ALTeGraD 2023 Data Challenge
Molecule Retrieval with Natural Language Queries

École Polytechnique

December 2023

1 Challenge Description

The goal of this project is to study and apply machine learning/artificial intelligence techniques to retrieve molecules (graphs) using natural language queries. Natural language and molecules encode information in very different ways, which leads to the exciting but challenging problem of integrating these two very different modalities.
In this challenge, given a text query and list of molecules (represented as graphs), without any reference or textual information of the molecule, you need to retrieve the molecule corresponding to the query. This requires the integration of two very different types of information: the structured knowledge represented by text and the chemical properties present in molecular graphs. The pipeline to deal with this task can be achieved by co-training a text encoder and a molecule encoder using contrastive learning. This involves simultaneously training two separate encoders—one specialised in handling textual data and the other focused on molecular structures. Through contrastive learning, the model learns to map similar text-molecule pairs closer together in the learned representation space while pushing dissimilar pairs apart.
The challenge is hosted on Kaggle, a platform for predictive modelling on which companies, organisations and researchers post their data, and statisticians and data miners from all over the world compete to produce the best models. The challenge is available at the following link: `https://www.kaggle.com/competitions/altegrad-2023-data-challenge`. To participate in the challenge, use the following link: `https://www.kaggle.com/t/a87fac011c7f4f7c91d38d8351e799ac`.

2 Dataset Description

As mentioned above, you will evaluate your methods on a dataset consisting of molecules and their descriptions. You are given the following files
(available at: `https://1drv.ms/u/s!AhcBGHWGY2mukdgpQZ1L-csK3qJ-1w?e=clJXho`).

1. **train.tsv**: A `tsv` file that contains the training data (26408 samples) in the following format:

   ```
   CID    Description
   ```

   Where `CID` represents the molecule (graph) ID and `Description` represents the corresponding textual description.

2. **val.tsv**: A `tsv` file that contains the validation data (3301 samples) in the following format:

```
CID    Description
```

Where `CID` represents the molecule (graph) ID and `Description` represents the corresponding textual description.

3. **token_embedding_dict.npy**: It is a dictionary mapping molecule's substructure tokens to their embeddings. It can be loaded with the following code:

```
import numpy as np
token_embedding_dict = np.load("token_embedding_dict.npy",
                               allow_pickle=True)[()]
```

4. **./data/raw/**: A folder contains 102981 `cid.graph` files. These are formatted first with the edgelist of the graph and then the correspondent substructure token for each node. For example:

```
edgelist:
0 1
1 0
1 2
2 1
1 3
3 1

idx to identifier:
0 3545365497
1 2664995851
2 1510328189
3 2807496773
```

Where `idx` represents the node index and `identifier` represents the token (node) identifier in `token_embedding_dict.npy`

5. **test_text.txt**: A text file contains 3301 textual descriptions from the test dataset. As the goal of this data challenge is to map each description in this data file to a molecule in `test_cids.txt`, **this dataset should not be shuffled**.

6. **test_cids.txt**: A text file contains 3301 `cid` (graph ID) from the test dataset. As the goal of this data challenge is to map each description in `test_text.txt` to this data file, **this dataset should not be shuffled**.

3 Evaluation

The performance of your models will be assessed using the label ranking average precision score[1]

$$LRAP(y, y') = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} \frac{1}{||y_i||_0} \sum_{j:y_{ij}} \frac{|\mathcal{L}_{ij}|}{rank_{ij}}$$

[1] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.label_ranking_average_precision_score.html

Where $\mathcal{L}_{ij} = \{k : y_{ik} = 1, y'_{ik} \geq y'_{ij}\}$, $rank_{ij} = |k : y'_{ik} \geq y'_{ij}|$, $|\cdot|$ computes the cardinality of the set and $||\cdot||_0$ is the ℓ_0 norm.
Label ranking average precision (LRAP) is the average over each ground truth label assigned to each sample, of the ratio of true vs. total labels with lower score. The goal is to give better rank to the label associated to each sample.
Label ranking average precision (LRAP) averages over the samples the answer to the following question: for each ground truth label, what fraction of higher-ranked labels were true labels? This performance measure will be higher if you are able to give better rank to the labels associated with each sample. The obtained score is always strictly greater than 0, and the best value is 1. If there is exactly one relevant label per sample (Which is our case here), label ranking average precision is equivalent to the mean reciprocal rank (MRR[2]).
In order to prepare your submission file, you need to create a `csv` file with an `ID` column representing the ID of each text sample (from 0 to 3300 in the same order of the given test_text file). And, 3301 columns representing the graph cid (numbered from 0 to 3300 in the same order of the given test_cid file). Where, each value at row `i` and column `j` represents the score of the graph `j` being the corresponding graph of the textual description `i` (can either be probability estimates of the positive class, confidence values, or non-thresholded measure of decisions). For example, in the provided source code, using the text encoder and the graph encoder to get the text representations (text_embeddings) and the graph representations (graph_embeddings) respectively, we can prepare the submission file in the following way:

```
from sklearn.metrics.pairwise import cosine_similarity

similarity = cosine_similarity(text_embeddings, graph_embeddings)
solution = pd.DataFrame(similarity)
solution['ID'] = solution.index
solution = solution[['ID'] + [col for col in solution.columns if col!='ID']]
solution.to_csv('submission.csv', index=False)
```

You can use a scoring system different than the cosine similarity.

4 Provided Source Code

You are given Python scripts that will help you get started with the challenge. The `main.py` script uses `DistilBERT` as a text encoder and `GCN` as a graph encoder (`Model.py`). The model is trained for five epochs by minimising the contrastive loss between the graph representation and the text representation. Then, the model with the best performance on the validation dataset is used to get the encodings of the text and the molecules in the test dataset. Finally, for each description we will compute the cosine similarity with all the molecules to get our solution.
The `dataloader.py` script contains three torch Dataset classes:

- `GraphTextDataset`: For each pair of (cid, description) in the `train` and `valid` datasets, it uses the `cid` to get the graph details (edge index and nodes) from `./data/raw/cid.graph` then it assign a token embedding from `token_embedding_dict.npy` for each node. Finally, it uses the tokenizer of `DistilBERT` to tokenize the textual description of the molecules.

[2]https://en.wikipedia.org/wiki/Mean_reciprocal_rank

- `GraphDataset`: It uses the `cid` from a lisr of cids to get the graph details (edge index and nodes) from `./data/raw/cid.graph` then it assign a token embedding from `token_embedding_dict.npy` for each node. This dataset class is used with the graphs in the test set since we do not have the corresponding description.
- `TextDataset`: It uses the tokenizer of `DistilBERT` to tokenize the textual description of the molecules. This dataset class is used with the descriptions in the test set since we do not have the corresponding graphs.

As part of this challenge, you are asked to write your own code and build your own models.

5 Useful Python Libraries

In this section, we briefly discuss some tools that can be useful in the challenge and you are encouraged to use.

- A very powerful machine learning library in Python: `scikit-learn`[3].
- A very popular deep learning library in Python is `PyTorch`[4]. The library provides a simple and user-friendly interface to build and train deep learning models.
- Since you will also deal with textual data, the Natural Language Toolkit (`NLTK`)[5] of Python can also be found useful.
- `Gensim`[6] is a Python library for unsupervised topic modeling and natural language processing, using modern statistical machine learning. The library provides all the necessary tools for learning word and document embeddings.
- Since you will deal with data represented as a graph, the use of a library for managing and analyzing graphs may be proven important. An example of such a library is the NetworkX3 library of Python that will allow you to create, manipulate and study the structure and several other features of a graph.
- `PyG`[7] (PyTorch Geometric) is a library built upon PyTorch to easily write and train Graph Neural Networks (GNNs) for a wide range of applications related to structured data.
- `Hugging Face's transformers`[8] an immensely popular Python library providing pretrained models that are extraordinarily useful for a variety of natural language processing (NLP) tasks.

6 Rules and Details about the Submission of the Project

Rules. The following rules apply to this challenge: (i) one account is allowed per participant (ii) there is a limit in the size of each team (at most 3 members), (iii) privately sharing code outside of teams is not permitted, (iv) there is a limit in the number of submissions per day (at most 4 entries per day), (v) the use of external data and code is **not allowed** (except from word embeddings, e.g. BERT, GPT, BART, WordVec, etc..). For instance, you are not allowed to use external data to determine if a summary is generated by a machine or written by a human. (vi) your code must be **reproducible**.

[3]`http://scikit-learn.org/`
[4]`https://pytorch.org/`
[5]`http://www.nltk.org/`
[6]`https://radimrehurek.com/gensim/`
[7]`https://pytorch-geometric.readthedocs.io/en/latest/`
[8]`https://huggingface.co/`

Evaluation and Submission. Each team must fill this form before **21/01/2024**.

Your final evaluation for the project will be based on (1) the presentation you will give (**50**%), (2) on your position on the private leader-board and the accuracy that will be achieved (**20**%), and (3) on your total approach to the problem and the quality of the report (**30**%). As part of the project, you have to submit the following:

- A 4-5 pages report (excluding references), in which you should describe the approach and the methods that you used in the project. Since this is a real classification task, we are interested to know how you dealt with each part of the pipeline, e.g., how you created your representation, which features did you use, which classification algorithms did you use and why, the performance of your methods (loss, accuracy and training time), approaches that finally didn't work but are interesting, and in general, whatever you think that is interesting to report.
- A directory with the code of your implementation (not the data, just the code).
- Create a `.zip` file containing the code and the report and submit it here. Make sure that the name of the file is as follows: `team name.zip`
- **Deadline (for both competition and submitting code and report): 04/02/2024 23:59**

Presentation: As mentioned above, you will be asked to present the approach you followed. Therefore, you will need to prepare some slides (using ppt or any other tool you like).
Date of presentation: We will publish the presentation date after the team submission deadline