Text Information Systems : CS410

Project Documentation

"2.2 ExpertSearch System"

--

Navyaa Sanan (navyaas2)
Srivardhan Sajja (sajja3)

Team

- **Navyaa Sanan** (navyaas2)
- **Srivardhan Sajja** (sajja3) : Team Captain

Overview

In our project, we were able to augment the ExpertSearch system by adding functionality which makes the system automatically crawl through faculty webpages given the primary university link/URL (illinois.edu, berkeley.edu, etc.), instead of having to explicitly identify them.

Our project has two main components. First, we implemented our own classifier, which given any URL uses text classification techniques mentioned in this course to judge whether the given URL is that of a faculty directory page. Second, given a primary university link we find all directory pages associated with that primary URL. We used the classifier built in part 1 for implementing part 2.

Software Implementation

Datasets: To train the extension, we used 800 manually labelled URLs. To test the extension, we used another 800 manually labelled URLs to check for accuracy. We were able to achieve 83.875% Accuracy, 89.28% precision, 77% recall and 82.68% F1-score.

Algorithms/Techniques: To get the list of all URLs from a university website, we used spider from scrapy package. We implemented a spider using the python scrapy package to recursively crawl through and identify all pages of a website with either ‘faculty’ or ‘staff’ in the URL, given the primary URL of the university (example: illinois.edu, berkeley.edu). Parameters such as time limit, page crawl limit, and results count limit can be set manually. To make the classifier, we used classification techniques like using stop words and filter words. We also used statistical indicators like mean length of URLs and standard deviation of the training set URLs. Lastly, we used a dictionary to look at the most common words in positive training samples to help us build a classification model.

Additionally, we would like mention that our extension is independent of the system. We consider this project to be an independent feature addition, which is inspired by the ExpertSearch System but does not directly rely on any preexisting code. We drew inspiration from MP2 and used the techniques taught to us in this course, but we did not have any direct reliance on any preexisting code whatsoever (aside from external Python packages).

Software Implementation Details

Crawler: To get the list of all URLs from a university website, we used spider from scrapy package. We implemented a spider using the python scrapy package to recursively crawl through and identify all pages of a website with either 'faculty' or 'staff' in the URL, given the primary URL of the university (example: illinois.edu, berkeley.edu). Parameters such as time limit, page crawl limit, and results count limit for the crawler can be set manually.

Model Training: To train the model, we looked at used classification techniques like using stop words and filter words. We also used statistical indicators like mean length of URLs and standard deviation of the training set URLs. Lastly, we used a dictionary to look at the most common words in positive training samples to help us build a classification model. We have a list of words we ignore in model_train/crawler.py. Any URL that contains a word also contained in ignore is automatically pulled out of consideration. The rest of the training is done in model_train/trainer.py and model_train.py where we calculate the statistical indicators and return those URLs which have the same words contained as the most popular words from the positively labelled training data. To train the extension, we used 800 manually labelled URLs. To test the extension, we used another 800 manually labelled URLs to check for accuracy. We were able to achieve 83.875% Accuracy, 89.28% precision, 77% recall and 82.68% F1-score.

Model Deploying: To deploy the model, most of the work is done in model_deploy /model_dep.py. There we use the model generated by us (model_deploy/ model_testing.json) and carry out the same classification process as we did to test our classification model.

Flask App: The Flask app refers mostly to the frontend work we did. It connects the deployed model to the website we made and provides a framework for piping input and output.

Installation and setup

Clone repository on your local machine and enter the project directory:

```
- git clone https://github.com/srivardhansajja/CourseProject.git
- cd CourseProject/
```

Setup a virtual environment. Executing the following two lines in a terminal will set up an environment using venv within the project directory.

```
- python 3 -m venv venv
- source venv/bin/activate
```

We have used Python3 for running and testing the project. Install all required packages by executing:

```
- python3 -m pip install -r requirements.txt
```

Change line 11 in crawler/crawler_handler.py to the python version that you are using. For example, `python3.8, python3, python3.6` or `py`. The project is now set up for you to use and test. Execute

- `python3 main.py`

from the primary project directory and access the locally hosted flask website by visiting `http://127.0.0.1:3000/` from a browser window. You can enter university domain names and the appropriate faculty directory URLs will be shown to you along with the crawling statistics.

The program by default uses our pre-generated model. If you wish to update the parameters or tweak the model, go to `model_train/trainer.py`, make your required changes, and run

- `python3 model_train/model_train.py`

This will output your testing statistics, including `accuracy, precision, recall` and `F1 score`, and generate `model_testing.json`. Once you are satisfied with your changes, if you wish to use your model in the crawling process instead, replace `model_testing.json` in line 50 of `model_train/model_train.py` with `model.json` and rerun the above statement in your terminal. Be careful as this will replace our original model, and re-cloning the project is the only way to revert it, unless you make a backup of it.

Project Structure

- Source Code:
  - Crawler
    - /crawler/crawler.py
    - /crawler/crawler_handler.py
  - Model Training
    - /model_train/trainer.py
    - /model_train/model_train.py
    - /model_train/train_data.txt
    - /model_train/dev_data.txt
  - Model Deployment
    - /model_deploy/model_dep.py
    - /model_deploy/model.json
  - Flask App
    - /main.py
    - /templates/
    - /static/
- Documentation:
  - ProjectProposal.pdf

    - ProjectProgressReport.pdf
    - ProjectDocumentation.pdf
- README.md
    - /

Team contributions

Throughout the course of this project, the team worked very closely and even though most of the tasks were divided by person both of us ended up working on everything in some capacity. Srivardhan focused more on implementing the web crawler, setting up the website, and making training/testing data sets. Navyaa focused mainly on doing research on URL classification, building the machine learning model, and integrating the classification model into the website.