UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Master in Innovation and Research in Informatics

Master's Thesis

Graph Neural Network Applications

Author:
Pau Rodríguez Esmerats

Professor:
Marta Arias Vicente

August 4, 2019

Contents

Abstract

abstract-text

Contents

1 Introduction

Introduction...

2 State of the art

SOTA...

3 Methodology

Methodology...

4 Implementation

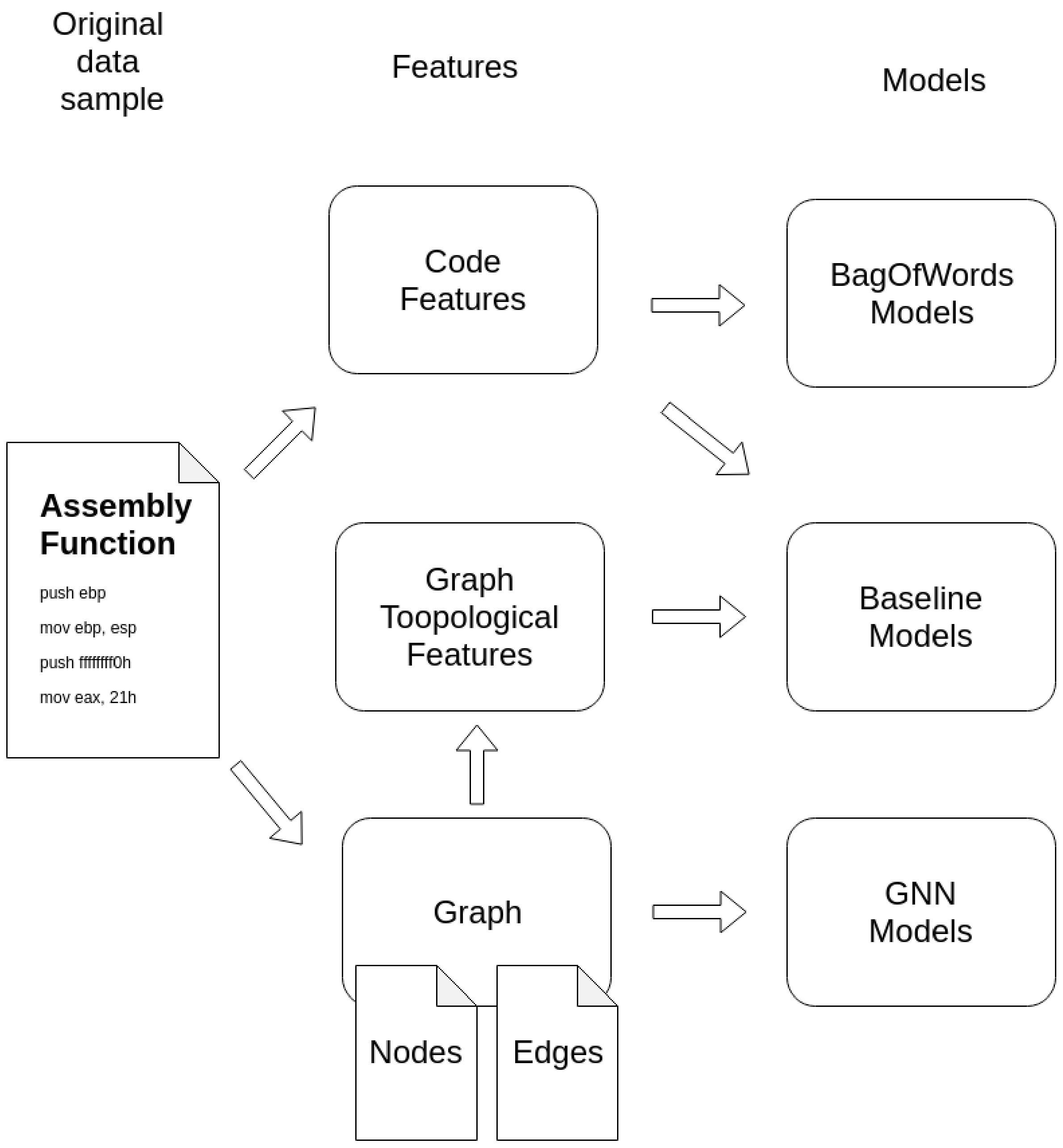

Figure 1: Features exracted from the dataset samples and their relationship with the models trained

5 Evaluation

6 Conclusion

References

[1]

[2]

[3]

Appendices

A Feature extraction details

This annex will present the details of the implementation of the feature extraction from the dataset of assembly function code listings, used in the task of assembly function code classification.

A.1 Graph of an assembly function code

In this thesis, the tasks that classifies binary code functions takes advantage of programs that are prepared for reading and parsing binary files into human readable formats, using a programing language called assembler or assembly. However, a binary file is a file that contains bytes organized following a convention, or file format like PE or COFF, completely dependent of the operating system, but not assembler directives. The bytes contained in those file formats represent machine language code. This is not a problem as, per wikipedia definition, an assembly language (or assembler language), often abbreviated asm, is any low-level programming language in which there is a very strong correspondence between the program's statements and the architecture's machine code instructions.

For analysing a binary file, programs that convert the bytes of the binary file, that represent machine code, into assembler instructions are called disassemblers. For this thesis the selected disassembler program is called IDA free, which is a free version of the corresponding professional program IDA pro. This program reads the binary file bytes, according to the corresponding file formats, like PE for Microsoft Windows or COFF2 in Unix operating systems, and is able to interpret the machine code and translate it into assembler code instructions. There is a direct correspondence between machine code and assembler directives.

The selected disassembler program, IDA free, allows for executing scripts written in the programming language python. This scripts are run from inside its user interface to do tasks, like inspection and modification of the database of the binary code loaded.

The python script is ran as a script, also known as "plugin", inside the disassembler program (IDA free). It takes advantage of series of exposed interfaces or methods that read the database that contains the information of the disassembled binary file and its instructions. The procedure followed by the python script is to process all the lines of code of each function contained in the binary file. For each function or subroutine that the disassembler has detected, the python script will write 2 separate text files that contain the list of elements found as nodes and the list of relationships between them as edges, conveniently following a file format that the python package NetworkX can read and import into a NetworkX graph representation. The elements considered are all the types of elements that can appear in assembly code and that the disassembler detects: instructions, registers, memory addresses, immediate values, displacements and called functions. The edges represent relationships between them. For example, the instruction "mov eax, 0x0457AB" is related to the register "eax", the memory address "0x0457AB", as well as the previous and next instruction found in the code listing when following its execution flow. When the instruction is calling another function, for example "call another_function_address", an edge relating this instruction and the function another_function_address.

Several assembly code functions are selected for testing the python plugin that generates the graph of the function code. The procedure is simple, the assembly code is printed as a code listing (each instruction is shown in assembly language in a separate line) like the disassembly program shows it inside its user interface. Then the contents of the file containing the description of the nodes, usually called function_name_nodes.txt where the function_name part is replaced by the actual function name being analysed, and the file containing the set of edges joining nodes, usually called function_name_edges.txt where the function_name part is replaced by the actual function name being analysed, are compared to the code listing to verify that every line of code, every register, memory address and the like is present in the nodes file, and every relationship between instructions, registers, memory addresses, called functions and the like is also present in the edges list file.

A.2 Feature engineering in assembly code classification

Some of the machine learning models for classifying code functions in assembly (compiled binaries), make use of features that have been designed outside of the algorithm. It's the case of the code features, those related to aspects of an assembly function code like number of instructions or number of called functions, and the topological features of the graph of a function, those derived from the function's graph based on the relations between instructions, registers, memory addresses and other instructions. For the rest of the models of this task, the ones based on graph neural networks, there is no feature engineering, as they use the graph as an input to learning and prediction.

The code features are aimed at summarizing or extracting relevant aspects of the underlying code. To this goal, counts of number of instructions, registers, memory addresses, function calls and the like are extracted from each assembly function code listing. The full assembly code listing is also extracted as a feature for models that use Bag of Words strategies. These features are simply obtained by reading the text files that list the nodes and the edges of the graph of the function, as the allow for recovering the basic elements on which the graph is constructed, the nodes, which are representations of instructions, registers, memory addresses and the like.

The code features extraction implementation is tested on several assembly fuction code listings, by comparing the counts of instructions, registeres, memory addresses and the like versus those values extracted by the code that reads the text files that list nodes and edges of the graph of the assembly function.

The features that are derived from the topological aspects of graphs, like assortativity and clustering coefficient, are obtained with the implementation included inside the NetworkX Python library.

No testing of these topological features is performed, as the Python library NetworkX is used by a large community and has a great reputation.