

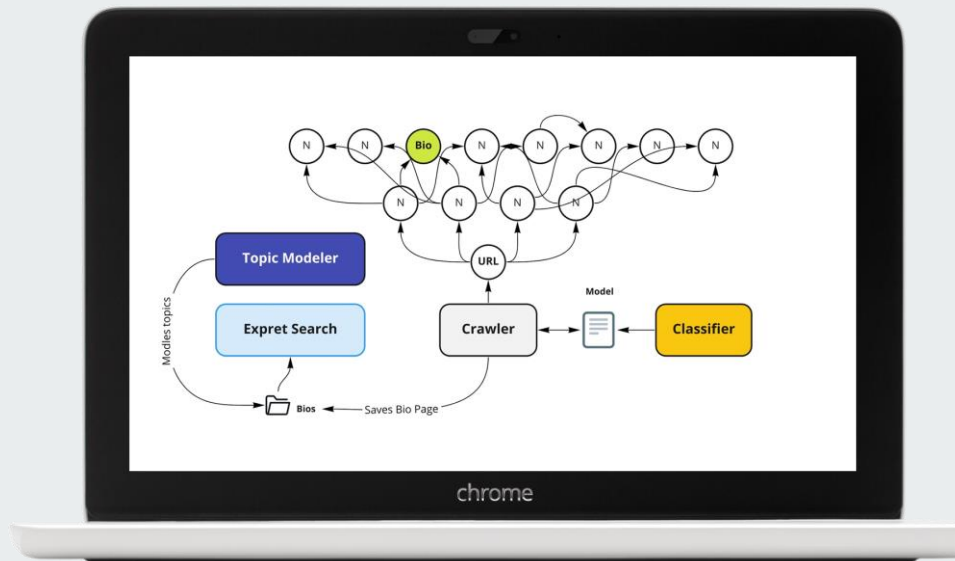
# Expert Search

Improvements to Expert Search  
System: Bio Page Classification,  
Automated Crawling, and Topic  
Modeling

Authors:

Bashir Partovi, Karthik Rajagopal, and Mohana Venkata Kalyan  
Cheerla

Fall 2020 - CS410 - Course Project



---

# Outline

Overview

Bio Classifier

Automatic Crawler

Topic Modeler

End Results

Demos



# Overview



# Requirements

- The user should be able to view topics for each search result
- University websites should be regularly crawled for automatic discovery of bio pages
- Crawler requires a classification model in order to identify faculty bio pages
- Classifier should use the provided compiled bios as positive samples and be able to generate negative samples by crawling the web

# Assumptions

- Expert Search system runs smoothly on its own
- It is easy to setup Expert Search system locally on developer's machine
- Expert Search system python runtime could be scaled to the latest version
- MeTA python library is also available in Python 3.8
- Training the classifier is not a computationally intensive task





**Solution**



# Solution description

The system consists of 3 independent components that run concurrently to provide data to Expert Search system to enhance the search results.

Bio Page Classifier utilizes compiled bios as positive training data along with a set of randomly crawled web pages as negative data to train a model to identify faculty bio pages.

The crawler uses such model to crawl universities websites in order to obtain faculty bio pages and stores such pages for Expert Search as seed data.

Topic Modeler runs on stored bio pages in order to model topics to be displayed as part of the search results



# Why it's better than the existing solution

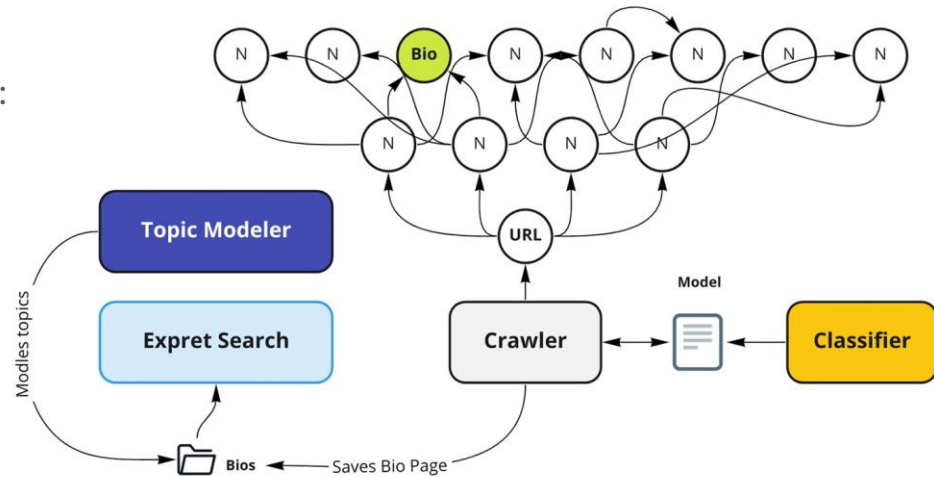
- New faculty pages are automatically discovered and indexed
- Search results include topic tags that could be used to create new search queries
- No manual intervention is needed by the administrator to prepare the bio data



# General Architecture

The system has 4 separate components:

- Expert Search
- Topic Modeler
- Bio Classifier
- Automated Crawler





# Bio Page Classifier

Creates a model to identify faculty bio pages

Bashir Partovi



# Why a Classifier?

- Crawler needs to be able to identify faculty bio pages in order to store them for Expert Search consumption
- Eliminates manual identification of web pages



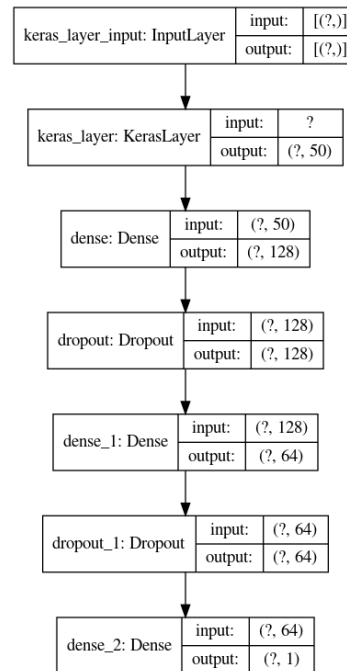
# Technology Stack

- Tensorflow and Keras were used to implement the training algorithm
- The classifier is written in Python due to ease of use of ML frameworks such as Tensorflow and Keras
- Compiled bios were used as positive examples and a new set of websites were crawled to obtain negative examples
- Multiple training algorithms were used to see whether the accuracy increases
- Reached accuracy of 99.95%

# Tensor Layers

The layers consists of

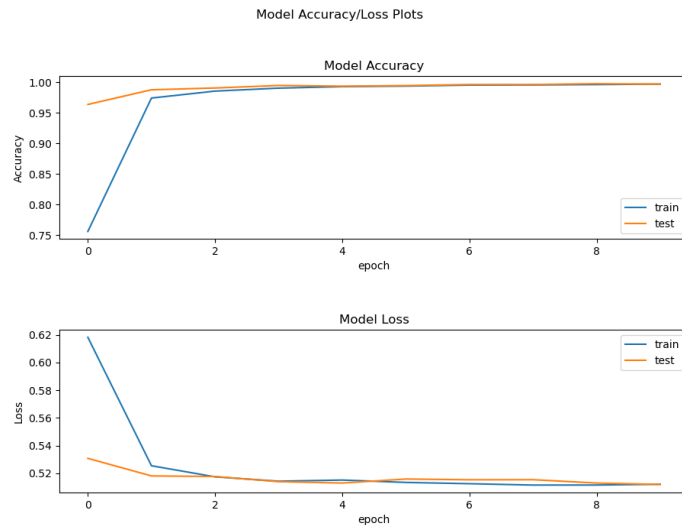
- An embedding layer (input shape)
- A dense layer with 128 units and relu activation
- A dropout layer of 0.5 ratio to avoid overfitting
- A dense layer of 64 units
- Another dropout layer of 0.5 ratio
- A dense layer of 1 unit with sigmoid activation



# Classifier in Action

How did it perform?

- Unfortunately, even though the accuracy is very high, the model overfits the data
- The accuracy spikes up to 95% on training data at the second epoch
- Better model? Yes, but it crashed due to hardware limitation





# Challenges

- Learning curve on Tensorflow and Keras was steep
- Model overfits the data
- Local hardware limitation in training the classifier (even crashed Google Colab virtual machine)



# Automated Crawler

Regularly runs to discover faculty bio pages  
by crawling universities websites

Mohana Venkata  
Kalyan Cheerla



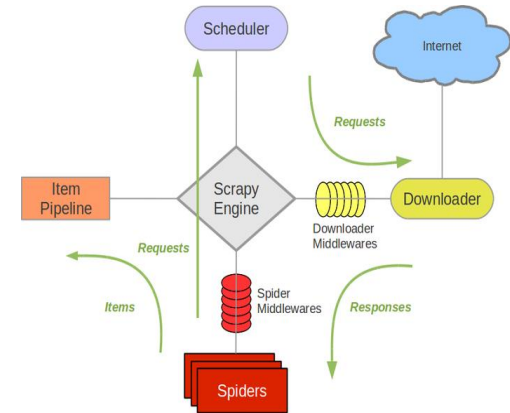


# Why an Automated Crawler?

- Manually searching the web and extracting faculty bio pages are time consuming
  - ◆ It's tedious
  - ◆ Doesn't scale
  - ◆ Navigation is difficult
- Our Automated Crawler crawls university websites to automatically fetches bio pages
- It converts the Unstructured Markup text into Semi-structured normalized text data to feed its downstream components

# Technology Stack

- The crawler runs on a cron schedule to to passively crawl universities websites
- The crawler is developed in Python. It utilizes the Scrapy library and follows Breadth First Search (BFS) up to the designated depth specified by the User/Administrator
- Crawler has Spiders that recursively follow all the web links of the University Website and extracts the web page data along with the Website's "URL Tree" to understand, validate, troubleshoot and trace



# Crawler in Action

```
C:\Windows\System32\cmd.exe - scrapy crawl faculty_pages_filtered

(py38) C:\CS410TIS\Projects\ExpertSearchCrawler>scrapy crawl faculty_pages_filtered
Err: error (200): program aborting due to control-C event
image PC Routine Line Source
libcoremd.dll 00007FFCF3613B58 Unknown Unknown Unknown
KERNELBASE.dll 00007FFD2F0762A3 Unknown Unknown Unknown
KERNEL32.dll 00007FFD2FF47C24 Unknown Unknown Unknown
ntdll.dll 00007FFD31C804D1 Unknown Unknown Unknown

(py38) C:\CS410TIS\Projects\ExpertSearchCrawler>scrapy crawl faculty_pages_filtered

Crawled 50 Bio-pages of Cornell University ...

Crawled 50 Bio-pages of Gatech University ...

Crawled 50 Bio-pages of Utxas University ...

Crawled 50 Bio-pages of Illinois University ...
```

Name	Date modified	Type	Size
00_30_brec.txt	12/15/2020 2:43 PM	Text Document	532 KB
00_100_brec.txt	12/15/2020 2:46 PM	Text Document	639 KB
00_150_brec.txt	12/15/2020 2:46 PM	Text Document	686 KB
00_200_brec.txt	12/15/2020 2:46 PM	Text Document	726 KB
00_250_brec.txt	12/15/2020 2:47 PM	Text Document	761 KB
00_300_brec.txt	12/15/2020 2:47 PM	Text Document	801 KB
00_350_brec.txt	12/15/2020 2:48 PM	Text Document	866 KB
00_400_brec.txt	12/15/2020 2:48 PM	Text Document	904 KB
00419771213c0e3da96dc5157962.txt	12/15/2020 2:46 PM	Text Document	11 KB
00a47146d1c10d718d27a62157950.txt	12/15/2020 2:47 PM	Text Document	16 KB
00a4909060854631aba0d0eb76176.txt	12/15/2020 2:40 PM	Text Document	3 KB
00a3f520ebd233261496dc0f068a0ed.txt	12/15/2020 2:42 PM	Text Document	4 KB
00a4c0e6c04a89595da2c747b13ba29f.txt	12/15/2020 2:48 PM	Text Document	4 KB
00a6c1ba4511a0d0e77a49159a7a1d.txt	12/15/2020 2:46 PM	Text Document	9 KB
00a85011978ac3482b1994a38c50b0e7.txt	12/15/2020 2:47 PM	Text Document	12 KB
00a3c48971c4a4e08977a709a20ab1f.txt	12/15/2020 2:47 PM	Text Document	8 KB
00a2803a1874a14803a0a74c03a18.txt	12/15/2020 2:46 PM	Text Document	4 KB
00a1a443196a15780733cda032960a4a.txt	12/15/2020 2:48 PM	Text Document	9 KB
00a31117f05ac15554a481c030862a.txt	12/15/2020 2:40 PM	Text Document	7 KB
00a4a9198620208a5358a8031a0ac7.txt	12/15/2020 2:39 PM	Text Document	13 KB
00a4a7a15a152a3617a0a0a71771a0.txt	12/15/2020 2:46 PM	Text Document	11 KB
00a2ba748024e4e96a74895f52ceda02.txt	12/15/2020 2:35 PM	Text Document	3 KB
00a28a1f5e41453a7a0a2a0a3c3a2.txt	12/15/2020 2:46 PM	Text Document	3 KB
00a4a52a15a4e5a0c35e08b16a051.txt	12/15/2020 2:42 PM	Text Document	4 KB
00a7521a1c3a0517a0a0a71771a0.txt	12/15/2020 2:46 PM	Text Document	4 KB
00a80a02a02a1a08b0a0237771a7.txt	12/15/2020 2:35 PM	Text Document	3 KB
00a71d1a0e4e050807b1a75c0a0a.txt	12/15/2020 2:48 PM	Text Document	9 KB
00a4161a1a71a71a0a0a0a0a0a0a.txt	12/15/2020 2:46 PM	Text Document	9 KB
00a26a7815a4a5a4a5120a0a71c0a5.txt	12/15/2020 2:47 PM	Text Document	9 KB
00a1a044a4a0a79a3a3a3a3a3a3a3a.txt	12/15/2020 2:36 PM	Text Document	9 KB
00a1a09546e4a3c291f3384435680aa2.txt	12/15/2020 2:38 PM	Text Document	17 KB
...	...	...	...

## Crawler in Action (contd...)

The image shows two side-by-side browser windows displaying lists of URLs from Stanford University. The left window shows a list of URLs, with 'https://cs.stanford.edu/directory/faculty' highlighted in red. The right window shows a list of URLs, with 'https://profiles.stanford.edu/david-karp' highlighted in green. A green arrow points from the highlighted URL in the right window to the text 'Faculty Bio Page URLs'.



# Challenges

- Crawler runs a long time and drains computational resources on the developer's machine
  - ◆ Parallelization over multiple machines?
- Optimizing the crawler was challenging
  - ◆ We had to eliminate different branches of crawl tree that were uninteresting, e.g. publications, news, sports



# Topic Modeler

Models topics for each search result

Karthik Rajagopal



# Why Topic Modeler?

- We wanted to enhance the Expert Search system with the topic modeling of the faculty bio pages to tag search results with top relevant skills
- This feature enhances user experience by enabling him/her to start a new search query based on a relevant tag



# Technology Stack

- Spacy is a fast library and provides a concise API to access its methods and properties
- We used POS Tagging, Named Entity Recognition (NER) and tokenization features of spacy
- Used Tokenization to process text for use with POS tagging and Named Entity Recognition
- POS is used for removing the unwanted and noisy words
- NER feature has helped us find the common entities such as persons, locations, organizations, etc. and distinguish them from noise words in bios
- TinyDB was used to store the final results of Topic Modeling





# Topic Modeler Code Snippet

Code for removing noise word before finding the keywords

```
def isNoise(token):
    is_noise = False
    if token.pos in noisy_pos_tags:
        is_noise = True
    elif token.is_stop == True:
        is_noise = True
    elif len(token.string) <= min_token_length:
        is_noise = True
    elif token.string.lower().strip() in stop_words:
        is_noise = True
    elif token.string.strip() in ents:
        is_noise = True
    return is_noise
```

Code for finding NER:

```
ents = [e.text for e in document.ents]
```

Code for finding top keywords

```
from collections import Counter
cleaned_list = [cleanup(word.string) for word in document if not isNoise(word)]
counts=Counter(cleaned_list).most_common(5)
```

Code for Saving data into TinyDB

```
for key in counts:
    a.append(key[0])

db = TinyDB('Topic_Model_Results.json')
db.insert({'File_Name': i, 'Topic_Modelled_word': a})
```



# Topic Modeler in Action

Below is sample file we used for Topic Modeling (highlights are the topics):

---

```
Home Bio Research Publications Research Group Contact Info Engineering at Illinois Vikram S. Adve Interim Head and Donald B. Gillies Professor|Computer Science
Department|University of Illinois at Urbana-Champaign Research Projects Prof. Vikram Adve Follow the individual project links for more details about each
project and relevant publications: ALLVM: Exploring the benefits for software performance, security and reliability if all software on a system (either all
userspace software or or userspace+OS software) is available in a rich virtual instruction set that can be analyzed and transformed by sophisticated compiler
techniques (think Java bytecode, but for all software). Heterogeneous Parallel Virtual Machine : A compiler infrastructure and parallel program
representation for heterogeneous parallel systems, with the goal of making it much easier to write performance-portable parallel programs. A single program in
the HPVM representation can be compiled to GPUs and multicore CPUs (with and without) vector extensions, while achieving performance close to separately
hand-tuned code for each of those systems. The project is also exploring code generation for FPGAs, for specialized deep-in-memory compute hardware, and
developing optimizing compilers for parallel languages like OpenMP and Domain Specific Languages. Automated Debugging for Software Failures : We are developing
automated static and dynamic analysis techniques to understand the causes of failures in software systems, in order to help programmers diagnose and fix software
bugs with as little effort as possible. The project is investigating automated fault localization and diagnosis techniques for both standalone and distributed
```

You can see he is part of research team and his interest is on compilers. Output from the model:

```
from tinydb import TinyDB, Query
db = TinyDB('Topic_Modelling.json')
result = db.get(Query()['File_Name'] == '2.txt')
print(result)
```

```
{'File_Name': '2.txt', 'Topic_Modelled_word': ['research', 'compiler', 'software', 'students', 'projects']}
```



# Challenges

One of the challenges we faced was to setup the Expert Search system working locally.

Our initial goal was to have the modeled topics displayed on the webpage. We tried different options and as we could not do that, we transformed our code to save the results in TinyDB and retrieve data whenever needed.

Setup instructions of the DB has been added in the Git README page.

---

# End Results



# In the end ...

- Unfortunately many of our assumptions were not correct
  - ◆ Expert System did not run on Linux or Mac
  - ◆ MeTA python library was incompatible with new version of Python
  - ◆ Expert Search Docker container continuously crashed
  - ◆ Even when the server ran, the indexer did not show any results
- Classifier overfits the data and generates inaccurate scores on most university pages

# Live Demo

---