ReadMe

ILS Data Project

The data

The raw data is sourced from:

- ATOMATIC1111/Stable Diffusion WebUI
- Succinctly/Midjourney Prompts

See the project in Google Colab

- Colab Repository ILS Project 2023 - Amanjyoti Mridha

All data can be found in the data directory. This directory has the following structure:

```
data/
|
├──processed/
|   |
|   ├──embeddings.pd
|   |
|   ├──flavors_combine_redundant.json
|   |
|   ├──graph.csv
|   |
|   └──node_ids.json
|
└──raw/
    |
    ├──artists.txt
    |
    ├──flavors.txt
    |
    ├──mediums.txt
    |
    ├──movements.txt
    |
    └──prompt data/
        |
        ├──data/
        |   |
        |   ├──test-00000-of-00001.parquet
        |   |
        |   ├──train-00000-of-00001.parquet
        |   |
        |   └──validation-00000-of-00001.parquet
        |
        ├──dataset_infos.json
        |
        ├──gitattributes
        |
        └──README.md
```

Here is a breakdown of the files and directories:

- data (directory containint data)
  - processed (directory containing processed network graph data)
    - embeddings.pd (pytorch dump file containing dictionary of the text embeddings for all the data from artists, flavors_combine_redundant, mediums, and movements. Can be opened using torch.load)
    - flavors_combine_redundant.json (file containing a json structure where each node has all the flavors that are very similar combined)
  - raw (directory containing raw prompt and dictionary data)
    - artists.txt (file containing a list of artists)
    - flavors.txt (file containing various prompt modifiers)
    - mediums.txt (file containing various artistic mediums)
    - movements.txt (file containing historical artistic movements)
    - prompt data (directory containing cloned midjourney prompt data)
      - dataset_infos.json (file with info about the data structure from original authors)
      - gitattributes (git log file from original repo)
      - README.md (readme about the prompt dataset from original author)
      - data (directory containing prompt data in train, test, val split, each as a parquet file)
        - test-00000-of-00001.parquet (contains a split of 12.3k prompts)
        - train-00000-of-00001.parquet (contains a split of 222k prompts)
        - validation-00000-of-00001.parquet (contains a split of 12.3k prompts)

Important information on data structure:

txt files:
Every line contains the value (a string) representing the node

parquet files:
These are from the midjourney prompt dataset and contain tabular data. There is a column called "text" and each row in this column contains a prompt. There is a train, test, validation split for the data and I use the train split.

flavors_combine_redundant.json:
This file has the following structure:

```
{
    "name" : "flavors_combine_redundant.json", // this is the name of the file
    "nodes" : [ // this is a list with the nodes
        { // each node is a dictionary containing the values which were deemed redundant and combined
            "n1" : "value 1", // n1 is the first redundant value and so forth
            "n2" : "redundant value 2",
            "n3" : "redundant value 3"
                        ...
        },
        {
                        "n1" : "value 1"
        },
        ...
    ]
}
```

embeddings.pd

> ***NOTE:*** This file is too large to upload to github so if you clone this repo from there, you will have to run `python create_embeddings.py` in order to generate this dump file

This is a serialized dump file made with torch.save that contains a dictionary of text embeddings. This dictionary, once loaded using torch.load("embeddings.pd"), has the following structure:

```
{
        "artists.txt" : [ // a list with the nodes
                (name, embedding), // every node is a tuple containing the name (the string value) as well as the embedding (a torch tensor)
                (name, embedding),
                // ...
        ],
        "mediums.txt" : [],
        "movements.txt" : [],
        "flavors" : []
}
```

node_ids.json
This is a json file that contains the id for the node in the graph as well as a set of dictionaries for which id's correspond to which categories. This file has the following structure:

```
{
    "name": "node_ids.json",
    "ids": {
            "0" : "node name" // this dict contains the node id and the corresponding node text
    },
    "cateogries": {
        "artists.txt": [0, 1, ...,], // this contains the list of ids which correspond to this category
        "mediums.txt": [...,],
        "movements.txt": [...,],
        "flavors": [...,]
    }
}
```

graph.csv
This is a csv file the contains the adjacency matrix for the network graph (each cell represents an edge between the node represented by the column index and the node represented by the row index). This can be loaded to a numpy array using np.loadtxt("graph.csv", delimiter=","). You can use other programs to edit this (it is just a csv) but because of how large the graph is (~1.3+ GB) it is better to programmatically do it. I provide a script called `network_graph.py` that opens this adjacency matrix and displays a sample of the nodes with the highest connections.

Code is provided to parse the json and pd files, and in case they no longer function, the scripts used to generate them are also provided so support can still exist for newer python or torch versions in case any major changes occur.

How to run the code:

1. It is recommended to make either a conda or python venv if running locally.
2. Having CUDA installed means that GPU acceleration can greatly speed up the computation. If you have CUDA (11.8), run `pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118` to install torch with cuda support.
3. Install the other requirements:

> `pip install -r requirements.txt`

3. To run the network graph:

> `python network_graph.py`

Here is a breakdown of the different scripts:

```
./
|
├──combine_nodes.py # this script combines entries in the flavors dataset to reduce the number of redundant
nodes (creates flavors_combine_redundant.json)
|
├──construct_graph.py # this script creates the bidirectional graph using the prompts to create edges between
the nodes and then saves the adjacency matrix representing the graph
|
├──convert_parquet_to_csv.py # this script takes the parquet file name as a command line argument and stores
the data to a csv with the same name
|
├──create_embeddings.py # this script generates the embeddings file (creates embeddings.pd)
|
├──directory_tree.py # this script generates and prints the directory trees (you can see the result in tree.md)
|
├──network_graph.py # this script actually generates the network graph from the node and edges
|
└──utils.py # this python file contains utility/helper functions
```

> To run `convert_parquet_to_csv.py` provide the parquet file name as a command line argument when running the program
>
> For example: `python convert_parquet_to_csv.py "data\raw\prompt data\data\train-00000-of-00001.parquet"`