Universität Bonn

MA-INF 4316: Graph Representation Learning

Examiner: Dr. Pascal Welke

Online Exam

The **5 problems** in this exam are graded out of **89 points**. Passing the exam requires to achieve at least **44 points**.

Note: You must hand in your solution within **12 hours**. Solutions received later than **22:00h CET, 2022-02-21** cannot be accepted. In other words: **Handing in solutions later than 22:00h CET, 2022-02-21 implies failing the exam**.

Note: When handing in your solutions, carefully follow the instructions on the next page.

Note: There are no restrictions on the use of auxiliary tools or resources during the examination. For example, you are allowed to use all materials from the lecture and the exercises, and you may access the internet. However, **during the examination period, you must not communicate with anybody except the examiner about the problems in this exam or their solutions**. This includes your fellow students, as well as asking questions on e.g. `StackOverflow`, `Quora`, `discord`, `Signal`, `WhatsApp`, `Telegram` groups, etc.

Note: At the begin of the exam during **10.00h–12.00h CET, 2022-02-21**, you may ask questions, regarding technical details of the exam or exercises in a zoom meeting. The meeting ID is 966 5236 0278 and the passcode is 218838 .

HowTo: Submit Your Solutions

1. Use the Jupyter notebook `exam.ipynb` to hand in your solution. You have received it with this task description. You are expected to know how to work with Jupyter notebooks.

2. Fill the first cell of the notebook with your personal information.

3. Enter your solutions to the individual tasks separately in one or more cells below the respective headings. Please ensure that code, as well as results (plots, printed results, etc.) are shown in the notebook. Please also make sure that the notebook can be rerun to compute the same results (possibly up to randomization). We may rerun your code to check your solutions.

4. Once you are done preparing the notebook, **export** it to `html` format. (Typically, this option is provided via File→Export Notebook as or File→Download as)

5. Check if the export worked by inspecting the resulting `html` file.

6. Create a file `affirmation.txt` containing the following text and fill it with your personal information.

   > Last name:
   > First name:
   > Student ID number:
   > Degree program:
   > I hereby swear that I completed the examination MA-INF 4316: Graph Representation Learning completely on my own and without any impermissible external assistance or through the use of non-permitted aids. I am aware that cheating during the execution of an examination (as detailed in § 63 Para 5 of the Higher Education Act NRW) is a violation of the legal regulations for examinations and an administrative offense. The submission of false affirmation in lieu of an oath is a criminal offense.

7. Upload the `html` file, the `ipynb` notebook, and the `txt` affirmation file within the ecampus module, as seen in the exercises.

8. **Only in case uploading with ecampus does not work,** you may attach a zip file with your solution to an email. In this case, please rename it to `YOUR_MATRICULATION_NO.[zip|tar|tar.gz]`. **Using your University email account** send this email with **subject** "Exam MA-INF 4316: Graph Representation Learning" to welke@cs.uni-bonn.de Your email must be received **no later than 22:00h CET, 2022-02-21**.

Task 1 Node Property Prediction [23 Points]

For this task, we will be using the twitch graph provided by SNAP. Download and unzip the `twitch-graph.zip` file that you find next to the exam on eCampus. The file can be loaded into memory using the following `igraph` function:

```
g = igraph.Graph.Read_Pickle('twitch-graph.pickle')
```

Let's call this *undirected* graph G for the remainder of this task.
Nodes represent twitch users, and edges indicate mutual friendships between them. Your task is to predict if the users swear too much during their streams. The vertices in G have the *binary* attribute 'label' which will be our target attribute. Furthermore they have the attribute 'lang' which can take values from the list

```
languages = ['E', 'ENGB', 'ES', 'FR', 'PTBR', 'RU']
```

The 'lang' attribute tells us in which language the user streams their content.

Task 1.1 Load the Graph [2 Points]

Load the graph into memory and print its number of vertices and its number of edges. Print the average degree of the nodes in G.

Task 1.2 Compute the Sane Density [3 Points]

Let $d(v)$ be the density of $v \in V(G)$, then the *sane density* is defined as

$$d'(v) = \begin{cases} d(v) & \text{if } \delta(v) > 1 \\ 0 & \text{otherwise} \end{cases}$$

Implement a function that computes the sane density for all vertices in a given graph. Compute and print the sane densities for the vertices with ids

```
[ 42, 123, 11024, 11585, 12280, 34117]
```

Task 1.3 Compute Vertex Features [3 Points]

Compute, for each vertex, the feature vector $r(v) := (\delta(v), p(v), deg(v), c'(v))$ where

- $\delta(v)$ is the degree,
- $p(v)$ is the page rank (with damping factor 0.85),

- $deg(v)$ is the degeneracy, and
- $c'(v)$ is the *sane* density.

Output the full feature vectors for the vertices with ids

```
[ 42, 123, 11024, 11585, 12280, 34117]
```

Task 1.4 Train/Validation/Test Split [4 Points]

Now create the three induced subgraphs

- $G_{train} = G[V_{train}]$ for
  $V_{train} = \{v \in V(G) \mid v[\text{`}lang'] \in \{\text{`}DE', \text{`}ENGB', \text{`}FR', \text{`}RU'\}\}$
- $G_{validate} = G[V_{validate}]$ for $V_{validate} = \{v \in V(G) \mid v[\text{`}lang'] = \text{`}ES'\}$
- $G_{test} = G[V_{test}]$ for $V_{test} = \{v \in V(G) \mid v[\text{`}lang'] = \text{`}PTBR'\}$

Report the number of vertices and edges for all three graphs individually.

Task 1.5 Grid Search [5 Points]

Now train a support vector machine on V_{train} using the feature vectors defined in Task 1.3. To this end, run a grid search over the SVM parameter $C \in \{0.001, 0.01, 0.1\}$ and $kernel \in \{rbf, linear\}$.[1] Evaluate the performance of your trained classifier on the vertices of $V_{validate}$ using accuracy scoring.
Print the accuracy of your classifiers on the validation set for each parameter combination $(C, kernel)$.

Task 1.6 Final Training and Prediction [3 Points]

Train a support vector machine on V_{train} and $V_{validate}$ using the parameters that resulted in the highest validation accuracy in Task 1.5. Print the accuracy of this classifier on the vertices of V_{test}.

Task 1.7 Interpretation [3 Points]

Compute and print the accuracy of a predictor that returns the majority class on V_{train} and $V_{validate}$. Is your result from Task 1.6 satisfactory? Give a short answer.

[1] Use the default parameter γ for the rbf kernel.

Task 2 Message Passing [13 Points]

We will continue working on the twitch graph G from Task 1.

Task 2.1 Implement a Message Passing Algorithm [3 Points]

Implement a message passing algorithm with the aggregate and update functions

- $\text{agg}\left(\{\{N\left(v\right)\}\}\right) := \sum_{u \in N(v)} r_k(u)$
- $\text{upd}\left(v\right) := r_0(v) + \text{agg}\left(\{\{N\left(v\right)\}\}\right)$

for $r_k(v) \in \mathbb{R}$ for all $v \in V(G)$ and $k \geq 0$.

Task 2.2 How to set $r_0(v)$? [3 Points]

How do you need to initialize $r_0(v)$ such that the following lemma holds:

Lemma 1. *For all $k \geq 0$ and all $v \in V(G)$ the message passing algorithm from Task 2.1 computes the number of vertices in the Weisfeiler Lehman unfolding tree centered at v of depth k.*

Give a short explanation of your reasoning.

Task 2.3 Compute Vertex Representations [3 Points]

Compute and print the representations of the vertices with ids

```
[ 42, 123, 11024, 11585, 12280, 34117]
```

in G for $k \in \{0, 1, 3\}$.

Task 2.4 r_k vs. r_k^{WL} [4 Points]

Assume that every node in G has label ϵ. Consequently, the level-0 Weisfeiler-Lehman labels of all vertices are equal, i.e., $r_0^{WL}(v) = \epsilon$ for all $v \in V(G)$. Prove or disprove the following two claims by providing a brief sketch of reasoning or a counter example.

For nodes $u, v \in V(G)$ and $k \geq 0$, it holds that

1. $r_k(u) = r_k(v) \;\Rightarrow\; r_k^{WL}(u) = r_k^{WL}(v)$
2. $r_k^{WL}(u) = r_k^{WL}(v) \;\Rightarrow\; r_k(u) = r_k(v)$

Task 3 (Message Passing) Graph Neural Networks [22 Points]

For this task, you will be working with the DHFR dataset from the TU-Dortmund collection. You can find some possibly useful code snippets in your jupyter notebook. However, you are not required to use them if you find another setup more to your liking. It is, however, beneficial to use the `pytorch_geometric` package.

Task 3.1 Layers [6 Points]

Create four neural networks with a different number of GIN (Graph Isomorphism Network) layers according to the following specifications:

- vary the number of GIN layers $k \in \{2, 3, 4, 5\}$
- let the vertex representations $r_k(v) \in \mathbb{R}^{2^{7-k}}$ for $k \in \{1, 2, 3, 4, 5\}$
- For each GIN layer, choose a fully connected MLP with one hidden layer of size 32 with ReLU activation function.
- aggregate the vertex representations $r_k(v)$ for all $v \in V(G)$ using a maximum aggregator (`global_max_pool` in `pytorch_geometric`)
- add a fully connected layer that reduces the dimension of the graph representation to the number of classes in DHFR
- Finally, apply `log_softmax` to obtain predictions for classes

Task 3.2 Training [4 Points]

Evaluate your four neural network architectures after 100 epochs with an Adam optimizer with learning rate of 0.005 which you halve after 50 and 75 epochs. You may use the code provided in the Jupyter notebook.
What do you observe? Briefly describe the results. Which number of layers would you choose? Give a short argument why.

Task 3.3 Re-Evaluation [4 Points]

Repeat the training from the previous task six times for six freshly initialized network architectures with the number of layers that you have chosen in Task 3.2. Print the mean and standard deviation of the train and test errors. What do you observe?

Task 3.4 Changing the MLP [5 Points]

Create two neural networks `NetSmall` and `NetLarge` with two GIN layers each. Choose 64-dimensional internal representations in all adequate places.

- For `NetLarge`, choose a fully connected MLP with one hidden layer of size 32 with ReLU activation function.
- For `NetSmall`, let the "MLP" have *no* hidden layer, i.e. let it consist of a single fully connected layer.

Train each model as in Task 3.2 and report the accuracy on the training and test split.

Task 3.5 From Practice to Theory [3 Points]

Interpret the results from Task 3.4. What do you observe? How does this relate to the theoretical results on the expressivity of GINs?

Task 4 Random Walk Kernels [21 Points]

Note: For this task, it is likely useful to use the grakel library. Recall that you can load a (standard) dataset using

```
from grakel.datasets import fetch_dataset

DATASET = fetch_dataset("DATASET", verbose=False, as_graphs=True)
graphs = DATASET.data
labels = DATASET.target
```

Task 4.1 Using the Random Walk Kernel [3 Points]

Load the `MUTAG` dataset and compute the Gram matrix of the geometric random walk kernel with the following parameters.

```
rwk = grakel.RandomWalk(lamda=0.0001, kernel_type='geometric',
                                      method_type='fast')
```

MUTAG contains 188 graphs. Let G_i for $i \in \{0, \dots, 187\}$ be the graph with index i in the MUTAG dataset. Print the kernel values for the following pairs of graphs:

1. G_{45} and G_{84}
2. G_{23} and G_{42}
3. G_{187} and G_{187}

Task 4.2 Crossvalidate an SVM Classifier [3 Points]

Evaluate the random walk kernel on MUTAG by running a 10-fold crossvalidation for a linear SVM classifier. Fix the SVMs C parameter to 1.0 and choose the random walk kernel parameters

```
rwk = grakel.RandomWalk(lamda=0.0001, kernel_type='geometric',
                                      method_type='fast')
```

Print the mean test accuracy and its standard deviation over the 10 folds.

Task 4.3 The λ Parameter of Random Walk Kernels [4 Points]

For reason of speed, consider only the dataset

```
small = graphs[:50]
```

for this subtask.
Which parameters $\lambda \in \{0.001, 0.01, 0.1, 0.2, 0.5, 0.9\}$ are suitable for the geometric random walk kernel on `small`? Briefly explain why.

Task 4.4 Random Walk Kernels and Isomorphism [4 Points]

Can you answer the following questions using the Random Walk Kernel computed in Task 4.1?

1. Are G_0 and G_{43} isomorphic?
2. Are G_0 and G_{69} isomorphic?

Print all necessary computations and/or kernel values and write down your answers and the reasoning that led you to it.

Task 4.5 Compute the Kernel Distance [3 Points]

Recall that given a kernel $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ there exists a distance function

$$d_k(G_1, G_2) := k(G_1, G_1) + k(G_2, G_2) - 2k(G_1, G_2)$$

Assume, that k is our random walk kernel

```
rwk = grakel.RandomWalk(lamda=0.0001, kernel_type='geometric',
                                      method_type='fast')
```

and let the distance matrix D_k be given by

$$D_k[i, j] = d_k(G_i, G_j) \ .$$

Compute the distance matrix D for the MUTAG dataset and print the distance values for the following pairs of graphs:

1. G_{45} and G_{84}
2. G_{23} and G_{42}
3. G_{187} and G_{187}

Task 4.6 Fix the Kernel Distance [2 Points]

As you know, distance functions should be nonnegative for all inputs. Ensure that your function indeed computes only nonnegative values by setting negative values to zero. Recompute the distance matrix and print the distance values for the following pairs of graphs:

1. G_{45} and G_{84}
2. G_{23} and G_{42}
3. G_{187} and G_{187}

Task 4.7 Plotting the MUTAG Dataset [2 Points]

Given the kernel distance matrix from Task 4.6, we can now plot the MUTAG dataset using some distance based low-dimensional embedding technique, such as t-SNE. Use the following snippet to plot the dataset.

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

tsne = TSNE(n_components=2, random_state=0, metric='precomputed')
emb = tsne.fit_transform(D)
plt.scatter(emb[:, 0], emb[:, 1], c=DATASET.target)
plt.show()
```

What do you observe? Judging from this illustration, is the random walk kernel distance a good fit for the classification task on MUTAG? Give short answers to those two questions.
Note: The code snippet assumes that you have loaded the dataset into the variable `DATASET` and created the distance matrix `D` in the previous task.

Task 5 R-Convolutional Graph Kernels [10 Points]

Recall that for some unlabeled graph G, and vertex $v \in V(G)$, and a connected graph H on $k \in \{3, 4, 5\}$ vertices, we have defined the *graphlet representation*

$$r_H(v) := |\{X \subseteq V \mid v \in X \wedge G[X] \text{ is isomorphic to } H\}| \ .$$

Note: This is a theory task. You do not need to implement anything.
Note: Jupyter notebooks support latex pidgin math mode. You can write formulas in markdown cells by enclosing latex formulas in dollars or double dollars (for display math) like this: `$k \in \{3,4,5\}$`

Task 5.1 Graphlet Representations for Graphs [4 Points]

Let $\mathcal{X}$ be a set of connected graphs on $k \in \{3, 4, 5\}$ vertices. Define a multiset graph representation

$$r_{\mathcal{X}} : \mathcal{G} \to \mathbb{N}^{|\mathcal{X}|}$$

for any unlabeled graph G based on the graphlet representations of all its vertices.

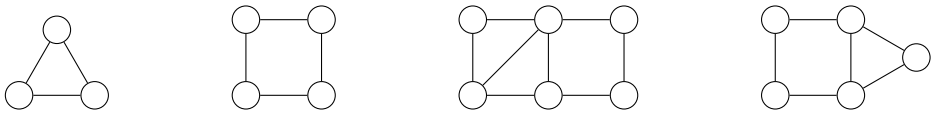
Task 5.2 Kernels on Graphlet Representations [4 Points]

Consider the graph kernel

$$k_{\mathcal{X}}(G_1, G_2) = \langle r_{\mathcal{X}}(G_1), r_{\mathcal{X}}(G_2) \rangle$$

where $\langle \cdot, \cdot \rangle$ is the scalar product / dot product between vectors. Show that $k_{\mathcal{X}}$ is an R-convolutional graph kernel.

Task 5.3 Compute the Kernel [2 Points]



C_3 C_4 G_1 G_2

Now, let $\mathcal{X}$ be the set that contains the triangle C_3 and the cycle C_4. Compute $k_{\mathcal{X}}(G_1, G_2)$.