAG/MUTAG_graph_labels.txt", header=None).to_numpy().
 ↪flatten()
node_labels = pd.read_csv("MUTAG/MUTAG_node_labels.txt", header=None).
 ↪to_numpy().flatten()
edges[:5]
```

[2]:
```
array([[2, 1],
       [1, 2],
       [3, 2],
       [2, 3],
       [4, 3]])
```

Next we will calculate, what nodes belong to what graph.

[3]:
```
node_lists = []
for graph in np.unique(graph_indicator):
    node_lists.append(np.asarray(graph_indicator == graph).nonzero()[0] + 1)
print(node_lists[0])
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17]
```

Next we will find the edges of each graph. This is not the most efficient way of doing it, since the list is sorted, but it is sufficiently fast.

[4]:
```
edges_lists = []
for node_list in node_lists:
    es = []
    for v, w in edges:
        if (v in node_list) and (w in node_list):
            es.append((v,w))
    edges_lists.append(es)
print(edges_lists[0])
```

```
[(2, 1), (1, 2), (3, 2), (2, 3), (4, 3), (3, 4), (5, 4), (4, 5), (6, 5), (5, 6),
(6, 1), (1, 6), (7, 5), (5, 7), (8, 7), (7, 8), (9, 8), (8, 9), (10, 9), (9,
10), (10, 4), (4, 10), (11, 10), (10, 11), (12, 11), (11, 12), (13, 12), (12,
13), (14, 13), (13, 14), (14, 9), (9, 14), (15, 13), (13, 15), (16, 15), (15,
16), (17, 15), (15, 17)]
```

Now we can create the `grakel.Graphs`.

[5]:
```
graphs = []
for es, ns in zip(edges_lists, node_lists):
    n_labels = {i: node_labels[i-1] for i in ns}
    g = grakel.Graph(es, node_labels=n_labels)
    graphs.append(g)
print(graphs[0])
print(len(graphs))
```

```
<grakel.graph.Graph object at 0x7f054c0d41d0>
188
```

Sanity check: do we have as many labels as graphs?

[6]:
```
print(len(graphs) == len(labels))
```

```
True
```

1.2 Classification and Cross validation for Graphlet Kernel

We can here use the crossvalidation algorithm from the last exercise (after slightly modifying it to take all Cs as an argument to avoid recomputing the kernel matrix unnecessarily). We will parametrize the function, such that it takes a function as one argument, which generates the kernel. This way we can easily reuse this function in the next task.

[7]:
```
def do_cv(Cs, X, Y, kernel):
    kf = KFold(n_splits=10, shuffle=True, random_state=42) # for reproducability
    X = np.asarray(X)
    Y = np.asarray(Y)

    # initialize output dict
```

```
    scores = {C: {
                "accuracy": [],
                "results": [],
                "truths": [],
                "classifiers": []
            } for C in Cs}

    # Crossvalidation-loop
    for train_indices, test_indices in kf.split(X):
        # generate training and test data
        Xtrain = X[train_indices]
        Ytrain = Y[train_indices]
        Xtest = X[test_indices]
        Ytest = Y[test_indices]

        # define kernel
        graphlet_kernel = kernel()
        # calculate kernels
        Ktrain = graphlet_kernel.fit_transform(Xtrain)
        Ktest = graphlet_kernel.transform(Xtest)

        for C in Cs:
            # initialize SV;
            e = svm.SVC(
                kernel="precomputed",
                C=C
            )

            # fit and evaluate
            e.fit(Ktrain, Ytrain)
            acc = e.score(Ktest, Ytest)
            scores[C]["accuracy"].append(acc)

            # save some more metadata in case we need it later
            results = e.decision_function(Ktest)
            scores[C]["results"].append(results)
            scores[C]["truths"].append(Ytest)

            scores[C]["classifiers"].append(e)

    for C in Cs:
        # calculate mean accuracy, save and print it
        scores[C]["accuracy"] = np.array(scores[C]["accuracy"]).mean()
        acc = scores[C]["accuracy"]
        print(f"Model (C={C}): accuracy = {acc:2.2%}")

    return scores
```

Define Cs

[8]:
```
Cs_graphlet = np.logspace(-4, 1, 6)
print(Cs_graphlet)
```

```
[1.e-04 1.e-03 1.e-02 1.e-01 1.e+00 1.e+01]
```

Define the function that will generate the graphlet kernel as described in the task

[9]:
```
def generate_grahplet_kernel():
    return grakel.GraphletSampling(
                random_state=42, # for reproducability
                k=3,
                sampling={
                    "n_samples": 1000
                }
            )
```

[10]:
```
scores_graphlet = do_cv(Cs_graphlet, graphs, labels, generate_grahplet_kernel)
```

```
Model (C=0.0001): accuracy = 81.93%
Model (C=0.001): accuracy = 81.93%
Model (C=0.01): accuracy = 82.49%
Model (C=0.1): accuracy = 82.49%
Model (C=1.0): accuracy = 82.49%
Model (C=10.0): accuracy = 82.49%
```

Find best accuracy:

[11]:
```
def print_best_accuracy(scores, Cs):
    accuracies = []
    for C in Cs:
        accuracies.append((C, scores[C]["accuracy"]))
    best_C, best_acc = max(accuracies, key=lambda x: x[1])
    print(f"The best accuracy is {best_acc:2.2%} which was achieved by␣
↪C={best_C}")
```

[12]:
```
print_best_accuracy(scores_graphlet, Cs_graphlet)
```

```
The best accuracy is 82.49% which was achieved by C=0.01
```

1.2.1 How many samples would we need?

For this we can use Theorem 6 from the script (graph kernels, page 20). For this we need to know, how many size 3 graphlets are there are in the largest graph (since the samples must be enough for all graphs, including the biggest graph). We can easily find the number of nodes in the largest graph:

[13]:
```
n = len(max(node_lists, key=len))
print(f"Number of nodes in the largest graph: n = {n}")
```

```
Number of nodes in the largest graph: n = 28
```

By page 18 of the same scipt, we know, that a graph with n nodes has $\begin{pmatrix} n \\ k \end{pmatrix}$ graphlets of size k. Since we are using $k = 3$ we get: $a = \begin{pmatrix} n \\ k \end{pmatrix} = \begin{pmatrix} 28 \\ 3 \end{pmatrix}$. What that is as a number, we can also calculate:

[14]:
```
k = 3
a = binom(n, k)
print(a)
```

```
3276.0
```

So we have $a = 3276$ graphlets in the biggest graph.

Now we can use Theorem 6:

We have $\mathcal{A} = \{1, ..., a\}$ as the set of graphlets in the largest graph. We then need:

$$m = \left\lceil \frac{2(\log 2 \cdot a + \log\left(\frac{1}{\delta}\right)}{\epsilon^2} \right\rceil$$

samples, in order to restrict the chance of more than ϵ deviation to a probability of δ.

Using the values from the task we get: $\epsilon = 0.05$ and $\delta = 1 - 0.9 = 0.1$ which leads us to:

[15]:
```
epsilon = 0.05
delta = 0.1

m = np.ceil(
    (2 * (np.log(2) * a + np.log(1/delta))) /
    (epsilon**2)
)
print(f"m = {m}")
```

```
m = 1818443.0
```

So we would need to sample 1818443 times in order to have a distribution of less, than 0.05 with probability 0.9. This number might be a stong overestimation since a lot of the graphlets might be isomorhic (see the paper cited in the scipt *Nino Shervashidze et al., Efficient graphlet kernels for large graph comparison* Chapter 3.2).

But as seen from the paper it is enough, to use the number of *different* graphlets of size k. For $k = 3$ there are only 4 possible graphlets:

```
- everything connected,
- everything unconnected,
- one edge (where that is does not matter),
- two edges (i.e. one edge missing, again, where that is does not matter).
```

So now we have $a = 4$ and can calculate a much tighter bound:

[16]:
```
a = 4
m = np.ceil(
    (2 * (np.log(2) * a + np.log(1/delta))) /
    (epsilon**2)
)
print(f"m = {m}")
```

```
m = 4061.0
```

Now we have a bound of $m = 4064$ samples which is much more reasonable.

1.3 Classification and Cross validation for Weisfeiler-Lehman subtree kernel

Given the way we have parametrized the funcion, we can now very simply do that second task:

[17]:
```
Cs_wlsk = np.logspace(-4, 2, 7)
print(Cs_wlsk)
```

```
[1.e-04 1.e-03 1.e-02 1.e-01 1.e+00 1.e+01 1.e+02]
```

Define the new kernel generating function

[18]:
```
# https://ysig.github.io/GraKeL/0.1a8/auto_examples/weisfeiler_lehman_subtree.
↪html?highlight=subtree
def generate_weisfeiler_lehman_subtree_kernel():
    return grakel.WeisfeilerLehman(
            n_iter=4,
            base_graph_kernel=grakel.kernels.VertexHistogram,
            normalize=True
        )
```

[19]:
```
scores_wlsk = do_cv(Cs_wlsk, graphs, labels,␣
↪generate_weisfeiler_lehman_subtree_kernel)
```

```
Model (C=0.0001): accuracy = 66.49%
Model (C=0.001): accuracy = 66.49%
Model (C=0.01): accuracy = 66.49%
Model (C=0.1): accuracy = 66.49%
Model (C=1.0): accuracy = 77.72%
Model (C=10.0): accuracy = 86.75%
Model (C=100.0): accuracy = 83.01%
```

[20]:
```
print_best_accuracy(scores_wlsk, scores_wlsk)
```

```
The best accuracy is 86.75% which was achieved by C=10.0
```