COL761 HW1

Deadline: 6 February 11:59 PM

This assignment has 3 parts:

1. Frequent Itemset Mining (15 marks)
2. Frequent Subgraph Mining (15 marks)
3. Graph classification (70 marks, competetive)

Instructions

- We'll count your latest `git push` as your submission time.
- There is no Moodle submission.
- Add `https://github.com/2023col761` as a collaborator to your github repository.
- Upload your assignment under `hw1` directory.
- The directory structure should be like this:

  ```
  hw1
  ├── q1/
  ├── q2/
  └── q3/
  ```

- The directory structures of `q1, q2, q3` are indicated under their problem statements.
- Do not submit data files.

Anti-Plagiarism Policy

Any detected attempts at plagiarism from parallel/past submissions will risk an F-grade in the course. Add references to any libraries used in your report.

Question 1: Frequent Itemset Mining (15 marks)

Conduct an empirical comparison of the Apriori and FP-tree algorithms for frequent itemset mining. Utilizing provided libraries, analyze their efficiency on the given dataset. Measure the runtime of both algorithms at five different support thresholds: `5%, 10%, 25%, 50%, 90%`. Subsequently, visualize the results by plotting line graphs on a single plot with the support threshold on the x-axis and runtime on the y-axis. Finally, provide a comprehensive analysis of your observations in a written report, comparing the performance characteristics of the algorithms.

Libraries

Linux binaries: OneDrive link

Refer to the documentation for the runtime arguments and input formats.

Apriori:

- Source code: https://borgelt.net/src/apriori.zip
- Documentation: https://borgelt.net/doc/apriori/apriori.html

FP-tree:

- Source code: https://borgelt.net/src/fpgrowth.zip
- Documentation: https://borgelt.net/doc/fpgrowth/fpgrowth.html

Dataset

- Dataset: http://fimi.uantwerpen.be/data/webdocs.dat.gz
- Description: http://fimi.uantwerpen.be/data/webdocs.pdf

Submission format

```
hw1/q1
├── q1.pdf
├── *.py
└── q1.sh
```

Your submission folder for this part should look as above.

- `q1.sh` should execute your entire pipeline. See more details in the next section.
- `*.py` indicate your python scripts.
- `q1.pdf` should contain the analysis of your observations comparing the performance characteristics of the algorithms.

Running your code

```
cd hw1/q1

bash q1.sh <path_apriori_executable> <path_fp_executable> <path_dataset> <path_out>
```

```
# <path_apriori_executable>: absolute filepath to apriori's compiled executable
# <path_fp_executable>: aboslute filepath to fp-tree's compiled executable
# <path_dataset>: absolute filepath to the dataset file

# <path_out>: absolute folderpath where the plot and the outputs at different thresholds will be saved
```

Running this code should run and time the algorithms, generate output files, and generate the plot. After running `q1.sh` the output folder should look like this where `ap10` corresponds to Apriori's output at 10% threshold:

```
<path_out>
├── ap10
├── ap25
├── ap5
├── ap50
├── ap90
├── fp10
├── fp25
├── fp5
├── fp50
├── fp90
└── plot.png
```

Code environment

```
- python=3.10
- matplotlib
```

Using libraries outside this environment will throw an error during evaluation.

Grading

- [5 marks] The output files and the plot will be crosschecked against the TA's outputs and plot.
- [10 marks] Your analysis in the report carries the maximum weightage.

Tips

- Use baadal for this part.
- You need not submit the binaries or the libraries.
- Don't forget the x and y labels in your plot and indicate the units of measurement.
- Your output filenames should match the above mentioned file names for the autograder to work.

Question 2: Frequent Subgraph Mining (15 marks)

This part will familiarize you with frequent subgraph mining tools. Run `gSpan`, `FSG` (also known as `PAFI`), and `Gaston` on the Yeast dataset at `minSup = 5%, 10%, 25%, 50%, 95%`. You may need to write a script to change the dataset format for each algorithm's library. In a single plot, plot the running times against the minimum supports for each method. In your report, explain the observed trends. Specifically, comment on the growth rates and why one technique is faster. You are advised to consult the respective papers.

Linux binaries: OneDrive link

Refer to the README files in the libraries for the runtime arguments and input formats.

gSpan:

- Paper: https://sites.cs.ucsb.edu/~xyan/papers/gSpan.pdf
- Library: https://sites.cs.ucsb.edu/~xyan/software/gSpan.htm

FSG:

- Paper: http://delab.csd.auth.gr/~manolopo/oikonomiko/fsg.pdf
- Library: OneDrive link
- The library used to be at: https://karypis.github.io/gkhome/pafi/download

Gaston:

- Paper: https://liacs.leidenuniv.nl/~nijssensgr/gaston/gaston-april.pdf
- Library: https://liacs.leidenuniv.nl/~nijssensgr/gaston/download.html
- Those trying to compile the library may face errors.
    - Add `#include <getopt.h>` to main.cpp before running `make`

Dataset

- Dataset: OneDrive link
- Dataset format:

```
#graphID
number of nodes
node label
node label
.
.
.
number of edges
source_node_id, destination_node_id, edge_type
source_node_id, destination_node_id, edge_type
.
.
.
```

Submission format

```
hw1/q2
├── q2.pdf
├── *.py
└── q2.sh
```

Your submission folder for this part should look as above.

- `q2.sh` should execute your entire pipeline. See more details in the next section.
- `*.py` indicate your python scripts.
- `q2.pdf` should contain the analysis of your observations comparing the performance characteristics of the algorithms.

Running your code

```
cd hw1/q2

bash q2.sh <path_gspan_executable> <path_fsg_executable> <path_gaston_executable>
<path_dataset> <path_out>

# <path_gspan_executable>: absolute filepath to gspan's compiled executable
# <path_fsg_executable>: aboslute filepath to fsg's compiled executable
# <path_gaston_executable>: aboslute filepath to gaston's compiled executable
# <path_dataset>: absolute filepath to the dataset file

# <path_out>: absolute folderpath where the plot and the outputs at different minimum
supports will be saved
```

Running this code should run and time the algorithms, generate output files, and generate the plot. After running `q2.sh` the output folder should look like this where `gspan10` corresponds to gspan's output at `minSup = 10%`:

```
<path_out>
├── fsg10
├── fsg25
├── fsg5
├── fsg50
├── fsg95
├── gaston10
├── gaston25
├── gaston5
├── gaston50
├── gaston95
├── gspan10
├── gspan25
├── gspan5
├── gspan50
├── gspan95
└── plot.png
```

Code environment

```
- python=3.10
- matplotlib
```

Using libraries outside this environment will throw an error during evaluation.

Grading

- [5 marks] The output files and the plot will be crosschecked against the TA's outputs and plot.
- [10 marks] Your analysis in the report carries the maximum weightage.

Tips

- Use baadal for this part.
- You need not submit the binaries or the libraries.
- Don't forget the x and y labels in your plots and indicate the units of measurement.
- Your output file names should match the indicated file names for the autograder to work.

Question 3: Graph classification (70 marks, competetive)

Graph classification is pivotal across various fields. In bio-informatics, it predicts protein functions and aids drug discovery by deciphering molecular interactions. Chemo-informatics employs it to predict compound activities and assess molecular structures and properties. In graph classification, we learn a function $$F$$ that takes a graph $$G$$ and maps it to a label from a predetermined label space $$L$$. In this assignment, we will classify molecular graphs.

Given a graph database, identify the subgraphs crucial for classification. Use them to convert each database graph into a binary presence/absence feature vector $$X$$ consisting of at most 100 dimensions. A naive approach would be to take top-k (k <= 100) frequent subgraphs as the features. A better approach may be to mine a larger set of frequent subgraphs and filter out discriminating ones to form the feature vector.

Datasets

Your are provided two molecular datasets for binary classification. You'll be evaluated on them and a third dataset which we have retained (also for binary classification).

Link to datasets: OneDrive link

The dataset format is as follows. The text in () are comments for your understanding.

```
# (new graph)
v 0 1 (node 0, label 1)
v 1 2 (node 1, label 2)
e 0 1 3 (edge from node 0 to node 1, with label 3)
# (new graph)
.
.
.
```

`labels.txt`: labels follow the same order as graphs in `graphs.txt`

```
0
1
1
.
.
.
```

You are advised to read the datasets' research papers for ideas:

- TUDataset: TUDataset: A collection of benchmark datasets for learning with graphs
- Mutagenicity: Derivation and Validation of Toxicophores for Mutagenicity Prediction
- NCI-H23: Comparison of descriptor spaces for chemical compound retrieval and classification

Submission format

```
hw1/q3
├── convert.sh
├── identify.sh
├── env.sh
├── *.py
└── q3.pdf
```

Your submission folder for this part should look as above where:

- `identify.sh`
    - It identifies the discriminative subgraphs from the input graphs whose presence/absence will act as features.
    - Only the training graphs and labels will be passed to this script.
    - For better generalization, you are advised to create a train-test split and only feed the training data.
    - The TA will create their own splits for grading and only pass the training data to this script.
- `convert.sh`
    - It converts graphs into feature vectors by performing subgraph isomorphism against the subgraphs stored earlier.
    - This script will be called twice seperately, once on the training graphs, once on the test graphs.
    - The feature vectors must be 2D numpy arrays.
- `*.py` indicates your python scripts.
- `env.sh` to setup your code enviroment.
- `q3.pdf` describes how you identified the discriminative subgraphs and the reasoning behind your approach.

Running your code

```
bash env.sh
```

```
bash identify.sh <path_train_graphs> <path_train_labels>
<path_discriminative_subgraphs>

# <path_train_graphs>: absolute filepath to the dataset of training graphs.
# <path_train_labels>: absolute filepath to the labels of training graphs.

# <path_discriminative_subgraphs>: absolute filepath to store the discriminative
subgraphs
```

```
bash convert.sh <path_graphs> <path_discriminative_subgraphs> <path_features>

# <path_graphs>: absolute filepath to the dataset of graphs. These can be train or
test graphs.
# <path_discriminative_subgraphs>: absolute filepath to the discriminative subgraphs.
# <path_features>: absolute filepath to store the input dataset mapped to the feature space. This must be a 2D numpy array.
```

```
python3 classify.py --ftrain <path_features_train> --ftest <path_features_test> --ltrain <path_labels_train> --ltest <path_labels_test> --proba <path_proba>

# <path_features_train>: absolute filepath to the training graphs' features.
# <path_features_test>: absolute filepath to the test graphs' features.
# <path_labels_train>: absolute filepath to the labels of training graphs.
# <path_labels_test>: absolute filepath to the labels of test graphs.

# <path_proba>: absolute filepath to store the predicted probabilities of class 1 on the test graphs.
```

You are provided `classify.py`. Your only focus is identifying the most discriminative subgraphs. Do not alter this script. Link to `classify.py`: OneDrive link

Algorithmic restrictions

- You are allowed to use libraries and are not limited to python libraries.
- Your features should not be any dense representations of graphs.
- They should be binary indicating the presence/absence of discriminative subgraphs.
- Do not use hacks like:
  - training a tree-based classification ML model on features made from frequent subgraphs to score the subgraphs.
  - applying PCA or LDA on the feature vector made from the presence/absence of frequent subgraphs.
  - training a neural network.
- Instead, research the graph classification literture from the pre-graph-neural-network era.
- The goal of this assignment is to identify the discriminative subgraphs. Your algorithm should be tailored for this.
- If you are in doubt about the validity of your algorithm choice, ask in private on Piazza for confirmation.
- Your code will be crosschecked against your report to catch any cheating attempts.

Code environment

- You have the freedom to build your environment in this part.
- Hence, you'll have to submit a script `env.sh` that creates the necesasry environment.
- You'll have internet access while setting up the environment.
- You can use baadal or hpc for this part.
- After the deadline passes, we'll float a form in which you'll let us know whether to grade your q3 on baadal or hpc as loading modules on hpc is specific to it.

Grading

- [10 marks] Report
- [60 marks] Competitive

  - The dataset score will be computed by taking the arithmetic mean of test ROC-AUC(%) over multiple train-test splits.
  - The overall score will be the minimum of the three dataset-scores.
  - The final score will be (overal score / max overall score in class) * 60

Tips

- Needless to say that you'll have read-only access to the data while grading. Any attempt to tamper with it will throw an error and show up in the logs. Strict action will be taken in such cases.