CS4624 Multimedia and Hypertext

Spring 2013

Focused Crawler

WIL COLLINS
WILL DICKERSON
CLIENT: MOHAMED MAGBY AND CTRNET

Department of Computer Science
Virginia Tech
Blacksburg, VA 24061

Date: 5/1/2013

Email: wil.collins@vt.edu
wdickers@vt.edu
mmagdy@vt.edu

EXECUTIVE SUMMARY

The original focused crawler code was very tightly coupled. First of all, constants such as thresholds, filenames, page limits, etc. were hard-coded and scattered amongst several classes. Secondly, there was no way to easily interchange components within the program (e.g. classifiers) since their initializations were hard-coded and their interfaces were not all compatible. In order to alleviate these problems, we created a configuration INI file that the user could easily see and modify. A single ConfigParser object was responsible for reading this file and was only accessed by the top-level driver method, which then passed the appropriate values to objects as they were initialized. The code was also refactored in order to better take advantage of inheritance in order to reduce code reuse and standardize interfaces. Now, the user can switch between classifiers, thresholds, seed files, keywords and more.

The purpose of the focused crawler is to shift the burden of sifting through web pages away from the user. However, it still required the user to categorize the training documents as relevant or not relevant. In an attempt to remove this task, we experimented with using a VSM filter. We vectorized the seed web pages using tf-idf or lsi space and then compared it to each of the training documents. By experimenting with the relevant and irrelevant thresholds, we could label enough training documents for the classifier without the user's input. Unfortunately, this process was rather slow. With over 1500 training documents, the Sikkim collection took upwards of 15 minutes to categorize.

In order to improve the efficiency of the process of training the data model, we utilize an algorithm that takes a small number of seed pages (5-15) provided by the user as input and generates a tf-idf model to test the crawled pages' relevance. To create a leaner data model, we implemented an input sterilizer to remove the data that we could not extract information from.We also wanted to create a more a robust model of the chosen topic so we implemented real-time updates to the model as relevant pages are found during the crawl. This creates a model that better encompasses the entire topic, therefore, better representing the data we are searching for and maintaining the quality of our results as the crawl persists.

TABLE OF CONTENTS

Problem Specification

1.1 Problem

The World Wide Web is incredibly vast, making it difficult to find information one is searching for. Searching for information and determining if it is relevant can take a very long time. We want an automated process to find relevant information.

1.2 What is a crawler?

A crawler is a program that downloads a portion of the web via a breadth-first search. Starting from seed URLs, a crawler will systematically download all links branching outward, and then the links branching from those web pages. And so on, in an exponential and unmindful fashion.

1.3 Why focused crawler?

The user only wants relevant results, so we should make a crawler that tries to maximize relevancy. This is accomplished by the use of a priority queue and a relevance classifier. This classifier is attuned to the interests of the user and is applied to each web page, only those with a high relevancy score are added to the priority queue. There are various vector space models used to determine relevancy (tf-idf, lsi) as well as self-learning statistical models (Naïve Bayesian, SVM).

User Manual

2.1 Repository

Github: https://github.com/wdickers/Focused_Crawler
VTechworks (older): http://vtechworks.lib.vt.edu/handle/10919/19085

2.2 Windows Installation

1. Install Python 2.7 32-bit (http://www.python.org/download/)
2. Add environment variables
   - Create %PYTHONHOME% (e.x. C:\Python27)
   - Add to %PATH%
     - %PYTHONHOME% and %PYTHONHOME%\Scripts
3. Install Nltk library:
   - Numpy: Numerical Python
     http://sourceforge.net/projects/numpy/files/NumPy/1.6.2/numpy-1.6.2-win32-superpack-python2.7.exe
   - NTLK
     http://pypi.python.org/pypi/nltk
   - PyYAML
     http://pyyaml.org/wiki/PyYAML
4. Install scikit-learn library
   - Install setuptools

     https://pypi.python.org/pypi/setuptools#files
   - Install scipy (Scientific Python)
     http://www.scipy.org/Download
   - Install sci-kit-learn
     http://sourceforge.net/projects/scikit-learn/files/
5. Install gensim library
   https://pypi.python.org/pypi/gensim#downloads
6. Install bs4 library (BeautifulSoup)
   http://www.crummy.com/software/BeautifulSoup/#Download
7. Download nltk “corpora\stopwords” data

   ```
   >>> import nltk
   >>> nltk.download()
   # select corporta->stopwards and download to %PYTHONHOME%\nltk_data or %PYTHONHOME%\lib\nltk_data or %USERPROFILE%\nltk_data
   ```

2.3 Usage

To run the focused crawler using the given settings, simply navigate to the folder where the project resides and invoke:

```
> python FocusedCrawler.py
```

In order to change the settings, alter the options within config.ini
In order to begin a new type of collection:

1. Find websites relevant to your topic and place their URLs into seeds.txt, one on each line. It is recommended to use 4 to 10
2. If you are using a classifier (e.g. Naïve Bayes) then you must have enough local documents. It is recommended to have at least 200 relevant and 200 irrelevant documents to train and test the classifier with. Each file path+name goes on a separate line within html_files.txt
3. (optional) Fill labels.txt with 0's for each irrelevant document and 1's for each relevant document. The line numbers in labels.txt and html_files.txt should line up. Leaving this field blank within the configuration file makes Focused Crawler construct a VSM for labeling, which can be inaccurate and slow.
4. Set the configurations within config.ini
5. Run FocusedCrawler.py

Developer's Manual

3.1 Technologies

Language: Python 2.7
Platform: Unix/Linux or Windows
Libraries: gensim, nltk, beautifulsoup, scikit-learn
Other: Internet Archive, Hanzo warc tools

3.2 Architecture

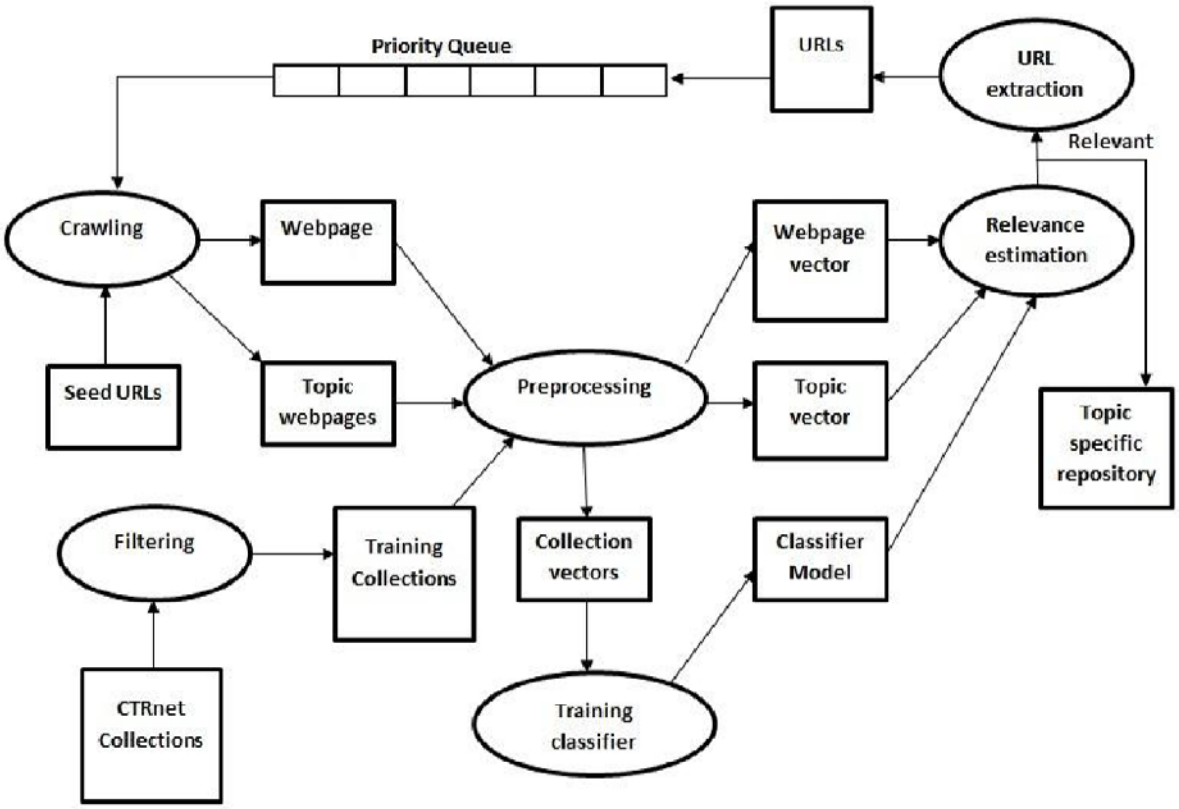


Figure 1: Focused Crawler Architecture

3.3 Model Training Flowchart

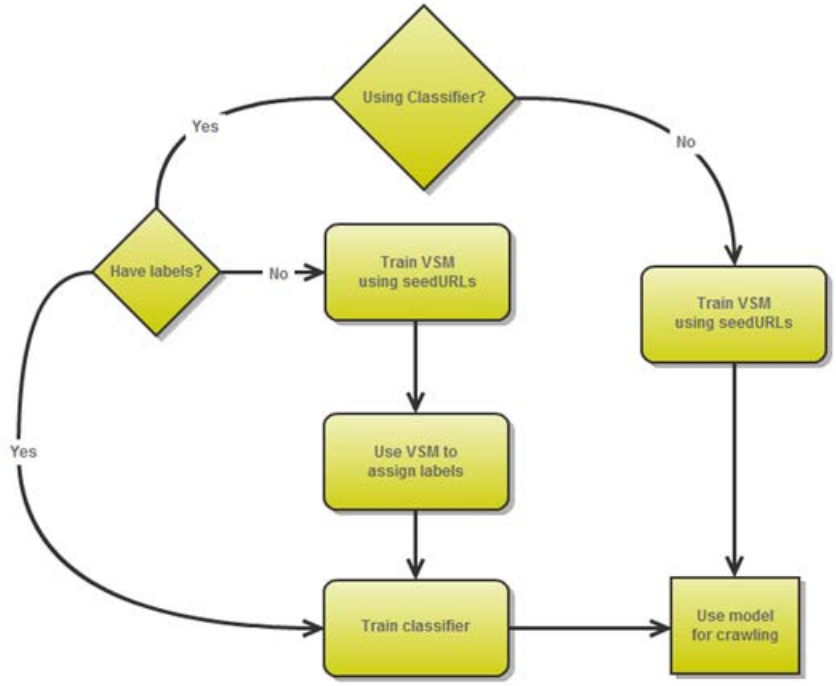


Figure 2: Model Training Flowchart

3.4 Crawling Flowchart

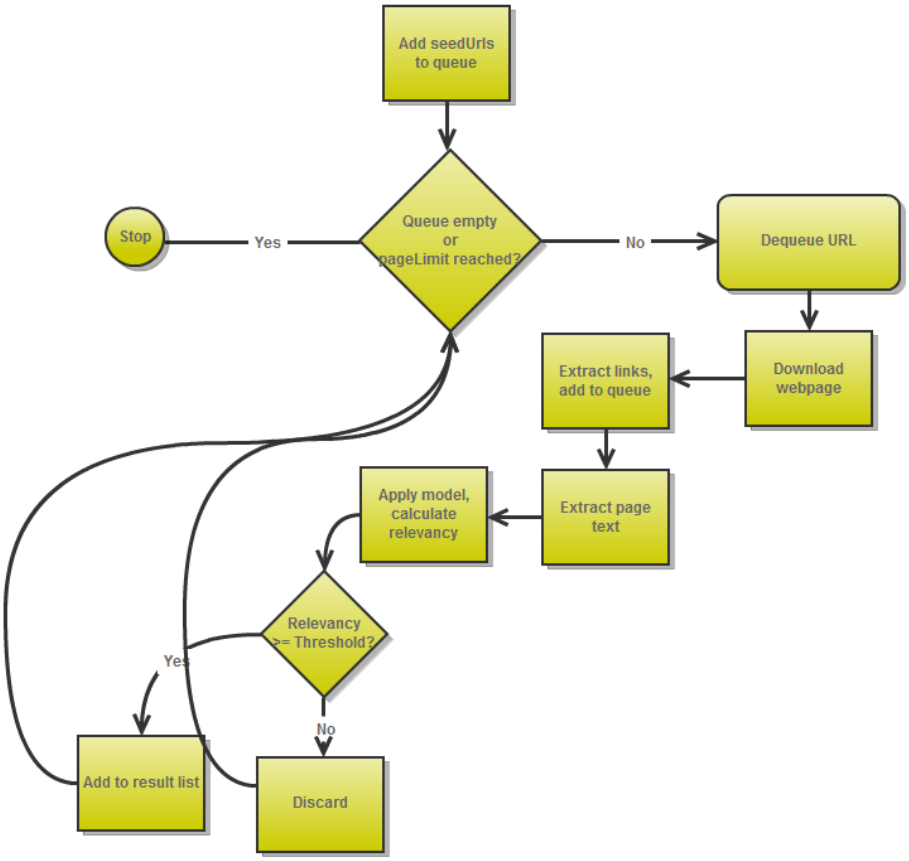


Figure 3: Crawling Flowchart

3.5 File List

FocusedCrawler.py

- Driver class for this project
- Responsible for creating configuration and classifier object and calling crawler

crawler.py

- Crawler class responsible for collecting and exploring new URLs to find relevant pages
- Given a priority queue and a scoring class with a calculate_score(text) method

classifier.py

- Parent class of classifiers (non-VSM) including NaiveBayesClassifier and SVMClassifier
- Contains code for tokenization and vectorization of document text using sklearn
- Child classes only have to assign self.model

config.ini

- Configuration file for focused crawler in INI format

fcconfig.py

- Class responsible for reading configuration file, using ConfigParser
- Adds all configuration options to its internal dictionary (e.g. config["seedFile"])

fcutils.py

- Contains various utility functions relating to reading files and sanitizing/tokenizing text

html_files.txt

- List of local files to act as training/testing set for the classifier ("repository docs")
- Default name, but can be changed in configuration

labels.txt

- 1-to-1 correspondence with lines of repository, assigning numerical categorical label
- 1 for relevant, 0 for irrelevant
- Optional

lsiscorer.py

- Subclass of Scorer representing an LSI vector space model

NBClassifier.py

- Subclass of Classifier, representing a Naïve Bayes classifier

priorityQueue.py

- Simple implementation of a priority queue using a heap

scorer.py

- Parent class of scorers, which are non-classifier models, typically VSM

seeds.txt

- Contains URLs to relevant pages for focused crawler to start
- Default name, but can be modified in config.ini

SVMClassifier.py

- Subclass of Classifier, representing an SVM classifier

tfidfscorer.py

- Subclass of Scorer, representing a tf-idf vector space model

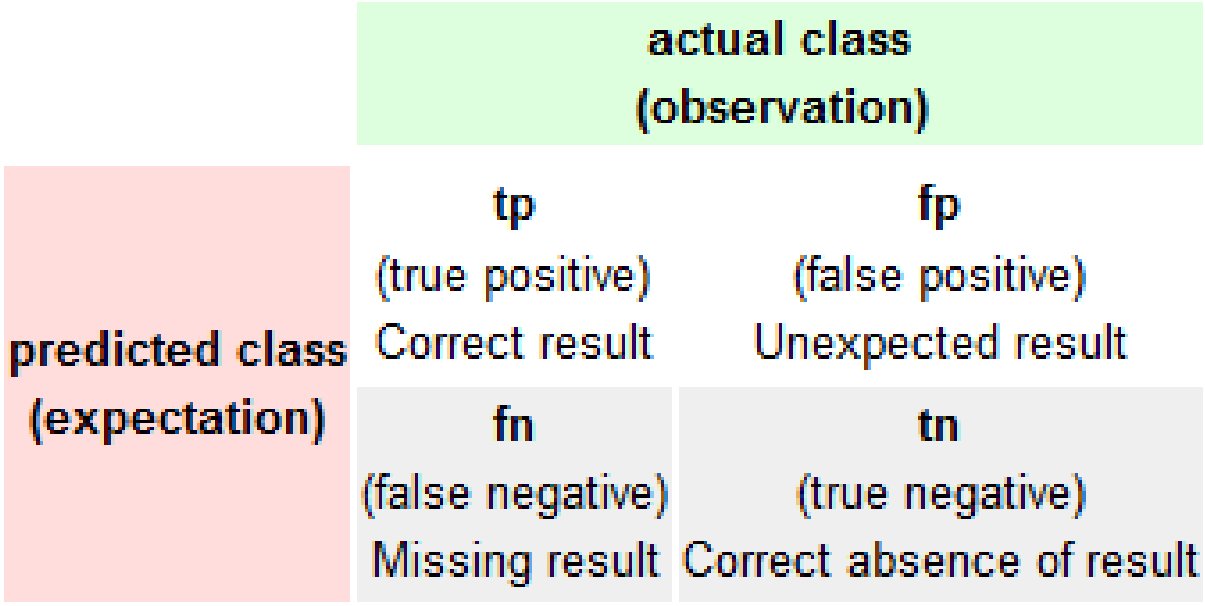
webpage.py

- Uses BeautifulSoup and nltk to extract webpage text

README.txt

- Documentation about Focused Crawler and its usage

3.6 Analysis Methods



Correctly Relevant	Incorrectly Relevant
Incorrectly Irrelevant	Correctly Irrelevant

Table 1: Error analysis

Precision: (correctly predicted relevant) / (predicted relevant)

Recall: (correctly predicted relevant) / (Actual relevant)

The Focused Crawler was run to build a collection about the Boston bombings using a tf-idf scorer. Initially, the scorer was trained only using the seed pages. The code was then improved to allow continuous training as it found more relevant pages.

Initial:	Recall: .4821	Precision: .2842
Final:	Recall: .4252	Precision: .9908

Timeline

2/12/2013: Meet with Mohamed for broad overview of crawlers

2/13/2013 ~ 3/09/2013: Research Python and crawlers (specifically Heretrix)

3/09/2013 ~ 3/16/2013: Spring Break

3/19/2013 : Meet with Mohamed for Focused Crawler demonstration

3/19/2013 ~ 3/24/2013: Look at Focused Crawler code and specific implementation

3/25/2013 ~ 3/27/2013: Midterm presentation

3/27/2013 ~ 4/2013: Meet bi-weekly with Mohamed, worked on Twitter seed generation and semantic analysis

4/20/2013 ~ 5/01/13: Meet bi-weekly with Mohamed, worked on VSM training labels, continuously-learnng tf-idf, and new CTRNet collection

5/06/2013 ~ 5/08/2013: Final presentation: powerpoint, video of focused crawler

Problems & Solutions

5.1 Twitter Seeds

One weakness of focused crawler is the need to supply it with seed URLs in order to begin searching for information about a topic. In an effort to automate this step, we looked at using links posted in Twitter feeds as seeds. However, it seemed that only the last 9 days are kept as logs and it was difficult to determine if a link would be relevant, since tweets do not have much context. We decided to focus more heavily on the classifiers than work on automatic seed generation.

5.2 Semantic Analysis Classifier

The goal was to create a new classifier based on the ontology of a document instead of relying on term frequency ("bag of words"). Having a machine understand text on this human-level turned out to be an ongoing dilemma in the technological community. The most promising solution was an online database of pre-built relational entities; however it was propriety and needed a subscription. The end result was adding the Latent Semantic Indexing (LSI) model as an option for focused crawler. It is still based upon term frequency, however it creates its own list of features that hope to better model the relationship between terms (e.g. "car" and "vehicle" are related).

5.3 Hard-coded Constants

Spread throughout multiple files were hard-coded constants such as relevancy threshold values, seedURL lists, input filenames, and which classifier to use. Not only was this coupling code and making it difficult to change, but it also prevented the user from having easy control over the focused crawler. Our solution was to create a config.ini file that abstracted these constants outside the program. A configParser was written to read this file and the driver method then passed these values to objects as it called them.

5.4 Poor Webpage Text Extraction

Web pages were downloaded in full then "cleaned" by removing comments and tags. However, they were still full of random bytes and junk terms (e.g. from an anchor tag) that were not related to the page's content. The solution we used was to filter out all terms that were not in the nltk dictionaries (which included cities and names).

Final Product

- **Improved Focused Crawler:**
  - Modifiable configurations in config.ini
  - Improved code documentation
  - Ability to use a VSM to determine training labels
  - Improved tf-idf scorer to be continuously learning

- **Github repository**
  - https://github.com/wdickers/Focused_Crawler

- **New CTRNet collection on subject of the Boston bombings**
  - https://github.com/wdickers/Focused_Crawler/blob/master/ctrnet.html

Glossary

LSI: Latent Semantic Indexing

A VSM based on creating new features as a basis for the vectors based on combinations of terms/tokens

NB: Naïve Bayes

A type of classifier model using a simple probabilistic approach by applying Bayes' theorem with strong independence conditions

SVM: Support Vector Machine

A type of classifier model in which the mapped examples are separated by wide gaps where f is the frequency, t is a term, d is a document, and D is the set of all documents

tf-idf: term frequency – inverse document frequency

A VSM based on the frequency of terms/tokens within each document as well as all the documents as a collection. This simple model is often used as the first stop to creating another.

VSM: Vector Space Model

A model in which the document text is composed into a vector of dimension n where n is the number of relevant terms in the model. This allows similarity between two documents to be computed by the cosine similarity (dot product) of their vectors.

References

Farag, M. M. G., Khan, M. S. A., Mishra, G., & Ganesh, P. K. (2012, 12 11). *Focused crawling*. http://vtechworks.lib.vt.edu/handle/10919/19085

Farag, M., Khan, M., Mishra, G., Ganesh, P., & Fox, E. (2012). *Project: Ctrnet focused crawler*. http://vtechworks.lib.vt.edu/bitstream/handle/10919/19085/Technical Report on FocusedCrawler_v2.0.pdf