Homework 3: Graph Classification
COL 761

- Due: **11th Nov 6:00PM**
- **30%** penalty for each late day.
- You are free to use any programming language but it must compile in an UNIX-based OS. Please start early. For submissions in C++, you must fix the optimization flag to **O3**.
- Testing system configuration: **Ubuntu 18.04 | 16GB | gcc 7.3.0**
- You can do the homework in groups of 3. **Only one zip file must be submitted per team.**
- You should upload all your code to the GitHub repo used for HW2. You are not allowed to make any uploads after the deadline. Otherwise, you will be penalized as per the late submission policy. Your repos will be cloned immediately after the deadline.

---

1. This question is about getting yourself familiar with frequent subgraph mining tools. You will use the dataset of molecules tested against AIDS . The format is the following:

```
#graphID
# of nodes
Series of Node Labels
# of edges
Series of "Source node, Destination Node, Edge label"
```

Run gSpan, FSG (also known as PAFI), and Gaston (you should be able to find it online) against frequency threshold in the AIDS dataset (you may need to write a script to change the format of the dataset) at minSup = **5%, 10%, 25%, 50%** and **95%**. Plot the running times and explain the trend observed in the running times. Specifically comment on the growth rates and why one technique is faster than the others. You are free to consult the respective papers. **(20 points)**

2. Consider the same AIDS graph dataset. We now provide you a label for a subset of the graphs: the active molecules against HIV virus and the inactive molecules. Molecules that do not have a class label should be ignored. Design a technique to classify graphs. More specifically, convert each graph into a binary feature vector where each dimension corresponds to the presence or absence of the corresponding subgraph. Ideally, you should not use all frequent subgraphs as features. Rather, you should only use the "discriminative" frequent subgraphs. We will classify the graphs (in the feature space) using the linear kernel of **LIBSVM** ("A Library for Support Vector Machines"). While training, we will also ensure that the trainset has an equal number of active and inactive molecules.

We will evaluate you on another dataset for molecules tested against AIDS using the **F-score** measure. **(60 points)**.

Your grades will be assigned as follows.

- Top 10% of all F-scores= 60 points
- Top 20% of all F-scores= 50 points
- Top 30% of F-scores = 40 points
- Top 50% of all F-scores or F-score > 0.5 = 30 points
- Top 90% of all F-scores=20 points
- F-score > 0=10 points

Submit a compilation script titled **compile.sh** that compiles all your sources files. For automated testing, you must also submit a shell script titled **classify.sh**. It should support the following operation.

sh classify.sh <trainset filename containing graphs> <active graph IDs filename> <inactive graph IDs filename> <testset filename containing graphs>

The output should be two files titled **train.txt** and **test.txt**. In **train.txt**, the i^{th} line contains the class label of graph i followed by the feature vector representation of the i^{th} graph in the trainset. The label of an active graph is "1" and an inactive graph is "-1". Each line must be of the following format:

```
<label> <index1>:<value1> <index2>:<value2> ...
.
.
.
```

The **test.txt** file should of the same format as above, with the only exception being that you do not need to include the class label. That is, it should be of the form

```
<index1>:<value1> <index2>:<value2> ...
.
.
.
```

If test set contains 100 graphs, **test.txt** should contain 100 lines (same for **train.txt** as well). We will run **LIBSVM** on your files. **If you do not adhere to the above format, you will receive 0 points. More specifically, you should run LIBSVM on train.txt and ensure that LIBSVM is able to train on your data format.**

Your shell script must complete within **30 minutes**. This includes both the training and testing time.

You may need to perform subgraph isomorphism for this task. To help you, a Java library with an example code is shared. To see an example of how subgraph isomorphism is performed, check the file **toolbox/subgraphFrequencyCounter.java**. You may use any other package as well.

Submission Instructions

Please follow these guidelines for Homework 3 submissions due **18:00 November 11, 2018.**

Managing GitHub Repo:

1. Either keep your Data Mining repository public or add us as a Collaborator for your private repository. Username: **COL761Staff**

2. All submitted repositories will be cloned immediately following the expiry of the deadline. The timestamp of the last pushed commit will be treated as your submission time. In addition to the existing submission files required, you are also required to (separately) submit an **install.sh** script on Moodle [Do not zip it].

3. **install.sh** should execute cloning in the current directory followed by executing bash commands to install all modules/dependencies/libraries in the correct path in your submission folder.

4. ONLY in case your dependencies cannot be downloaded through a bash command, please add a description of the module/library to be imported along with its download path and installation instructions as comments (each module used with installation description in a new line) in **install.sh**

5. This is the ONLY script in your submission that we will review manually to ensure successful imports. After this, please ensure your project compiles and executes successfully. The evaluation would terminate if your executables fail our scripts and this would incur a significant penalty.

6. Execution: **sh install.sh** [all Bash scripts will be executed using **sh** bash command]

7. It is absolutely essential to conform to the **folder hierarchy and naming conventions** (see below)

8. **README.txt** should list the entry numbers and names of all members of your group (in separate lines) in the following format: <EntryNo>:<Name>

Submission Organisation & How to Submit

You have to submit one **install.sh** file per group on **Moodle**.

We will run **install.sh** which should clone your **remote GitHub Data Mining repository**. Inside the cloned (local) repository, we should be able to locate **HW3_<EntryNo>.zip** which corresponds to your Homework 3 submission. Further, as mentioned before, **install.sh** should contain bash commands that install any dependencies you might require. We will be executing **install.sh** with root privileges. For example: pip install HeapDict

At this stage, all imports are resolved. Now, we will unzip **HW3_<EntryNo>.zip** which should create a folder named **HW3_<EntryNo>** containing **compile.sh**, **classify.sh, README.txt** and all your **source files.** **We will change the working directory to the unzipped directory HW3_<EntryNo>**, then inside it, execute **compile.sh** followed by **classify.sh** which should generate output files **train.txt** and **test.txt** (corresponding to the flag), in the working directory.

Python submissions must conform to Python-3.6.5 only.

The Data-Mining remote repository should have Homework 1 & 2 files also, although we will not be explicitly checking for this. You can have any files/scripts/folders related to the all assignments hosted on your repo. We will only concern ourselves with the zipfile corresponding to Homework 3 **(HW3_<EntryNo>.zip)** which should be **directly** inside your repository.

Testing system: Ubuntu 18.04/16GB/gcc 7.3.0. Timeout will be set for 30 minutes. Your program must generate all required output files within this time limit.

Use HTTPS cloning:

git clone https://github.com/ABC/XYZ.git

Do NOT use SSH:

git clone git@github.com:ABC/XYZ.git

Anti-Plagiarism Policy for Homework 3

All submitted source files for Homework 3 will be scanned via Moss. Any detected attempts at plagiarism either from parallel/past submissions or the Internet will risk an F-grade in the course.