Course Project Documentation

Topic

The project topic fits the *Free Topics* theme and is the following: a recommendation system for open source (OS) projects based on OS projects a user has liked or contributed to.

Overview

In this project, through a command-line interface (CLI), one is able to get recommendations for projects on GitHub based on projects they like. The input to this recommender system can be one of three types: the user's GitHub profile, manual entry of any GitHub repository in the form *githubUser/repositoryName*, or manual entry of any repository in the original dataset. The dataset this system uses is from Kaggle which contains the top 980 starred repositories at the time, with the following features: *username* (repository owner), *repository name* (name of the repository), *description* (repository description), *last update date* (when the repository was last updated), *language* (the dominant programming language used), *number of stars* (repository popularity), *tags* (repository tags), and *url* (the link to the repository). With this tool, the user can find projects relevant to their interests or their given input for any purpose, e.g. learning from other approaches to a problem space or simply looking for a project to contribute to.

Implementation

The recommender system for OS projects is implemented as a collection of several components:

- Data processing (`data_processing.py`)
- GitHub API queries (`queries.py`)
- Model evaluation (`evaluation.py`)
- Recommendation model (`model.py`)
- CLI interface / core (`main.py`)

Data processing and Queries

These components are responsible for querying the GitHub API to fetch additional information about the repositories in the original dataset, as well as new data that the user may supply (e.g. a user-liked project not in the dataset as it is not popular/old enough). The additional property fetched for a repository is the text data from its readme file if one exists. A project readme is expected to contain a summary of the project and may capture some details the short description and tags from the existing columns may not, which is why it was retrieved - to improve model accuracy. The GitHub API follows both REST and GraphQL protocols; in this project, GraphQL

is used where the relevant queries are stored in `queries.py`. GraphQL has a rich and declarative syntax allowing one to define the entirety of the desired data and its structure, as opposed to the 'legacy' REST protocol which would require more data processing to get the desired shape of data. Once all the data from the user, if any, is fetched and existing data enriched all the data is then processed and cleaned to make the model generation straightforward. This final step involves concatenating all the useful feature columns (`description`, `tags`, `readme`, `language`) to be fed into the model into one column and rows where the computed column was empty to be removed. Below are the main functions implemented in this component:

- `import_user()`
    - Input: GitHub username
    - Output: detailed data for each repository name the user has starred
- `fetch_repos()`
    - Input: list of repositories to fetch
    - Output: detailed data for each given repository name
- `enrich_data()`
    - Input: locally stored raw dataset file
    - Output: new dataset file containing new features
- `prepare_data()`
    - Input: locally stored raw dataset file
    - Output: cleaned and enriched data including original dataset and any provided by the user

Evaluation

Implicit feedback is used to evaluate the model performance. Due to the project interface being restricted to a CLI, a user must "simulate" clicks by entering which recommendations they would normally click on by entering the recommendations that interest them. The entered values are fed to a tiny model that performs a simple calculation to output the utility of the model. Below are the main functions implemented in this component:

- `evaluate()`
    - Input: indices of the recommended observations in the dataset
    - Output: utility of the recommendations, after taking user input

Model

The concept the recommendation system implements is content-based filtering, where the input is a set of items (repositories) a user likes and the set of items most similar to those are the output. This is achieved with the use of `sklearn` python library, where term-frequency inverse-document-frequency (TF-IDF) generator for each document / observation / repository is first constructed. Common stop-words in the English language are removed in the TF-IDF

construction step to reduce noise/variance. Then, the feature that contains all the relevant repository details is extracted from the data and fed into this *vectorizer*. In content-based filtering, a similarity measure is needed for which cosine similarity using the TF-IDF vector for each row was used. At this point, the *model* is complete and now the user input is read in and fed to the model; for each input a similarity score is computed against the repository corpus, where the top 10 ranking documents/repositories are kept stored. Finally, the model takes the set of candidate recommendations (proportional to the size of the input) and returns the top 10 ranking repositories among them as the final recommendations presented to the user. Below are the main functions implemented in this component:

- `recommend()`
    - Input: list of names of the user's liked repositories
    - Output: list of indices corresponding to recommended repositories

CLI / Core

The core module mainly provides the CLI interface the user interacts with and invokes the corresponding other modules for the relevant tasks. It first prompts the user for how they would like to specify the set of repositories that interest them which invokes the data processing module. Afterwards, the data is fetched, enriched, cleaned (all by the data module) and finally forwarded to the model module which outputs predictions/recommendations. Finally, the model output is piped to the evaluation model which the user interacts with to determine the utility of the model. Below are the main functions referenced in this component:

- `input()`: gets different types of input from the user
- `import_user()`: imports a user's GitHub profile
- `fetch_repos()`: fetches of a user's list of repositories they like
- `recommend()`: recommends repositories to the user
- `evaluate()`: determines the utility of the recommendations

Usage

The system is fairly simple to use via any command-line shell, such as iTerm or Terminal on OSX. It is required that Python3 is installed on the system as well as its package management system, pip3. Then, the following steps are necessary to run the project:

1) Clone the project repository with `git`:
   `git clone git@github.com:arhamahmed/CourseProject.git`
2) Install library dependencies with `pip3`:
   `pip3 install -r requirements.yaml`
3) Generate a GitHub personal access token:

   a) Login into your account at github.com
   b) Go to settings
   c) Go to “Developer settings”
   d) Go to “Personal access tokens”
   e) Click “Generate a new token”, set any expiry, and give it at least all the “repo” permissions
   f) Click “Generate token”
   g) Make sure to copy the value of this token somewhere as it won’t later be accessible/viewable
   h) Create a new file in the root of the cloned project repository called `github_pat.txt` and paste the value of your new generated token
4) Run the project with `python3`:
   `python3 main.py`
5) Follow the prompts. You will be prompted to choose how to select OS projects:
   a) Via importing of repositories that were starred on your GitHub account given your GitHub username, e.g. arhamahmed
   b) Manual entry of repository names in the format *owner/repoName*, e.g. `arhamahmed/CourseProject`
   c) Manual entry of indices corresponding to rows in the dataset by directly visiting the dataset source on Kaggle: https://www.kaggle.com/chasewillden/topstarredopensourceprojects. A sample would be: 3 5 7 (project 4, 6, and 8 in the dataset).
6) Once you have selected a method of entry for your liked projects and entered them, there will be 10 recommendations printed out for you. Since this is a CLI and your terminal may or may not have the ability to click on links, manually enter a space separated list of the recommendation numbers you would have clicked or interested you.
7) Enjoy and look up the recommended projects!

Contributions

The entire project was implemented completely by me, with a little reference to lecture material for relevant concepts.