Project Final Report

Theme: Free Topics
Focus: Topic Mining
Title: Team Projects Topic Mining and Browsing
Team Name: Topic Thunder
Members:

- Creon Creonopoulos (team captain): creonc2@illinois.edu
- Suhas Ashok Bhat: suhasb2@illinois.edu
- Kurt Tuohy: ktuohy@illinois.edu

Table of Contents

Project Overview

The Team Topic Thunder project is designed to simplify browsing through past and present CS 410 projects. Rather than clicking each project and reading the documentation to learn what it covers, the web app displays keywords from the top topic associated with each project. Users can filter the topics and projects by these keywords or tags.

This project was designed with the intention that the scripts/application can be re-used by interested staff and students year over year. The scripts in python/collect_data are written such that they can be run to pull in the latest data.

1. We initially extracted only last year's projects. When this year's forks began to get created, we simply re-ran our code and scraped the new project's content. The scraper script loops through each available forked repository, checks for textual files, and collates their content by appending the strings. This means we concatenate the content of the 'project proposal', 'project progress report' and 'project final report' and create a sufficiently large document per project.
2. The next step is to clean the data to get rid of stop words, punctuation, digits etc. We have mentioned this in detail in the project overview and code comments.
3. Once the data is tokenized, we perform topic mining/modelling using the Gensim library. The topics generated are then assigned to the document (project's content) for which it has the highest probability match. The top 10 weighted words of this topic become the word tags associated with this project. These word tags are then placed in a JSON structure that can be stored on firebase.
4. Data uploaded to our database using a cloud function hosted on firebase that accepts our script's final output and is directly fed into the front-end web app which auto-updates with every change that happens in the database. The Web App is written in REACT and is built and deployed on Google's Firebase Hosting Platform.

We originally considered text categorization to classify projects by theme, but we chose topic modeling because CS 410 themes have changed over time and will continue to evolve. For example, many 2020 projects focus on reproducing the results of research papers, which is no longer a theme in 2021. Topic modeling is more flexible and future-proof than categorization, if less predictable.

The project uses 3 kinds of technologies:

- Python 3.8
    - Dependencies: beautifulsoup4, datetime, docx2txt, pymupdf, GitPython, markdown, pandas, python-pptx, unidecode,
      regex, nltk, gensim, pyLDAvis, matplotlib, requests, python-dotenv
- Google Cloud Platform (Firebase)
- React

Project Structure and Overview of Code

The project's git repository is split into 4 main folders python, topic-files, web and firebase-functions

1. python

This folder contains all the python scripts related to data collection, data cleaning, topic modeling and updating of the database. It is made up of the **sub folders** below:

a.collect_data

This folder contains scripts which harvest text from document files of past CS 410 projects. To harvest the text, first run **clone_forks.py** as below to clone forks of the base CS 410 project repository. Then run **get_project_text.py** to collect text from the clones.

Requirements
Git must be installed for clone_forks.py to work.

Scripts

clone_forks.py
Takes a URL of GitHub forks from the main CS 410 project and returns a CSV of information and links to each project. Optionally gets shallow clones of projects that were last pushed more than n months ago.

Arguments
Call clone_forks.py with no arguments to use default settings.
--forksurl: URL for GitHub API to retrieve desired forks.
--destination: local top-level folder to clone to.
--doclone: whether to perform cloning step. "yes" to clone, "no" to simply collect info about the forks.
--minmonthsold: only clone forks that were last pushed at least this number of months ago.

Functions
find_forks - Query the GitHub API for all forks of a repository.
github_resp_next_page - Check to see if the GitHub response has a next link. If the response, look for the 'link' header and see if there is a value pointed to by next.
min_months_old_met - Determine whether two dates are at least min_months_old months apart. Assume date_2 is greater than date_1.
shallow_clone_forks - Given a dataframe of git repo forks, clone all forks that are at least min_months_old. Clone them to the "destination" folder. Use depth = 1 for a shallow clone.

Output
When **--doclone** is "yes", shallow clones of GitHub repos more than **--minmonthsold** months old are saved to a folder called repo_clones on the same level as the script.
repo_forks.csv: file of information about each fork. Will be used in the following step to help harvest text.

Usage
```
python ./clone_forks.py --
forksurl "https://api.github.com/repos/CS410Assignments/CoursePr
oject/forks" --destination "./repo_forks" --doclone "yes" --
minmonthsold 0
```

get_project_text.py
Given a repo_forks.csv file (see above) and a directory of cloned repositories, harvests text from document files at the top level of each repository. Processes PDFs, Word documents, Power Point documents, and markdown files. The script converts sequences of whitespace characters into single space characters. The script also converts non-ASCII Unicode characters to their closest ASCII equivalents.

Arguments
Call get_project_text.py with no arguments to use default settings.
--projectlist: CSV file containing information about the cloned CS 410 projects. Helps associate project URLs with harvested text.
--projectroot: folder containing cloned CS 410 projects. Walks through the top level of each project directory and harvests textual documents.
--outputdir: Destination directory of output files.

Functions
get_pdf_text - Return text contents of the given PDF. Return as string.
get_docx_text - Return text of the given Word document. Return as string.
get_pptx_text - Return text of the given Power Point document. Return as string.
get_md_text - Return flattened string consisting of the contents of the given markdown file (or any text file).
normalize_text - Given a string, remove or replace non-ASCII characters, normalize spaces, and make text a single line.
process_project_files - Given a project folder, identify textual files and pull contents. Return the name of each file, plus the text as a single line. Look in top level of directory only.
process_project_list - Iterate through dataframe of info about GitHub projects. Look for textual files in each project, read the text, and log it.

Output
project_file_text.tsv: tab-delimited file of project URLs, filenames, and the text harvested from those files. One row per project file.
project_text.tsv: same as above, but with a single line per project. Concatenates text from each project's files.

Usage
```
python ./get_project_text.py --projectlist "./repo_forks.csv" --
projectroot "./repo_forks" --outputdir "."
```

b.text_mining

The scripts present here perform data cleaning on the web scraped data made available in the collect_data folder and then proceed to perform LDA Topic Mining and analysis.

Scripts

text_cleaning.py
Takes the scraped content from the forked github projects that are stored in python/collect_data/project_text.tsv and performs a series of data cleaning steps on it. This script makes all the content lower case, removes punctuation and digits, tokenizes, lemmatizes and removes high frequency words which are not considered as high value words for tag creation.

Usage
```
python ./text_cleaning.py
```

Output

project_clean_text.tsv: tab-delimited file of project URLs, filenames, text harvested from these files and list of tokens produced from the text. One row per project file.

LDA.py
Takes the tokens generated by the above script and runs the LDA algorithm on it to produce several topics. Each topic is then matched against a CS410 project based on the highest probability match. The number of topics are decided after a thorough study on the LDA model, this study is done in python/text_mining/topic_model_eval.ipynb.

Usage
```
python ./LDA.py
```

Output

AlgOutput.tsv**:** tab-delimited file of project URLs, filenames, text harvested from those files, list of tokens produced from the text and list of words/tags identified from the highest probability topic. One row per project file.
firebase-output.json**:** JSON output that is used by web api and firebase to create the front end.

topic_model_eval.ipynb
This is a Jupyter notebook that provides a guided walkthrough of the script in LDA.py and follows up with a detailed study on which would be the best fit number of topics based on Coherence and perplexity. It also uses the pyLDAvis library to help with this study.
There is an extensive walkthrough of our model evaluation in the project video submission.

c.firebase

The script here is the last step to update the WebApp with the output found in python/text_mining/firebase-output.json. You should not need to run this script to test the functionality of the Web App, but you certainly can. *(See note on api-key authentication below)

Important Note
For security reasons (so that not everyone can write to our database), you will also need to create a file called .env in the same folder as this README, which will contain the firebase api key to update the database. If you want to test our database upload system, and you are a project reviewer, please look at the CMT submission under the "Abstract" section for the API KEY. Otherwise, **please reach out to Creon Creonopoulos (creonc2@illinois.edu) and we will be more than happy to share the key with you.**

Scripts

uploadToFirebase.py
Reads the content of the file **firebase-output.json** and uploads it the Firebase Real-Time Database instance tied to our team. Once there, our WebApp is coded to listen to any changes to the database and re-render with the updated information.

Output
Database updated: Success Message

2. topic-files

We have placed our files that we used in the manual evaluation process in this folder.

- **Ground_Truth_Topics.xlsx -** This document attempts to list "ground truth" topics for selected CS 410 projects.

- **Manual_Topic_Evaluation.xlsx -** This document takes discovered topics and tags for CS 410 projects and compares them to the "ground truth" topics for matching example projects. Team members have rated the relevancy of the match between the actual content of the projects versus the tags that have been allotted to them. We have done this exercise on both the 8 topics and 16 topics to get a feel of which we can use for the front end.

- **Tag_Friendliness_8_Topics_and_16_Topics.xlsx -** This document was used as a voting mechanism between teammates to decide the usefulness of tags, we have dropped certain tags from the front end based on the result.

3. web

This project is using React for the frontend and the website is hosted on Firebase. No further changes should be needed even if the data source changes. It is setup to dynamically listen to changes on the Firebase Real-Time Database instance setup for this project.

Scripts

yarn start
Runs the app in development mode.
Open http://localhost:3000 to view it in the browser. The page will be reloaded if you make edits. You will also see any lint errors in the console.

yarn build
Builds the app for production to the build folder. It correctly bundles React in production mode and optimizes the build for the best performance.
See the section about deployment in the **firebase-functions** below, for more information on deploying updates to the website.

4. firebase-functions

This folder contains the code of a cloud function created to update the real-time database on firebase. It is already hosted on firebase as a firebase function and it can be called as a REST-full API call to pass the topic mining output data to the database, to feed the Web App. This cloud function is called by the python script uploadToFirebase.py, to update the database with the latest output from the topic mining algorithm.

Deployment
To deploy changes to the website UI or the cloud function, **please contact Creon Creonopoulos (creonc2@illinois.edu) for access to the firebase console tied to this project.**

Output Evaluation

Evaluation was done at different checkpoints in the project.

During text mining we use the **pyLDAvis** library to scan through the topics generated which gave us valuable feedback that we were on the right track. We also plotted topic coherence and perplexity values to evaluate whether we need to use more parameters to fine tune the model. Finally, once the topics were mined and tagged with each project, we performed extensive self-evaluation by creating ground-truth documents.

First, **Ground_Truth_Topics.xlsx** was built to serve as a reference for us. We determined the focus of sample CS 410 projects and listed our interpretation of their topics.

We later realized that LDA-generated topics may need not be an exact word to word match to the content of a project's documentation. This is because the model generates topics in a certain probability and words in each topic have certain weights.

This led to creation of **Manual_Topic_Evaluation.xlsx.** For each generated topic, we chose 2 random projects for which the topic had greatest coverage. We determined the "ground truth" tags and topics for these projects, then rated their fit to the generated topics.

Each team member validated the projects from their perspective and added a relevancy score for the word tags generated. A score of 5 means that highly relevant tags have been associated with the project while a score of 1 implies poor relevancy. The mean value over all the teammates' scores is the final relevancy score of the project.

By having multiple checkpoints and distributing the evaluation across all team members we believe we have the expected outcome. Many projects received scores in the range of higher relevancy, i.e., 3-5.

This scoring was done for both an original 10-topic model and a later 16-topic model. Considering that the corpus of documents is slightly skewed towards topics around 'twitter sarcasm detection', we chose the 16-topic model as it was bringing up more useful words which we saw as good potential to use in the front end as filter tags.

We did expect LDA-generated topics to be more coherent in terms of subject matter. One likely reason is the CS 410 project documentation is noisy. Along with the high-level subject of each project, the documentation includes boilerplate language for the questions the documentation addresses, low-level implementation details, and installation instructions. Below we list potential ways to address this.

To address noisy topics, we filtered non-reader-friendly words both out of the original text and out of the final topic tags. However, some tags are still indirect or poorly match the projects they label.

Future Improvements

One improvement would be to scrape CS 410 project text directly from the web, rather than from local clones of Git repositories.

All the models we generated included topics that seemed to cover multiple subtopics, such as "topic modelling" plus "text classification". Many CS 410 projects did not seem to fit the highest-coverage topics, either.

To generate more coherent topics that relate more strongly to their documents, we could try several approaches:

- Perform better-focused text collection. Each project's text includes a lot of noise for topic-mining purposes, such as team member names and installation instructions. The collection process could identify the Overview or Abstract sections of the documentation for more relevant text.
- Generate models with higher topic counts. This may separate the omnibus topics into their component subtopics. Such models may have better perplexity scores.
- Use a different model type or implementation. We chose Gensim's implementation of LDA because it is commonly used and well-documented. The Mallet implementation of LDA may generate models with higher coherence scores, though.
- Use a background topic to compartmentalize terms that are common across all CS 410 projects, such as "project" and "report". LDA supports specification of prior word distributions, but we were unable to find examples or documentation for how to code these. Instead, we approximated the effect of a background topic by making a list of high-usage words and filtering them out in the preprocessing stage.

This project can also be extended to mine and browse topics of other collections related to the class. For example, to get and display grouped lists of the most popular topics of submitted Tech Reviews.

Team Contributions

Kurt: Data aggregation, Output Evaluation
Suhas/Kurt: Text cleaning and topic mining scripts, Output Evaluation
Creon: Frontend build and deploy, general admin stuff, Output Evaluation

Links to Deliverables

Project GitHub URL: https://github.com/kreonjr/CourseProject
Web App: https://topic-thunder-a7103.web.app/
Submission Video: https://youtu.be/6lFScAFhaHc