Improving ExpertSearch Progress Report

Plan

Automate scraping process
- To identify faculty directory pages
- To identify faculty home pages

Perform topic mining
- To identify top-k topics associated with each faculty

Additional improvement
- To improve email extraction for each faculty

Improve UI
- To display top-5 topics associated with each retrieved faculty
- To allow search based on any of the topics from the displayed topic cloud
- To prepopulate email content when clicked on a faculty's email address

Progress

Item	Owner	Status
Automated Scraping	Mriganka Sarma	Completed: - Automated scraper - Data Handler - Scraper - Text Classifier Remaining: - Optimizing parameters for classification - Integration testing Challenges: - None
Topic Mining	Zacharia Rupp	Completed: - Topic model - Function to return top-10 words associated with query topic Remaining: - Further exploration of best topics - Clean up code

		- Integration testing Challenges: - Inferring topics takes considerable processing time
Improved Email Extraction		
Improved UI	Sai Ranganathan	Completed: - To display top 5 topics associated with each faculty member - To prepopulate email field when clicked on email address - To improve email extraction part 1. Challenges: - None

More detailed description is provided in the below sections.

Automated scraping process

Deliverables:

- Automated Scraper (auto_scraper.py)
- Data Handler (data_handler.py)
- Scraper (scraper.py)
- Text Classifier (text_classifier.py)

Outputs:

- Corpus of classified Faculty Directory URLs
- Corpus of classified Faculty Bio URLs
- Documents of bios for each faculty generated by scraping the classified Faculty Bio URLs

Challenges:

- None

Automated Scraper

Automated scraper module (auto_scraper.py) automates the process in the following way:

- Uses the data handler to prepare a train and test set of Faculty Directory URLs
- Uses the scraper to scrape these URLs to prepare the train and test corpus
- Uses the text classifier to build and train a Doc2Vec model on the documents in the train corpus of directory contents
- Uses the text classifier to predict the category of the test URLs as “Directory” or “Non-Directory”
- Saves the classified directory URLs to a file
- Uses the data handler to prepare a train and test set of Faculty Bio URLs
- Uses the scraper to scrape these URLs to prepare the train and test corpus
- Uses the text classifier to build and train a Doc2Vec model on the documents in the train corpus of faculty bios
- Uses the text classifier to predict the category of the test URLs as “Faculty” or “Non-Faculty”
- Saves the classified bio URLs to a file
- Uses the scraper to scrape the faculty bios from the classified bio URLs
- Generates one document per faculty bio and saves under ExpertSearch/data/compiled_bios

The automated scraper can be invoked as follows:

```
$ cd ExpertSearch/AutoScraper
$ python ./auto_scraper.py –d –t
```

-d option specifies to generate/regenerate the train and test dataset.
The dataset will be generated even if -d is not provided if the dataset doesn’t exist yet.
When -d is not provided, the existing dataset will be used.

-t option specifies to retrain the Doc2Vec model on the train dataset.
The model will be trained even if -t is not provided if the model wasn’t trained and saved yet.
When -t is not provided, the saved model will be loaded.

The following sections describe the text classification tasks for Faculty Directory URLs and Faculty Bio URLs.

Directory URL classification

1. Dataset preparation

First we need to prepare the dataset for training and testing the model. The following approach was used to prepare the dataset.

- Downloaded the known faculty directory pages from the sign-up sheet for MP 2.1. These will serve as the “positive” examples.
- Collected top URLs from Alexa. These will serve as the “negative” examples.
  - Collected the global top-50 pages of Alexa.
  - Collected the top-50 pages for different countries. Manually verified that the pages are in English.
- About 900 URLs were obtained from the sign-up sheet data, which was partitioned into 500 for training and 400 for test data.
- URLs for total 14 countries + top-50 global URLs from Alexa were collected. This gave 750 “negative” URLs.
- Wrote a python module (data_handler.py) for data handling that does the following:
  - Converts the MP 2.1 sign-up data from csv to a file containing only the directory URLs. Performs any cleanup as necessary and labels them as “directory”.
  - Combines the top-50 Alexa URLs for 10 countries and labels them as “alexa_dir”. Uses these 500 pages for training.
  - Combines the top-50 Alexa URLs for 5 countries and labels them as “test_dir”. Uses these 250 pages for testing.
  - Mix the 500 Faculty Directory training URLs with the 500 Alexa training URLs. Remove duplicates if any. This gives 734 URLs as the final training URLs.
  - Mix the 400 Faculty Directory test URLs with the 250 Alexa training URLs. Remove duplicates if any. This gives 548 URLs as the final test URLs.

2. Scraper

Wrote a python module (scraper.py) for scraping the URLs collected from the above step.

The scraper does the following:

- Gets the contents of each URL as text.
- Performs clean-up of non-ascii characters from the content.
- Performs other clean-ups such as substituting newlines, tabs, multiple whitespaces into single whitespace.
- Substitutes contents such as “403 Forbidden”, “404 Not found”, etc. with “Error: Content Not Found”.

- Writes contents of each webpage as a single line of space separated words in a file meant to be the final corpus.
    - This is done to prepare both the training corpus ("train_dataset.cor") and the test corpus ("test_dataset.cor").

3. Text classification

Wrote a python module (text_classifier.py) for performing the text classification task of identifying valid Faculty Directory pages from the test corpus.

The classification module does the following:

- Uses gensim to build a Doc2Vec model for feature vector representation of each document.
- Uses the train_dataset.cor to build the vocabulary and train the model.
- Saves the model so that it can be reloaded while running next time on the same dataset.
- Uses LogisticRegression as the classifier from scikit-learn module.
- Uses LogisticRegression to predict the categories of the test URLs given the test dataset.

Faculty URL classification

1. Dataset preparation

The following approach was used to prepare the dataset:

- Use the top 1000 URLs from the currently existing Faculty Bio URLs in the ExpertSearch project as the train URLs.
- Use 250 URLs from the Alexa test URLs set as the "negative" train URLs.
- Use the data handling module to do the following:
    - Tag the faculty bio URLs as "faculty" and save to a file.
    - Tag the Alexa URLs as "alexa_faculty" and save to the same file.
    - This will be the final file with all the train URLs.

2. Scraper

Since the ExpertSearch project already contains the faculty bios as documents, the top 1000 faculty bios are copied to the train corpus file ("train_bio_dataset.cor").

Then the scraper does the following:

- Scrapes the remaining train URLs from the train URLs file and appends to the train corpus ("train_bio_dataset.cor").
- Uses the classified Faculty Directory URLs from the classified Directory URLs file above and gets all embedded potential faculty bio URLs as the test URLs.

- Scrapes the test URLs from above and adds to a test corpus ("test_bio_dataset.cor").

3. Text classification

The classification module does the following:

- Uses gensim to build a Doc2Vec model for feature vector representation of each document.
- Uses the train_bio_dataset.cor to build the vocabulary and train the model.
- Saves the model so that it can be reloaded while running next time on the same dataset.
- Uses LogisticRegression as the classifier from scikit-learn module.
- Uses LogisticRegression to predict the categories of the test URLs given the test dataset.

Topic Mining

Deliverables:

- Python script to create topic model and retrieve top-10 terms associated with query topic (miner.py)

Outputs:

- Trained topic model ('lda_mallet_model')
- Bag-of-words representation of corpus to be used with miner.py ('corpus_dictionary')

Challenges:

- Inferring topics takes considerable processing time.

Topic Miner:

The topic miner uses gensim and mallet to create a model from the entire corpus. The process is as follows:

1. Corpus preparation

- Read in compiled bios as strings
- Filter the string representation of each bio to:
  - Remove stop words
  - Extract HTML tags and elements
  - Strip non-alphanumeric characters

        - Strip numbers
        - Strip words that exist in lists of terms extracted from the bios
        - Strip words that exist in a manually defined list of words that were creating incoherent topic clusters (unwanted_words.txt)
        - Remove words shorter than four characters
        - Split all words into a list of tokens
    - Create list of documents which is comprised of lists of tokens for each document as described above
    - Append bigrams and trigrams to each token list for each document
    - Create a gensim dictionary from the above documents
    - Create a bag-of-word representation of our documents: this will be our corpus.

2. Model creation

Model creation required a good deal of manual work to ensure that the term clusters were understandable. The process consisted of a lot of trial and error, using the steps below:

- Create a general model with gensim.models.ldamodel.LdaModel class with 10 models
- Visually inspect term clusters to ensure they were meaningful
- Visualize clusters with pyLDAvis to assess clusters
- If the above criteria were not satisfactory:
    a. Tweak corpus construction
- After the above criteria was deemed satisfactory:
    a. Using gensim.models.wrappers.LdaMallet with the mallet library, I:
        i. Varied number of topics to create new model
        ii. Assessed coherence of each model with varying number of topics
        iii. Manually inspected output of models with high coherence, looking for term clusters that made intuitive sense and that appeared distinct given knowledge of the separate domains
        iv. Chose the best model according to above criteria and saved it and the created dictionary for query inference

3. Term extraction

With a model and a dictionary, I wrote a method that can infer the topic of a given query and fetch the top-10 terms associated with that topic, a method that can infer the topic of a single document and fetch the top-10 terms associated with that document's topic, and a method that can infer the topics of multiple documents and fetch the top-10 terms associated with each document's topic. These terms will eventually be pushed to the user to help them potentially refine their query.

Improved Email Extraction

Deliverables:

- email-extraction.py

Regex Improvement:

- There are certain edge cases that we had noticed in some of the email web pages where the format was different than the traditional email formatting
- Added more regex matches in order to match with these edge cases such as ex. rohini[@]buffalo[DOT]edu

UI Improvements

Deliverables:

- Server (server.py)
- UI Front (index.js)

Challenges:

- None

Info Button:

- Information button is created at the top of each of the retrieved faculty.
- When the button is clicked there a table pops up that appears below the selected retrieved faculty
- The table will contain additional information regarding the research topics that the faculty does
- When one of the info buttons associated with a faculty is clicked the other one will close, and the new one will open.

Top 5 Topics Display:

- Display the top 5 topics from the preview for each of the faculty.
- Display these topics in a table format when the information button is clicked
- Underneath each topic is a ‘Learn More’ button which when clicked leads you to a page talking more about the topic in detail from the web.

Email Automation:

- Email comes pre-populated with a set subject and body.
- The body talks about one of the research topics that were extracted from the top 5 topics and how the user would like to connect with the faculty regarding research in this topic.