Project 2: Model Engineering

Using Feed Forward Neural Network (FFNN), Recurrent Neural Network (RNN) and Graph Neural Network (GNN)

1 Objective

The goal of this project is to learn how to perform model engineering. For this, students will have to model the same problem and data using different architectures, to understand how to use the different preprocessing tecnquies based on the architectures used to face a classification tasks.

Specifically, students will:

- Learn how to preprocess categorical data using different methodologies.
- Learn how to preprocess based on the Neural Network architectures.
- Experiment with different architectures (FFNN, RNN, and GNN) using different hyper-parameters, optimizers and architectures.

2 Submission Rules

- Groups consist of 3 students.
- Each group must submit a zip file containing the following:
  - A report (maximum 10 pages) describing the approach, experiments and results, including tables and plots.
  - The Juypyter notebook(s) and code (e.g., libraries with classes and functions written by you) to solve the tasks.
    - **Best practice**: Add comments and headers (Markdown) sections to understand what you are doing. They will i) help you tomorrow to understand what you did today and ii) help us to interpret your solution correctly.
    - **The notebook needs to be executed**: code and results must be visible so that we can interpret what you have done and what the results look like.
    - **Must run**: the code must work if we need to run it again.
    - **Submission format**: Include the notebook file (`.ipynb`) and an HTML export for easier review.
- Each group must upload the zip file to the teaching portal via Moodle before the deadline.

3 Dataset: Malware Analysis Datasets: API Call Sequences

This dataset is part of research on malware detection and classification using Deep Learning. It contains 42,797 malware API call sequences and 1,079 goodware API call sequences. Each API call sequence is composed of **up to** 100 non-repeated consecutive API calls associated

with the parent process, extracted from the 'calls' elements of Cuckoo Sandbox reports. Some of the API sequence can be comade upewer API calls.

More information about the dataset can be found at:

- Kaggle dataset
- and on the paper Oliveira, Angelo; Sassi, Renato José (2019), "Behavioral Malware Detection Using Deep Graph Convolutional Neural Networks.", TechRxiv. Preprint. at `https://doi.org/10.36227/techrxiv.10043099.v1`

The dataset is in form of a `Json` file where each document describing a API call sequence is composed of

- **md5hash**: MD5 hash of the example encoded with 32 bytes string.
- **api_call_sequence**: the sequence of API calls, each describe by its name. (**notice this sequence can have variable length**)
- **is_malware**: th class label coded with an integer: 0 (Goodware) or 1 (Malware).

4 Tasks

Students must create a Machine Learning pipeline based on a Feed Forward Neural Network (FFNN), Recurrent Neural Network (RNN), and Graph Neural Network (GNN) that performs all required steps. Each task describes the steps to follow.

4.1 Task 1: BoW Approaches

Each sequence of processes is a string, all the processes are words in a centence –> Fit a CountVectorizer object from sklearn!

- How many columns do you have after applying the CountVectorizer? What does that number represent?
- What does each row represent? Can you still track the order of the processes (how they were called)?
- Do you have *out-of-vocabulary* from the test set? If yes, how many? How does CountVectorizer handle them?
- Try to fit a classifier (at your choice, shallow deep or neural network). Report how you chose the hyperparameters of your classifier, and the final performance on the test set.

4.2 Task 2: Feed Forward Neural Network

Now, obtain some stats on the number of processes called per sample.

- Do you have the same number of calls for each sample? Is the training distribution the same as the test one? shorter than the testing ones.
- Let's say that you really want to use a simple FeedForward neural network to solve the problem. Can a Feedforward Neural Network handle a variable number of elements? Why?

- Which technique do you adopt to bring everything to a fixed size during training? What happens if at test time you have more processes to handle?
- Each process is actually a string: first solve the problem using sequential identifiers, then try using learnable embeddings. Use a FeedForward network on both cases. Report how you selected the hyper-parameters of your final model, and justify your choices. Can you obtain the same results for the two alternatives (sequential identifiers and learnable embeddings)?

  Articulate (report results, whether one training was longer/more unstable than the other, etc.)

4.3 Task 3: Recursive Neural Network (RNN)

Next, use a Recursive Neural Network (RNN) to model your problem.

- Do you still have to pad your data? If yes, how?
- Do you have to truncate the testing sequences? Justify your answer with your understanding of why it is/it is not the case.
- Is there any memory advantage on using an RNN wrt a FF when you process your dataset? Why?
- Start with a simple one-directional RNN. Is your network fast as the FF? If not, where do you think that time-overhead comes from?
- Train and tne three variations of networks:
  - Simple one-directional RNN
  - Bi-Directional RNN
  - LSTM (choose whether you want it one-directional or bi-directional)

  Can you see any differences during their training? Can you obtain the same

4.4 Task 4: Graph Neural Network (GNN)

Finally, use a Graph Neural Network (GNN) to model your problem. Consider each sequence of API call as a graph.

- Do you still have to pad your data? If yes, how?
- Do you have to truncate the testing sequences? Justify your answer with your understanding of why it is/it is not the case.
- What is the advantage of modelling your problem with a GNN with respect to FFNN and RNN? What do you lose?
- Start creating a first simple GCN and train/test it on CPU first and GPU next. How long does it take for training and testing in each configuration? How does it differ with respect to previous architectures? Can you guess why?

- Finally train and tune variations of GNN considering different message and aggregation functions and architectures:
  - Simple GCN
  - GraphSAGE
  - and GAT

  Can you see any differences during their training? Can you obtain the same