

Xpantrac Connection with IDEAL

David Cabrera
dcabrera@vt.edu

Erika Hoffman
herika6@vt.edu

Samantha Johnson
sjf2728@vt.edu

Sloane Neidig
sloane10@vt.edu

Client: Seungwon Yang (syang20@gmu.edu)
CS4624, Edward A. Fox

Blacksburg, VA
May 8, 2014

Table of Contents

Table of Contents	2
Table of Figures	4
Abstract	5
User's Manual	6
Command Line.....	6
Developer's Manual.....	8
Inventory of Data Files	8
Xpantrac Explained.....	9
Expansion.....	10
Extraction.....	10
How to Setup Apache Solr.....	11
Download.....	11
Starting the Server.....	11
Indexing	12
Querying	12
WARC Files with IDEAL Documents	13
Python Script to Remove HTML.....	14
Indexing Documents into Solr	14
Attempting to use the IDEAL Pages Script	14
Manually Indexing Documents into Solr.....	15
Concept Map.....	17
Xpantrac for Yahoo Search API	17
File Hierarchy	17
Input Text Files.....	18
Yahoo Search API Authorization	19
Output	20
Xpantrac for Solr.....	20
Finding a Larger Solr Collection	20
Removing Code from Xpantrac_yahooWeb.py.....	20
Changing the URL in Xpantrac	20

Handling the Content Field.....	21
Changing the Xpantrac parameters.....	21
Connecting with IDEAL in the Future.....	22
Configuration File.....	23
Evaluation of Extracted Topics.....	24
File Hierarchy	24
How to Run.....	24
Human Assigned Topics.....	24
Gold Standard Files.....	24
Evaluation Metrics	25
Evaluation	25
Lessons Learned.....	27
Special Note	27
Acknowledgements.....	28
References.....	28

Table of Figures

Figure 1: The 0.txt file used to run the Xpantrac script	6
Figure 2: How to run Xpantrac from the command line (with output).....	7
Figure 3: Components of Xpantrac grouped into two parts.....	10
Figure 4: Shows the command to start the server and initialization output.....	11
Figure 5: Shows the Solr administration page	12
Figure 6: A query of ‘*:*’ that returns all of the documents in the collection	13
Figure 7: URL to the query response	13
Figure 8: Python script to remove all other files except HTML from a directory	14
Figure 9: Text file containing the information from a CNN article.....	15
Figure 11: Command to index ‘50docs.xml’ into Solr	15
Figure 12: XML file using the correct format	16
Figure 13: IOException from indexing the ‘50docs.xml’ file into Solr	16
Figure 14: Xpantrac concept map	17
Figure 15: Creates a list of all file IDs from “plain_text_ids.txt”.....	18
Figure 16: Shows how each input text file is accessed.....	18
Figure 17: Authorization and query information for Yahoo Search API	19
Figure 18: Output from Xpantrac_yahooWeb.py script	20
Figure 19: Importing urlopen to be used for the query request	21
Figure 20: Shows the new query_assembled with ‘content’ as the field name to query in the collection. This can be found in the ‘makeMicroCorpus’ function.	21
Figure 21: Shows the return of the first 30 words of the content field.	21
Figure 22: num_topics represents the number of topics to be found for each input document	21
Figure 23: ‘u_size’ represents the query unit size and a_size represents the API return size.	22
Figure 24: A document from the IDEAL collection in Solr	22
Figure 25: First 30 words of the content field from the IDEAL collection in Solr	22
Figure 26: Xpantrac configuration file	23

Abstract

Title: Integrating Xpantrac into the IDEAL software suite, and applying it to identify topics for IDEAL webpages

Identifying topics is useful because it allows us to easily understand what a document is about. If we organize documents into a database, we can then search through those documents using their identified topics.

Previously, our client, Seungwon Yang, developed an algorithm for identifying topics in a given webpage called Xpantrac. This algorithm is based on the Expansion-Extraction approach. Consequently, it is also named after this approach. In the first part, the text of a document is used as input into Xpantrac and is *expanded* into relevant information using a search engine. In the second part, the topics in each document are identified, or *extracted*. In his prototype, Yang used a standard data set, a collection of one thousand New York Times articles, as a search database.

As our CS4624 capstone project, our group was asked to modify Yang's algorithm to search through IDEAL documents in Apache Solr. In order to accomplish this, we set up and became familiar with a Solr instance. Next, we replaced the prototype's database with the Yahoo Search API to understand how it would work with a live search engine. Then we indexed a set of IDEAL documents into Solr and replaced the Yahoo Search API with Solr. However, the amount of documents we had previously indexed was far too few. In the end, we used Yang's Wikipedia collection in Solr instead. This collection has approximately 4.2 million documents and counting.

We were unable to connect Xpantrac to the IDEAL collection in Solr. This issue is discussed in detail later (along with a future solution). Therefore, our deliverable is Xpantrac for Yang's Wikipedia collection in Solr along with an evaluation of the extracted topics.

User's Manual

Command Line

In the command prompt, the user must navigate to Xpantrac's project directory. Before running the Xpantrac script, the user must ensure there is a document named "0.txt" in that project directory. This document will be used as input to Xpantrac. To run the Xpantrac script, simply type 'python ./Xpantrac.py'. The output in the console will show the query size, each query performed, and a list of topics found in the relevant documents.

(CNN) -- Nine Ringling Bros. and Barnum and Bailey circus performers were among 11 people injured Sunday in Providence, Rhode Island, after an apparatus used in their act failed, circus spokesman Stephen Payne said.

Eight performers fell when the hair-hang apparatus -- which holds performers by their hair -- failed, Payne added. Another performer was injured on the ground, he said.

The performers were among 11 people hospitalized with injuries related to the accident, Rhode Island Hospital spokeswoman Jill Reuter told CNN. One of those people was listed in critical condition, Reuter said.

It was not immediately clear who the other two victims were.

Multiple emergency units responded to the accident at the Dunkin' Donuts Center.

Eyewitnesses told CNN affiliate WPRI that they saw acrobats up on a type of aerial scaffolding doing a "human chandelier" when a cable snapped.

Payne told CNN's Fredricka Whitfield the apparatus had been used for multiple performances each week since Ringling Bros. and Barnum & Bailey launched its "Legends" show in February.

"Each and every time that we come to a new venue, all of the equipment that is used by this performer -- this group of performers as well as other performers -- is carefully inspected. We take the health and safety of our performers and our guests very seriously, and our company has a safety department that spends countless hours making sure that all of our equipment is indeed safe and effective for continued use," he said.

The circus and local authorities are investigating the incident together, Payne said.

"Legends" began a short Providence residency on Friday. The final five performances there were slated for 11 a.m., 3 p.m. and 7 p.m. on Sunday, and 10:30 a.m. and 7 p.m. on Monday.

"The rest of the (11 a.m. Sunday) show was canceled and we're making a determination about the remainder of the shows for the Providence engagement," Payne said.

Figure 1: The 0.txt file used to run the Xpantrac script

```

PS C:\Users\sloan_000\Desktop\project> python .\Xpantrac.py
Input text: 0.txt is being processed.....
---- List of queries (query size:5) ----
[1]: cnn+ringling+bros+barnum+bailey
[2]: bailey+circus+performers+injured+providence
[3]: providence+rhode+island+apparatus+failed
[4]: failed+circus+spokesman+stephen+payne
[5]: payne+performers+fell+hair+hang
[6]: hang+apparatus+holds+performers+hair
[7]: hair+failed+payne+performer+injured
[8]: injured+ground+performers+hospitalized+injuries
[9]: injuries+accident+rhode+island+hospital
[10]: hospital+spokeswoman+jill+reuter+told
[11]: told+cnn+listed+critical+condition
[12]: condition+reuter+clear+victims+multiple
[13]: multiple+emergency+units+responded+accident
[14]: accident+dunkin+donuts+center+eyewitnesses
[15]: eyewitnesses+told+cnn+affiliate+wpri
[16]: wpri+acrobats+type+aerial+scaffolding
[17]: scaffolding+human+chandelier+cable+snapped
[18]: snapped+payne+told+cnn+fredricka
[19]: fredricka+whitfield+apparatus+multiple+performances
[20]: performances+week+ringling+bros+barnum
[21]: barnum+bailey+launched+legends+time
[22]: time+venue+equipment+performer+group
[23]: group+performers+well+performers+carefully
[24]: carefully+inspected+health+safety+performers
[25]: performers+guests+seriously+company+safety
[26]: safety+department+spends+countless+making
[27]: making+equipment+safe+effective+continued
[28]: continued+circus+local+authorities+investigating
[29]: investigating+incident+payne+legends+began
[30]: began+short+providence+residency+final
[31]: final+performances+slated+rest+anceled
[32]: canceled+making+determination+remainder+providence
[33]: providence+engagement+payne

---- Micro corpus is created ----

---- Vector Space Model is applied for topic extraction ----

---- Topics (separated by ',') ----
payne, island, rhode, circus, providence, reuter, american, county, john, state
-----

```

Figure 2: How to run Xpantrac from the command line (with output)

Developer's Manual

Inventory of Data Files

File	Description
./project	Directory containing all project files
./project/Xpantrac.py	Script containing Xpantrac algorithm to be used with Apache Solr
./project/0.txt	Sample input file to be used by algorithm
./project/pos_tagger.py	Part of speech tagger; Trained using the CoNLL2000 corpus provided by the Natural Language Tool Kit (NLTK)
./project/pos_tagger.pyc	Compiled version of './project/pos_tagger.py'
./project/get-pip.py	Package installer
./project/stopwords.txt	A list of words to exclude from the topic identification
./project/custom_stops.txt	A list of words to exclude from the topics identification
./project/Xpantrac_yahooWeb.py	Script containing the Xpantrac algorithm to be used with the Yahoo Search API
./project/plain_text_ids.txt	Text file containing a list of file IDs Used in './project/Xpantrac_yahooWeb.py'
./project/files	Directory of text files with corresponding IDs Used in './project/Xpantrac_yahooWeb.py'
./project/processWarcDir.py	Unpacks a WARC file and returns only html files
./project/CTR_30	A directory of 30 CTR files
./project/VARIOUS_30	A directory of 30 various files
./project/gold_ctr30.csv	The "gold standard" of merged human topics
./project/gold_various30.csv	The "gold standard" of merged human topics
./project/human_topics_CTR30.csv	Human assigned topics for 30 CTR articles

./project/human_topics_VARIOUS30.csv	Human assigned topics for 30 various articles
./project/xpantrac_ctr30_10topics.csv	Xpantrac assigned topics for 30 CTR articles; 10 topics per article
./project/xpantrac_ctr30_20topics.csv	Xpantrac assigned topics for 30 CTR articles; 20 topics per article
./project/xpantrac_various30_10topics.csv	Xpantrac assigned topics for 30 various articles; 10 topics per article
./project/xpantrac_various30_20topics.csv	Xpantrac assigned topics for 30 various articles; 20 topics per article
./project/computePRF1.py	Computes the precision, recall, F1 score of the extracted topics

Xpantrac Explained

Xpantrac is an algorithm that combines Cognitive Informatics with the Vector Space Model to retrieve topics from an input of text. The name Xpantrac came from the **Expansion-Extraction** approach it takes when expanding the query and eventually extracting the topics. Consider this use case of Xpantrac in the following scenario:

Rachel is a librarian working at a children’s library. This library received about 100 short stories, each of which was written by young writers who recently started their literary career. To make these stories accessible online, Rachel decides to organize them based on the topic tags. So, she opens a Web browser and enters a URL of the Xpantrac UI. After loading documents that contain 100 stories, she selects each document to briefly view it, and then extracts suggested topic tags using the UI. After selecting several suggested tags from the Xpantrac UI, and also coming up with additional tags by herself, she enters them as the topic tags representing a story. A library patron, Jason, accesses the library homepage at home, clicks a tag “Christmas”, which lists 5 stories about Christmas. He selects a story that might be appropriate for his 4-year daughter, and reads the story to her. (Yang, 90)

The design of Xpantrac has two parts: Expansion and Extraction. The flow of the algorithm can be shown in the figure below.

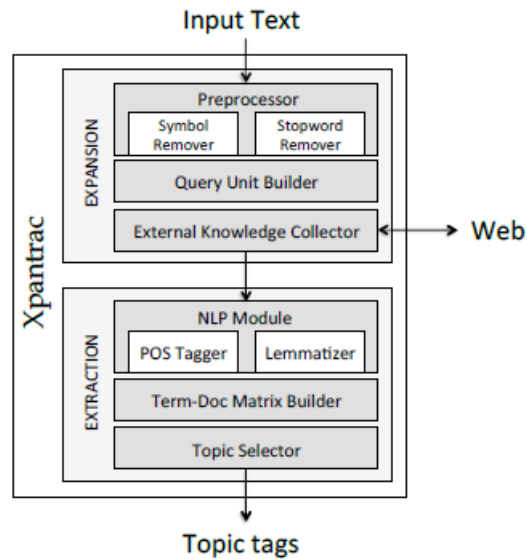


Figure 3: Components of Xpantrac grouped into two parts

Because of the modular design of Xpantrac, any component can be flexibly replaced. For our project, we used a web API as the External Knowledge Collector on the first run and later replaced it with a Solr system.

Expansion

The Expansion part of the algorithm is responsible for building a “derived corpus” of relevant information by accessing an external knowledge source by expanding input text. This part contains three parts:

- 1) Preprocessor: removes symbol characters (e.g. &, #, \$,) and stopwords (e.g. ‘a’, ‘and’, ‘the’)
- 2) Query Unit Builder: segments the preprocessed input texts into uniform sized groups of words. The words are grouped with neighboring words to keep the context.
- 3) External Knowledge Collector: accesses a knowledge source, located outside the system, to search and retrieve relevant information on the queries sent

Extraction

The extraction part is where a list of words is derived from the corpus created from expansion. This part contains three parts:

- 1) NLP Module: applies a POS (Part of Speech) tagger to the corpus to select only nouns, verbs, or both. It also finds “lemmas” of the nouns or verbs to resolve singular and plural forms

- 2) Term-Doc Matrix Builder: develops a term index using the unique words from the derived corpus and constructs a term-document matrix as in the Vector Space Model
- 3) Topic Selector: identifies significant words representative of the input text

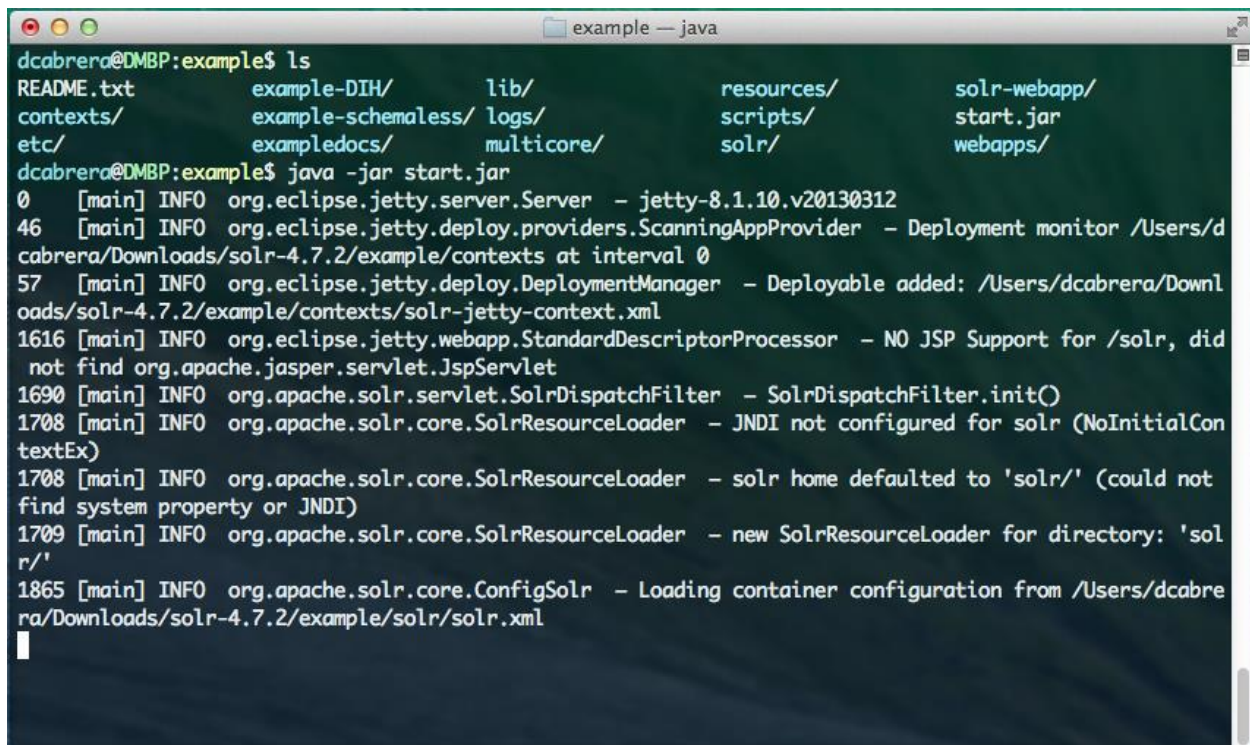
How to Setup Apache Solr

Download

In order to setup Solr, you need to have the latest Java JRE installed on your system. At the time of this writing, the current version of Java, Java 8, is fully compatible with Apache Solr but previous versions can be used if desired. Once the latest Java is installed, you can download Apache Solr.

Starting the Server

Once Solr is downloaded, you can run the server in its template form by navigating to [solr download]/example. From here, running “java -jar start.jar” starts the server. You can then navigate to <http://localhost:8983/solr/>. If the server is successfully started, you should be able to see the administrator page. The figure below shows the command to start the server and what a developer should see when initializing the server.



```
dcabrera@DMBP:example$ ls
README.txt      example-DIH/    lib/            resources/      solr-webapp/
contexts/       example-schemaless/ logs/           scripts/        start.jar
etc/            exampledocs/   multicore/     solr/           webapps/
dcabrera@DMBP:example$ java -jar start.jar
0 [main] INFO org.eclipse.jetty.server.Server - jetty-8.1.10.v20130312
46 [main] INFO org.eclipse.jetty.deploy.providers.ScanningAppProvider - Deployment monitor /Users/dcabrera/Downloads/solr-4.7.2/example/contexts at interval 0
57 [main] INFO org.eclipse.jetty.deploy.DeploymentManager - Deployable added: /Users/dcabrera/Downloads/solr-4.7.2/example/contexts/solr-jetty-context.xml
1616 [main] INFO org.eclipse.jetty.webapp.StandardDescriptorProcessor - NO JSP Support for /solr, did not find org.apache.jasper.servlet.JspServlet
1690 [main] INFO org.apache.solr.servlet.SolrDispatchFilter - SolrDispatchFilter.init()
1708 [main] INFO org.apache.solr.core.SolrResourceLoader - JNDI not configured for solr (NoInitialContextEx)
1708 [main] INFO org.apache.solr.core.SolrResourceLoader - solr home defaulted to 'solr/' (could not find system property or JNDI)
1709 [main] INFO org.apache.solr.core.SolrResourceLoader - new SolrResourceLoader for directory: 'solr/'
1865 [main] INFO org.apache.solr.core.ConfigSolr - Loading container configuration from /Users/dcabrera/Downloads/solr-4.7.2/example/solr/solr.xml
```

Figure 4: Shows the command to start the server and initialization output

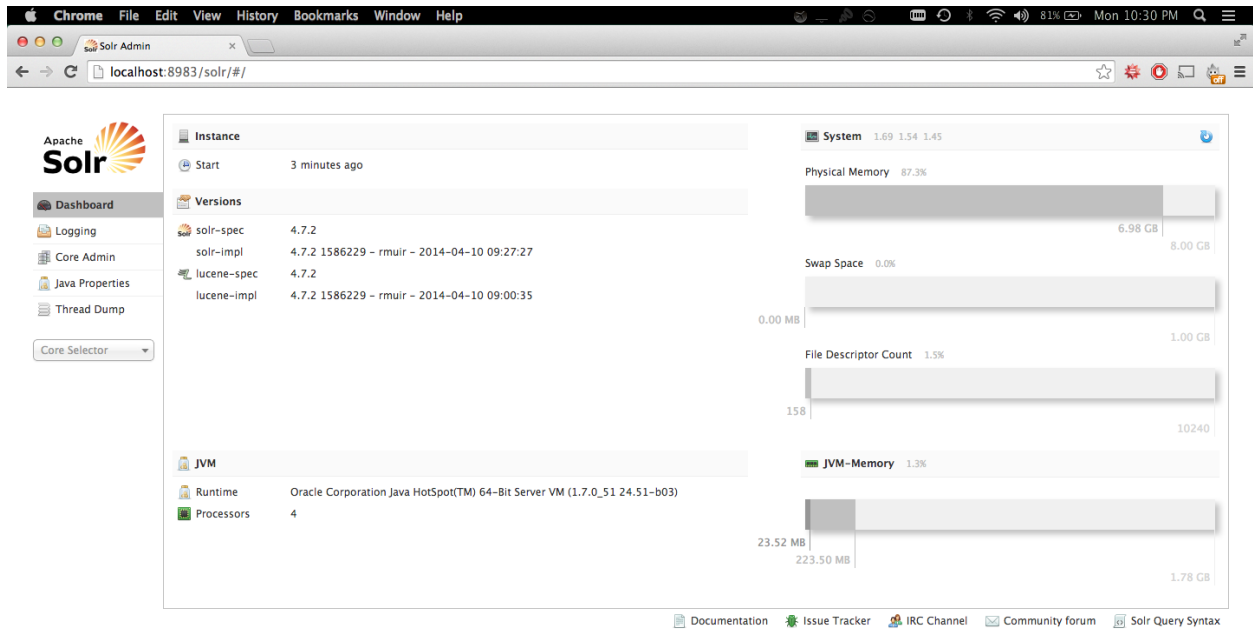


Figure 5: Shows the Solr administration page

Indexing

To index documents with the default setup of the Solr server, you can use the post.jar file that is located in the exampledocs folder. You can copy and paste the post.jar file into any folder and do the command “java -jar post.jar [file name here]”. Once you run post, it uploads the files to servers and they are indexed.

Querying

To query the files you have indexed, you choose the Solr collection to search (for the default setup, the collection is named Collection 1). Once you choose the collection from the administrator page, you can select the Query tab to see the Query menu. From here you have a lot of options when you search. What we are most concerned about is the ‘q’ box containing the “*:*” query. The left asterisk indicates the tag you want to search in (you can leave the * to search all tags) and the right asterisk indicates the content you want to search within the tag. Searching “*:*” returns all of the documents that are contained within the server.

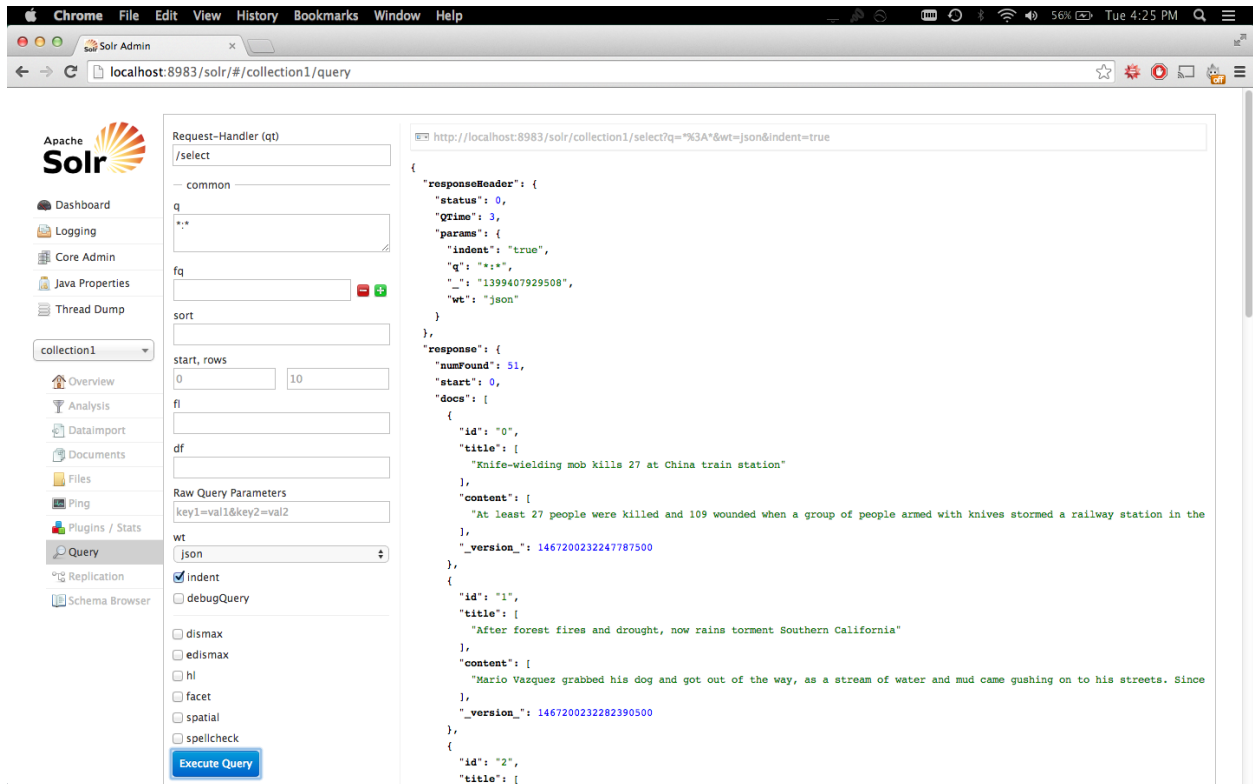


Figure 6: A query of ‘*:*’ that returns all of the documents in the collection

The link at the top of the query gives you the general structure of a query if you do not want to use the Admin page.

http://localhost:8983/solr/collection1/select?q=%3A*&wt=json&indent=true

Figure 7: URL to the query response

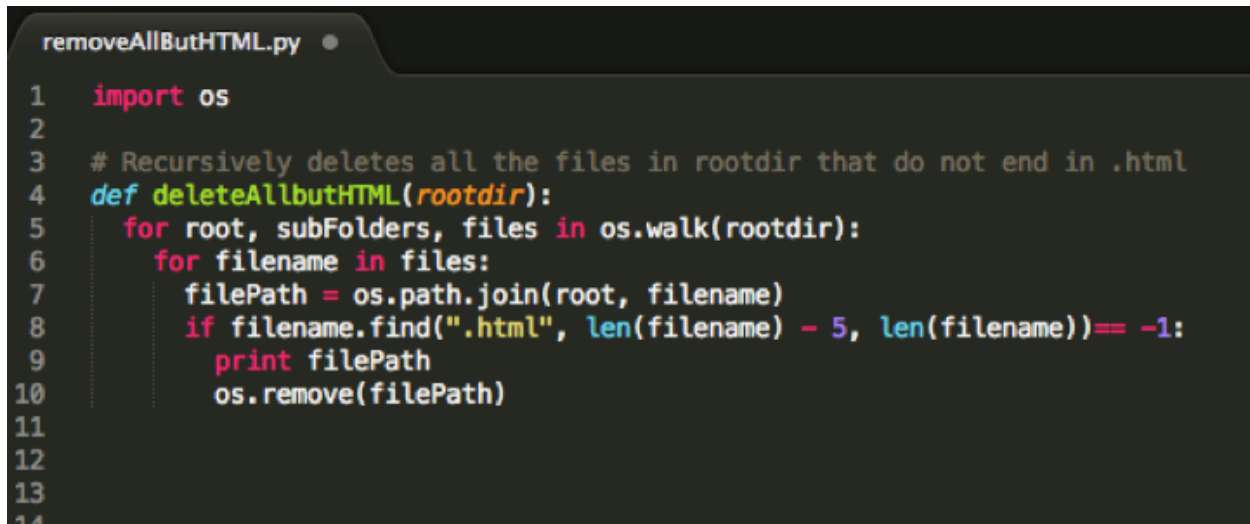
From here, the “*” in the link represent the the things we search for and you can replace the asterisks with the queries of your choice. This link stays constant for all queries. Another option that you see is the part that says “json”. Here, you can change it to return “json”, “xml”, “python”, “ruby”, “php”, or “csv”.

WARC Files with IDEAL Documents

Our group collaborated with the IDEAL Pages group for the initial part of our project since we were both working with IDEAL and Solr. The IDEAL Pages group goal was to index the IDEAL documents into Solr. To achieve this goal, they had been given a set of WARC files containing IDEAL documents in the form of HTML pages. However, the WARC files also contained non-HTML documents that were unnecessary for our purposes. After the IDEAL Pages group created a Python script to unpack the WARC files, they sent it to us for further modification.

Python Script to Remove HTML

As stated before, the WARC files included the HTML documents we needed, but they also included a lot of other files we did not need. Figure 8 shows the Python script we created to remove all of the unnecessary documents.



```
removeAllButHTML.py ●
1  import os
2
3  # Recursively deletes all the files in rootdir that do not end in .html
4  def deleteAllbutHTML(rootdir):
5      for root, subFolders, files in os.walk(rootdir):
6          for filename in files:
7              filePath = os.path.join(root, filename)
8              if filename.find(".html", len(filename) - 5, len(filename)) == -1:
9                  print filePath
10                 os.remove(filePath)
11
12
13
14
```

Figure 8: Python script to remove all other files except HTML from a directory

This script recursively deletes all of the files in a root directory that do not end with the HTML extension. When running the script, the only parameter needed is the path to the root directory where the files are located. The full path to each deleted file is printed as it is removed.

Indexing Documents into Solr

Attempting to use the IDEAL Pages Script

As mentioned before, the IDEAL Pages group goal was to index IDEAL documents into Solr. Our group also needed to do this in order to later use IDEAL documents with Xpantrac. After speaking with our professor and primary contacts, our groups were asked to work together. The IDEAL Pages group would supply the Xpantrac group with the script to index documents into Solr and the Xpantrac group would manually index the documents until that script was created.

When the IDEAL Pages script was finally received, it would not run with our Solr instance. Our group spent a lot of time trying to fix the script and get it to run with our instance. The IDEAL Pages group was also unable to help. Eventually, we realized that we would rather spend time manually indexing the files into Solr instead of trying to fix a script that may never work for us.

Manually Indexing Documents into Solr

Initially, we had 50 text documents from CNN that were supposed to be indexed into Solr (See Figure 9). These documents would represent documents from the IDEAL collection. However, Solr needed those documents to be in XML format (See Figure 10).

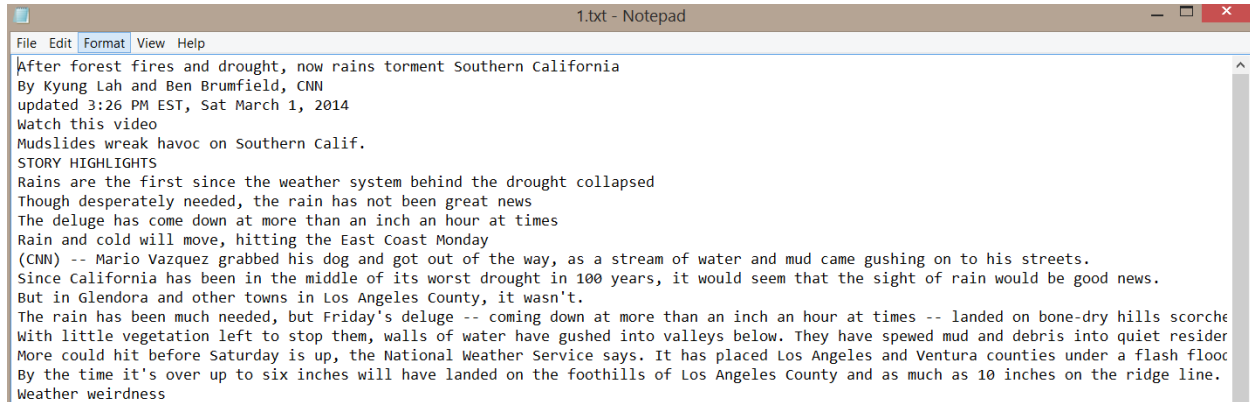


Figure 9: Text file containing the information from a CNN article

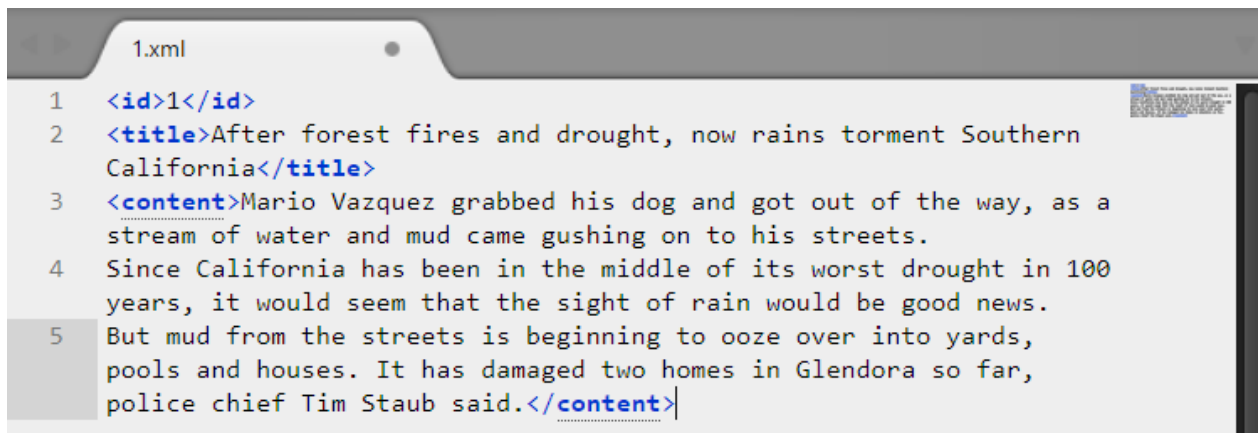


Figure 10: XML file containing the information from the text file in Figure 9

Next, we tried to manually index those XML files into Solr using the command line.



Figure 11: Command to index '50docs.xml' into Solr

However, we ran into an error. After examining Solr's schema.xml file and reviewing some tutorials, we realized that we had been formatting our XML files incorrectly for Solr. The correct formatting can be seen in Figure 12.

```
50docs.xml
1  <add>
2  <doc>
3      <field name="id">0</field>
4      <field name="title">Knife-wielding mob kills 27 at China train station</field>
5      <field name="content">At least 27 people were killed and 109 wounded when a group of people armed
6      with knives stormed a railway station in the southwest Chinese city of Kunming, authorities said,
7      according to state news agency Xinhua.
8      It was an organized, premeditated terrorist attack, authorities told the news agency. No motive has
9      been provided. A doctor with the Kunming No.1 People's Hospital told Xinhua over the phone they're
10     not sure of the number of casualties. Xinhua said the Kunming Railway Station is one of the largest
11     stations in southwest China.</field>
12 </doc>
13 <doc>
14     <field name="id">1</field>
15     <field name="title">After forest fires and drought, now rains torment Southern California</field>
16     <field name="content">Mario Vazquez grabbed his dog and got out of the way, as a stream of water and
17     mud came gushing on to his streets. Since California has been in the middle of its worst drought in
18     100 years, it would seem that the sight of rain would be good news.
19 </doc>
```

Figure 12: XML file using the correct format

Initially, we had 50 separate XML files for each of the 50 articles. However, we learned that we were able to combine these into one long XML file, with each article in its own `<doc>` tag. When we tried to index the '50docs.xml' file into Solr, we received the error seen in Figure 13.

```
dcabrera@DMBP:exampledocs$ java -jar post.jar 0.xml
SimplePostTool version 1.5
Posting files to base url http://localhost:8983/solr/update using content-type application/xml..
POSTing file 0.xml
SimplePostTool: WARNING: Solr returned an error #500 Server Error
SimplePostTool: WARNING: IOException while reading response: java.io.IOException: Server returned HTTP
response code: 500 for URL: http://localhost:8983/solr/update
1 files indexed.
COMMITting Solr index changes to http://localhost:8983/solr/update..
Time spent: 0:00:00.072
dcabrera@DMBP:exampledocs$
```

Figure 13: IOException from indexing the '50docs.xml' file into Solr

The issue was caused by the existence of ampersand characters ('&') in the XML file we tried to index. To fix this problem we removed the ampersands and then ran the indexing command again. The files were then able to be indexed into our local Solr instance without any more issues.

Concept Map

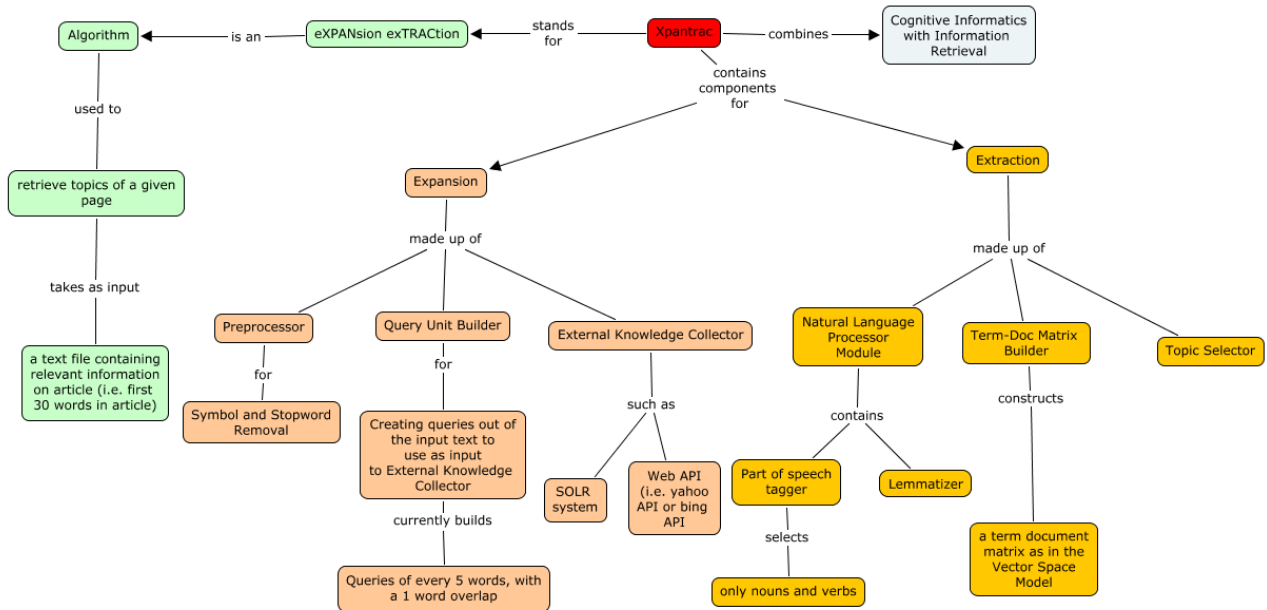


Figure 14: Xpantrac concept map

Xpantrac for Yahoo Search API

For our midterm presentation, we tried to modify Yang's original Xpantrac script that used a database to instead use the Bing Search API. However, we ran into multiple authentication issues. As a result of these problems, we modified the original Xpantrac script to use the Yahoo Search API.

File Hierarchy

File	Description
./project	Directory containing all project files
./project/Xpantrac_yahooWeb.py	Script containing the Xpantrac algorithm to be used with the Yahoo Search API
./project/plain_text_ids.txt	Text file containing a list of file IDs Used in './project/Xpantrac_yahooWeb.py'
./project/files	Directory of text files with corresponding IDs Used in './project/Xpantrac_yahooWeb.py'

Input Text Files

The Xpantrac_yahooWeb.py script used a plain_text_ids.txt file to identify all of the IDs of the text files to be used as input. These text files can be found in the ./project/files directory. The IDs for the text files are simply 0-50 and the text files themselves are named 0.txt -50.txt, respectively. Figures 15 and 16 show how the files are accessed in the Xpantrac for Yahoo script.

```
# develop id list
fi = open("plain_text_ids.txt", "r")
li = fi.read().split()
fi.close()
```

Figure 15: Creates a list of all file IDs from "plain_text_ids.txt"

```
print "\n----- Document ID: %s is being processed -----\n" % doc_id
filename = str(filenum) + ".txt"
# for linux/mac machine
text = open("files/"+filename, "r").read()
# for Windows machine:
text = open("files/"+filename, "r").read()
```

Figure 16: Shows how each input text file is accessed

Yahoo Search API Authorization

Querying the Yahoo Search API required authorization. Therefore, this script had a few extra authorization lines than normal. Figure 17 shows the necessary authorization and query information.

```
if query != "":
    try:
        url = ""
        if yahoo_api_type == "web":
            url = "http://yboss.yahooapis.com/ysearch/web?q=" + query
        else:
            url = "http://yboss.yahooapis.com/ysearch/news?q=" + query
        consumer = oauth2.Consumer(key=OAUTH_CONSUMER_KEY,secret=OAUTH_CONSUMER_SECRET)
        params = {
            'oauth_version': '1.0',
            'oauth_nonce': oauth2.generate_nonce(),
            'oauth_timestamp': int(time.time()),
        }

        oauth_request = oauth2.Request(method='GET', url=url, parameters=params)
        oauth_request.sign_request(oauth2.SignatureMethod_HMAC_SHA1(), consumer, None)
        oauth_header=oauth_request.to_header(realm='yahooapis.com')

        # Get search results
        http = httplib2.Http()
        resp, content = http.request(url, 'GET', headers=oauth_header)
        # print resp
        # print content
        results = simplejson.loads(content)
```

Figure 17: Authorization and query information for Yahoo Search API

Output

See Figure 18 for instructions on how to run the Xpantrac for Yahoo script in the command prompt. This figure also shows the list of topics (output) for each document processed.

```
PS C:\Users\sloan_000\Desktop\project> python .\Xpantrac_yahooWeb.py
1399317485.37

----- Document ID: 0 is being processed -----
----- m39 10 Topics -----
station,people,attack,news,railway,xinhua,china,train,group,knife

29.3789999485 seconds
1399317514.75

----- Document ID: 1 is being processed -----
----- m39 10 Topics -----
water,rain,california,weather,drought,los,angeles,storm,fire,street

81.75 seconds
1399317596.51

----- Document ID: 2 is being processed -----
```

Figure 18: Output from Xpantrac_yahooWeb.py script

Xpantrac for Solr

Finding a Larger Solr Collection

After we successfully indexed our 50 CNN documents into Solr, we found out that 50 files is too small a number to enable Xpantrac to work correctly. Instead, we ended up using Yang's collection of Wikipedia articles on Solr. This collection currently holds 4.2 million documents (and counting).

Removing Code from Xpantrac_yahooWeb.py

First, we removed all of the database code (and 'db' variables) from the Xpantrac_Yahoo.py script. This database held one thousand New York Times articles. Solr will replace this database, so we can remove it and the 'import MySQLdb' statement.

Changing the URL in Xpantrac

After obtaining the URL to Yang's Wikipedia collection in Solr, we created a new query request in Xpantrac. First, we had to import 'urlopen' as seen in Figure 19.

```
from urllib import urlopen
```

Figure 19: Importing urlopen to be used for the query request

Next, we had to modify the 'query_assembled' with the correct URL and field name.

```
for item in query_list: # [[query unit 1], [query unit 2], [query unit 3],...]
    query = "+".join(item)
    num_results_returned = 0
    if query != "":
        try:
            query_assembled = 'http://[redacted]/solr/collection1/select?q=content%3A'+ query + '&wt=json&indent=true&rows=5';
            conn = urlopen(query_assembled)
            rsp = eval( conn.read() )
            results = rsp['response']['docs']
```

Figure 20: Shows the new query_assembled with 'content' as the field name to query in the collection. This can be found in the 'makeMicroCorpus' function.

Handling the Content Field

In addition to changing the query field to 'content' in the query_assembled for the request, we also had to change the field name in the configuration for the results seen later in the code. First, we changed the field name to 'content'. Next, we returned on the first 30 words of the content field. Only the first 30 words are used because they tend to represent the key issues of an entire document. The field change can be seen in Figure 21.

```
# for M_43 configuration (only 10 results merged) -----//
for result in results[0:10]:
    short_result = " ".join(result['content'][0].split()[:30])
    clean_result = short_result.replace("...", "").strip().replace("\'", "'")
```

Figure 21: Shows the return of the first 30 words of the content field.

Because we are no longer using the Yahoo Search API, we also removed all of the authorization code that enabled us to access that API.

Changing the Xpantrac parameters

With Yang's help, the number of topics for Xpantrac to find was changed to 10, the number of API results to return to be 10, and the query unit size to be 5. These changes can be seen in Figures 22 and 23.

```
def main():
    num_topics = 10
    window_overlap = 1
```

Figure 22: num_topics represents the number of topics to be found for each input document

```

# input text ----- (Control inputs)
iter_id = 1
# for u_size in [20, 15, 10, 5, 1][3:4]:
for u_size in [20, 15, 10, 5, 2, 1][3:4]:      # [3:4] -> group 5 words together
    for a_return in [50, 10, 5, 1][1:2]:      # [1:2] -> ask Solr to return 10 matching documents

```

Figure 23: 'u_size' represents the query unit size and a_size represents the API return size. This can be found in the 'main' function.

Connecting with IDEAL in the Future

In the future, Xpantrac should connect to the IDEAL collection in Solr. This collection can be found at <http://nick.dlib.vt.edu:8080/solr/#/collection1/query>. While this collection does contain a 'content' field, it does not meet the specifications of our project at this time.

The IDEAL Pages group was given a different specification to use for the content of their Solr collection. Their group was instructed to collect the entire content of an HTML page. This means that all of the text in the <body> of an HTML page will be put into their 'content' field. Figure 24 shows an example of a content field.

```

{
  "content": [
    "Google Newsvar GLOBAL_window=window; (function() {function
    d(a) {this.t={};this.tick=function(a,c,b) {b=void 0!=b?b:(new
    Date).getTime();this.t[a]=[b,c]};this.tick(\"start\",null,a)}var a=new
    d;GLOBAL_window.jstiming={Timer:d,load:a};if(GLOBAL_window.performance&&GLOBAL_windo
    w.performance.timing) {var
    a=GLOBAL_window.performance.timing,c=GLOBAL_window.jstiming.load,b=a.navigationStart
    ,a=a.responseStart;0<b&&a>=b&&(c.tick(\"_wtsrt\",void
    0,b),c.tick(\"_wtsrt_\", \"_wtsrt\",a),c.tick(\"tbsd_\", \"_wtsrt_\"))}try{a=null,GLOBAL
    _window.chrome&&GLOBAL_window.chrome.csi&&\n(a=Math.floor(GLOBAL_window.chrome.csi()
    .pageT),c&&0<b&&(c.tick(\"_tbnd\",void...\"),
    "collection_id": "3650",
    "id": "7f74825401865487f671bd0fd388ce2b",
    "_version_": 1465938356823130000
  },

```

Figure 24: A document from the IDEAL collection in Solr

As you can see, there is a lot of unnecessary text and JavaScript inside of the 'content' field. If Xpantrac thought that this page was a match and we returned the first 30 words of the content field, it would look like:

```

"Google Newsvar GLOBAL_window window function function d a this t this tick
functiona a c b b void b b new Date getTime this t a b c this tick start null a
var a new d GLOBAL_window jstiming"

```

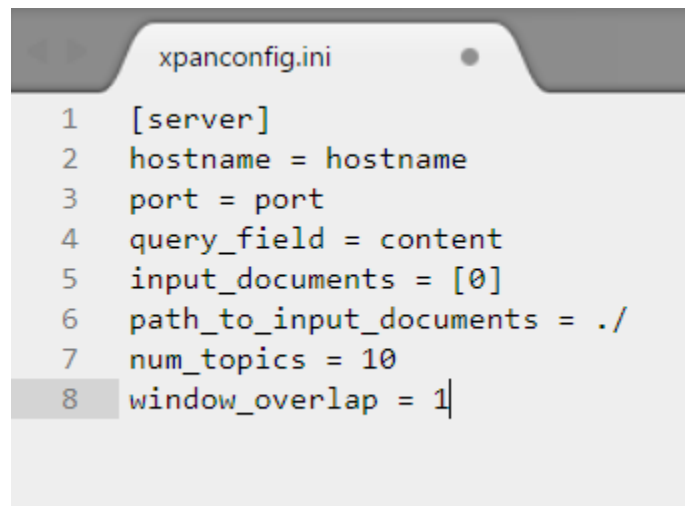
Figure 25: First 30 words of the content field from the IDEAL collection in Solr

Therefore, we are unable to use the Solr collection at this time because the project specifications for IDEAL Pages and Xpantrac were not the same. If an IDEAL collection is created to match our specifications, then you would only need to change the URL to match the collection URL and the field name to match the field containing the relevant content information.

For example, if the current IDEAL collection in Solr is changed to suit our specifications, then you would only need to change the hostname and port to match the corresponding URL.

Configuration File

In order to help create an easier Xpantrac experience for future developers, we have created a configuration file. This file will allow users to enter commonly changed variables, such as hostname, port, query field, title of input documents, path to input documents, number of topics to be found, and window overlap.

A screenshot of a text editor window titled 'xpanconfig.ini'. The window contains the following configuration text:

```
1 [server]
2 hostname = hostname
3 port = port
4 query_field = content
5 input_documents = [0]
6 path_to_input_documents = ./
7 num_topics = 10
8 window_overlap = 1
```

Figure 26: Xpantrac configuration file

Because of this configuration file, there is no longer a need to change variables directly in the Xpantrac script. This will help ensure that all variables are changed correctly when a new user wishes to use the system.

Evaluation of Extracted Topics

File Hierarchy

File	Description
./project/CTR_30	A directory of 30 CTR files
./project/VARIOUS_30	A directory of 30 various files
./project/gold_ctr30.csv	The “gold standard” of merged human topics
./project/gold_various30.csv	The “gold standard” of merged human topics
./project/human_topics_CTR30.csv	Human assigned topics for 30 CTR articles
./project/human_topics_VARIOUS30.csv	Human assigned topics for 30 various articles
./project/xpantrac_ctr30_10topics.csv	Xpantrac assigned topics for 30 CTR articles; 10 topics per article
./project/xpantrac_ctr30_20topics.csv	Xpantrac assigned topics for 30 CTR articles; 20 topics per article
./project/xpantrac_various30_10topics.csv	Xpantrac assigned topics for 30 various articles; 10 topics per article
./project/xpantrac_various30_20topics.csv	Xpantrac assigned topics for 30 various articles; 20 topics per article
./project/computePRF1.py	Computes the precision, recall, F1 score of the extracted topics

How to Run

```
> python computePRF1.py gold_ctr30.csv xpantrac_ctr30_#topics.csv  
> python computePRF1.py gold_various30.csv xpantrac_various30_#topics.csv
```

Human Assigned Topics

Two sets of test files, CTR_30 and VARIOUS_30, were included in this project. These files have been tagged with topics by multiple human sources. The people who tagged these articles were from the Library Sciences field, so they were experienced taggers. The human assigned topics for each file can be found in human_topics_CTR30.csv and human_topics_VARIOUS30.csv.

Gold Standard Files

The gold standard files are a merged version of the human assigned topics. That means that if Tagger A said that a file’s topics are “Florida, marsh, tropical, coast” and Tagger B said that same file’s topics are “marsh, storm, Jacksonville”, then those topics would be merged in the

gold standard file. Therefore, the gold standard of topics for that file would be “Florida, marsh, tropical, coast, storm, Jacksonville”.

Evaluation Metrics

This evaluation of topics measures precision, recall, and F1. Precision can be used to compute the proportion of matching topics (i.e., C) from all the retrieved topics (i.e., A) by the following formula:

$$precision = \frac{|C|}{|A|} = P(relevant | retrieved)$$

Recall is the proportion of the matching topic (i.e., C) from all of the retrieved topics (i.e., B), which are assigned by the human topic indexers or exist as the gold standard:

$$recall = \frac{|C|}{|B|} = P(retrieved | relevant)$$

Ideally, both the precision and recall values should be 1. This would mean that the sets of topics compared would be exactly the same.

The F1 score is used to compare precision and recall with the following formula:

$$F_1 = \frac{2 * precision * recall}{precision + recall}$$

Evaluation

The tables below show the evaluation of average precision, recall, and F1 of the gold standard of topics versus 10 Xpantrac topics.

```
> python computePRF1.py gold_ctr30.csv xpantrac_ctr30_10topics.csv
```

Evaluation	Value
Average Precision	0.4534
Average Recall	0.2110
Average F1	0.2800

```
> python computePRF1.py gold_various30.csv xpantrac_various30_10topics.csv
```

Evaluation	Value
Average Precision	.5922
Average Recall	.1640
Average F1	.2547

Above, the number of human assigned topics are much larger than the number of Xpantrac topics (10). Because of this, the recall value will be somewhat low. Increasing the number of Xpantrac topics from 10 to a larger number, such as 20, will increase the recall value. Eventually, the F1 measure will increase as well. However, the precision value may decrease slightly. Below are the average precision, recall, and F1 scores for the increased number of topics (20).

```
> python computePRF1.py gold_ctr30.csv xpantrac_ctr30_20topics.csv
```

Evaluation	Value
Average Precision	0.3067
Average Recall	0.2608
Average F1	0.2777

```
> python computePRF1.py gold_various30.csv xpantrac_various30_20topics.csv
```

Evaluation	Value
Average Precision	0.4000
Average Recall	0.2221
Average F1	0.2824

As expected, the precision value has decreased and the recall value has increased. Overtime, we should still expect the F1 score to increase.

Lessons Learned

This capstone project was definitely an eye-opening experience for all of us. We had never done this type of work in any of the courses from our past semesters before. Because of this, we felt that we learned a lot of lessons and gained a lot of experience.

While all of our group members had previous experience working in a team, none of us had ever had to coordinate with another separate team before. Overall, we felt that there was a good deal of miscommunication between our group and the IDEAL Pages group. Throughout the semester, we were under the impression that some of our project goals overlapped with their project goals. However, this was not the case. In hindsight, we should have made our objectives more clear with the other group and ensured that we had a better understanding of their project goals. We had initially thought they could help us accomplish some of our tasks, so we waited for them to finish one of their deliverables so that they could share it with us. It turned out that this particular deliverable did not accomplish the same thing we needed, so we wasted time waiting on it.

Another lesson learned was dealt with Apache Solr. We were very confused about the purpose of Solr when we first started our project. Additionally, we were unsure how to use it. We did not understand how to index or query files, so we had to find a lot of tutorials (some of which were misleading) or ask our primary contact. However, these tasks became more clear after we had the guest lecture from Tarek Kanan about Solr and completed the Solr assignments for homework. We hope that in the future the Solr activity will be moved toward the beginning of the semester instead of the end. We believe that we would have experienced less troubles if the course had been structured this way.

Overall, we gained a lot of knowledge regarding tools that were new to us, such as Solr and Yahoo Search API. We are glad to have the experience of working with Yang's code and hope that his research can be carried on in the future.

Special Note

Yang has requested that the URL to the GMU Wikipedia Solr collection be redacted as it should not yet be public. This explains the blackened hostname and port in Figure 20.

Acknowledgements

We would first like to thank Seungwon Yang for taking the time out of his busy schedule at George Mason University to help our group better understand the Xpantrac algorithm and goals for this capstone project.

We would also like to mention Mohamed Magdy and IDEAL Pages group (consisting of Mustafa Aly and Gasper Gulotta) for their contributions to the initial part of our project. The IDEAL Pages project goal was to index the IDEAL documents into Solr.

Lastly, we would like to thank Dr. Edward Fox for presenting us with the opportunity to work on and improve this project for our capstone class and the National Science Foundation (NSF) for supporting the Integrated Digital Event Archiving and Library (IDEAL) organization.

References

Yang, Seungwon. Automatic Identification of Topic Tags from Texts Based on Expansion-Extraction Approach. Diss. Virginia Polytechnic Institute and State University, 2013, 230 pages. <<http://hdl.handle.net/10919/25111>>.