INF-5050: Artificial Intelligence

ML Project

SOFTWARE ENGINEERING

Elite Graduate Program

Universität Augsburg University

Recap: Course Outline

- 5 ECTS for this course
  - ML project: working in groups of 2 students on a given ML task (50%)
  - Oral exam (30 min) at the end of January (50%)

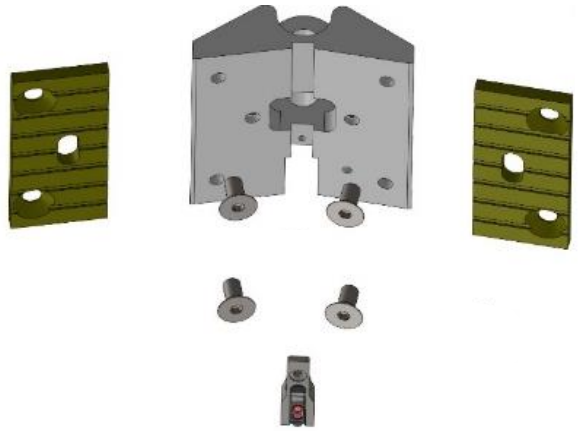
- Course Agenda

6th December	Recurrent Neural Networks	**Project start**
	CHRISTMAS HOLIDAYS	
10th January	Attention-based Models, State-of-the-Art Models	
17th January	Unsupervised Learning	
24th January	Reinforcement Learning	
27nd January	*Submission Deadline of Project*	
29th and 31st January	*Oral Exams*	**Project end**

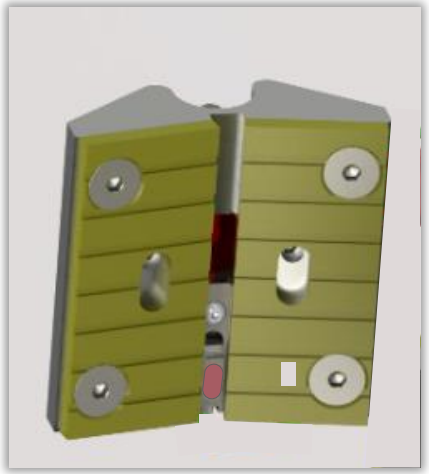
The ML Projekt: Your Task

Building assemblies based on a set of given parts.

Given: a set of parts



Target: a graph connecting these parts



Data and code for the project can be found in Digicampus.

About the Code: Graph, Node and Part

```python
class Part:
    """
    A class to represent pseudonymized parts.
    A part is described by its ID (part_id) and the ID of its corresponding family (family_id).
    Multiple parts can belong to the same family (i.e. different value for part_id but same value for family_id).

    """
    def __init__(self, part_id: int, family_id: int):
        assert part_id and family_id, 'Creation of Part failed. Fields `part_id` and `family_id` must not be empty.'
        self.__part_id: int = part_id
        self.__family_id: int = family_id
```

```python
class Node:
    """
    A class to represent nodes of a graph.
    A part is described by its ID (id) and its containing part (part).
    """
    def __init__(self, node_id: int, part: Part):
        self.__id: int = node_id
        self.__part: Part = part
```

```python
class Graph:
    """
    A class to represent graphs. A Graph is composed of nodes and edges between the nodes.
    Specifically, these are *undirected*, *unweighted*, *non-cyclic* and *connected* graphs.
    """

    def __init__(self, construction_id: int = None):
        self.__construction_id: int = construction_id  # represents unix timestamp of creation date
        self.__nodes: Set[Node] = set()
        self.__edges: Dict[Node, List[Node]] = {}
        ...
```

About the Code

- `evaluation.py` contains
  - an abstract class for your prediction models
  - the evaluation method

```python
class MyPredictionModel(ABC):
    """
    This class is a blueprint for your prediction model(s) serving as base class.
    """

    @abstractmethod
    def predict_graph(self, parts: Set[Part]) -> Graph:
        """
        Returns a graph containing all given parts. This method is called within the method `evaluate`.
        :param parts: set of parts to form up an assembly (i.e. graph)
        :return: graph
        """
        # TODO: implement this method
        ...
```

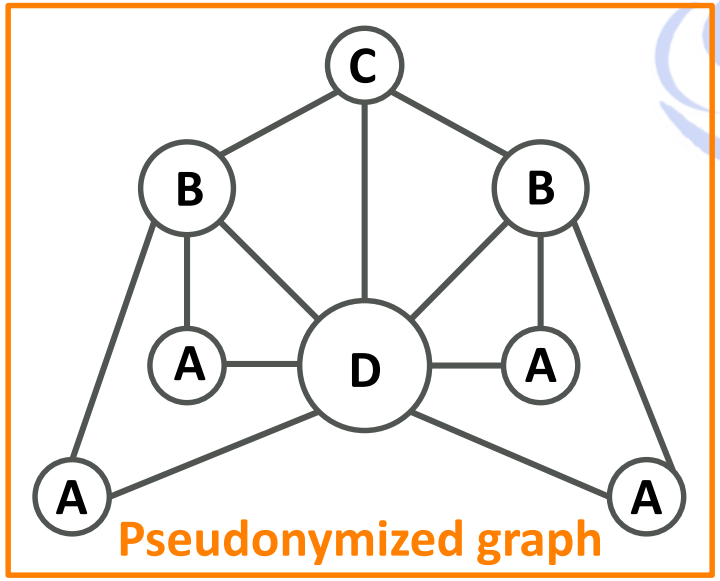
```python
def evaluate(model: MyPredictionModel, data_set: List[Tuple[Set[Part], Graph]]) -> float:
    """
    Evaluates a given prediction model on a given data set.
    :param model: prediction model
    :param data_set: data set
    :return: evaluation score
    """
```

About the Data

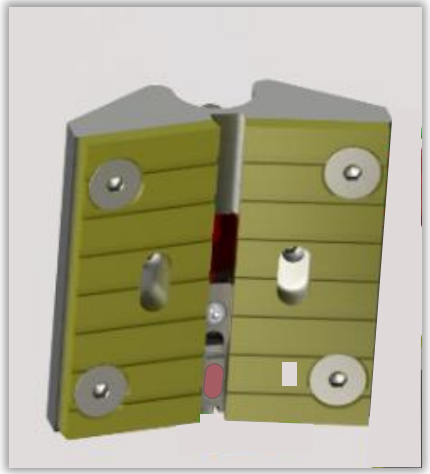
- The data stems from a real-world data set
- `graphs.dat` contains 11.159 graphs (`graph.py`) representing assemblies
  - Undirected, unweighted, non-cyclic and connected without self-loops.
  - They can contain multiple instances of the same part
  - 1.089 different parts; parts are pseudonymized
- We built a holdout set for testing your final models
  (all parts used there are already seen in the training data)

About the Data: Assemblies as Graphs

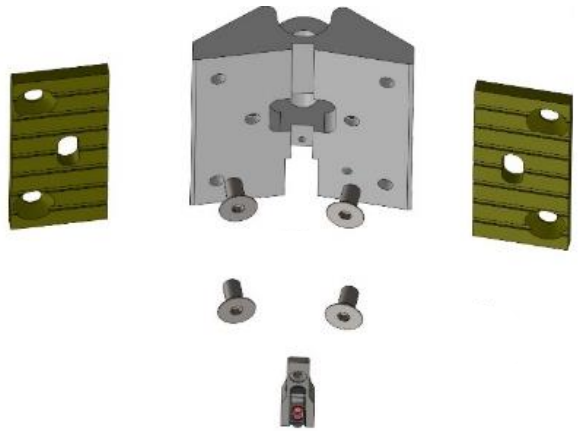
Elementary parts ≙ Nodes
Physical connections ≙ Edges



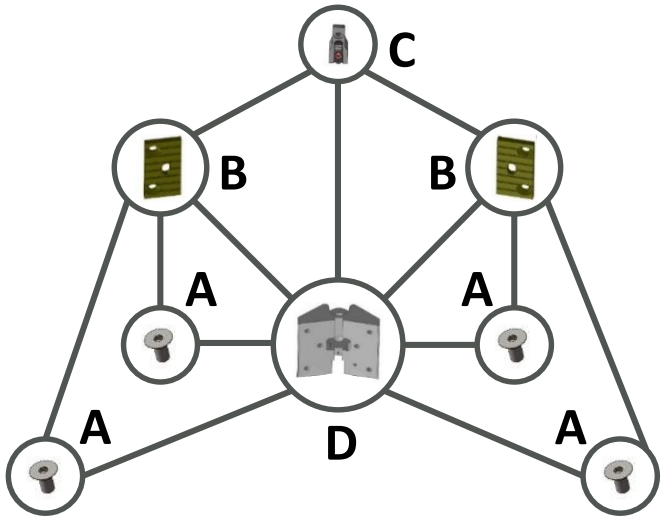
Assembly model



Exploded-view drawing



Extracted graph



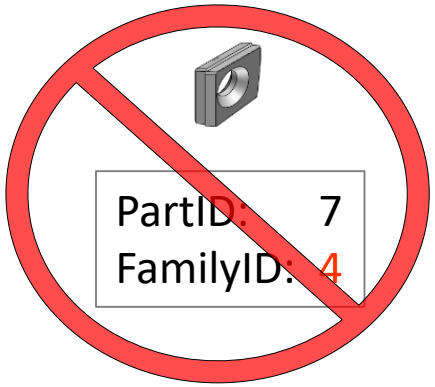
About the Data: PartID vs FamilyID in Parts

Different parts may belong to the same part family. They are like variants.
Every part belongs to exactly one part family.

PartID: 7
FamilyID: 5

PartID: 4
FamilyID: 5

PartID: 9
FamilyID: 2



PartID: 7
FamilyID: 4

```python
class Part:
    """
    A class to represent pseudonymized parts.
    A part is described by its ID (part_id) and the ID of its corresponding family (family_id).
    Multiple parts can belong to the same family (i.e. different value for part_id but same value for family_id).

    """
    def __init__(self, part_id: int, family_id: int):
        assert part_id and family_id, 'Creation of Part failed. Fields `part_id` and `family_id` must not be empty.'
        self.__part_id: int = part_id
        self.__family_id: int = family_id
```

Expected Resources at the Deadline

- Your code (in Python and PyTorch)
- Documentation about your project, **your decisions** and performed experiments

- Saved model that can be loaded to perform the final evaluation.
- Code for loading your model (see `evaluation.py`)

Correct, reasonable procedure takes precedence over performance.

Possibly helpful

- Execute Python code in browser, e.g. Google Colab
  https://colab.research.google.com/

- Tracking experiments, e.g. mlflow https://mlflow.org/